# AVERTISSEMENT



# LIENS

Code la Propriété Intellectuelle – Articles L. 122-4 et L. 335-1 à L. 335-10
Loi n° 92-597 du 1er juillet 1992, publiée au *Journal Officiel* du 2 juillet 1992

http://www.cfcopies.com/V2/leg/leg-droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Université Fédérale
Toulouse Midi-Pyrénées

# THÈSE

## En vue de l'obtention du

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

### Délivré par l'Université Toulouse 1 Capitole

---

**Présentée et soutenue par**

**Rabah TIGHILT FERHAT**

Le 16 novembre 2021

## Extraction des modèles d'une base de données NoSQL orientée-documents basée sur une approche dirigée par les modèles

---

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
**IRIT : Institut de Recherche en Informatique de Toulouse**

Thèse dirigée par
**Gilles ZURFLUH**

Jury

**Mme Elisabeth METAIS,** Rapporteure
**M. Slimane HAMMOUDI,** Rapporteur
**Mme Faten ATIGUI,** Examinatrice
**M. Jérôme DARMONT,** Examinateur
**Mme Fatma ABDELHEDI,** Examinatrice
**M. Gilles ZURFLUH,** Directeur de thèse

Université Fédérale
Toulouse Midi-Pyrénées

# THÈSE

## En vue de l'obtention du

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

### Délivré par l'Université Toulouse 1 Capitole

---

**Présentée et soutenue par**

**Rabah TIGHILT FERHAT**

Le 16 novembre 2021

## Extraction des modèles d'une base de données NoSQL orientée-documents basée sur une approche dirigée par les modèles

---

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
**IRIT : Institut de Recherche en Informatique de Toulouse**

Thèse dirigée par
**Gilles ZURFLUH**

Jury

**Mme Elisabeth METAIS,** Rapporteure
**M. Slimane HAMMOUDI,** Rapporteur
**Mme Faten ATIGUI,** Examinatrice
**M. Jérôme DARMONT,** Examinateur
**Mme Fatma ABDELHEDI,** Examinatrice
**M. Gilles ZURFLUH,** Directeur de thèse

# Résumé

Nos travaux s'inscrivent dans le contexte des bases de données gérées par des SGBD NoSQL. La majorité de ces systèmes sont schema-less (sans schéma), ce qui signifie que le modèle de données n'est pas fourni lors de la création d'une BD. Cette absence du modèle introduit une difficulté de compréhension de la sémantique des données et un manque de visibilité sur l'organisation de la BD NoSQL. Autrement dit, l'absence du modèle de données ne permet pas à l'utilisateur de connaître comment les données sont stockées (sous quel nom et quel type) et reliées dans la BD. Or, cette connaissance est indispensable pour exprimer des requêtes. L'objectif de cette thèse est de proposer une approche qui vise à extraire les modèles physique et conceptuel nécessaires pour la manipulation d'une BD NoSQL schema-less. En se basant sur l'ingénierie dirigée par les modèles, nous proposons l'élaboration de deux modèles :

- Le modèle physique qui décrit l'organisation interne des données et permet d'exprimer des requêtes,
- Le modèle conceptuel qui fait abstraction des aspects techniques et se concentre sur la sémantique de données.

Ces deux modèles sont extraits successivement de la BD NoSQL par des techniques de méta-modélisation et de transformation automatique.

# Abstract

Our work falls within the context of databases managed by NoSQL DBMS. The majority of these are schema-less, which means that the data model is not provided when creating a database. This absence of the data model introduces a lack of understanding and visibility into the organization of data in a NoSQL database. In other words, the absence of the data model does not allow the user to know how the data is stored (under what name and what type) and linked in the database. However, this knowledge is essential to express requests. The objective of this thesis is to propose an approach which aims to extract the physical and conceptual models necessary for the manipulation of a NoSQL schema-less database. Based on model-driven engineering, we propose the development of two models:

- The physical model which describes the internal organization of the data and makes it possible to express queries,
- The conceptual model which disregards technical aspects and focuses on data semantics.

These two models are successively extracted from the NoSQL database by meta-modeling and automatic transformation techniques.

# Remerciements

Je remercie très sincèrement :

Monsieur Gilles ZURFLUH, Professeur à l'Université Toulouse Capitole pour avoir dirigé mes travaux, pour sa totale disponibilité à mon égard, pour son aide, ses encouragements, ses critiques et ses précieux conseils qu'il m'a donnés et sans lesquels ce travail n'aurait pas été possible. Je tiens également à lui exprimer toute ma reconnaissance pour la confiance qu'il m'a toujours témoignée, pour m'avoir laissé une grande liberté d'action et pour m'avoir offert d'excellentes conditions pour mener à bien mes travaux de recherche.

Monsieur Jérôme DARMONT, Professeur à l'Université Lumière de Lyon, pour avoir accepté de juger mon travail, pour ses remarques et pour l'honneur qu'il me fait en présidant mon jury.

Madame Elisabeth METAIS, Professeure au Conservatoire National des Arts et Métiers (CNAM) de Paris, pour avoir accepté d'être rapporteur de mes travaux de thèse. Je tiens à lui exprimer ma plus vive reconnaissance pour la lecture approfondie de ce mémoire qui a permis d'en améliorer la qualité, pour ses remarques pertinentes et pour sa participation au jury.

Monsieur Slimane HAMMOUDI, Professeur à l'ESEO d'Angers, pour avoir accepté d'être rapporteur de ce mémoire, pour ses remarques et critiques constructives qui m'ont permis d'améliorer grandement la qualité du manuscrit, ainsi que pour sa participation à mon jury. Je tiens tout particulièrement à le remercier pour les discussions que nous avons eues et ses nombreux conseils.

Madame Fatma ABDELHEDI, Directrice du laboratoire CBI2 au sein de la société TRIMANE, qui a participé à mes travaux et qui m'a offert la possibilité de valider mes propositions à partir d'applications réelles ; je la remercie également d'avoir participé au jury.

Madame Faten ATIGUI, Maître de conférences au Conservatoire National des Arts et Métiers (CNAM) de Paris, pour l'intérêt qu'elle a porté à mes travaux en examinant ce mémoire et pour avoir accepté d'être membre de mon jury.

Madame Amal AIT BRAHIM, Responsable Recherche et Innovation en Big Data Analytics & IA chez la société NEXESS, pour m'avoir encadré rigoureusement, de m'avoir fait découvrir l'environnement de travail à mon arrivée à Toulouse et d'avoir participé à la définition de ma problématique de recherche. J'ai une reconnaissance très particulière envers Amal qui a continué à encadrer mes

travaux malgré ses engagements professionnels. Je tiens également à lui exprimer toute ma reconnaissance pour son entière disponibilité à mon égard, sa générosité, ses encouragements, ses conseils de qualité et pour tous les échanges constructifs sur nos recherches depuis le début de ma thèse. J'ai eu énormément plaisir d'avoir fait sa connaissance.

Les gestionnaires de la Faculté d'informatique Michèle CUESTA et Florence THERY pour la disponibilité, l'aide généreuse et la gentillesse dont elles ont toujours fait preuve à mon égard.

Les collègues enseignants avec qui j'ai collaboré pour dispenser des cours et avec qui j'ai pu échanger sur la pédagogie. Plus particulièrement, je tiens à remercier Madame Geneviève PUJOLLE et Monsieur Khalid TAZI.

Pour finir, je souhaiterais remercier toute ma famille qui n'a eu de cesse de me soutenir et de croire en moi tout au long de mes études ; notamment, mes chers parents Seddik et Fatima, mes frères Djamal, Kamal, Ali, Samir et Hocine et mes sœurs Djamila, Rahma, Leila, Dahbia et Kahina, à qui je dédie cette thèse. Je leur suis très reconnaissant de m'avoir appris à être ambitieux et à ne jamais baisser les bras. Mes remerciements ne pourront jamais égaler leur assistance et leur amour qui m'a apporté du soutien au moment où j'avais besoin d'aide. C'est grâce à eux et pour eux que je suis qui je suis aujourd'hui.

# Table des matières

# Liste des figures

# Liste des tableaux

# Chapitre 1 : Introduction

## 1.1 Contexte

Au cours de ces dernières années, la transformation digitale des entreprises et plus largement celle de la société a entrainé une évolution des bases de données (BD) relationnelles vers les BD massives (Big data). Celles-ci permettent de stocker non seulement de grandes quantités de données mais également différents types et formats de données provenant de sources hétérogènes. De plus, ces données sont souvent saisies à très haute fréquence et doivent donc être filtrées et agrégées en temps réel pour éviter toute saturation inutile de l'espace de stockage. Ces caractéristiques ont eu un impact sur les outils nécessaires au stockage et à la gestion des données. Ainsi, sont apparus de nouveaux systèmes de gestion des données : les systèmes NoSQL. Ceux-ci sont plus adaptés que les systèmes relationnels pour gérer de gros volumes de données présentant des types et des formats variés.

Dans la majorité des SGBD NoSQL, les bases de données (BD) sont schema-less (sans schéma), ce qui signifie que le modèle de données n'est pas fourni lors de la création d'une BD. Autrement dit, dans une table, les noms des attributs ne sont précisés qu'au moment de l'insertion de leurs valeurs. Le mécanisme de schema-less est en opposition à ce que l'on trouve dans les SGBD relationnels où le modèle de données est fourni lors de la définition de cette table.

La propriété d'absence du schéma offre une flexibilité indéniable qui :

- facilite l'évolution du modèle de données au fur et à mesure de l'utilisation de la BD,
- et permet aux utilisateurs finaux d'ajouter de nouvelles informations sans avoir recours à l'administrateur de BD.

Il est important de noter ici que, selon la manière dont nous définissons leur modèle de données, les SGBD NoSQL peuvent être classés en trois catégories [Amal AIT BRAHIM, 2018] :

(a) Des SGBD où le modèle est fixé préalablement à toute alimentation de données. Autrement dit, comme dans les systèmes relationnels, chaque table doit être préalablement décrite en précisant le nom et le type des attributs qu'elle contiendra ; la saisie des données ne peut se faire qu'une fois le modèle entièrement défini ; c'est par exemple le cas de Cassandra (système orienté-colonnes) et de Riak TS (système clé-valeur),

(b) Des SGBD où seulement une partie du modèle est déclaré avant l'alimentation ; généralement, il s'agit de préciser les noms des tables ; par exemple dans les systèmes MongoDB (système orienté-documents) et HBase (système orienté-colonnes),

(c) Des SGBD où le modèle est spécifié au fur et à mesure de la saisie des données. L'utilisateur insère chaque ligne en précisant le nom de la table ainsi

que les noms des attributs ; c'est le cas des systèmes Neo4j (système orienté-graphes) et Redis (système clé-valeur).

Nos travaux s'inscrivent dans les deux dernières catégories, c'est-à-dire les systèmes qui n'exigent pas la définition complète du modèle avant la saisie des données ; ils sont ainsi qualifiés de schema-less.

### Avertissement

La conception d'une base de données passe par une modélisation à trois niveaux [ANSI/SPARC, 1975] et [OMG, 2003] : (1) conceptuelle (indépendante de toute technique), (2) logique (liée à une famille de SGBD) et (3) physique (propre à un SGBD particulier). A l'inverse, la rétroconception permet de fournir une représentation abstraite de haut niveau d'une base de données existante. Elle suit alors le cheminement contraire en passant par les niveaux : (1) physique, (2) logique et (3) conceptuel.

L'objectif de cette thèse est de proposer une démarche de rétroconception de BD NoSQL orientées-documents. En partant du niveau physique (la BD source intégrant le modèle physique), nous pouvons extraire le modèle logique puis le modèle conceptuel.

Notre processus vise bien à extraire un modèle logique commun à toute la famille des BD NoSQL orientées-documents. Mais comme la BD source prend en compte des éléments spécifiques à MongoDB, qui est le SGBD NoSQL orienté-documents le plus utilisé à ce jour [DB-Engines Ranking, n.d.], nous avons employé le terme "modèle physique" pour désigner le résultat de l'extraction. Il convient de noter que les spécificités MongoDB sont systématiquement ignorées lorsque ce processus est appliqué à d'autres systèmes NoSQL orientés-documents.

## 1.2 Problématiques

Comme nous l'avons introduit dans la section précédente, la propriété schema-less offre une souplesse indéniable en facilitant l'évolution du modèle de données (MD) au fur et à mesure de l'alimentation des tables. Par exemple, elle permet l'ajout de nouveaux descripteurs pour une ligne existante ou bien la modification de la structure d'une ligne sans modifier les autres lignes préalablement stockées. De plus, elle permet aux utilisateurs finaux d'ajouter de nouvelles informations sans avoir recours à l'administrateur de BD [Herrero et al., 2016].

Mais, en contrepartie, cette propriété introduit un manque de visibilité sur l'organisation des données d'une BD NoSQL. Ceci est dû à l'absence du MD qui est un élément de connaissance essentiel pour une gestion des données efficace [Abelló, 2015], [Herrero et al., 2016], [Daniel et al., 2016]. Autrement dit, l'absence du MD ne permet pas à l'utilisateur de connaitre comment les données

sont stockées (sous quel nom et quel type) et reliées dans la BD ; cette connaissance est désormais indispensable pour l'expression des requêtes. En effet, pour écrire ses requêtes, l'utilisateur doit disposer de la structure de la BD où les noms des tables présentes dans cette BD, les noms des attributs ainsi que leurs types sont mentionnés ; cette structure est représentée par un MD.

Actuellement dans l'industrie et pour surmonter cette difficulté, le même développeur (ou la même équipe de développement) se voit généralement confier la création de la base de données, le développement des interfaces de saisie des données et la rédaction des requêtes. Ainsi, le développeur connaît implicitement le modèle de données et peut donc exprimer des requêtes en connaissant la structure des données. Mais cette solution pragmatique n'est pas efficace lors de la maintenance des applications puisque le développeur qui en est chargé, ne connaît pas le modèle de données. Il en va de même pour un décideur qui souhaite interroger une BD alors qu'il n'a pas participé à sa création.

Pour manipuler une BD NoSQL schema-less, nous sommes convaincus qu'il est important de fournir une réponse à deux problématiques :

- La première est liée à l'extraction du modèle physique de données qui décrit l'organisation interne de la BD et qui permet aux utilisateurs (généralement des développeurs ou des décideurs) d'exprimer des requêtes sur cette base.
- La deuxième concerne la génération du modèle conceptuel de données qui fait abstraction des aspects techniques et se concentre sur la sémantique de données de la BD. Le modèle conceptuel a pour objectif de bien comprendre la BD avant l'expression des requêtes sur celle-ci.

Ce sont donc ces deux problématiques que nous allons traiter dans cette thèse. L'utilité est donc d'assister les utilisateurs à comprendre la BD NoSQL schema-less et à rédiger des requêtes sur cette même base.

## 1.3 Contributions

Notre objectif est de répondre aux problématiques présentées dans la section précédente : (1) l'extraction du modèle physique de données (MPD) d'une BD NoSQL et (2) la transformation du MPD résultat en un modèle conceptuel de données (MCD). Pour cela, nous proposons d'adopter une approche basée sur l'architecture MDA (Model Driven Architecture) qui est une norme du consortium OMG[1] pour le développement dirigé par les modèles. A partir d'une BD NoSQL schema-less, notre approche applique deux processus automatiques et successifs : *ToPhysicalModel* et *ToConceptuelaModel*. Ces processus se résument comme suit :

[1] http ://www.omg.org/

- **(1) processus *ToPhysicalModel* :** produit et met à jour le MPD d'une BD NoSQL schema-less. Il est composé de deux sous-processus qui s'appliquent successivement : *ModelExtraction* et *ModelUpdate*. Le sous-processus *ModelExtraction* permet de générer un MPD NoSQL à partir d'une BD de type orienté-documents. Il s'agit du sous-processus « à froid ». Son principe consiste à balayer la totalité de la BD à un instant *t* et à fournir une structure interne conforme à l'organisation des données à l'instant *t*. Le sous-processus *ModelUpdate* recalcule le MPD au fur et à mesure de l'exécution des requêtes de mise à jour sur la BD. Il s'agit du sous-processus « à chaud ». Son principe consiste à utiliser des métadonnées pour le traitement des requêtes de mise à jour. Selon l'architecture MDA, les modèles utilisés dans les deux sous-processus sont conformes à des métamodèles. Pour assurer le passage automatique entre les modèles sources et cibles, nous avons défini des règles de transformations de type Model-To-Model (M2M) formalisées avec le standard QVT.

- **(2) processus *ToConceptualModel* :** consiste à transformer le modèle physique déjà extrait par le processus *ToPhysicalModel* en un modèle conceptuel. Celui-ci est exprimé en utilisant le formalisme UML. Son principe consiste à trouver les classes que comporte la BD ainsi que les différents liens entres elles. Le passage entre le modèle physique et conceptuel est assuré par la définition d'un ensemble de règles de transformation de type Model-To-Model (M2M) formalisées avec le standard QVT.

Nos deux processus s'appuient sur :

- Deux niveaux de modélisation : physique et/ou conceptuel,
- Des métamodèles permettant de vérifier la validité des modèles à chaque niveau,
- Des normes de l'OMG pour formaliser l'entrée du processus et l'ensemble des règles de transformation.

Afin de vérifier la faisabilité de notre solution, nous avons développé un prototype composé de deux modules. Le premier est chargé de générer un modèle physique de données à partir d'une DB NoSQL schema-less et le mettre à jour au fur et à mesure de l'exécution des requêtes de mise à jour sur la BD. Le modèle physique résultant décrit l'organisation interne des données de la base et permet d'exprimer des requêtes. Le second module a pour objectif de transformer le modèle physique déjà extrait en un modèle conceptuel de données. Celui-ci fait abstraction des aspects techniques et se concentre sur la sémantique des données.

Nous avons ensuite effectué des expérimentations sur l'extraction des modèles physique et conceptuel d'une application médicale afin de tester le fonctionnement des modules composant notre prototype.

En outre, notre solution a été validée dans un cadre professionnel par des ingénieurs de la Société Trimane spécialisée en Big Data et en informatique décisionnelle et située à Saint Germain en Laye. L'objectif est de comparer le temps de rédaction de requêtes sur une BD MongoDB avec ou sans la présence des modèles physique et conceptuel.

## 1.4 Organisation du mémoire

Cette thèse est organisée de la façon suivante :

Le chapitre 2 présente le contexte technologique dans lequel s'intègrent nos travaux de thèse. Il s'agit notamment des Big Data, des systèmes NoSQL et de l'ingénierie dirigée par les modèles. Nous présentons également le cas d'étude qui a motivé nos travaux.

Le chapitre 3 situe l'état de l'art en matière d'extraction des modèles de données d'une BD NoSQL schema-less. Nous présentons dans ce chapitre les solutions industrielles ainsi que les travaux de recherche qui traitent de l'extraction du modèle physique de données. Nous présentons également les travaux qui ont étudié les approches d'extraction du modèle conceptuel de données. Enfin, nous abordons le positionnement de nos travaux au regard des travaux de l'état de l'art.

Nos contributions sont présentées dans les chapitres 4 et 5 comme suit. Le chapitre 4 décrit le processus *ToPhysicalModel* qui vise à formaliser et à automatiser l'extraction et la mise à jour du modèle physique de données à partir d'une BD gérée par un système NoSQL de type schema-less. Nous détaillons dans ce chapitre les composants de ce processus en précisant, pour chacun, les trois éléments suivants : la source, la cible et les règles de transformation associées. Le chapitre 5 présente le processus *ToConceptualModel* qui consiste à transformer le modèle physique (déjà extrait dans le chapitre 4) en un modèle conceptuel. Celui-ci est exprimé en utilisant le formalisme UML. Le passage entre le modèle physique et conceptuel est assuré par la définition d'un ensemble de règles de transformation de type Model-To-Model (M2M) formalisées avec le standard QVT. Ces règles de transformation ainsi que le langage utilisé pour les formaliser sont présentés dans le chapitre 6 intitulé Expérimentation et Validation.

Le chapitre 6 décrit la réalisation de notre prototype d'extraction des modèles physique et conceptuel à partir d'une BD MongoDB. Ce prototype a permis de montrer la faisabilité de notre approche dirigée par les modèles. Ce chapitre présente également une évaluation de notre processus automatisé dans un cadre professionnel au sein d'une société spécialisée en Big Data et en informatique décisionnelle.

Le chapitre 7 conclut notre mémoire en mettant en exergue les points forts de notre contribution. Nous terminons en ouvrant quelques perspectives.

# Chapitre 2 : Contexte Technologique et Cas d'Etude

Ce chapitre comporte deux parties. La première présente le contexte technologique de notre thèse. Il s'agit en l'occurrence des Big data, des systèmes NoSQL et de l'ingénierie dirigée par des modèles (IDM). La seconde présente le cas d'étude qui a motivé notre travail.

# 2.1 Données massives : « Big Data »

Le Big Data est un type de BD ayant des caractéristiques spécifiques. Nous présentons dans cette section les trois règles qui ont été largement utilisées pour définir le Big Data. Il s'agit des règles « 3V », « 4V » et « 5V ».

**Règle « 3V » :** Une description du Big Data a été établie dans un rapport de recherche en 2001 par l'analyste Douglas Laney du groupe Gartner [Douglas, 2001]. Elle définit les enjeux inhérents à la croissance des données comme étant tridimensionnels. Il s'agit du *Volume*, de la *Variété* et de la *Vélocité*. En 2012, Gartner[2] a donné une définition plus détaillée des Big Data comme suit :

> **Définition 1.** « *Les Big Data sont des données volumineuses, très variées, générées et traitées à grande vitesse. Ces données exigent des formes efficaces et innovantes de traitement de l'information pour permettre une meilleure prise de décision* » [Douglas, 2001].

**Règle « 4V » :** Le cabinet d'analyse IDC[3] (International Data Corporation) a délimité, dans un rapport de 2011 [Gantz and Reinsel, 2011], quatre dimensions pour caractériser le Big Data, à savoir : le *Volume*, la *Variété*, la *Vélocité* et la *Valeur*.

> **Définition 2.** « *Les Technologies Big Data décrivent une nouvelle génération de technologies et d'architectures conçues pour extraire économiquement de la valeur à partir de grands volumes de données très variées, en permettant leur capture et leur analyse à grande vitesse* » [Gantz and Reinsel, 2011].

**Règle « 5V » :** Deux nouveaux V sont apparus en 2013 dans un article de recherche [Demchenko et al., 2013] où le groupe SNE[4] (System and Network Engineering) propose une définition plus large pour le Big Data au travers de la règle des 5V.

> **Définition 3.** « *Les Technologies Big Data visent à traiter des données de grand volume, grande vélocité et grande variété pour extraire de la valeur, assurer une forte véracité des données originales et obtenir des informations qui exigent des formes novatrices de traitement des données, afin d'améliorer la prise de décision et le contrôle des processus* » [Demchenko et al., 2013].

[2] https://www.gartner.com/en
[3] https://www.idc.com/about
[4] https://ivi.fnwi.uva.nl/sne/

Les définitions associées aux dimensions utilisées dans les règles ci-dessus sont les suivantes :

- Volume : représente la taille de l'ensemble des données à traiter,
- Variété : fait référence à la diversité des sources, des types et des formats de données,
- Vélocité : correspond à la vitesse à laquelle les données sont collectées et traitées,
- Valeur : équivaut au profit que l'on peut tirer de l'usage des Big Data,
- Véracité : recouvre la qualité et la fiabilité des données ; c'est la dimension qualitative des Big Data.

Les SGBD relationnels sont utilisés par de nombreuses entreprises. Mais face aux caractéristiques du Big Data, ces systèmes rencontrent des limites dont les principales sont :

- La mise à l'échelle horizontale[5] : une BD relationnelle a été principalement conçue pour des configurations à serveur unique [Kumar et al., 2015] ; mettre à l'échelle cette base consiste à la distribuer sur plusieurs serveurs. Ceci pose des contraintes financières (le coût des serveurs) et techniques (le nombre de serveurs est généralement limité à 10). De plus, gérer des tables sur différents serveurs reste une tâche complexe,

- La définition d'un modèle lors de la création de la BD et avant la saisie des données : dans un contexte Big Data, l'utilisateur doit être capable d'intégrer facilement de nouvelles données. Les SGBD relationnels n'offrent pas cette souplesse ; le modèle relationnel est difficile à modifier de manière dynamique pendant l'exploitation sans affecter les performances ou mettre la BD hors ligne.

Ainsi, les technologies de stockage ont dû évoluer pour introduire de nouveaux SGBD qui sont capables de gérer des données volumineuses, variées et qui peuvent évoluer très rapidement. Il s'agit des SGBD NoSQL [Han et al, 2011] présentés dans la section suivante.

## 2.2 NoSQL

Le terme NoSQL désigne un type de systèmes de gestion de base de données (SGBD) qui va au-delà des systèmes relationnels associés au langage SQL en acceptant des structures de données plus complexes. Selon leurs modèles

[5] La mise à l'échelle horizontale fait référence à l'ajout de plusieurs machines au système pour améliorer les performances. A la différence de la mise à l'échelle verticale qui correspond à l'ajout de ressources supplémentaires (CPU, RAM) à une seule machine existante afin d'améliorer les performances.

physiques, les BD gérées par ces systèmes se répartissent en quatre catégories : colonnes, documents, graphes et clé-valeur [Angadi et al., 2013]. Chacune d'elles offrant des fonctionnalités spécifiques. Par exemple, dans une BD orientée-documents comme MongoDB [6], les données sont stockées dans des tables imbriquées. Cette organisation des données est couplée à des opérateurs qui permettent d'accéder aux données imbriquées [Kumar et al., 2015].

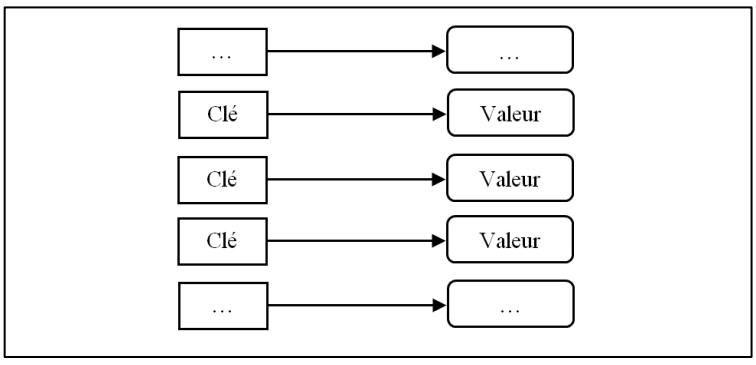
Le choix de la catégorie du SGBD la plus adaptée à une application donnée, est lié à la nature des traitements (les requêtes) appliqués sur les données. Mais ce choix n'est pas exclusif puisque, dans chaque catégorie, les SGBD peuvent assurer tous les types de traitements, au prix parfois d'une certaine lourdeur ou d'une programmation plus importante.

Dans ce qui suit, nous présentons les modèles de données adoptés par chaque catégorie de SGBD NoSQL.

### 2.2.1 Modèle orienté clé-valeur

Il s'agit du modèle NoSQL le plus basique et qui a été le précurseur dans les SGBD NoSQL. Il organise les données sous forme de paires clé-valeur (cf. figure 2.1), où la clé est le point d'entrée unique qui permet d'accéder à la donnée.

Du fait de leur modèle d'accès simplifié qui utilise exclusivement la clé, les SGBD qui adoptent le modèle clé-valeur, comme Redis[7] et Riak[8], permettent d'atteindre des performances élevées en termes de temps d'accès aux données. Cependant, ils n'offrent que des fonctionnalités simplifiées en termes d'expression des requêtes [Angadi et al., 2013]. Ainsi, ces SGBD sont principalement utilisés dans des contextes où les besoins en termes de requêtes sont très réduits. Par exemple, nous utilisons ces systèmes pour conserver les sessions d'un site web en tant que cache. Nous citons comme cas d'usage la gestion des paniers sur les sites d'achat comme Amazon et la collecte d'évènements que nous trouvons dans les jeux en ligne.




[6] https://www.mongodb.com/
[7] https://redis.io/
[8] http://docs.basho.com/

**Figure 2.1 -** Organisation des données dans une BD clé-valeur

Les autres modèles de données orientés colonnes, graphes et documents, présentés dans ce qui suit, sont des évolutions du modèle clé-valeur.

## 2.2.2 Modèle orienté-colonnes

Le modèle orienté-colonnes est un modèle structuré où les données sont organisées en familles de colonnes, ce qui équivaut au concept de table dans le modèle relationnel. Les lignes possèdent un identifiant appelé clé de ligne et sont composées d'un ensemble de valeurs ; chacune est associée à une colonne. Ainsi, la recherche d'une valeur revient à parcourir la séquence : *clé de ligne -> famille de colonnes -> colonne*.

Bien que le modèle orienté-colonnes se rapproche du modèle relationnel, l'organisation des données dans les deux modèles est différente [Abadi et al., 2008]. En opposition à ce que l'on trouve dans une BD relationnelle où les colonnes sont statiques et présentes dans chaque ligne, les colonnes d'une BD orientée-colonnes sont dynamiques et apparaissent uniquement dans les lignes concernées. Autrement dit, chaque ligne a un nombre différent de colonnes (cf. figure 2.2) et on peut lui ajouter à tout moment de nouvelles colonnes ; on gagne ainsi en extensibilité au niveau du modèle de données.

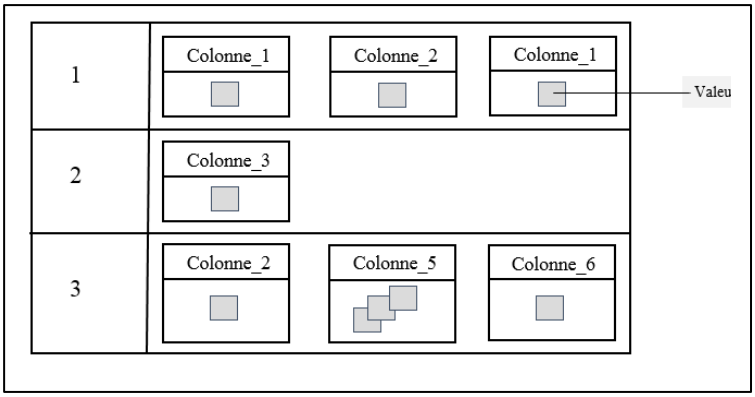



**Figure 2.2 -** Organisation d'une famille de colonnes dans une BD orientée-colonnes

De plus, le modèle orienté colonnes a pour avantage d'améliorer l'efficacité du stockage et d'éviter de consommer de l'espace par rapport au modèle relationnel. En effet, du fait de leur conception par allocation de blocs, dans un SGBD relationnel, une colonne vide consommera quand même de l'espace. Dans un SGBD orienté-colonnes, le coût de stockage d'une colonne vide est nul.

Cassandra[9], HBase[10] et Accumulo sont des exemples de SGBD où les données sont stockées suivant un modèle orienté-colonnes.

### 2.2.3 Modèle orienté-graphes

Ce modèle organise les données sous forme de nœuds et de relations. Les nœuds et les arcs (i.e. présentant les relations) peuvent porter un ensemble de propriétés exprimées sous la forme de paires clé-valeur (cf. figure 2.3).

Ce modèle est utile pour stocker et interroger des données complexes fortement liées ; c'est le cas par exemple des données issues des réseaux sociaux.

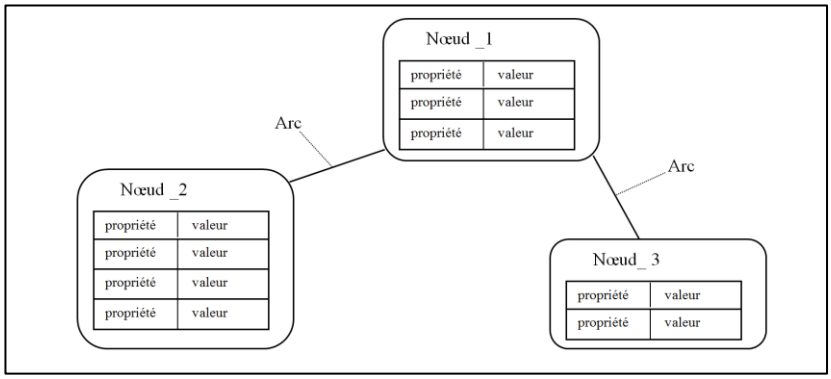



**Figure 2.3 -** Organisation des données dans une BD orientée-graphes

Les SGBD orientés-graphes les plus connus sont Neo4j [11], OrientDB [12] et FlockDB[13].

### 2.2.4 Modèle orienté-documents

Les données dans un modèle orienté-documents sont organisées en collections de documents. Un document est constitué d'un identifiant (clé) et d'une valeur composée de couples clé-valeur qui peuvent être hiérarchisés (cf. figure 2.4). Cela veut dire qu'une valeur peut-elle même contenir une ou plusieurs paires clé-valeur.

Au sein de la même collection, les documents peuvent être définis par des structures différentes. Autrement dit, les couples utilisés pour définir les documents d'une collection n'ont pas nécessairement les mêmes noms. De plus, comme les autres modèles NoSQL, le modèle orienté-documents est flexible ; on

[9] Datastax. (2012). Apache Cassandra™ Documentation. From http://www.odbms.org/wp-content/uploads/2013/11/cassandra10.pdf.
[10] Apache HBase Team. (2017). Apache HBase ™ Reference Guide, version 3(0). From https://hbase.apache.org/book.html

[11] https://neo4j.com/
[12] https://orientdb.com/
[13] https://db-engines.com/en/system/FlockDB

peut ajouter des couples à chaque insertion d'un nouveau document ou lors de sa modification.

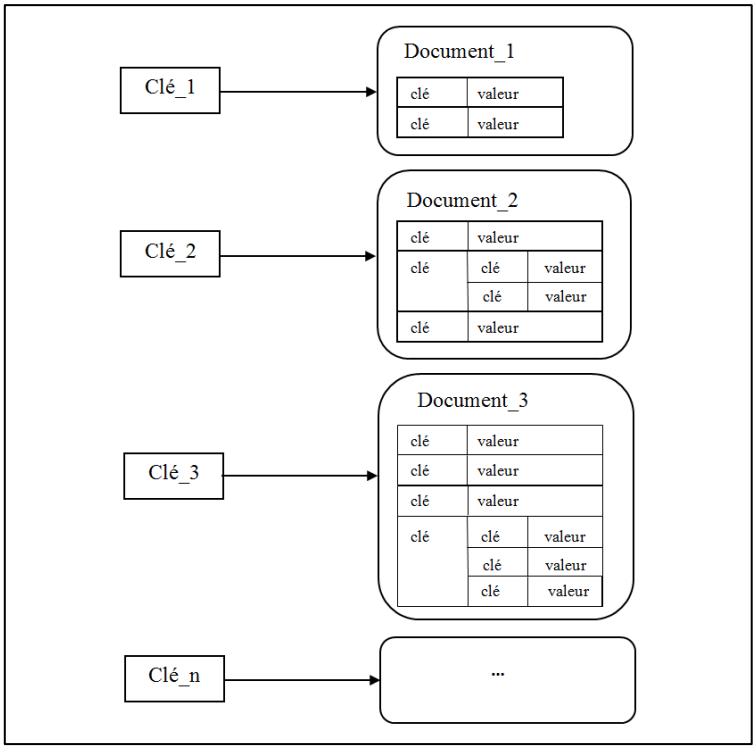



**Figure 2.4 -** Organisation d'une collection dans une BD orientée-documents

Les SGBD orientés-documents, comme MongoDB, couchDB [14] et couchDB server, apportent des fonctionnalités avancées pour la manipulation des documents. Avec le principe d'imbrication du modèle de données qu'ils adoptent (le modèle orienté-documents), nous pouvons modéliser les données de façon à ce qu'elles supportent des fonctionnalités plus avancées d'interrogation, telles que la capacité de manipuler le contenu du document (recherche en plein texte) [Arora and Aggarwal, 2013].

Dans ce mémoire, nous avons limité notre étude aux BD gérées par des systèmes NoSQL de type orientés-documents. Ceux-ci sont les plus complets en termes d'expression des liens. En effet, pour exprimer un lien dans une BD orientée-documents, l'utilisateur choisit l'une des deux manières suivantes :

- Utilisation de références : consiste à pointer l'identifiant (clé) d'un document à partir d'un autre.

[14] Apache CouchDB 2.1 Documentation". [Online]. Available: http://docs.couchdb.org/en/2.1.1/.

- Utilisation d'imbrication : consiste à définir un document sous la forme d'un champ multivalué structuré. Celui-ci est déclaré dans un autre document.

De plus, dans notre contexte d'étude (cf. section 2.4), l'utilisation de ces deux options (référence et imbrication) est nécessaire, d'où le choix d'une BD orientée-documents pour stocker les donnes de notre application.

# 2.3 Ingénierie dirigée par les modèles (IDM)

Nous présentons dans cette section les concepts de base sur lesquels s'appuie notre mémoire. Il s'agit notamment des concepts de l'ingénierie dirigée par des modèles (IDM).

Pour faire face à la complexité croissante des applications informatiques à développer, les spécialistes du génie logiciel ont proposé un nombre important de méthodes d'ingénierie informatique. Après la méthode Objet qui est basée sur le concept « tout est objet », une nouvelle méthode intitulée Ingénierie Dirigée par les Modèles (IDM) est apparue. Celle-ci quant à elle est basée sur le concept « tout est modèle » [Combemale, 2008]. Elle consiste à adopter des modèles comme éléments centraux dans toutes les phases (ou une partie) du cycle de vie de développement de logiciels. Le but est de séparer la logique métier des plateformes techniques. Cela a apporté des avantages considérables en termes de gain de temps et de facilité de maintenance.

Dans cette optique, l'Object Management Group[15] (OMG) a rendu publique son initiative MDA (Model Driven Architecture) [Kleppe et al., 2003]. Il s'agit d'une démarche de développement qui vise à définir un cadre formel pour l'IDM. L'idée de base de MDA est de décrire séparément les spécifications fonctionnelles et les spécifications d'implantation d'une application sur une plateforme donnée [Hutchinson et al., 2011]. Pour cela, MDA préconise l'utilisation de trois modèles. Il s'agit du (1) modèle des besoins, dit CIM (pour Computation Independent Model) dans lequel aucune considération informatique n'apparait, du (2) modèle d'analyse et de conception, dit PIM (pour Platform Independent Model) indépendant des détails techniques des plateformes d'exécution et du (3) modèle de code dit PSM (pour Platform Specific Model) spécifique à une plateforme particulière. Nous détaillons ces différents modèles dans la section 2.3.2. Dans cette section, nous présentons d'abord les concepts généraux manipulés par l'ingénierie dirigée par les modèles. Ensuite, nous présentons l'architecture MDA. Les définitions présentées dans cette section reprennent

[15] http ://www.omg.org/

principalement les résultats des travaux de [Bézivin and Gerbé, 2001], de [Kleppe et al., 2003] ainsi que des recommandations de l'OMG [OMG, 2003].

### 2.3.1 Principes généraux

Dans cette section, nous définissons les concepts de bases de l'IDM ainsi que les relations entre ces concepts.

Le premier concept important est le modèle. Celui-ci est défini comme suit :

**Définition 1 :** Un modèle est une représentation simplifiée d'un système construite dans un but précis. Autrement dit, un modèle est une abstraction d'un aspect particulier d'un système. Il est une abstraction dans la mesure où il contient un ensemble restreint d'informations sur un système [Bézivin and Gerbé, 2001].

Cette définition signifie qu'un système peut être représenté par un ensemble de modèles, où chacun d'eux capture un aspect particulier.

Notons que ce que nous connaissons communément sous le nom du « Schema » dans le paradigme Objet ou Entité/Association est désigné par « Modèle » dans notre mémoire de thèse. En effet, les deux termes appartiennent au même niveau d'abstraction dans une vision dirigée par les modèles. Ils sont donc considérés comme synonymes.

Afin de rendre un modèle utilisable, il est nécessaire d'indiquer ses concepts, leurs sémantiques ainsi que les liens entre eux. Cela se fait au travers de la définition d'un méta-modèle que nous décrivons comme suit :

**Définition 2 :** Un méta-modèle est la spécification explicite d'une abstraction (simplification). Autrement dit, un méta-modèle est un modèle qui décrit les différents éléments (concepts et sémantique) d'un modèle ainsi que la manière dont les éléments sont organisés [Bézivin and Gerbé, 2001].

On dit qu'un modèle est conforme à son méta-modèle si l'ensemble des éléments du modèle est défini par le métamodèle. Pareillement qu'un modèle, l'utilisation d'un méta-modèle nécessite une description de sa syntaxe. Cela se fait au travers de la définition d'un méta-méta-modèle Celui-ci est défini comme suit :

**Définition 3 :** Un méta-méta-modèle est un modèle qui décrit les éléments d'un méta-modèle.

Théoriquement, il n'y a pas de limite par rapport aux niveaux de modélisation. Ainsi, un méta méta-modèle peut être décrit par un méta-méta-méta modèle. Cependant, en pratique, l'idée retenue est de définir des méta-méta-modèles de manière qu'ils soient auto-descriptifs ; c'est-à-dire qu'ils peuvent se définir eux-mêmes.

L'OMG a défini une hiérarchie de modélisation à quatre niveaux comme le montre la figure 2.5 :

**- Niveau M0 :** c'est le niveau du monde réel. Il correspond aux données concrètes que l'on souhaite modéliser. Ces données sont des instances du modèle du niveau M1.

**- Niveau M1 :** c'est le niveau modèle. Il décrit les données du niveau M0. Par exemple, un modèle UML appartient à ce niveau. Les données d'un niveau M1 sont des instances du niveau M2.

**- Niveau M2 :** c'est le niveau méta-modèle. Par exemple, le métamodèle UML.

**- Niveau M3 :** c'est le niveau méta-méta-modèle. Il permet de décrire la structure des méta-modèles. Il permet aussi d'étendre ou de modifier des méta modèles existants. L'OMG a défini un langage unique et auto-descriptif appelé Meta Object Facility (MOF). Le MOF est utilisé pour décrire les différents métamodèles et aussi pour décrire sa propre sémantique.

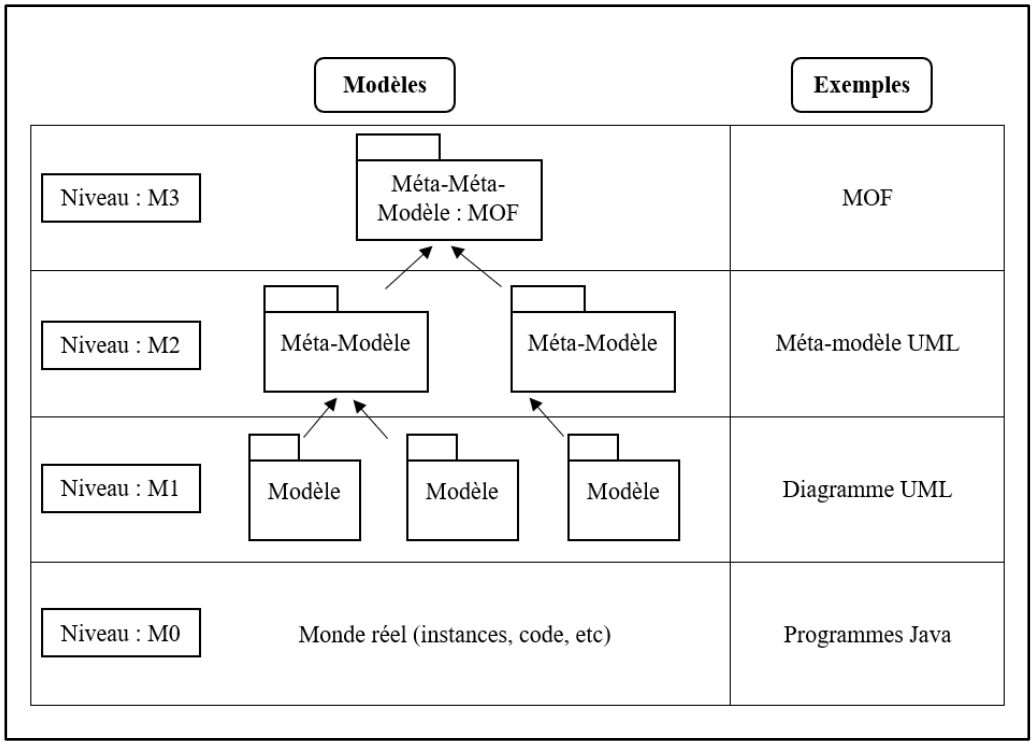



**Figure 2.5 -** Architecture de modélisation à quatre niveaux [OMG, 2003]

## 2.3.2 Architecture dirigée par les modèles (MDA)

MDA est l'initiative recommandée par l'OMG pour définir un cadre formel pour l'ingénierie dirigée par les modèles. Une démarche MDA repose sur l'utilisation des modèles dans toutes les phases (ou une partie) du cycle de vie de développement de logiciels. Le principe de base est de décrire séparément les spécifications fonctionnelles et les spécifications d'implantation d'une application sur une plateforme donnée [Hutchinson et al., 2011]. Pour ceci, MDA préconise l'utilisation trois modèles représentant les niveaux d'abstraction de l'application : CIM, PIM et PSM. Cependant, le nombre et le type de modèles sont propres à chaque démarche. La figure 2.6 présente ces modèles que nous décrivons comme suit :

- **CIM** : ce modèle décrit les besoins fonctionnels de l'application en utilisant le vocabulaire du métier (exprimés par un spécialiste du domaine de l'application). Les détails de la structure et de l'implantation de l'application restent encore indéterminés à ce niveau.

Les besoins recensés dans le CIM seront pris en compte dans les constructions des PIM et des PSM.

- **PIM** : ce modèle représente la logique métier spécifique à l'application en faisant abstraction des aspects techniques liés à la technologie de mise en œuvre.

- **PSM** : ce modèle correspondant à la spécification d'une application après projection sur une plateforme technologique donnée.

- **Code** : représente le code de l'application généré à partir du PSM.

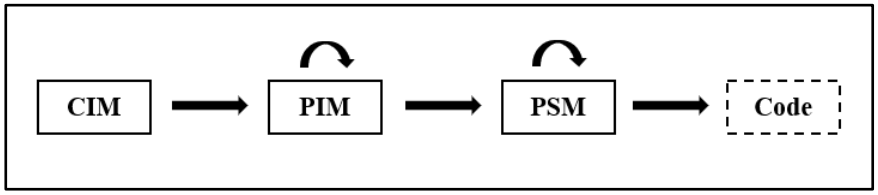



**Figure 2.6 -** Modèles MDA [OMG, 2003]

Le passage entre les différents modèles MDA se fait via une succession de transformations, essentiellement entre PIM et PSM. Notons toutefois que MDA permet plusieurs possibilités de transformations entre les modèles (cf. figure 2.6).

Une transformation correspond à l'application d'un ensemble de règles qui décrivent comment dériver un modèle cible à partir d'un modèle source. Les transformations de modèles peuvent être classées selon deux catégories : (1) la transformation d'un modèle vers du texte (M2T pour Model-To-Text) lorsque le résultat généré est du code et (2) la transformation d'un modèle vers un autre modèle (M2M pour Model-To-Model) quand le résultat est un modèle.

Pour exprimer les règles de transformation, l'OMG a défini le langage QVT[16] (Query/View/Transformation) ; il s'agit d'un langage standardisé de transformation de modèles que nous présentons dans le chapitre 6.

## 2.4 Cas d'étude

Pour illustrer et motiver notre travail, nous utilisons un cas extrait d'une application médicale dont la base de données est décrite dans le formalisme UML. Il s'agit de la mise en place de programmes scientifiques consacrés au suivi d'une pathologie déterminée, qui regroupent une cinquantaine d'établissements hospitaliers européens (hôpitaux, cliniques et centres de soins spécialisés)

L'objectif premier d'un tel programme est de collecter des données significatives sur l'évolution temporelle de la pathologie, d'étudier ses interactions avec des maladies opportunes et d'évaluer l'influence de ses traitements à court et moyen termes. La durée d'un programme est décidée lors de son lancement et peut atteindre entre trois et dix ans.

Au début du programme, un médecin-coordonnateur est nommé au sein de chaque établissement participant ; il a la charge de :

- Déterminer une cohorte de patients sur la base du volontariat (en moyenne une trentaine de patients),
- Fixer les protocoles de soins en accord avec les objectifs du programme,
- Affecter des praticiens de l'établissement pour réaliser les mesures, les soins, les traitements et les consultations.

Avant le démarrage du programme, chaque coordonnateur doit saisir l'ensemble des données de départ :

- Les caractéristiques de chaque participant : identification, date de naissance, coordonnées de l'organisme social, médecin traitant, personne(s) de confiance,
- Les éventuels antécédents médicaux du patient (maladies graves, opérations chirurgicales, accidents) ; à cette occasion, peuvent être enregistrés des documents sous forme d'image : comptes rendus, radiographies, séquences vidéo,
- Les professionnels de santé affectés au programme : identification, spécialité, etc.,

[16] OMG, Q. (2011). Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1. 1.

- Le protocole de soin trimestriel spécifique à chaque patient et comportant les éléments suivants : mesures à collecter, traitements et contrôles à effectuer ; chacun de ces éléments étant associé à une fréquence ou à une date de réalisation et, le cas échéant, affecté à un professionnel de santé.

Dès le démarrage du programme, les données sont collectées soit à la volée grâce à des capteurs (thermomètre, tensiomètre, …) à la disposition des patients, soit par l'intermédiaire des personnels médicaux utilisant des tablettes mobiles ou des matériels spécialisés (scanner, IRM, etc.).

La consultation d'un patient par un médecin peut être effectuée soit en chambre, lorsque le patient est hospitalisé, soit en consultation ambulatoire. Elle est nécessairement programmée, c'est-à-dire prévue dans le protocole trimestriel ou ajoutée par un médecin. Lors d'une consultation, le médecin établit et enregistre un compte rendu ; il peut également enregistrer des mesures et tout type de documents remis par le patient ; enfin il peut rédiger une ou plusieurs ordonnances (radiographie, médicaments non prévus dans le protocole, suspension temporaire ou définitive d'un traitement).

Cette étude de cas est donc un exemple typique d'application Big Data où l'utilisation d'un SGBD NoSQL est nécessaire compte tenu de la spécificité des données. En effet, les données collectées par les établissements impliqués dans le cadre du programme médical, présentent les caractéristiques généralement admises pour le Big Data (les 3 V). Le volume des données médicales recueilli quotidiennement auprès des patients, peut atteindre, pour l'ensemble des établissements et sur trois années, plusieurs téraoctets. D'autre part, la nature des données saisies (mesures, radiographie, scintigraphies, etc.) est diversifiée et peut varier d'un patient à un autre selon son état de santé. Enfin, certaines données sont produites en flux continu par des capteurs ; elles doivent être traitées quasiment en temps réel car elles peuvent s'intégrer dans des processus sensibles au temps (mesures franchissant un seuil qui impliqueraient l'intervention d'un praticien en urgence par exemple).

L'extrait de diagramme UML de la figure 2.7 montre quelques classes pour la base de données d'un programme médical.

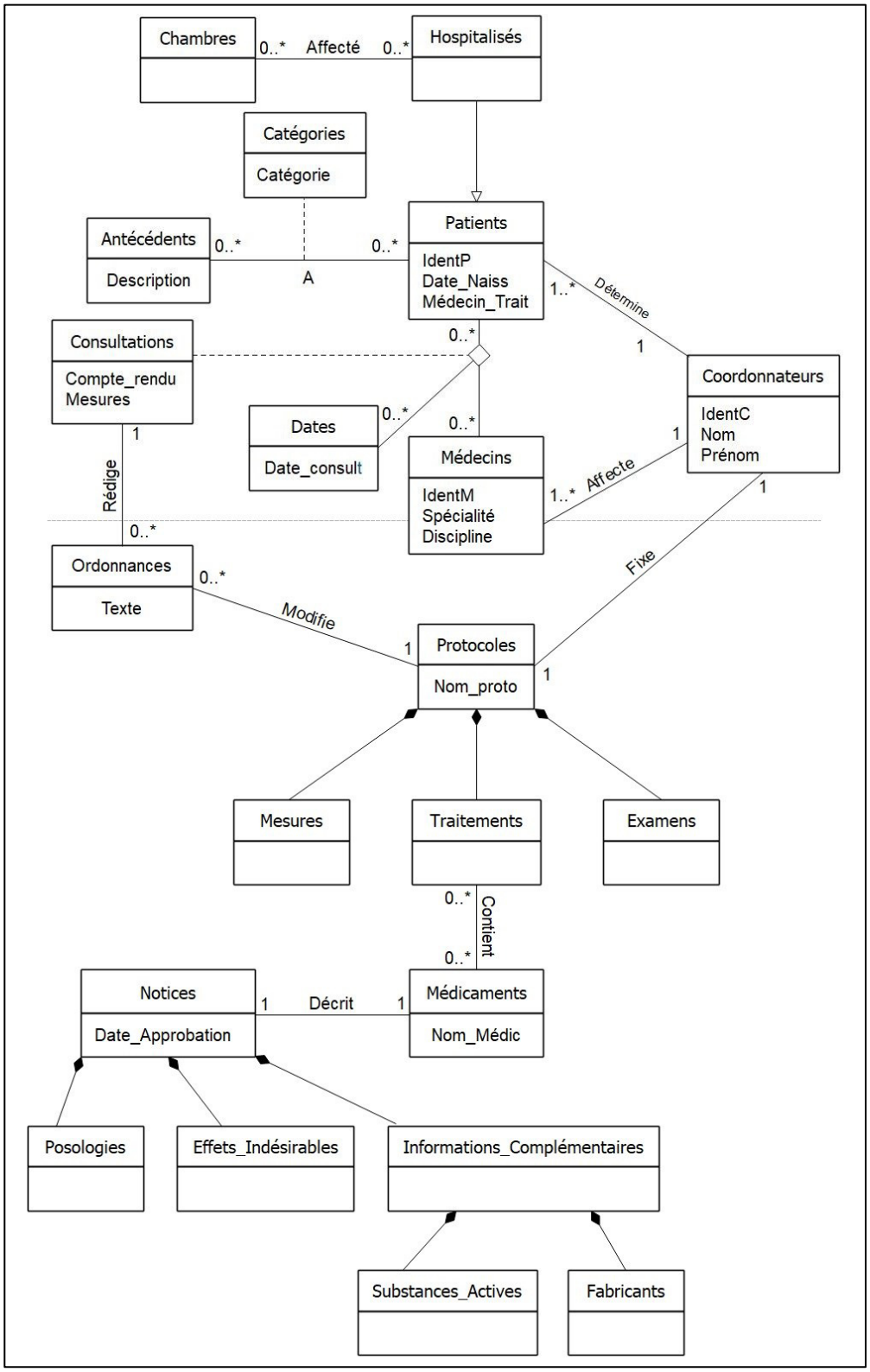



**Figure 2.7 -** Extrait d'un modèle de données du programme médical

# Chapitre 3 : Etat de l’art

Nous présentons dans ce chapitre les travaux portant sur la rétroconception des BD NoSQL schema-less. L'objectif est de fournir aux utilisateurs (qui peuvent être des développeurs ou des décideurs) les modèles de données nécessaires pour la manipulation de ce type de BD. Il s'agit notamment :

- du modèle physique qui décrit l'organisation interne des données et permet aux utilisateurs d'exprimer des requêtes.
- du modèle conceptuel qui fait abstraction des aspects techniques et met en évidence la sémantique de données.

Dans l'industrie, des solutions ont été proposées pour générer le modèle des BD NoSQL schema-less, mais ces solutions présentent des limites. D'autre part, des travaux de recherche proposent des solutions pour extraire les modèles de données des BD NoSQL. La plupart de ces travaux se focalisent uniquement sur le niveau physique. Cependant, quelques travaux ont abouti à la génération d'un modèle conceptuel.

La section 3.1 présente les solutions industrielles et de recherche permettant d'extraire le modèle physique de données. La section 3.2 quant à elle montre les approches d'extraction du modèle conceptuel de données. Dans la section 3.3, nous présentons une synthèse sur ces différentes approches. Enfin, la section 3.4 aborde le positionnement de nos travaux au regard des travaux de l'état de l'art.

# 3.1 Extraction du modèle physique de données

L'extraction d'un modèle physique d'une BD NoSQL « schema less » a fait l'objet de plusieurs solutions industrielles et de recherche, ceci principalement pour les BD de type documents gérées par MongoDB[17].

## 3.1.1 Solutions industrielles

Dans l'industrie, des solutions ont été proposées pour extraire le modèle physique d'une BD NoSQL ; nous citons MongoDB et le système Apache Drill.

MongoDB est un SGBD NoSQL orienté-documents le plus utilisé au monde, comme le montre une étude récente de DB-Engines [DB-Engines Ranking, n.d.]. Il est conçu pour offrir de hautes performances en lecture et en écriture ainsi qu'une mise à l'échelle automatique de la base de données. MongoDB stocke les données dans des collections regroupant des documents. Ceux-ci sont présentés en format BSON (Binary JSON) qui est une représentation textuelle de données en clé/valeur.

MongoDB offre une possibilité de visualiser le modèle physique d'une BD gérée par ce même système. En effet, sur la version graphique « Compass[18] » de

[17] https://www.mongodb.com/fr
[18] https://www.mongodb.com/products/compass

MongoDB, les données d'une collection sont présentées sous forme d'un tableau. Celui-ci contient comme lignes les documents de la collection et comme colonnes les champs de la collection associés à leurs types.

Quant à Apache Drill, il est un logiciel open-source conçu pour l'intégration de Big Data. Il permet d'assurer une analyse interactive de jeux de données à grande échelle sur des applications distribuées. Drill est capable de traiter en quelques secondes des pétaoctets de données et des milliards d'enregistrements sur 10.000 serveurs ou plus. Drill peut supporter de nombreuses BD relationnelles, BD NoSQL et des systèmes de gestion des fichiers tels que HDFS d'Hadoop et Amazon S3, Azure Blob Storage, Google Cloud Storage, etc. L'avantage de Drill est qu'il peut joindre des données de tous ces systèmes dans la même requête. Par exemple, il est capable d'interroger une collection MongoDB avec des données stockées sur le système HDFS d'Hadoop.

Drill permet aussi d'extraire le modèle physique d'une BD NoSQL schema-less de type MongoDB par exemple. Pour cela l'utilisateur doit exécuter manuellement un script shell composé d'une suite de requêtes sur la BD en question.

Cependant, ces deux outils présentent deux limites majeures concernant l'extraction du modèle physique. D'abord, ils ne considèrent pas les champs structurés figurant sur les collections. En effet, seuls les champs du premier niveau sont affichés sur le modèle physique. Par conséquent, les champs composant un champ structuré ne sont pas affichés. Par exemple, pour le champ structuré *Adresse* qui est composé de trois champs atomiques : *NumRue*, *NomRue* et *CodePostal*, ces outils affichent uniquement le champ *Adresse* mais pas les champs composants. De plus, ces outils ne prennent pas en compte les liens entre les collections constituant la BD.

### 3.1.2 Travaux de recherche

Un processus a été proposé dans [Klettke et al., 2015] pour extraire le modèle d'une collection de documents JSON stockée sur MongoDB. Le modèle retourné par ce processus est en format JSON ; il est obtenu en capturant les noms des attributs qui figurent dans les documents en entrée et en remplaçant leurs valeurs par leurs types. Les valeurs d'attributs peuvent être de type atomique, des listes et des documents imbriqués.

Un travail similaire [Izquierdo & Cabot, 2016] propose un processus de génération d'un modèle physique à partir d'une collection de documents JSON. Ce processus consiste à (1) d'abord extraire le modèle de chaque document en remplaçant les paires (Clé, Valeur) par des paires (Clé, Type) et (2) à unifier ensuite tous les modèles obtenus pour avoir un seul modèle pour l'ensemble de la collection. Le modèle généré est au format JSON.

Dans l'article de [Sevilla et al., 2015], les auteurs proposent un autre processus d'extraction du modèle de données sur une BD NoSQL de type documents pouvant comporter plusieurs collections. Le résultat retourné n'est pas un modèle unifié pour toute la base mais il donne les différentes versions de modèles pour chaque collection. Le processus d'extraction est composé de deux étapes successives. La première parcourt la BD et, pour chaque version de modèle distincte, génère un document dans une collection intitulée « Modèle ». Dans la seconde étape, le processus fournit un modèle de chaque version en instanciant le méta-modèle JSON.

Nous citons aussi le travail de [Gallinucci et al., 2018] qui propose un processus intitulé BSP (Build Schema Profile) pour classifier les documents d'une collection en appliquant des règles correspondantes aux exigences des utilisateurs. Ces règles sont exprimées grâce à un arbre de décision dont les nœuds représentent les attributs des documents ; les arêtes spécifient les conditions sur lesquelles la classification est basée. Ces conditions traduisent soit l'absence ou la présence d'un attribut dans un document soit sa valeur. Comme dans l'article précédent [Sevilla et al., 2015], le résultat retourné par cette approche n'est pas un modèle unifié mais un ensemble de modèles de versions ; chacun d'eux est commun à un groupe de documents.

Dans [Maity et al., 2018], les auteurs décrivent le passage d'une BD NoSQL orientée-documents ou orientée-graphes en un schéma relationnel. Le processus regroupe l'ensemble des documents (ou des objets) qui ont les mêmes noms de champs. Pour chaque classe d'objets ainsi obtenus, il génère une table ayant comme attributs les noms des champs ; et comme lignes les valeurs de ces champs.

D'autre part, dans [Baazizi et al., 2017], les auteurs proposent un processus d'extraction du schéma d'une collection volumineuse de documents JSON en utilisant le système MapReduce. La phase Map consiste à extraire le schéma de chaque document de la collection en inférant des couples (champ, type) à partir des couples (champ, valeur). La phase Reduce consiste à unifier tous les schémas produits dans la phase Map dans le but de fournir un schéma global de tous les documents de la collection.

Dans un autre article [Baazizi et al., 2019], les mêmes auteurs ont proposé d'étendre ce processus en intégrant le paramétrage de l'extraction au niveau de la phase Reduce. Ainsi, l'utilisateur peut choisir soit d'unifier tous les schémas des documents de la collection, soit d'unifier uniquement les schémas ayant les mêmes champs (noms et types).

## 3.2 Extraction du modèle conceptuel de données

A notre connaissance, peu de travaux permettent l'extraction d'un modèle conceptuel pour les BD NoSQL ; ici le terme conceptuel qualifie un modèle sémantique exempt de toute considération technique.

Ainsi dans [Comyn-Wattiau et al., 2018], les auteurs proposent un processus d'extraction d'un modèle conceptuel pour une BD NoSQL orientée-graphes (Neo4J). Dans ce type particulier de BD NoSQL, la BD contient des nœuds (objets) et leurs liens binaires. Le processus proposé prend en entrée les requêtes d'insertion d'objets et de liens puis retourne un modèle Entité/Association ; il repose sur une architecture MDA et applique successivement deux transformations. La première consiste à construire un graphe (Nœuds + Arêtes) à partir des requêtes Neo4j. La deuxième consiste à extraire de ce graphe un modèle Entité/Association en transformant les nœuds portant la même étiquette en entités et les arêtes en associations. Ces travaux sont spécifiques aux BD NoSQL orientées-graphes généralement utilisées pour stocker et interroger des données fortement liées comme celles issues des réseaux sociaux.

Par ailleurs d'autres travaux [Izquierdo et al., 2016] proposent un processus en deux étapes pour l'extraction d'un modèle conceptuel (diagramme de classes UML) à partir d'un fichier JSON. La première étape consiste à extraire un modèle physique tel qu'il apparait dans le fichier JSON. La seconde étape génère le diagramme de classes UML en transformant le modèle physique en une classe racine *CR* puis les champs de type primitif (Number, String et Boolean) en des attributs de *CR* et les champs structurés en des classes composantes *CC* liées à *CR* par des liens de composition. Ainsi, ces travaux prennent uniquement en compte les liens de composition et ignorent les liens d'association.

D'autre part dans [Chillón et al., 2019], les auteurs proposent un processus qui décrit le passage d'une BD orientée-documents en un modèle conceptuel. Celui-ci est composé d'entités reliées par des liens. Une entité comporte une ou plusieurs versions selon les attributs qu'elles contiennent. Un attribut peut être atomique ou multivalué d'éléments atomiques et un lien peut être de type association ou composition. Un lien d'association entre deux entités est obtenu en transformant un champ *référence* (utilisant une syntaxe spécifique proposée par les auteurs). De plus, tout champ structuré dans le modèle physique est systématiquement transformé en un lien de composition. Ainsi, ces travaux ne considèrent ni les attributs structurés ni les classes d'associations dans le modèle conceptuel.

Enfin, dans l'article [Munoz-Sanchez et al., 2020], les auteurs ont proposé une approche d'extraction du modèle conceptuel d'une BD MongoDB. Cette approche comporte deux processus : (1) l'extraction du modèle physique, puis (2) la transformation de celui-ci en un modèle conceptuel (appelé « Logical » dans l'article). Le processus d'extraction du modèle physique est composé de quatre étapes. La première consiste à appliquer une fonction Map / Reduce à l'ensemble des documents composant les collections de la BD. Comme dans les articles [Sevilla et al., 2015] et [Gallinucci et al., 2018], le résultat retourné par cette étape est un ensemble de versions de modèles en format JSON ; chacune d'elles est commune à un groupe de documents d'une collection. La deuxième étape consiste à extraire les champs de référence utilisés pour exprimer des liens entre documents. Ces champs sont ajoutés ensuite aux versions de modèles déjà

extraites. La troisième étape quant à elle consiste à la construction d'un modèle physique unifié pour toute la BD en fusionnant les versions de modèles appartenant aux mêmes collections. La quatrième étape consiste à récupérer les statistiques en affichant pour chaque champ du modèle les fréquences minimales et maximales dans les documents de la BD. Le processus d'extraction du modèle conceptuel consiste à transformer le modèle physique déjà extrait en un modèle Entité/Association. Celui-ci est obtenu en appliquant une succession de quatre transformations. La première consiste à transformer une collection en une entité de même nom. La deuxième consiste à transformer les champs atomiques et multivalués en des attributs atomiques et multivalués respectivement. La troisième consiste à transformer un champ imbriqué en un lien d'agrégation. Enfin, la quatrième consiste à transformer un champ de référence en un lien d'association. Ainsi, ces travaux considèrent uniquement les liens d'association et d'agrégation et ignorent les autres types de liens, notamment les liens de composition, d'héritage et les classes d'associations. De plus, ils ne prennent pas en compte les attributs structurés dans le modèle conceptuel.

## 3.3 Synthèse

Nous présentons dans cette section une synthèse des travaux de recherche précédents en faisant apparaitre leurs caractéristiques suivantes comme le montre le tableau 3.1 :

- La composition de la BD NoSQL,
- Le type de la BD NoSQL,
- Le traitement des liens entre objets,
- Le niveau de modélisation.

| | BD comportant | | Système NoSQL | | | Traitement des liens entre objets | Niveau de modélisation | |
|---|---|---|---|---|---|---|---|---|
| | Une seule classe | Plusieurs classes | Type documents | Type graphes | | | Physique | Conceptuel |
| [Klettke et al., 2015] | X | | X | | | | X | |
| [Sevilla et al., 2015] | | X | X | | | | X | |
| [Gallinucci et al., 2018] | X | | X | | | | X | |
| [Maity et al., 2018] | X | | X | X | | | X | |
| [Baazizi et al., 2017 et 2019] | X | | X | | | | X | |
| [Comyn-Wattiau et al., 2018] | | X | | X | | X | X | X |
| [Izquierdo et al., 2016] | | X | X | | | X | X | X |
| [Chillón et al., 2019] | | X | X | | | X | X | X |
| [ Munoz-Sanchez et al., 2020] | | X | X | | | X | X | X |

**Tableau 3.1.** Tableau comparatif des travaux d'extraction des modèles de données à partir d'une BD NoSQL « schema less »

À travers le tableau 3.1, il apparaît que les solutions de l'état de l'art ne répondent que partiellement à notre problème. En effet, dans ([Klettke et al., 2015], [Izquierdo et al., 2016], [Gallinucci et al., 2018], [Maity et al., 2018], [Baazizi et al., 2017] et [Baazizi et al., 2019], les auteurs proposent des processus qui prennent en entrée une seule classe d'objets (documents ou nœuds) et ne prennent pas en compte les liens entre objets. De même, les travaux de [Sevilla et al., 2015] n'abordent pas les liens entre les collections. Par ailleurs, les travaux de [Comyn-Wattiau et al., 2018] ne prennent pas en compte les attributs structurés ; ceci est dû au fait qu'un système orienté graphe tel que Neo4J ne permet pas de déclarer ce type d'attributs.

D'autres parts, peu de travaux permettent l'extraction d'un modèle conceptuel pour les BD NoSQL « schema-less » ; ici le terme conceptuel qualifie un modèle sémantique exempt de toute considération technique. En effet, les travaux de [Comyn-Wattiau et al., 2018] portent uniquement sur les systèmes orientés-graphes qui ne prennent en compte ni les attributs structurés ni les liens de composition. D'autre part, dans l'article [Izquierdo et al., 2016], les auteurs ne considèrent pas les liens d'association qui s'avèrent les plus courants dans notre cas d'étude. Les travaux de [Chillón et al., 2019] ne prennent pas en compte les attributs structurés dans une classe ainsi que le concept de classe d'associations qui permet de caractériser une relation en lui rattachant des attributs. Enfin, les travaux de [Munoz-Sanchez et al., 2020] ne prennent en compte que les liens d'association et d'agrégation et ignorent les autres types de liens, notamment les liens de composition, d'héritage ainsi que les classes d'associations. De plus, ils ne prennent pas en compte les attributs structurés dans le modèle conceptuel généré.

## 3.4 Positionnement de nos travaux

A présent, nous effectuons le positionnement de nos travaux au regard des travaux présentées ci-dessus.

Dans la littérature, l'extraction du modèle de données des BD NoSQL « schema less » a fait l'objet de plusieurs travaux de recherche. La plupart de ces travaux se focalisent uniquement sur le niveau physique ([Klettke et al., 2015], [Izquierdo et al., 2016], [Gallinucci et al., 2018], [Maity et al., 2018], [Baazizi et al., 2017], [Baazizi et al., 2019] et [Sevilla et al., 2015]). Pour notre part, nous avons proposé le processus *ToPhysicalModel* qui extrait et met à jour automatiquement le modèle physique d'une BD NoSQL de type documents (cf. chapitre 4). Ce processus, basé sur l'architecture MDA (Model Driven Architecture), vise à extraire de la BD un modèle physique permettant aux utilisateurs d'exprimer des requêtes de type SQL. Le modèle physique obtenu est mis à jour ensuite au fur et à mesure de l'exploitation de la base (expression de requêtes de mise à jour). Le modèle physique de la BD est généré en appliquant une séquence de transformations formalisées avec le standard QVT. Le modèle retourné décrit les

collections qui composent la BD. L'apport majeur de notre solution se situe dans :

- (1) la prise en compte des liens entre collections,

- (2) l'extraction dynamique du modèle au fur et à mesure de l'exploitation de la base. En effet, le volume des données d'une BD NoSQL pouvant atteindre plusieurs téraoctets et la génération du modèle physique nécessite le balayage de la totalité de la BD. Il n'est pas donc envisageable de relancer le processus d'extraction du modèle après chaque mise à jour de la BD. Pour cela, nous préconisons un sous-processus qui consiste à mettre à jour le modèle physique au fur et à mesure que l'expression des requêtes de mise à jour sur la BD ; le modèle physique est ainsi disponible à tout moment.

Quant aux solutions industrielles MongoDB et Drill d'Apache, elles ne permettent d'afficher modèle physique d'une BD MongoDB qu'à minima. En effet, ces deux outils n'affichent pas les champs composant un champ structuré. De plus, ils ne considèrent pas les liens entre collections. Dans notre processus, le modèle généré prend en considération tous les champs des collections quels que soient leurs niveaux ainsi que les liens entre collections.

Pour l'extraction du modèle conceptuel, nous positionnons notre proposition au regard des travaux de recherche présentés dans la section 3.2 [Comyn-Wattiau et al., 2018], [Izquierdo et al., 2016], [Chillón et al., 2019] et [Munoz-Sanchez et al., 2020].

L'article [Comyn-Wattiau et al., 2018] traite de l'extraction d'un modèle Entité/Association à partir d'une BD NoSQL orientée-graphes. Or ce type de BD ne considère pas les attributs structurés et ne permet pas de distinguer les différents types de liens (association et composition) que l'on rencontre dans notre cas d'étude. D'autre part, les travaux présentés dans [Izquierdo et al., 2016] sont basés sur une BD NoSQL orientée documents. Ils font apparaître les liens de composition mais ne prennent pas en compte les liens d'association entre les collections. Or ce dernier type de liens constitue une pièce maîtresse dans les relations entre objets pour notre cas d'étude. Les travaux de [Chillón et al., 2019] présentent un processus d'extraction d'un modèle conceptuel pour une BD orientée-documents. Ils prennent en compte les liens d'association et les liens de composition. De plus, ils considèrent les attributs atomiques et multivalués d'éléments atomiques. Cependant, ces travaux n'étudient pas les attributs structurés ainsi que les classes d'associations. Enfin, les travaux de [Munoz-Sanchez et al., 2020] présentent une approche d'extraction d'un modèle Entité/Association à partir d'une BD MongoDB. Ces travaux prennent en considération les liens d'association et d'agrégation. Cependant, ils ignorent les autres types de liens, notamment les liens de composition, d'héritage ainsi que les classes d'associations. De plus, ils ne considèrent pas les attributs structurés.

Pour notre part, nous avons proposé le processus *ToConceptualModel* (cf. chapitre 5). Il s'agit d'un processus dirigé par les modèles qui consiste à transformer le modèle physique déjà extrait en un modèle conceptuel. Celui-ci est exprimé en utilisant le formalisme UML. Le passage entre le modèle physique et conceptuel est assuré par la définition d'un ensemble de règles de transformation de type Model-To-Model (M2M) formalisées avec le standard QVT. L'originalité du processus *ToConceptualModel* réside dans la prise en compte des liens d'association, de composition, d'héritage, d'agrégation, les classes d'associations ainsi que les attributs structurés.

# Chapitre 4 : Extraction du modèle physique de données d'une BD NoSQL orientée-documents

Dans la majorité des SGBD NoSQL, les bases de données (BD) sont schema-less (sans schéma), ce qui signifie absence du modèle de données lors de la création d'une BD. Autrement dit, dans une table, les noms des attributs ne sont précisés qu'au moment de l'insertion de leurs valeurs.

Cette propriété d'absence du schéma offre une flexibilité indéniable qui :

- facilite l'évolution du modèle de données (MD) au fur et à mesure de l'utilisation de la BD,
- et permet aux utilisateurs finaux d'ajouter de nouvelles informations sans avoir recours à l'administrateur de BD.

Mais, en contrepartie, cette propriété introduit un manque de visibilité sur l'organisation des données dans une BD NoSQL. Autrement dit, l'absence du MD ne permet pas à l'utilisateur de connaitre comment les données sont stockées (sous quel nom et quel type) et reliées dans la BD ; cette connaissance est pourtant indispensable pour l'expression des requêtes. En effet, pour écrire ses requêtes, l'utilisateur doit disposer du modèle de la BD où sont précisés les noms des tables, les noms des attributs ainsi que leurs types.

Actuellement dans l'industrie et pour surmonter cette difficulté, le même développeur (ou la même équipe de développement) se voit généralement confier la création de la base de données, le développement des interfaces de saisie des données et la rédaction des requêtes. Ainsi, le développeur connaît implicitement le modèle de données et peut donc exprimer des requêtes en connaissant la structure des données. Mais cette solution pragmatique n'est pas efficace dans la maintenance des applications puisque le développeur qui en est chargé, ne connaît pas le modèle de données. Il en va de même pour un décideur qui souhaite interroger une BD alors qu'il n'a pas participé à sa création.

Confronté à ce constat, nous sommes convaincus qu'il est important de fournir aux utilisateurs (qui peuvent être des développeurs ou des décideurs) deux modèles complémentaires décrivant la BD NoSQL schema-less :

- Le modèle physique qui décrit l'organisation interne des données et permet d'exprimer des requêtes.
- Le modèle conceptuel qui fait abstraction des aspects techniques et se concentre sur la sémantique de données (cf. Chapitre 5).

Dans ce chapitre, nous étudions uniquement la génération du modèle physique. Pour cela, nous proposons le processus *ToPhysicalModel.* Celui-ci est basé sur une approche dirigée par les modèles qui vise à formaliser et à automatiser l'élaboration d'un modèle physique à partir d'une BD NoSQL schema-less. Le processus de génération du modèle conceptuel sera étudié dans le chapitre 5.

Ce chapitre 4 est structuré comme suit. La section 4.1 montre un aperçu de notre processus d'extraction du modèle physique. Les sections 4.2 et 4.3 présentent

respectivement les sous-processus composant notre solution « à froid » et « à chaud ». Enfin, la section 4.4 conclut ce chapitre en mettant en relief nos contributions en termes d'extraction du modèle physique.

# 4.1 Aperçu de notre processus

L'objectif de ce chapitre est d'automatiser l'extraction du modèle physique de données (MPD) d'une BD gérée par un système NoSQL de type schema-less. A la suite de la justification donnée dans le chapitre 2 (cf. Section 2.2.4), nous avons limité notre étude aux BD gérées par des systèmes NoSQL de type orientés-documents.

Le MPD décrit la structure physique de données, c'est-à-dire la manière dont les données sont implantées. En effet, il indique les noms des tables, les noms des attributs ainsi que leurs types comme on le fait dans une BD relationnelle où le modèle est fourni par le système.

Pour ce faire, nous proposons le processus *ToPhysicalModel* qui produit et met à jour le MPD d'une BD NoSQL orientée-documents. Comme le montre la figure 4.1, *ToPhysicalModel* est composé de deux sous-processus qui s'appliquent successivement :

- *ModelExtraction* : qui consiste à extraire le MPD à partir de la BD d'entrée ; il s'agit du sous-processus « à froid ».
- *ModelUpdate* : qui recalcule le MPD pendant l'exploitation des données, c'est-à-dire au fur et à mesure de l'exécution des requêtes de mise à jour sur la BD ; il s'agit du sous-processus « à chaud ».

Notons que dans le cas où la BD est vide, le sous-processus *ModelExtraction* retournera un MPD vide. Celui-ci ne sera alimenté que si le sous-processus *ModelUpdate* est déclenché à la suite de l'exécution d'une requête de mise à jour. Ces deux sous-processus se résument comme suit :

**(1) Sous-processus *ModelExtraction* :**

- Source : Une BD NoSQL de type orienté-documents.

- Cible : - Un MPD NoSQL (version i) décrivant les collections composants la BD ainsi que les liens entre elles,

- Des métadonnées de la BD (version i) qui seront utilisées pour le traitement des requêtes de suppression et de modification dans le processus *ModelUpdate*.

- Nature des transformations : le passage entre la source et la cible est assuré par des transformations Model-To-Model ; celles-ci sont exprimées par un ensemble de règles formalisées en QVT[19].

[19] OMG, Q. (2011). Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1. 1.

- Description synthétique : ce sous-processus permet de générer un MPD NoSQL à partir d'une BD de type orienté-documents. Son principe consiste à balayer la totalité de la BD et à fournir une structure interne commune aux données. Parallèlement, *ModelExtraction* crée et enrichit les métadonnées de la BD.

**(2) Sous-processus *ModelUpdate* :**

**-** Source : - Des requêtes de mise à jour,

- Un MPD NoSQL ; notons que la première version du MPD de ce sous-processus correspond au résultat du sous-processus *ModelExtraction,*

- Des métadonnées de la BD ; la première version des métadonnées de ce sous-processus est générée par sous-processus *ModelExtraction,*

- Cible : - un MPD (version i+1),

- des métadonnées (version i+1),

- Nature des transformations : le passage entre la source et la cible est assuré par des transformations Model-To-Model.

- Description synthétique : le sous-processus *ModelUpdate* procède comme suit. D'abord, il analyse la requête de mise à jour pour déterminer la collection sur laquelle elle porte ainsi que ses attributs. Ensuite, il compare ces éléments avec la version actuelle du MPD. Si celle-ci ne correspond pas aux éléments de la requête, alors il met à jour le MPD. Sinon, il garde la version précédente. Parallèlement à l'élaboration du modèle, *ModelUpdate* met à jour des métadonnées.

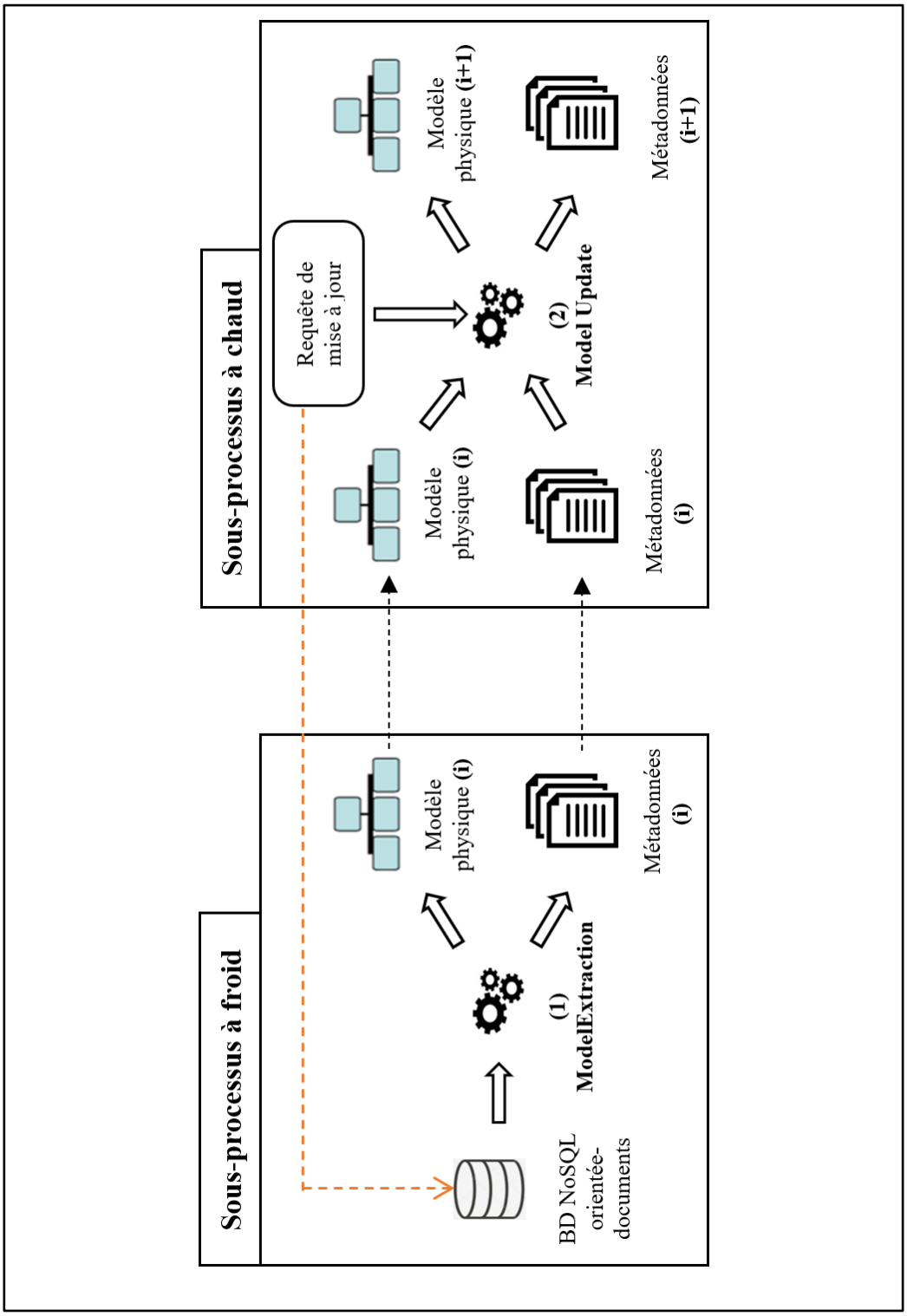



**Figure 4.1 -** Approche d'extraction et de mise à jour du MPD d'une BD

Comme annoncé dans les chapitres précédents, nos travaux s'inscrivent dans l'ingénierie dirigée par les modèles (IDM). Celle-ci consiste à utiliser les modèles dans les différentes phases du cycle de développement d'un logiciel [Combemale, 2008]. L'IDM préconise l'utilisation de l'architecture MDA (Model Driven Architecture) de l'OMG qui offre un cadre formel pour l'automatisation des transformations de modèles. L'objectif de cette architecture est de décrire séparément les spécifications fonctionnelles et les spécifications d'implantation d'une application sur une plateforme donnée [Hutchinson et al., 2011]. Comme nous l'avons indiqué précédemment (cf. Chapitre 2 – Section 2.3.2), une démarche MDA préconise l'utilisation de trois modèles CIM, PIM et PSM ainsi que deux types de transformations : M2M ou M2T. Cependant, le nombre et le type de modèles et de transformations sont propres à chaque démarche. Etant donné que dans le processus *ToPhysicalModel* nous avons manipulé les quatre éléments suivants : (1) la BD NoSQL, (2) les métadonnées, (3) les requêtes de mise à jour et (4) le MPD, nous retenons uniquement le niveau PSM. Comme le montre la figure 4.2, nous distinguons quatre modèles à ce niveau :

- PSM-BD : il s'agit du modèle décrivant le contenu de la BD.
- PSM-MPD : c'est le modèle physique de la BD.
- PSM-MetaData : décrit les métadonnées de la BD utilisées lors du sous-processus « à chaud ».
- PSM-Query : modèle décrivant les requêtes de mise à jour appliquées à la BD et qui sont manipulées lors du sous-processus « à chaud ».

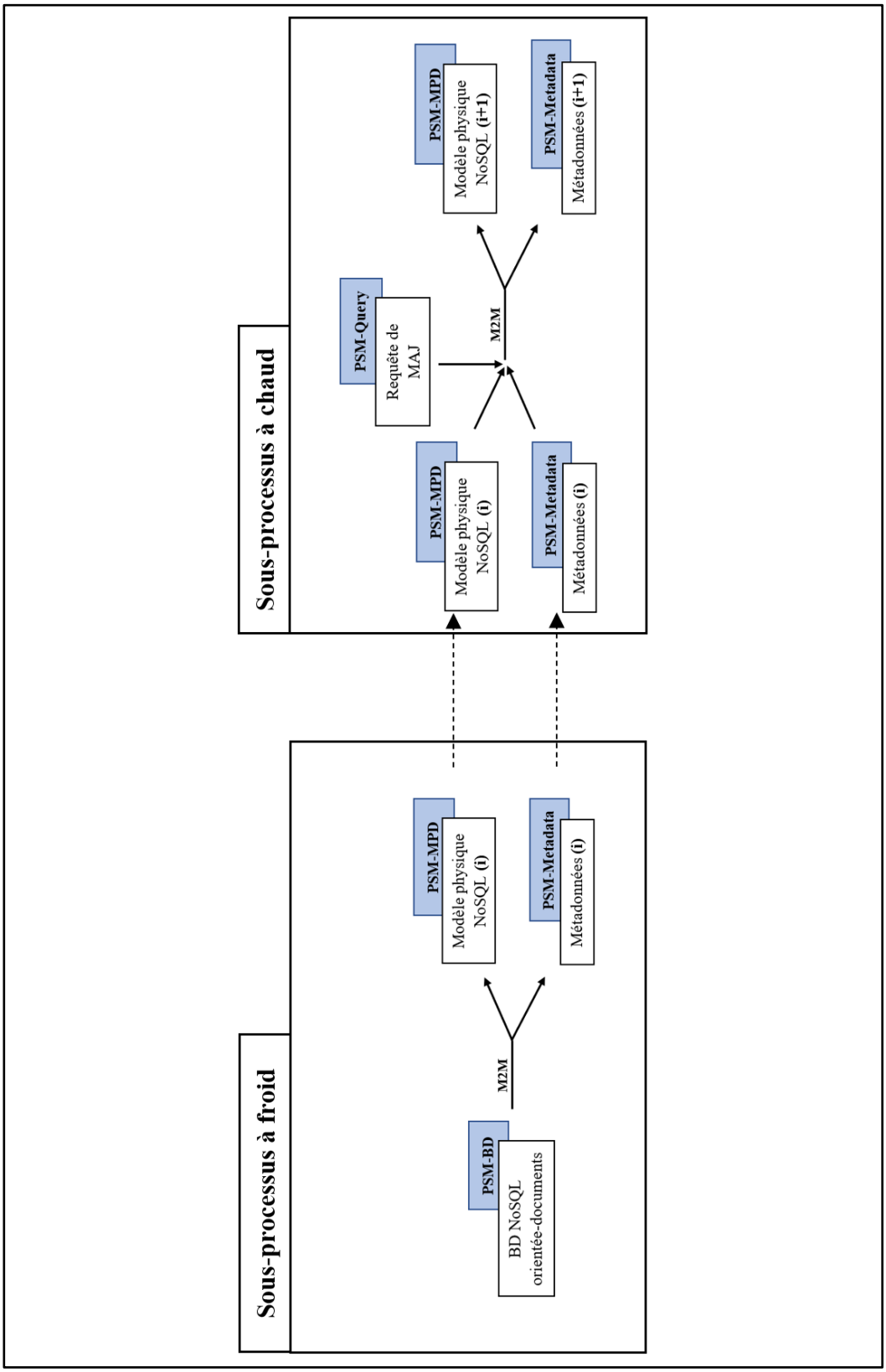



**Figure 4.2 -** Niveau de modélisation du processus *ToPhysicalModel*

Le passage d'un modèle à un autre se fait en utilisant des transformations de type M2M (Model-To-Model). Nous allons formaliser par la suite ces transformations en utilisant le formalisme standard QVT (Query View Transformation) défini par l'OMG pour la transformation de modèles (voir les Sections 4.2.3 et 4.3.3).

Dans les sections suivantes, nous détaillons les composants de chaque sous-processus de notre solution en précisant les trois éléments suivants : (a) la source, (b) la cible et (c) les règles de transformation associées.

# 4.2 Sous-processus « à froid »

Dans cette section, nous présentons le sous-processus d'extraction dit « à froid » du MPD. Ce sous-processus consiste à parcourir la totalité d'une BD NoSQL existante à un instant t ; le MPD résultant décrit les structures de données en t. Un sous-processus complémentaire dit « à chaud » sera étudié plus tard dans ce chapitre pour mettre à jour le MPD tout au long de l'exploitation de la BD (cf. Section 4.3).

*ModelExtraction* est le premier sous-processus de *ToPhysicalModel*. Il part d'une BD NoSQL orientée-documents et elle retourne son MPD ainsi qu'un ensemble de métadonnées. Ces dernières sont stockées sous la forme d'une BD distincte notée PSM-MetaData. Celle-ci est composée d'un ensemble de collections. Chacune d'elles correspond à une collection dans la BD source et contient un ensemble de champs de la forme (Nom-champ, compteur), où le compteur représente le nombre d'apparitions du champ en question dans la collection correspondante. Ces métadonnées sont utiles dans le deuxième sous-processus *ModelUpdate* pour le traitement des requêtes de mise à jour de type suppression et de type modification. En effet, elles indiquent si la suppression ou la modification d'un champ donné affecte le MPD ou non. Par exemple, si le nombre d'apparition d'un champ X dans la BD est 1, alors la suppression de ce champ de la BD implique aussi sa suppression du MPD ; sinon, le sous-processus *ModelUpdate* maintient la dernière version du MPD avant l'exécution de la requête en question.

Notons que les éléments source (BD NoSQL orientée-documents) et cible (MPD + Métadonnées) du sous-processus *ModelExtraction* sont respectivement conformes aux métamodèles des PSM-BD, PSM-MPD et PSM-MetaData présentés dans les sections ci-dessous.

Dans la figure 4.3, nous présentons un aperçu du sous-processus « à froid ».

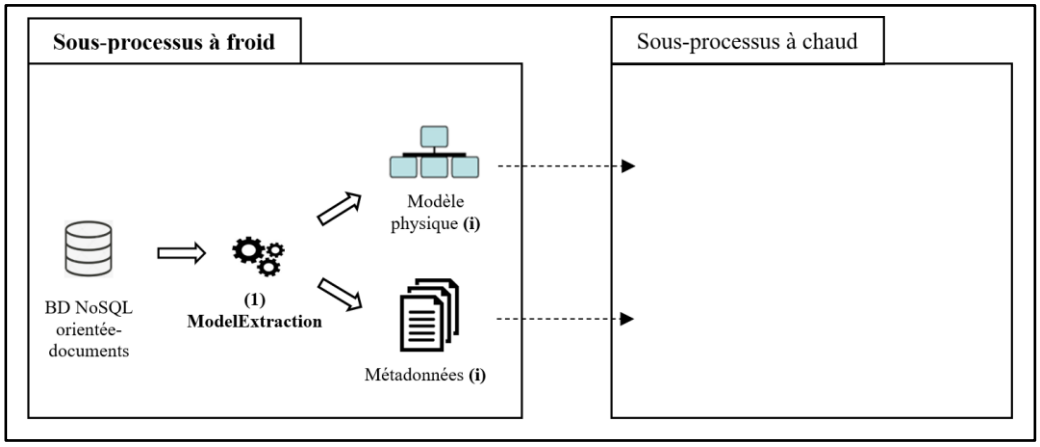



**Figure 4.3 -** Sous-processus d'extraction à froid

Nous commençons d'abord par définir la source (BD NoSQL orientée-documents) et la cible (MPD + Métadonnées), puis nous présentons les règles de transformation.

### 4.2.1 La source : PSM-BD

Avant d'assurer une extraction automatique du MPD d'une BD NoSQL orientée-documents, nous formalisons préalablement les concepts présents dans cette BD. Pour cela, Nous nous appuyons sur les caractéristiques et le vocabulaire du système MongoDB. Celui-ci est le SGBD NoSQL orienté-documents le plus utilisé au monde, comme le montre une étude récente de DB-Engines [DB-Engines Ranking, n.d.].

Dans MongoDB, une BD est composée de collections. Chaque collection contient un ensemble de documents. Ainsi, à l'instar d'une table composée de lignes dans une BD relationnelle, une collection regroupe des documents dans une BD MongoDB.

Formellement :

**Définition 1.** Une base de donnes NoSQL orientée-documents *BD* est définie par un couple *(N, C)*, où :

- *N* est le nom de *BD*,
- $C = \{c_1, \dots, c_n\}$ est l'ensemble de collections qui composent *BD*.

**Définition 2.** $\forall\ i \in [1..n]$, une collection $c_i \in C$ est définie par un couple *(N, D)*, où :

- $c_i.N$ est le nom de la collection $c_i$,
- $c_i.D = \{d_1, \dots, d_m\}$ est l'ensemble de documents de la collection $c_i$.

Un document est toujours identifié par une clé système, notée *_id*, à laquelle est associé un agrégat de champs sous la forme nom/valeur. La valeur d'un champ peut être atomique, structurée (i.e. qu'elle est composée d'autres champs) ou multivaluée (composée d'un ensemble de valeurs). Notons que les documents regroupés dans la même collection ne possèdent pas forcément les mêmes champs.

Formellement :

**Définition 3.** $\forall\, j \in$ [1..m], un document $d_j \in D$ est défini par un couple ($Id$, $F$) où :

- $d_j.Id$ est l'identifiant du document $d_j$. Il est noté par _id.
- $d_j.F = AF \cup SF \cup MF$ est l'ensemble de champs du document $d_j$, où :
  - $AF = \{af_1, \ldots, af_p\}$ est l'ensemble des champs atomiques du document $d_j$,
  - $SF = \{sf_1, \ldots, sf_q\}$ est l'ensemble des champs structurés du document $d_j$.
  - $MF = \{mf_1, \ldots, mf_r\}$ est l'ensemble des champs multivalués du document $d_j$.

Formellement :

**Définition 4.** $\forall$ k $\in$ [1..p], un champ atomique $af_k \in AF$ est défini par un couple *(N, v)* où :

- $af_k$.N est le nom du champ $af_k$,
- $af_k$.v est la valeur du champ $af_k$ ; elle est d'un type de données prédéfini (String, Integer, Boolean, Date, etc.)

**Définition 5.** $\forall$ l $\in$ [1..q], un champ structuré $sf_l \in SF$ est défini par un couple ($N$, $F'$) où :

- $sf_l$.N est le nom du champ $sf_l$ ,
- $sf_l$ . $F' = AF' \cup SF' \cup MF'$ est l'ensemble de champs atomiques, structurés et multivalués qui composent $sf_l$ . Les sous-ensembles $AF'$, $SF'$ et $MF'$ sont respectivement définis de la même manière que $d_j.AF$, $d_j.SF$ et $d_j.MF$ avec $j \in$ [1..m].

**Définition 6.** $\forall$ z $\in$ [1..r], un champ multivalué $mf_l \in MF$ est défini par un couple ($N$, V) où :

- $mf_z$.N est le nom du champ $mf_z$ ,
- $mf_z.V = \{v_1, \ldots, v_s\}$ est l'ensemble de valeurs du champ $mf_z$. Les valeurs $v_u$ avec u $\in$ [1..s], sont toutes de type atomique (cf. Définition 4), de type structuré (cf. Définition 5) ou de type multivalué.

Pour exprimer un lien entre deux collections, nous avons utilisé un champ intitulé *DBRef* conformément au principe préconisé dans la documentation de MongoDB [MongoDB, 2018]. Un champ de référence *DBRef* est un champ structuré. Il permet de lier un document à un autre en utilisant deux informations : l'identifiant du document référencé et le nom de la collection où se trouve celui-ci. Notons cependant que le système MongoDB ne vérifie pas l'intégrité référentielle lors de l'alimentation de la BD ; cette vérification reste à la charge du logiciel de saisie et de mise à jour.

Formellement :

**Définition 7.** Un champ *DBRef* est défini par un couple $(N, F')$ où :

- $sf_l$.N est le nom du lien.
- $F'$ contient deux champs atomiques **:** le premier, de la forme ***$id: ObjectId*,** identifie le document référencé ; le second, de la forme ***$Ref: NomCollection,*** désigne le nom de la collection cible.

Dans la figure 4.4, nous donnons un exemple d'utilisation du champ DBRef. Dans cet exemple le DBRef est nommé *MédecinTraitant* et il relie les deux collections *Patients* et *Médecins*.

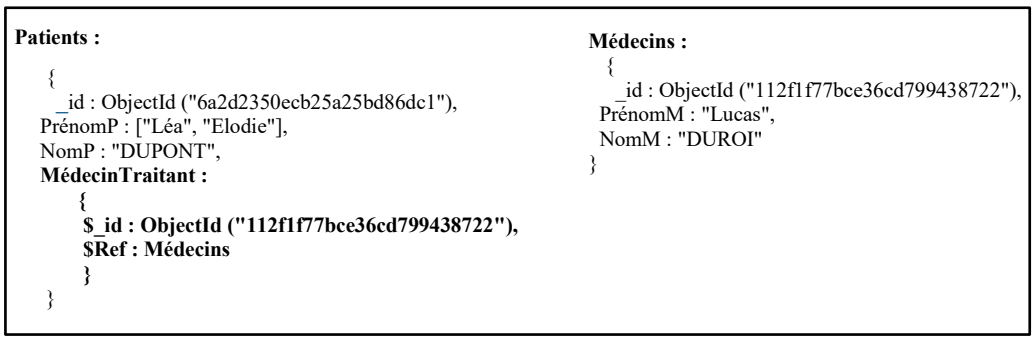



**Figure 4.4 -** Exemple de champ *DBRef dans MongoDB*

Nous avons étendu la syntaxe de *DBRef* pour prendre en compte les liens multivalués, les liens n-aires ainsi que les classes d'associations. Dans cette nouvelle syntaxe (*DBRef étendu*), $F'$ peut contenir plusieurs paires ***($id, $Ref)*** et éventuellement d'autres champs atomiques. Nous illustrons cette nouvelle syntaxe par les figures 4.5 et 4.6.

Formellement :

**Définition 8.** Un champ *DBRef* étendu est défini par un couple ($N$, $F'$) où :

- $sf_l$.N est le nom du lien.
- $F'$ peut contenir plusieurs paires ***($id: [ObjectId_1, \ldots, ObjectId_y], $Ref: NomCollection).*** Chaque paire permet de référencer des documents d'une collection. $\forall$ x $\in$ [1..y], $ObjectId_x$ est l'identifiant d'un document référencé appartenant à la collection dont le nom est *NomCollection*. Ainsi, si y $\geq$ 2, le lien est multivalué ; il est monovalué sinon. $F'$ peut contenir aussi d'éventuels champs. Ceux-ci sont utilisés pour simuler l'utilisation d'une classe d'associations. Ce choix nous a été dicté par la définition qui stipule qu'une classe d'associations est une association qui contient des attributs.

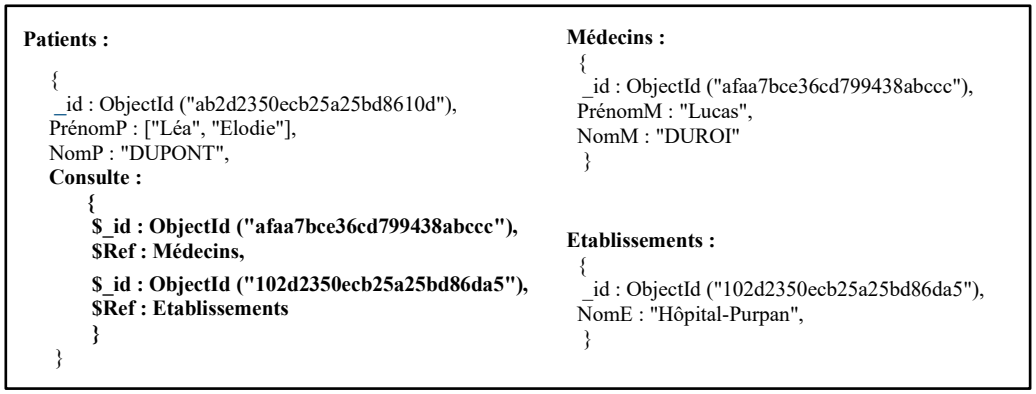



**Figure 4.5 -** Exemple d'un lien ternaire dans MongoDB

La figure 4.5 présente un lien ternaire intitulé *Consulte* déclaré dans la collection *Patients* et qui pointe vers les collections *Médecins* et *Etablissements*. La figure 4.6 quant à elle montre une classe d'associations intitulée *Exerce* entre Les collections *Médecins* et *Spécialités.* Cette classe d'association est caractérisée par le champ *DateSpécialisation*

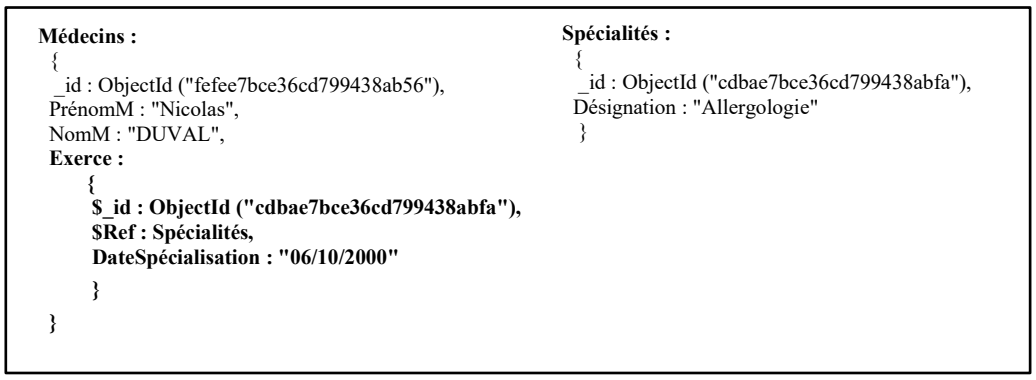



**Figure 4.6 -** Exemple d'une classe d'associations dans MongoDB

MongoDB ne permet pas d'exprimer des liens d'héritage, de composition et d'agrégation entre collections[20]. Nous proposons des modes d'implantation pour simuler l'utilisation de ces liens. Pour cela, nous suggérons trois champs : *DBSub*, *DBComp* et *DBAgg* en s'appuyant sur le principe de *DBRef*.

Pour exprimer un lien d'héritage entre collections dans une BD MongoDB, nous proposons d'utiliser un champ *DBSub* dans la sous-collection. *DBSub* est un champ structuré qui contient deux champs atomiques : l'un référence le document générique et l'autre désigne le nom de la collection générique.

Formellement :

**Définition 9.** Un champ *DBSub* est un champ structuré défini par un couple $(N, F')$ où :

- $sf_1$.N est *ISA*.
- $F'$ contient deux champs atomiques **:** le premier, de la forme ***$id: ObjectId*,** identifie le document générique ; le second, de la forme ***$ISA: NomCollection,*** correspond au nom de la collection générique.

Dans la figure 4.7, nous exprimons un lien d'héritage entre les collections *Spécialistes* et *Médecins*.

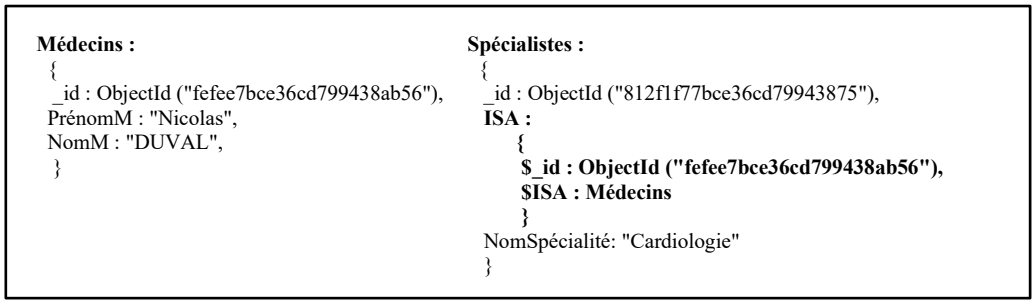



**Figure 4.7 -** Exemple d'un lien d'héritage dans MongoDB

Pour exprimer un lien de composition entre collections dans une BD MongoDB, nous proposons d'utiliser un champ *DBComp* dans la collection composite. *DBComp* est un champ structuré qui contient deux champs. Le premier est multivalué ; il sert à identifier les documents composants. Le second est atomique ; il désigne le nom de la collection composante.

Pour exprimer la dépendance du cycle de vie de la collection composante avec celui de la collection composite, nous définissons une contrainte d'intégrité dans

[20] Certains auteurs proposent d'exprimer des liens de composition sous la forme d'un attribut multivalué structuré. Par exemple, les enfants de personnes peuvent être imbriqués dans la collection *Personne* sous la forme d'un attribut de la collection. Mais dans ce cas, les enfants ne peuvent pas être référencés (liés) depuis une autre collection.

le code de l'application qui utilise la BD. Cette contrainte consiste à supprimer les documents composants une fois que leur document composite est supprimé.

Formellement :

**Définition 10.** Un champ *DBComp* est un champ structuré défini par un couple $(N, F')$ où :

- $sf_1$.N est *Comp*.
- $F'$ contient deux champs. Le premier est multivalué, de la forme ***$id: [ObjectId$_1$, … , ObjectId$_y$]***. ∀ x ∈ [1..y], *ObjectId*$_x$ est l'identifiant d'un document composant. Le second champ est atomique, de la forme ***$Comp: NomCollection ;*** il désigne le nom de la collection composante.

Nous présentons dans la figure 4.8 un exemple d'un lien de composition les collection *Hôpitaux* et *Services*.


**Figure 4.8 -** Exemple d'un lien de composition dans MongoDB

Pour exprimer un lien d'agrégation entre collections dans une BD MongoDB, nous proposons d'utiliser un champ *DBAgg* dans la collection agrégée. *DBAgg* est un champ structuré qui contient deux champs atomiques : l'un identifie le document agrégat et l'autre désigne le nom de la collection agrégat.

Formellement :

**Définition 11.** Un champ *DBAgg* est un champ structuré défini par un couple $(N, F')$ où :

- $sf_l$.N est *Agg*.
- $F'$ contient deux champs atomiques **:** le premier, de la forme ***$id: ObjectId*****,** identifie le document agrégat ; le second, de la forme ***$Agg: NomCollection,*** désigne le nom de la collection agrégat.

Un exemple de lien d'agrégation entre collection d'une BD MongoDB est donné dans la figure 4.9.

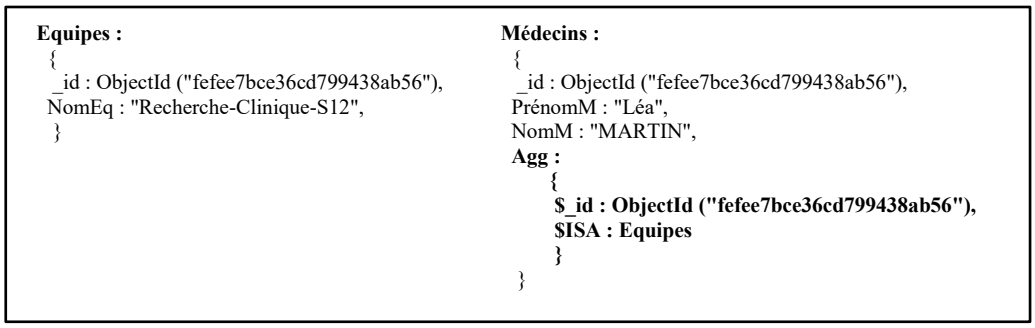



**Figure 4.9 -** Exemple d'un lien d'agrégation

Nous présentons ces différents concepts à travers un métamodèle de la figure 4.10. Ce métamodèle ainsi que tous les métamodèles présentés dans ce mémoire sont implantés et validés en utilisant la plateforme Eclipse Modeling Framework (EMF) présentée dans le chapitre 6.

Ce métamodèle montre les éléments composant une BD NoSQL orientée-documents. Ainsi, une BD NoSQL orientée-documents est composée de collections. Chaque collection contient un ensemble de documents. Un document est constitué d'un ensemble de champs ; chacun d'eux peut être atomique, structuré (i.e. composé d'autres champs) ou multivalué.

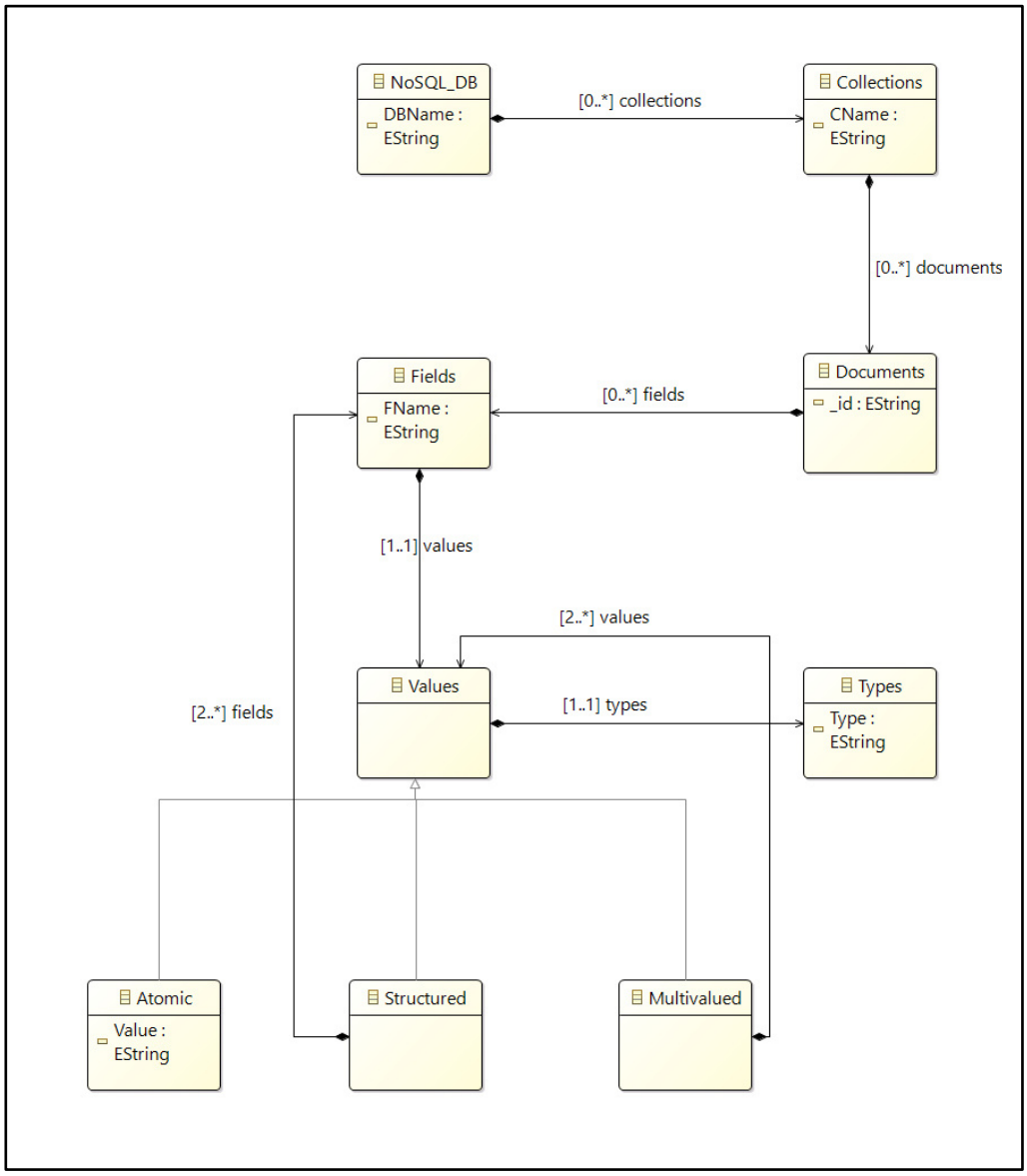



**Figure 4.10 -** Métamodèle du PSM-BD

## 4.4.2 La cible :

### 4.2.2.1 PSM-MPD

Le MPD généré par le processus *ToPhysicalModel* fait apparaître l'organisation interne des données de la BD NoSQL ; il permet aux utilisateurs d'exprimer des requêtes sur la BD.

Nous avons choisi de stocker le MPD sous la forme de collections dans MongoDB. Chacune d'elles contient un seul document. Celui-ci décrit le modèle d'une collection en entrée. Ce document est composé d'un ensemble de champs ;

Chacun d'eux peut être soit atomique, structuré ou multivalué. Un champ atomique est présenté sous la forme d'un couple (Nom, Type) ; Type correspond à un type de données prédéfini comme Integer, Boolean et String, Date, etc. Un champ structuré est composé d'un ensemble de champs qui, à leurs tours, peuvent être atomiques, structurés ou multivalués. Un champ multivalué est composé d'un ensemble de valeurs qui peuvent être atomiques, structurées ou multivaluées.

Formellement :

**Définition 12.** Le modèle physique de données *MPD* est défini par un couple *(N, $C'$)*, où :

- *N* est le nom de *MPD*,
- $C' = \{c'_1, \dots, c'_n\}$ est un ensemble de collections (cf. Définition 2). $\forall\, i \in$ [1..n], une collection $c'_i \in MPD.\ C'$ représente le modèle physique d'une collection en entrée $c_i \in BD.C$.

Une collection $c'_i$ est définie par un couple *(N, $F'$)* où :

- $c'_i.N$ est le nom de la collection $c'_i$,
- $c_i'.F' = AF' \cup SF' \cup MF'$ est l'ensemble de champs figurant dans les documents de la collection $c_i \in BD.C$.
  - $AF' = \{af'_1, \dots, af'_p\}$ est l'ensemble des champs atomiques de la collection $c_i$,
  - $SF' = \{sf'_1, \dots, sf'_q\}$ est l'ensemble des champs structurés de la collection $c_i$,
  - $MF' = \{mf'_1, \dots, mf'_r\}$ est l'ensemble des champs multivalués de la collection $c_i$.

$\forall$ k $\in$ [1..p], un champ atomique $af'_k \in AF'$ est défini par un couple *(N, T)* où :

  - $af'_k$.N est le nom du champ $af'_k$,
  - $af'_k$.T est le type du champ $af'_k$ ; T peut être String, Integer, Boolean, Date, etc.

$\forall$ l $\in$ [1..q], un champ structuré $sf'_l \in SF'$ est défini par un couple ($N$, $F''$) où :

  - $sf'_l$.N est le nom du champ $sf'_l$ ,
  - $sf'_l\ .\ F'' = AF'' \cup SF'' \cup MF''$ est l'ensemble de champs atomiques, structurés et multivalués qui composent $sf'_l$. Les sous-ensembles $AF''$, $SF''$ et $MF''$ sont définis respectivement de la même manière que $c'_i.F'.AF'$, $c'_i.F'.SF'$ et $c'_i.F'.MF'$,

∀ z ∈ [1..r], un champ multivalué $mf'_z \in MF'$ est défini par un triplet (*N*, T, $F''$) où :

- $mf'_z$.N est le nom du champ $mf'_z$,
- $mf'_z$.T est le type du champ $mf'_z$ dans le cas où celui-ci est multivalué de valeurs atomiques ; $mf'_z$.T est *Null* sinon.
- $mf'_z.F'' = AF''' \cup SF''' \cup MF'''$ est l'ensemble de champs atomiques, structurés et multivalués qui composent le champ $mf'_z$. Si celui-ci est multivalué de valeurs atomiques $mf'_z.F'' = \emptyset$.

Les différents concepts du MPD sont présentés au travers du méta-modèle de la figure 4.11.

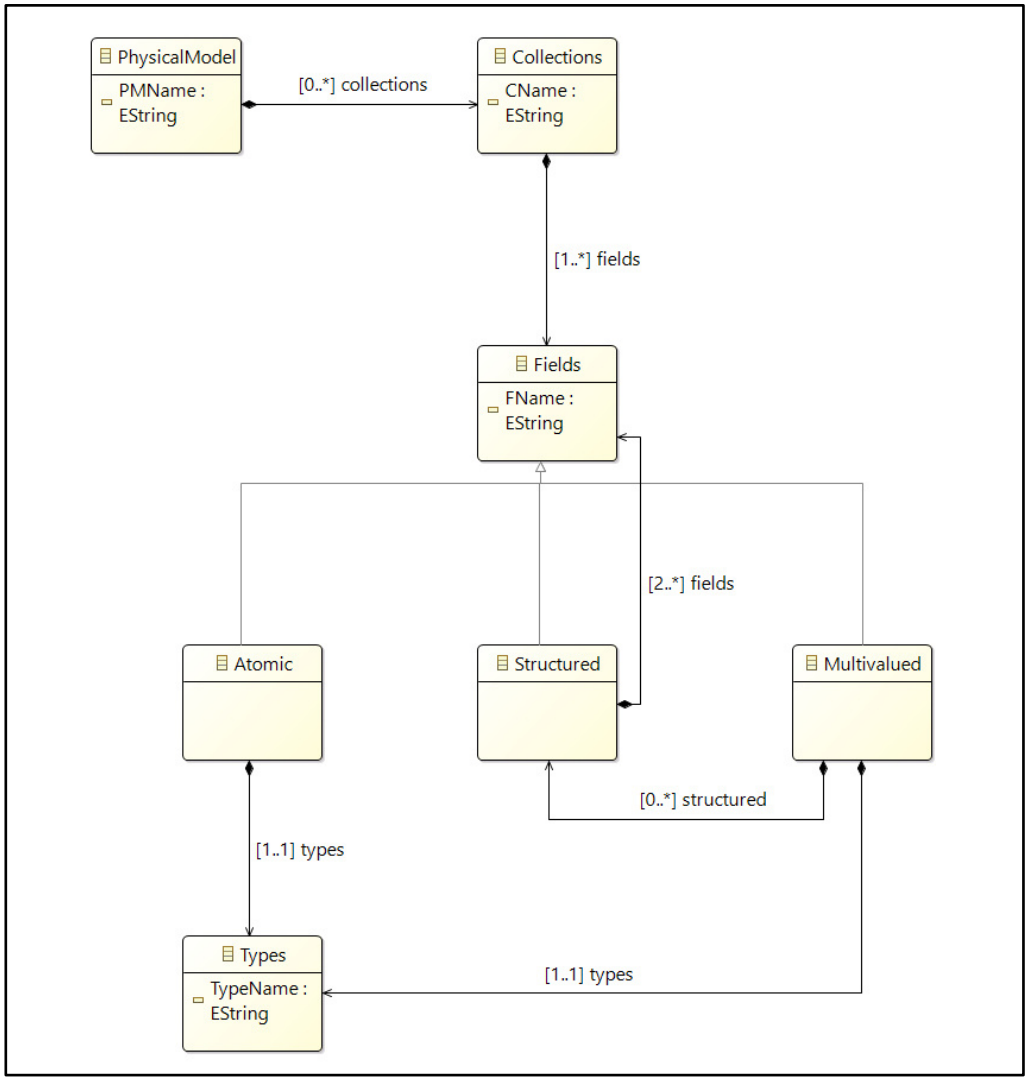



**Figure 4.11 -** Métamodèle du PSM-MPD

#### 4.2.2.2 PSM-MetaData

La deuxième sortie du processus *ModelExtraction* correspond aux métadonnées *MD*. Celles-ci seront utilisées plus tard dans le sous-processus *ModelUpdate* (extraction à chaud) pour le traitement des requêtes de mise à jour de type suppression et modification. En effet, elles indiquent si la suppression ou la modification d'un champ donné affecte le MPD ou non.

Nous avons choisi de stocker les métadonnées sous forme de collections dans MongoDB. Chacune d'elles contient un seul document. Celui-ci contient les métadonnées de la collection correspondante en entrée. Un document contient un ensemble de champs de la forme (nom-champ, compteur), où le compteur représente le nombre d'apparition du champ en question dans la collection correspondante. Par exemple, si le compteur associé à un champ X est 1, alors la suppression de ce champ de la BD implique aussi sa suppression au niveau du MPD ; sinon, le sous-processus garde la dernière version du MPD avant l'exécution de la requête en question.

Bien que les métadonnées soient utilisées uniquement que dans le sous-processus « à chaud », elles sont initialisées dans le sous-processus « à froid ».

Formellement :

**Définition 13.** Les métadonnées *MD* sont définies par un couple *(N,* $C^{MD}$*)*, où :

- *N* est le nom des *MD*,
- $C^{MD} = \{c^{MD}{}_1, \dots, c^{MD}{}_n\}$ est un ensemble de collections. $\forall\, i \in [1..n]$, une collection $c^{MD}{}_i \in MD.C^{MD}$ contient les métadonnées d'une collection en entrée $c_i \in BD.C$.

Une collection $c^{MD}{}_i$ est définie par un couple *(N, F′)* où :

- $c^{MD}{}_i.N$ est le nom de la collection $c^{MD}{}_i$,
- $c^{MD}{}_i.F' = AF' \cup SF' \cup MF'$ est l'ensemble de champs figurant dans les documents de la collection $c_i \in BD.C$.
  - $AF' = \{af'_1, \dots, af'_p\}$ est l'ensemble des champs atomiques de la collection $c_i$.
  - $SF' = \{sf'_1, \dots, sf'_q\}$ est l'ensemble des champs structurés de la collection $c_i$
  - $MF' = \{mf'_1, \dots, sf'_r\}$ est l'ensemble des champs multivalués de la collection $c_i$

$\forall$ k $\in$ [1..p], un champ atomique $af'_k \in AF'$ est défini par un couple *(N,* $Cpt$*)* où :

  - $af'_k.N$ est le nom du champ $af'_k$,

- $af'_k.Cpt$ est le nombre d'occurrence de $af'_k$ dans la collection $c_i$.

∀ l ∈ [1..q], un champ structuré $sf'_l \in SF'$ est défini par un triplet ($N$, $F''$, $Cpt$) où :

- $sf'_l$.N est le nom du champ $sf'_l$ ,
- $sf'_l.F'' = AF'' \cup SF'' \cup MF''$ est l'ensemble de champs atomiques, structurés et multivalués qui composent $sf'_l$. Les sous-ensembles $AF''$, $SF''$ et $MF''$sont définis respectivement de la même manière que $c^{MD}{}_i.AF'$, $c^{MD}{}_i.SF'$ et $c^{MD}{}_i.MF'$,
- $sf'_l.Cpt$ est le nombre d'occurrence de $sf'_l$ dans la collection $c_i$.

∀ z ∈ [1..r], un champ multivalué $mf'_z \in MF'$ est défini par un triplet ($N$, $F''$, $Cpt$) où :

- $mf'_z$.N est le nom du champ $mf'_z$,
- $mf'_z.F'' = AF''' \cup SF''' \cup MF'''$ est l'ensemble de champs atomiques, structurés et multivalués qui composent le champ $mf'_z$. Si celui-ci est multivalué de valeurs atomiques $mf'_z.F'' = \emptyset$,
- $mf'_z.Cpt$ est le nombre d'occurrence de $mf'_z$ dans la collection $c_i$.

Nous présentons les différents concepts des métadonnées au travers le méta-modèle de la figure 4.12.

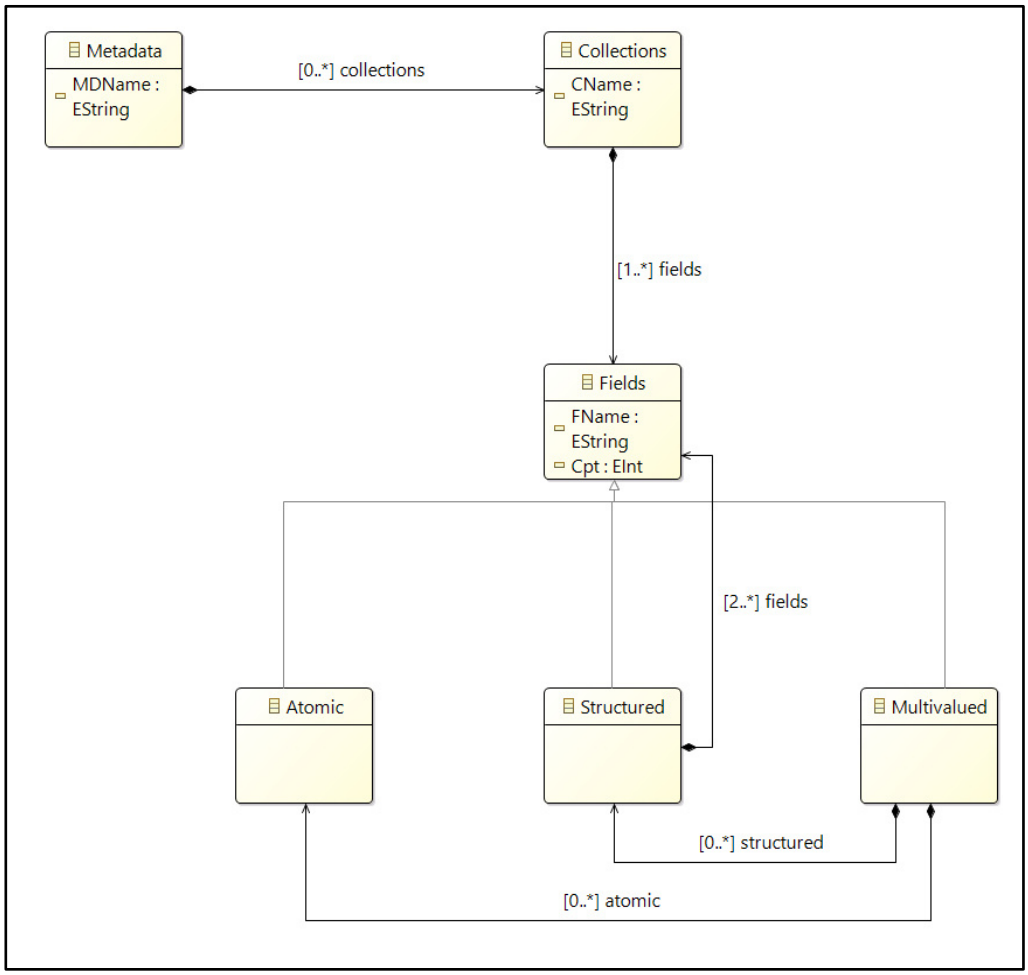



**Figure 4.12 -** Métamodèle du PSM-MetaData

Nous présentons dans la figure 4.13, un exemple du PSM-MetaData. Celui-ci contient les métadonnées de la collection *Patients*. Par exemple, le compteur associé au champ *NomPatient* est 18. Cela veut dire que le total de nombre d'apparitions de ce champ dans tous les documents de collection *Patients* au niveau de la BD est 18. Quant aux autres champs de la collection, ils peuvent avoir des valeurs de compteurs différentes. Cela est assuré par la variété des données d'une BD MongoDB. En d'autres termes, les documents constituant une collection n'ont pas nécessairement le même schéma ; ainsi, ils n'ont pas le même nombre d'apparitions.

**Patients :**

{
_id : 20,
NSS : 20,
NomPatient : 18,
PrénomPatient : 15,
DateNaissance : 14
}

**Figure 4.13 -** Exemple de métadonnées

### 4.2.3 Règles de transformation

Après avoir formalisé les concepts présents dans le modèle source (PSM-BD) et dans le modèle cible (PSM-MPD), nous allons décrire le passage automatique entre les deux modèles sous forme d'une suite de cinq règles de transformation.

**R1 :** Une base de données NoSQL orientée-documents *BD* est transformée en un modèle physique *MP* où :

- *MP*.N = *BD*.N.

**R2 :** $\forall\ i \in [1..n]$, une collection $c_i \in BD.C$ est transformée en une collection $c'_i \in MP.\ C'$ où :

- $c'_i$.N = $c_i$.N,

**Exemple :**

Dans l'exemple de la figure 4.14, notre processus applique la règle R1 et R2 comme suit :

- La BD est transformée en un modèle physique MP en appliquant la règle R1.
- Les collections *Patients*, *Doctors*, *Hospitals* et *Specialties* sont transformées en appliquant la règle R2.

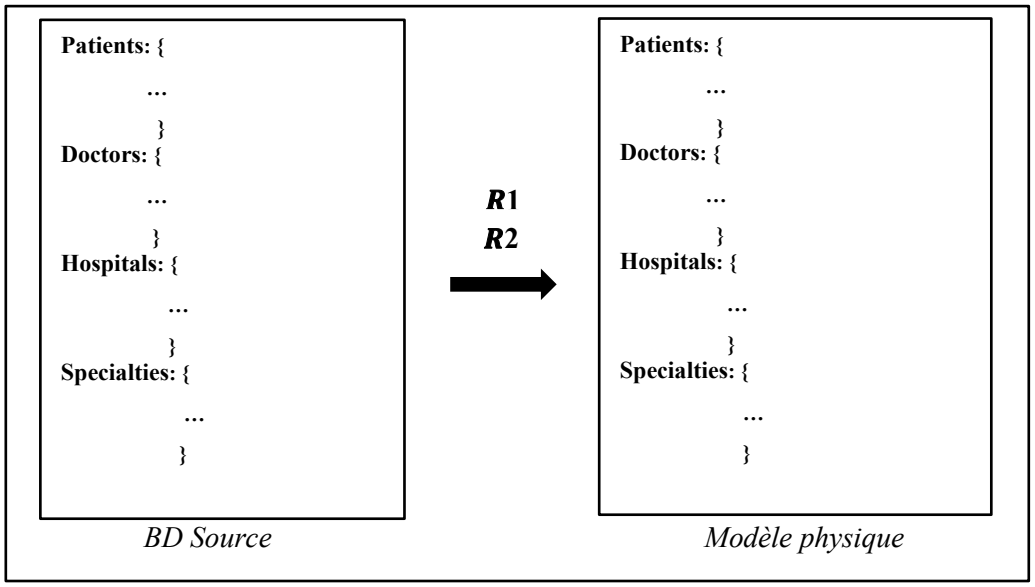



**Figure 4.14 -** Exemple de transformation de collections

**R3 :** $\forall\, j \in [1..m]$, un document $d_j \in c_i.D$ est transformé en appliquant respectivement sur chacun de ses champs atomiques, structurés ou multivalués les règles R4, R5 ou R6.

**R4 :** $\forall$ k $\in$ [1..p], un champ atomique $af_k \in d_j.AF$ est transformé en un champ atomique $af'_k \in c'_i.AF'$ où :

- $af'_k$. N = $af_k$.N,
- $af'_k$. T est généré en fonction des caractéristiques de $af_k$.V. Par exemple, si $af_k$.V est de la forme : "xxx", alors $af'_k$. T = "String" ; si $af_k$.V est de la forme : xxx, alors $af'_k$. T = "Number",
- $AF' = \{af'_k\} \cup AF'$.

Dans l'exemple de la figure 4.15, notre processus applique la règle R4 sur les champs atomiques *_id*, *Designation* et *MedicalStaff* de la collection *Specialties*.

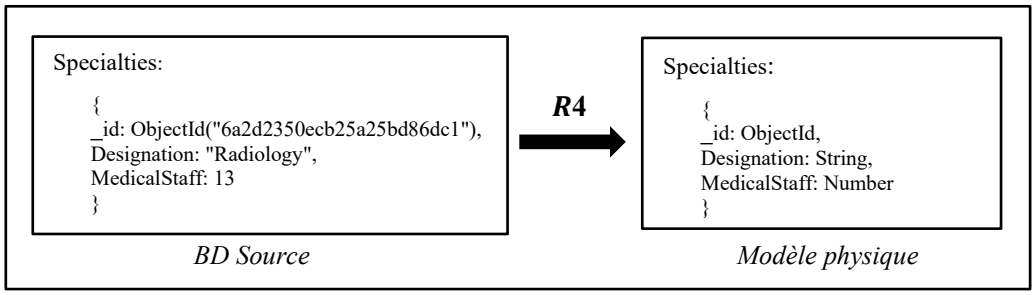



**Figure 4.15 -** Exemple de transformation de champs atomiques

**R5 :** ∀ l ∈ [1..q], un champ structuré $sf_l \in d_j.SF$ est transformé en un champ structuré $sf'_l \in c'_i.SF'$ où :

- $sf'_l$. N = $sf_l$.N,
- $sf'_l.F''$ est généré en transformant $sf_l.F'$ comme suit :
  - Générer $sf'_l.AF''$ en appliquant R4 sur chaque champ atomique appartenant à $sf_l.AF'$,
  - Générer $sf'_l.SF''$ en appliquant R5 sur chaque champ structuré appartenant à $sf_l.SF'$,
  - Générer $sf'_l.MF''$ en appliquant R6 sur chaque champ multivalué appartenant à $sf_l.MF'$
- $SF' = \{sf'_l\} \cup SF'$.

Dans l'exemple de la figure 4.16, notre processus applique la règle R5 sur le champ structuré *Address* de la collection *Hospitals*.

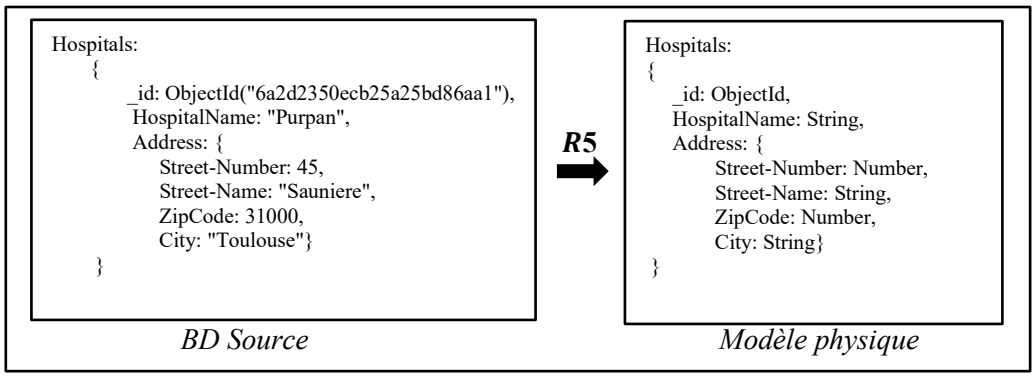



**Figure 4.16 -** Exemple de transformation des champs structurés

**R6 :** ∀ z ∈ [1..r], un champ multivalué $mf_z \in d_j.MF$ est transformé en un champ multivalué $mf'_z \in c'_i.MF'$ où :

- $mf'_z$. N = $mf_z$.N,
- $mf'_z.F''$ est généré en transformant $mf_z.F'$ comme suit :
  - Générer $mf'_z.AF''$ en appliquant R4 sur chaque champ atomique appartenant à $mf_z.AF'$,
  - Générer $mf'_z.SF''$ en appliquant R5 sur chaque champ structuré appartenant à $mf_z.SF'$,
  - Générer $mf'_z.MF''$ en appliquant R6 sur chaque champ multivalué appartenant à $mf_z.MF'$
- $MF' = \{mf'_z\} \cup MF'$.

Dans l'exemple de la figure 4.17, notre processus applique la règle R6 sur les champs multivalués *Prénom et Adresses* de la collection *Employés*.

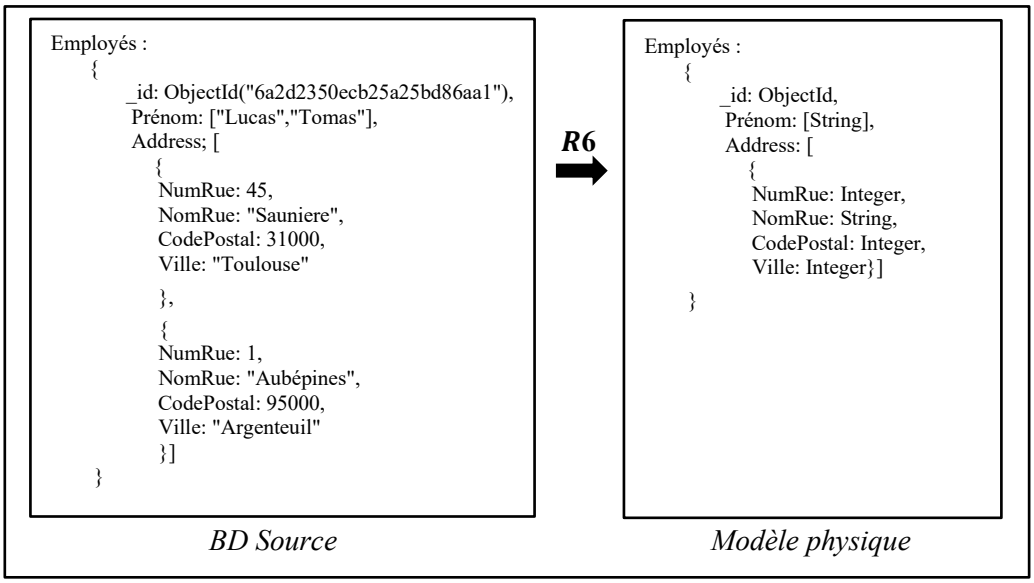



**Figure 4.17 -** Exemple de transformation des champs multivalués

Nous appliquons la même règle R6 pour la transformation des champs *DBRef*, *DBRef* étendu, *DBSub*, *DBComp* et *DBAgg*. Ceux-ci sont en effet des champs structurés.

Par exemple, nous montrons dans la figure 4.18 la transformation du champ DBRef intitulé *Treating-Doctor* qui relie les collections *Patients* et *Doctors*.

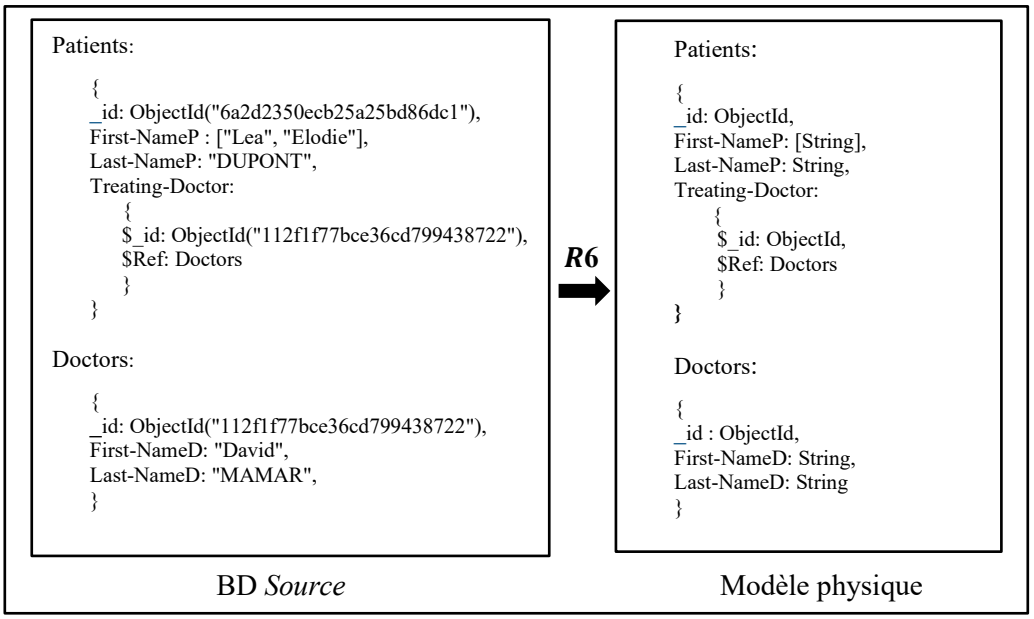



**Figure 4.18 -** Exemple de transformation d'un champ *DBRef*

## 4.3 Sous-processus « à chaud »

Après avoir extrait le MPD de la BD NoSQL (sous-processus à froid), nous présentons dans cette section un sous-processus d'extraction complémentaire dit « à chaud ». En effet, le volume des données de la BD pouvant atteindre plusieurs téraoctets et la génération du MPD nécessite le balayage de la totalité de la BD. Il n'est pas donc envisageable de relancer le sous-processus *ModelExtraxtion* après chaque mise à jour de la BD. Nous présentons donc le sous-processus *ModelUpdate* qui consiste à mettre à jour le MPD au fur et à mesure que l'expression des requêtes de mise à jour sur la BD ; le MPD est ainsi disponible à tout moment.

Comme le montre la figure 4.19, l'entrée du sous-processus *ModelUpdate* est constituée de trois éléments (1) une requête de mise à jour (2) un MPD courant (version i) et (3) des métadonnées courantes (version i). Une requête de mise à jour peut être de type adjonction, suppression ou modification. La sortie de *ModelUpdate* est constituée d'une version modifiée du MPD (version i+1) et d'une version modifiée des métadonnées (version i+1).

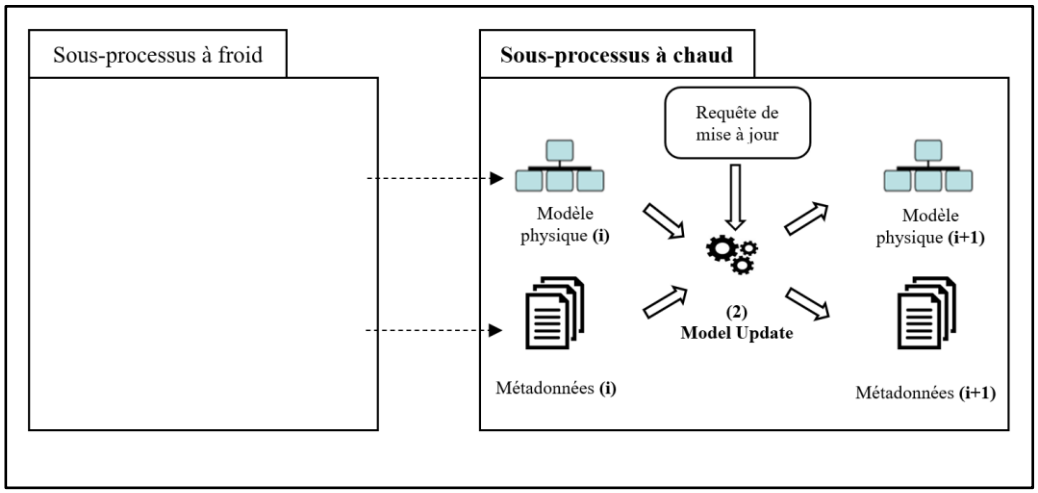



**Figure 4.19 -** Sous-processus d'extraction « à chaud»

Le sous-processus *ModelUpdate* procède comme suit. D'abord, il analyse la requête de mise à jour pour déterminer la collection sur laquelle elle porte ainsi que ses champs. Ensuite, il compare ces éléments avec la version courante du MPD. Si celle-ci ne correspond pas aux éléments de la requête, alors il met à jour le MPD. Sinon, il garde la version précédente. Parallèlement à l'élaboration du modèle, *ModelUpdate* met à jour des métadonnées, le cas échéant.

Dans les sections suivantes, nous définissons la source (MPD + Métadonnées + Requête de mise à jour) et la cible (MPD + Métadonnées), puis nous présentons les règles de transformation.

## 4.3.1 Source

### 4.3.1.1 PSM-MPD

Comme le montre la figure 4.16, l'une des entrées du sous-processus *UpdateModel* est la version (i) du MPD. Celle-ci est le modèle physique courant de la BD au moment de l'exécution de la requête de mise à jour. Notons que la première version du MPD dans le sous processus à chaud correspond au MPD résultant du sous-processus à froid (cf. section 4.2.2.1).

### 4.3.1.2 PSM-MetaData

Les métadonnées courantes de la BD constituent une entrée du sous-processus *UpdateModel.* La première version des métadonnées dans l'extraction à chaud correspond aux métadonnées résultantes de l'extraction à froid (cf. section 4.2.2.2).

### 4.3.1.3 PSM-Query

Une requête de mise à jour est une requête qui apporte une modification au niveau des données de la BD et au niveau du modèle de la BD le cas échéant. Elle peut être de type adjonction, suppression ou modification. Nous définissons dans ce qui suit chaque type.

**Requête d'adjonction**

Une requête d'adjonction $IQ$ permet d'insérer un document dans une collection de la BD. Elle contient une valeur pour chaque champ du document.

Formellement :

> **Définition 14.** Une requête d'adjonction $IQ$ est définie par un couple (c, $d$) où :
>
> - $IQ.c$ est le nom de la collection sur laquelle porte $IQ$,
> - $IQ.d$ représente le document à insérer (cf. définition 3 de la section 4.2.1).

**Exemple :**

La requête suivante permet d'insérer un document dans la collection *Patients*.

*db. Patients. insert ({NSS : "1 90 10 99 350 638 60", Nom : "DUPONT", Prénom : "Luc", Adresse : { NumRue : "6", NomRue : "Kabylie", CodePostal : 75015, Ville : "Paris" }}) ;*

**Requête de suppression**

Une requête de suppression $DQ$ permet d'extraire un ou plusieurs documents d'une collection de la BD. Le(s) document(s) supprimé(s) correspond(ent) à une condition donnée comme paramètre dans la requête.

Formellement :

**Définition 15.** Une requête de suppression $DQ$ est définie par un couple (c, $cond$) où :

- $DQ.c$ est le nom de la collection sur laquelle porte $DQ$,
- $DQ.cond = AF \cup SF \cup MF$ est l'ensemble des champs constituant la condition de suppression de la requête $DQ$, où :
  - $DQ.AF = \{af_1, \dots, af_k\}$ est l'ensemble des champs atomiques de la condition (cf. définition 4 de la section 4.2.1),
  - $DQ.SF = \{sf_1, \dots, sf_l\}$ est l'ensemble des champs structurés de la condition (cf. définition 5 de la section 4.2.1),
  - $DQ.MF = \{mf_1, \dots, mf_z\}$ est l'ensemble des champs multivalués de la condition (cf. définition 6 de la section 4.2.1).

**Requête de modification**

Une requête de modification $UQ$ permet de modifier les données d'un ou plusieurs documents d'une collection de la BD. Elle peut s'exprimer de différentes manières comme le montre le tableau 4.1.

| Cas | Description |
|---|---|
| **1** | Modification de la valeur d'un champ dans le document |
| **2** | Ajout d'un nouveau champ dans le document |
| **3** | Suppression d'un champ dans le document |
| **4** | Renommage d'un champ dans le document |

**Tableau 4.1 -** Cas d'une requête de modification

Par exemple, la requête suivante permet de modifier un document de la collection *Patients* en ajoutant un nouveau champ *DateNais* :

*db. Patients. update ({_id : ObjectId ("545a4asd4ad6a")}, {$set {DateNais : 10/12/1960}) ;*

Notons que, contrairement aux cas 2, 3 et 4, le cas 1 n'apporte aucun changement ni sur le MPD ni sur les métadonnées. En effet, la modification d'une valeur d'un champ n'entraine pas la modification de son type au niveau du MPD. De plus, le compteur au niveau des métadonnées, qui indique le nombre d'apparitions du champ en question dans la BD, n'est pas affecté.

Nous présentons dans ce qui suit une définition formelle d'une requête de modification $UQ$ :

**Définition 16.** Une requête de modification $UQ$ est définie par quadruplet (c, $cond, AddF, RemoveF$) où :

- $UQ.c$ est le nom de la collection sur laquelle porte $UQ$,
- $UQ.cond = AF \cup SF \cup MF$ est l'ensemble des champs constituant la condition de modification de la requête $UQ$, où :
  - $UQ.AF = \{af_1, \dots, af_k\}$ est l'ensemble des champs atomiques de la condition (cf. définition 4 de la section 4.2.1),
  - $UQ.SF = \{sf_1, \dots, sf_l\}$ est l'ensemble des champs structurés de la condition (cf. définition 5 de la section 4.2.1),
  - $UQ.MF = \{mf_1, \dots, mf_z\}$ est l'ensemble des champs multivalués de la condition (cf. définition 6 de la section 4.2.1).
- $UQ.AddF = AddAF \cup AddSF \cup AddMF$ est l'ensemble de champs à ajouter dans le document , où :
  - $UQ.AddAF = \{af_1, \dots, af_k\}$ est l'ensemble des champs atomiques à ajouter dans le document (cf. définition 4 de la section 4.2.1),
  - $UQ.AddSF = \{sf_1, \dots, sf_l\}$ est l'ensemble des champs structurés à ajouter dans le document (cf. définition 5 de la section 4.2.1),
  - $UQ.AddMF = \{ mf_1, \dots, mf_z \}$ est l'ensemble des champs multivalués à ajouter dans le document (cf. définition 6 de la section 4.2.1).
- $UQ.RemoveF = RemoveAF \cup RemoveSF \cup RemoveMF$ est l'ensemble de champs à retirer du document.
  - $UQ.RemoveAF = \{ af_1, \dots, af_k \}$ est l'ensemble des champs atomiques à retirer du document (cf. définition 4 de la section 4.2.1),
  - $UQ.RemoveSF = \{ sf_1, \dots, sf_l \}$ est l'ensemble des champs structurés à retirer du document (cf. définition 5 de la section 4.2.1),
  - $UQ.RemoveMF = \{ mf_1, \dots, mf_z \}$ est l'ensemble des champs multivalués à retirer du document (cf. définition 6 de la section 4.2.1).

Notons que $UQ.RemoveF = \emptyset$ pour le cas 2 et $UQ.AddF = \emptyset$ pour le cas 3. Pour le cas 4, l'ancien nom du champ appartient à $UQ.RemoveF$ et son nouveau nom appartient $UQ.AddF$.

Nous présentons les différents concepts des requêtes de mise à jour au travers le métamodèle de la figure 4.20.

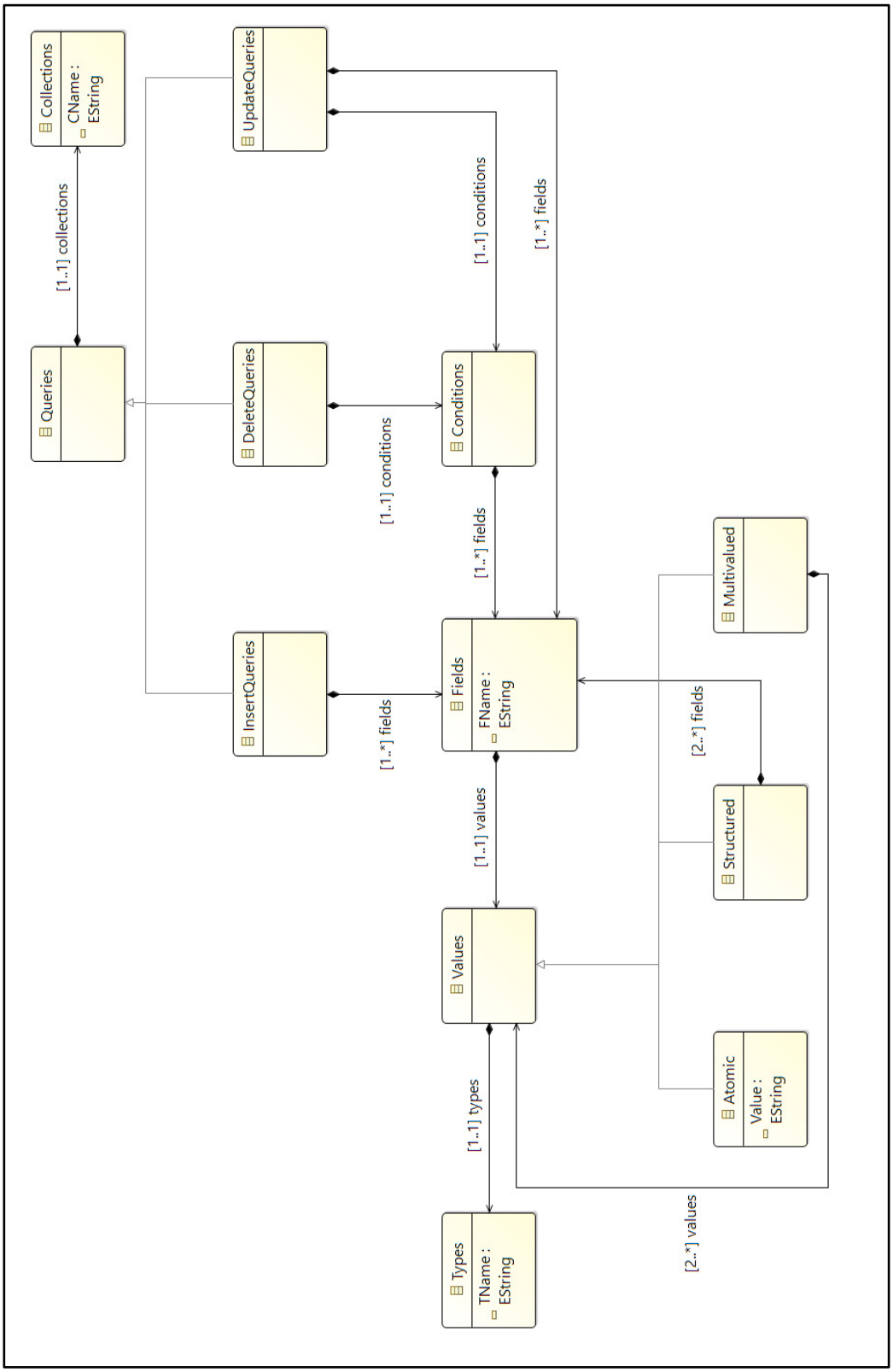



**Figure 4.20 -** Métamodèle du PSM-Query

## 4.3.2 Cible

La sortie du sous-processus *UpdateModel* correspond à une version évoluée du MBD (version i+1) et une version évoluée des métadonnées (version i+1). Ces deux modèles résultants sont calculés après le traitement de la requête de mise à jour.

## 4.3.3 Règles de transformation

Pour chaque type de requête de mise à jour, le sous-processus *UpdateModel* analyse la requête pour déterminer la collection sur laquelle elle porte ainsi que ses attributs. Puis, il met à jour le MPD et les métadonnées, le cas échéant.

Nous présentons dans cette section le passage entre la source (requête de mise à jour, MPD (version i) et les métadonnées (version i)) et la cible (MPD (version i+1) et les métadonnées (version i+1)) sous forme d'une séquence de règles de transformation pour chaque type de requête.

### 4.3.3.1 Cas d'une requête d'adjonction

Pour chaque requête d'insertion $IQ$, le sous-processus *UpdateModel* applique deux règles de transformation : R6 sur le MPD et R7 sur les métadonnées.

La R6 consiste à trouver la collection correspondante dans le MPD. Ensuite elle compare les champs figurant dans la requête d'insertion $IQ$ avec ceux de la version courante du MPD. Si ceux-ci ne correspondent pas aux champs de $IQ$, alors le sous-processus *UpdateModel* met à jour le MPD. Sinon, il garde la version précédente.

La R7 consiste à trouver la collection correspondante dans les métadonnées. Pour chaque champ de la requête d'insertion $IQ$, la règle R7 met à jour la valeur de son compteur dans les métadonnées.

Formellement, R6 et R7 sont présentées comme suit ;

**R6 :**

- S'il existe une collection $c'_i \in MP.\ C'$ où $c'_i.N = IQ.\mathrm{c}$, avec i ∈ [1..n], alors :
    - Pour chaque champ atomique $af_\mathrm{k} \in IQ.AF$, avec k ∈ [1..p],
        - Transformer le couple *(N, V)* en un couple *(N, T)*.
        - S'il n'existe pas un champ atomique $af' \in c'_i.AF'$ où $af'.N = af_\mathrm{k}.N$ et $af'.T = af_\mathrm{k}.T$, alors créer le champ $af'$ et l'ajouter à $c'_i.AF'$ comme suit :
            - $af'.N = af_\mathrm{k}.N$,
            - $af'.T = af_\mathrm{k}.T$,

- $c'_i.AF' = \{af'\} \cup c'_i.AF'$

- Pour chaque champ structuré $sf_l \in IQ.SF$, avec $l \in [1..q]$,
  - Transformer le couple *(N, F')* en un couple *(N, F'')*.
  - S'il n'existe pas un champ structuré $sf' \in c'_i.SF'$ où $sf'.N = sf_l.N$ et $sf'.F'' = sf_l.F''$, alors créer le champ $sf'$ et l'ajouter à $c'_i.SF'$ comme suit :
    - $sf'.N = sf_l.N$,
    - $sf'.F'' = sf_l.F''$,
    - $c'_i.SF' = \{sf'\} \cup c'_i.SF'$
- Pour chaque champ multivalué $mf_l \in IQ.MF$, avec $z \in [1..r]$,
  - Transformer le couple *(N, F')* en un couple *(N, F'')*.
  - S'il n'existe pas un champ multivalué $mf' \in c'_i.MF'$ où $mf'.N = mf_z.N$ et $mf'.F'' = mf_z.F''$, alors créer le champ $mf'$ et l'ajouter à $c'_i.MF'$ comme suit :
    - $mf'.N = mf_z.N$,
    - $mf'.F'' = mf_z.F''$,
    - $c'_i.MF' = \{mf'\} \cup c'_i.MF'$

- Sinon, créer $c'_i$ et l'ajouter à *MP.* $C'$comme suit :
  - *MP.* $C' = \{c'_i\} \cup$ *MP.* $C'$avec $c'_i.N = IQ.\text{c}$.
  - Pour chaque champ atomique $af_\text{k} \in IQ.AF$, avec k ∈ [1..p],
    - Transformer le couple *(N, V)* en un couple *(N, T)*.
    - Créer le champ $af'$ et l'ajouter à $c'_i.AF'$ comme suit :
      - $af'.N = af_\text{k}.N$,
      - $af'.T = af_\text{k}.T$,
      - $c'_i.AF' = \{af'\} \cup c'_i.AF'$
  - Pour chaque champ structuré $sf_l \in IQ.SF$, avec $l \in [1..q]$,
    - Transformer le couple *(N, F')* en un couple *(N, F'')*.
    - Créer le champ structuré $sf'$ et l'ajouter à $c'_i.SF'$ comme suit :
      - $sf'.N = sf_l.N$,
      - $sf'.F'' = sf_l.F''$,
      - $c'_i.SF' = \{sf'\} \cup c'_i.SF'$
  - Pour chaque champ multivalué $mf_z \in IQ.MF$, avec $z \in [1..r]$,
    - Transformer le couple *(N, F')* en un couple *(N, F'')*.
    - Créer le champ multivalué $mf'$ et l'ajouter à $c'_i.MF'$ comme suit :
      - $mf'.N = mf_z.N$,
      - $m.F'' = mf_z.F''$,
      - $c'_i.MF' = \{mf'\} \cup c'_i.MF'$

**R7 :**

- S'il existe une collection $c^{MD}{}_i \in$ *MD.* $C^{MD}$ où $c^{MD}{}_i.N = IQ.\text{c}$, avec i ∈ [1..n], alors :

- Pour chaque champ atomique $af_{\mathrm{k}} \in IQ.AF$, avec k ∈ [1..p],
  - S'il n'existe pas un champ atomique $af' \in c^{MD}{}_i.AF'$ où $af'.N = af_{\mathrm{k}}.N$, alors créer le champ $af'$ et l'ajouter à $c^{MD}{}_i.AF'$ comme suit :
    - $af'.N = af_{\mathrm{j}}.N$,
    - $af'.Cpt$ =1,
    - $c^{MD}{}_i.AF' = \{af'\} \cup c^{MD}{}_i.AF'$
  - Sinon, mettre à jour le champ $af'$ dans $c^{MD}{}_i.AF'$ comme suit :
    - $af'.Cpt = af'.Cpt$ +1
- Pour chaque champ structuré $sf_l \in IQ.SF$, avec $l$ ∈ [1..q],
  - S'il n'existe pas un champ structuré $sf' \in c^{MD}{}_i.SF'$ où $sf'.N = sf_l.N$, alors créer le champ $sf'$ et l'ajouter à $c^{MD}{}_i.SF'$ comme suit :
    - $sf'.N = sf_l.N$,
    - $sf'.Cpt$ = 1,
    - $c^{MD}{}_i.SF' = \{sf'\} \cup c^{MD}{}_i.SF'$
  - Sinon, mettre à jour le champ $sf'$ dans $c^{MD}{}_i.SF'$ comme suit :
    - $sf'.Cpt = sf'.Cpt$ +1
- Pour chaque champ multivalué $mf_z \in IQ.MF$, avec $z$ ∈ [1..r],
  - S'il n'existe pas un champ multivalué $mf' \in c^{MD}{}_i.MF'$ où $mf'.N = mf_l.N$, alors créer le champ $mf'$ et l'ajouter à $c^{MD}{}_i.MF'$ comme suit :
    - $mf'.N = mf_z.N$,
    - $mf'.Cpt$ = 1,
    - $c^{MD}{}_i.MF' = \{mf'\} \cup c^{MD}{}_i.MF'$
  - Sinon, mettre à jour le champ $mf'$ dans $c^{MD}{}_i.MF'$ comme suit :
    - $mf'.Cpt = mf'.Cpt$ +1

- Sinon, créer $c^{MD}{}_i$ et l'ajouter à *MD.* $C^{MD}$ comme suit :
  - *MD.* $C^{MD} = \{c^{MD}{}_i\} \cup$ *MD.* $C^{MD}$ avec $c^{MD}{}_i.N = IQ.$c.
  - Pour chaque champ atomique $af_{\mathrm{k}} \in IQ.AF$, avec k ∈ [1..p],
    - Créer le champ $af'$ et l'ajouter à $c^{MD}{}_i.AF'$ comme suit :
      - $af'.N = af_{\mathrm{k}}.N$,
      - $af'.Cpt$ =1,
      - $c^{MD}{}_i.AF' = \{af'\} \cup c^{MD}{}_i.AF'$
  - Pour chaque champ structuré $sf_l \in IQ.SF$, avec $l$ ∈ [1..q],
    - Créer le champ $sf'$ et l'ajouter à $c^{MD}{}_i.SF'$ comme suit :
      - $sf'.N = sf_l.N$,
      - $sf'.Cpt$ =1,
      - $c^{MD}{}_i.SF' = \{sf'\} \cup c^{MD}{}_i.SF'$
  - Pour chaque champ multivalué $mf_z \in IQ.MF$, avec $z$ ∈ [1..r],
    - Créer le champ $mf'$ et l'ajouter à $c^{MD}{}_i.MF'$ comme suit :
      - $mf'.N = mf_z.N$,

- $mf'.Cpt$ =1,
- ${c^{MD}}_i.MF' = \{mf'\} \cup {c^{MD}}_i.MF'$

Dans l'exemple illustré dans de la figure 4.21, *UpdateModel* applique les règles R6 et R7 pour le traitement d'une requête d'adjonction portant sur la collection *Patients*. Nous supposons que la BD est initialement vide. Par conséquent, son MPD et ses métadonnées résultant de l'exécution du sous-processus *ExtractionModel* (extraction à froid) sont vides.

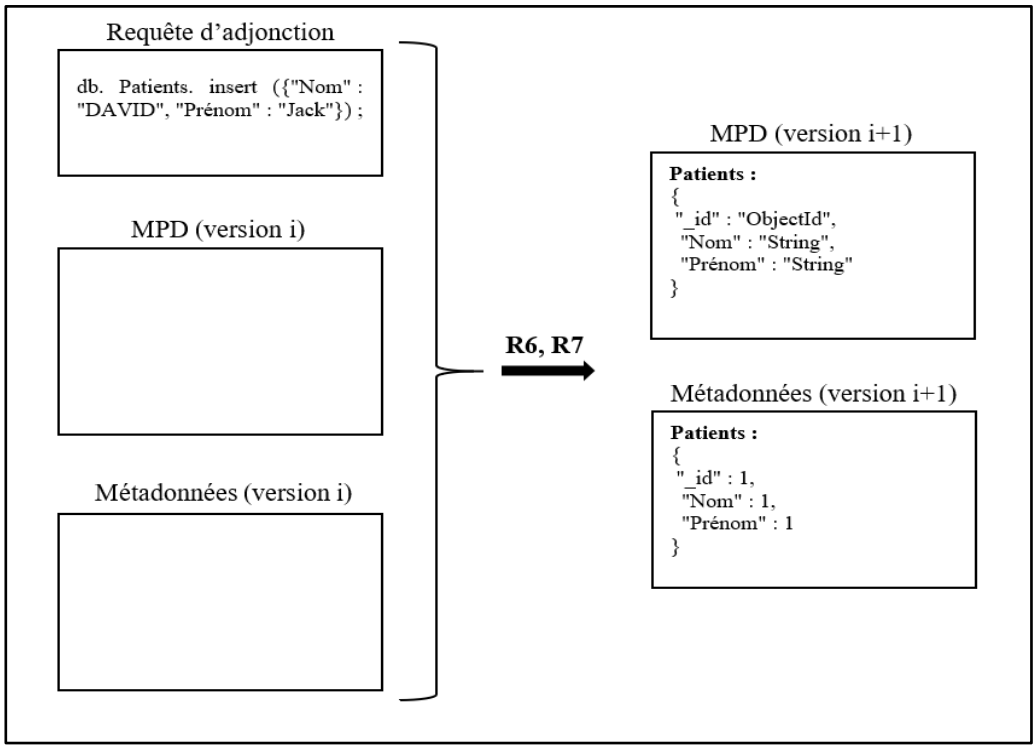



**Figure 4.21 -** Exemple de traitement d'une requête d'adjonction

#### 4.3.3.2 Cas d'une requête de suppression

Pour chaque requête de suppression $DQ$, notre processus applique deux règles de transformation : R8 sur le MPD et R9 sur les métadonnées. R8 et R9 sont appliquées respectivement selon le même principe que R6 et R7.

Dans l'exemple de la figure 4.22, le sous-processus *UpdateModel* applique les règles R8 et R9 pour le traitement d'une requête de suppression. Le document à supprimer par celle-ci figure dans la BD de la même figure.

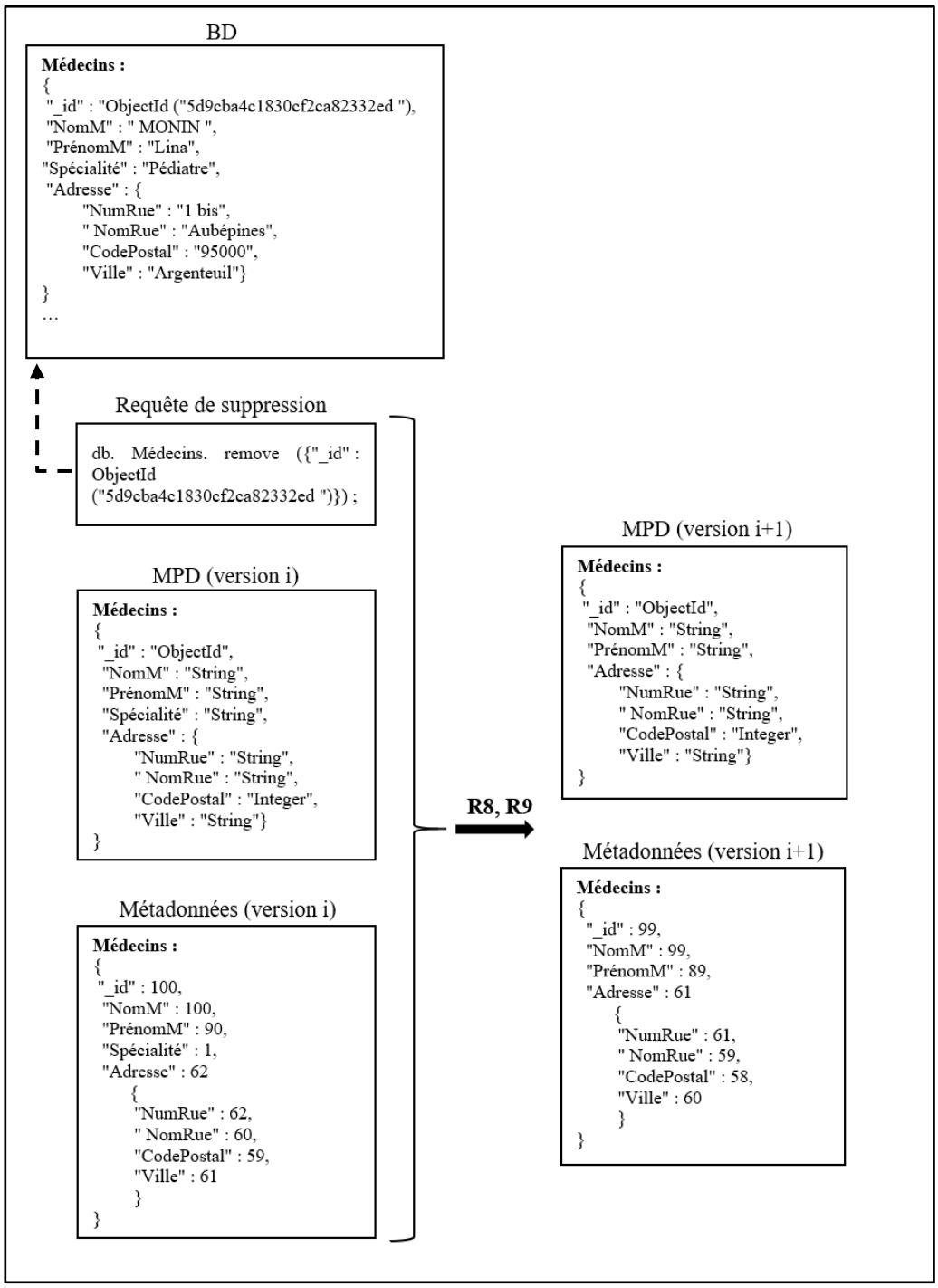



**Figure 4.22 -** Exemple de traitement d'une requête de suppression

### 4.3.3.3 Cas d'une requête de modification

Pour chaque cas d'une requête de modification $UQ$, notre processus applique deux règles de transformation comme suit :

- Cas 2 : R10 sur le MPD et R11 sur les métadonnées,
- Cas 3 : R12 sur le MPD et R13 sur les métadonnées,
- Cas 4 : R14 sur le MPD et R15 sur les métadonnées.

Les règles R10, R12 et R14 suivent le même principe que R6. Quant aux règles R11, R13 et R15, elles suivent le même principe que R7.

Dans l'exemple illustré dans de la figure 4.23, le sous-processus *UpdateModel* applique les règles R10 et R11 pour le traitement du cas 2 d'une requête de modification (ajout d'un nouveau champ dans un document).

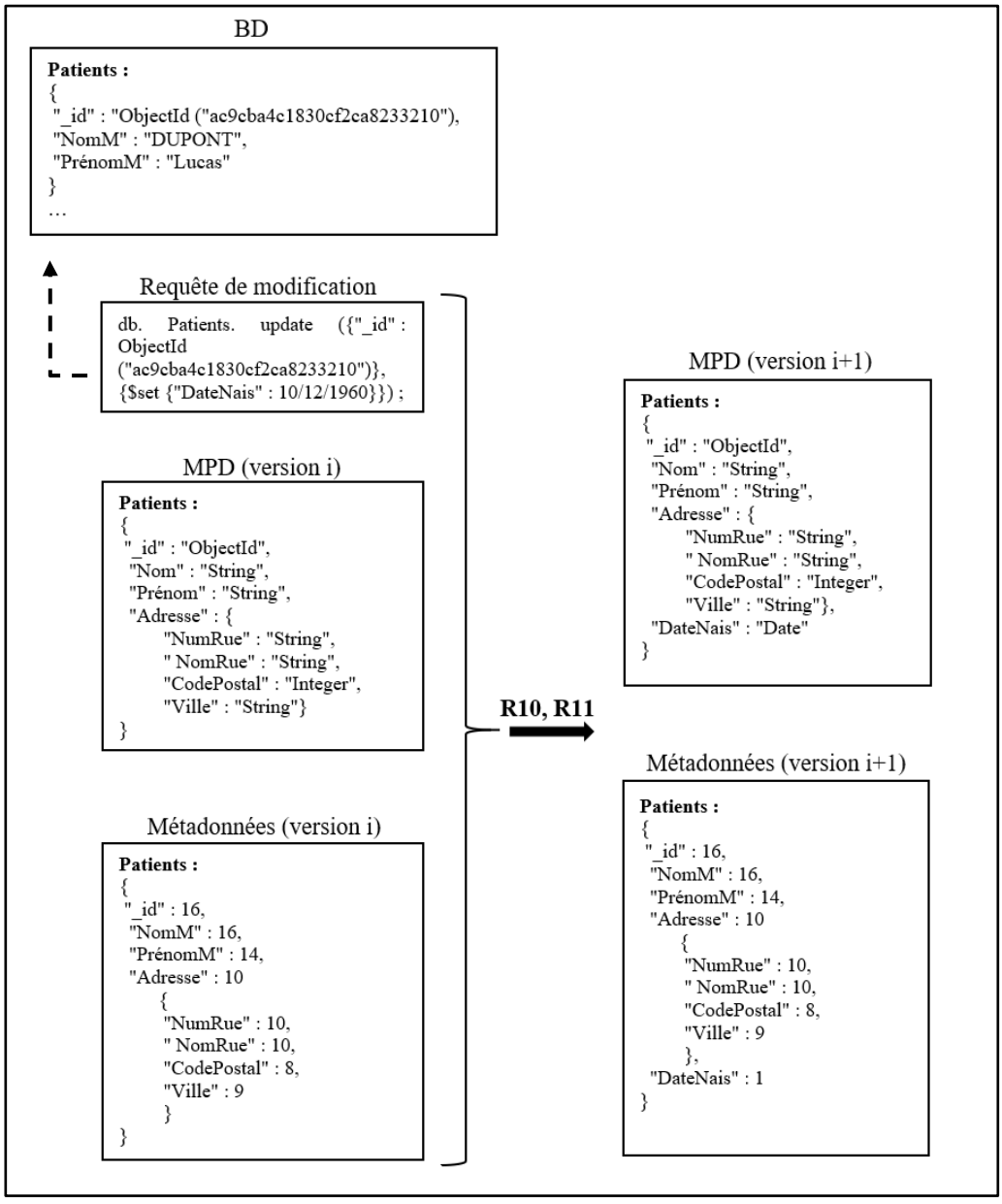



**Figure 4.23 -** Exemple de traitement d'une requête de modification (ajout d'un champ)

Dans l'exemple illustré dans de la figure 4.24, le sous-processus *UpdateModel* applique les règles R14 et R15 pour le traitement du cas 4 d'une requête de modification (renommage d'un champ dans un document).

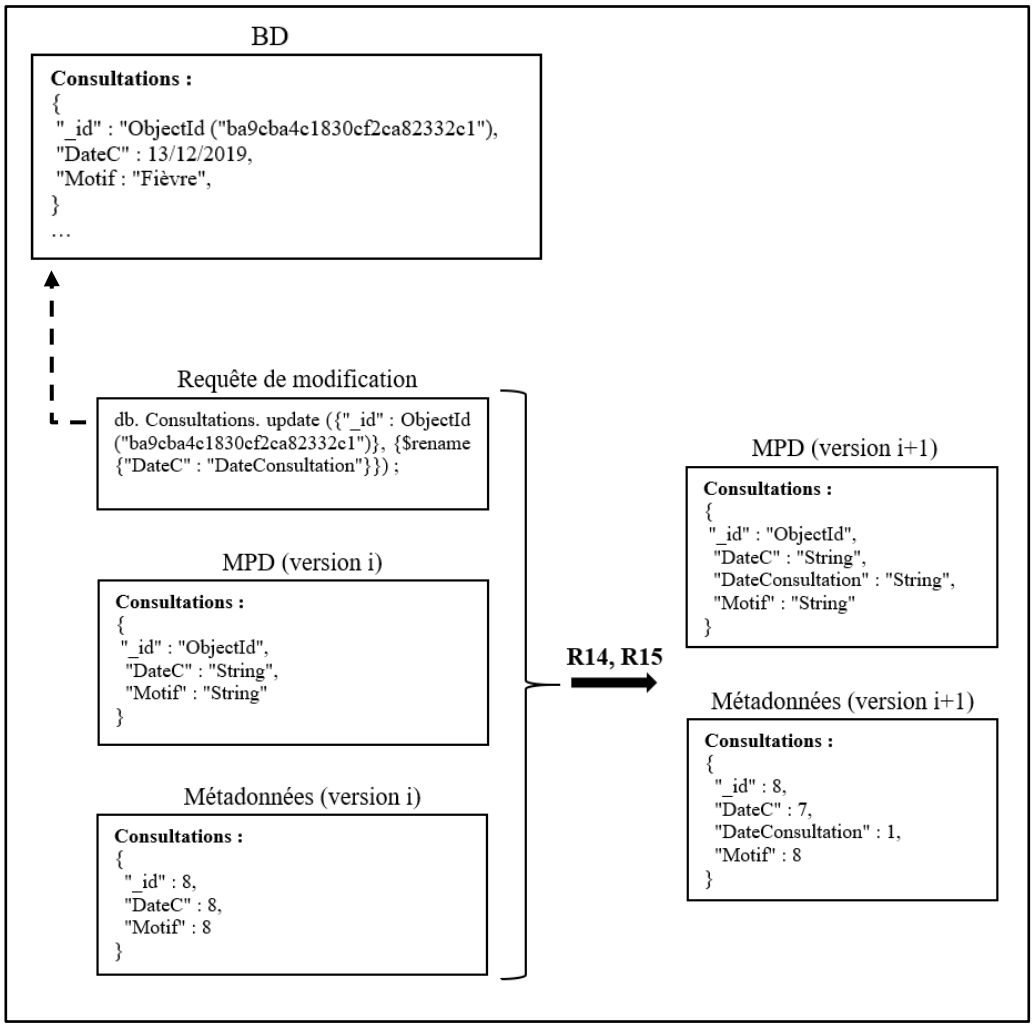



**Figure 4.24 -** Exemple de traitement d'une requête de modification (renommage d'un champ)

## 4.4 Bilan du processus d'extraction du modèle physique

### 4.4.1 Synthèse

La plupart des SGBD NoSQL sont caractérisés par la propriété « schema less » (sans schéma) qui signifie absence du modèle de données lors de la création d'une BD. Cette propriété apporte une souplesse indéniable en permettant l'évolution du modèle de données au fur et à mesure de l'exploitation de la BD. Cependant, l'absence du modèle ne permet pas de connaître comment les données sont stockées et reliées dans la BD. Cette méconnaissance du modèle de données rend difficile l'expression des requêtes sur cette BD. Pour surmonter cette difficulté, nous sommes convaincus qu'il convient de fournir aux utilisateurs (développeurs ou décideurs) les modèles nécessaires pour manipuler des BD NoSQL. Il s'agit de deux modèles complémentaires décrivant la BD : (1) le modèle physique qui décrit l'organisation interne des données et permet d'exprimer des requêtes et (2) le modèle conceptuel qui fournit un haut niveau d'abstraction et met en exergue la sémantique de la BD. Dans ce chapitre, nous avons étudié la génération du modèle physique. Pour cela, nous avons proposé le processus dirigé par les modèles *ToPhysicalModel*. Quant au processus de génération du modèle conceptuel, il sera étudié dans le chapitre 4.

Le processus *ToPhysicalModel* vise à formaliser et à automatiser l'élaboration du modèle physique de données à partir d'une BD gérée par un système NoSQL de type schema-less. A la suite à la justification donnée dans le chapitre 1 (cf. Section 2.2.4), nous avons limité notre étude aux BD gérées par des systèmes NoSQL orientés-documents. Le processus *ToPhysicalModel* comporte deux sous-processus successifs. Il s'agit du (1) sous-processus « à froid » qui consiste à extraire le modèle physique d'une BD existante et du (2) sous-processus à chaud qui consiste à élaborer d'une manière incrémentale le modèle physique à chaque fois que l'utilisateur exécute une requête de mise à jour sur la BD. Les modèles utilisés dans les deux sous-processus sont conformes à des métamodèles. Pour assurer le passage automatique entre les modèles sources et cibles, nous avons défini des règles de transformations de type Model-To-Model (M2M) formalisées avec le standard QVT.

Les travaux de ce chapitre ont été présentés dans les publications suivantes :

[Ait Brahim et al., 2018], [Ait Brahim et al., 2019a], [Ait Brahim et al., 2019b], [Ait Brahim et al., 2019c] et [Ait Brahim et al., 2020].

### 4.4.2 Positionnement

Dans cette section, nous effectuons le positionnement de nos travaux au regard des travaux portant sur l'extraction d'un modèle physique de données (MPD) à partir d'une BD NoSQL « schema less ».

L'extraction du MPD d'une BD NoSQL « schema less » a fait l'objet de plusieurs solutions industrielles et de recherche.

Pour les solutions industrielles, notamment MongoDB et Drill d'Apache, elles ne permettent d'afficher le MPD d'une BD MongoDB qu'à minima. En effet, ces deux outils n'affichent pas les champs composant un champ structuré. De plus, ils ne considèrent pas les liens entre collections. Quant aux travaux de recherche, ils ne répondent que partiellement à notre problématique. En effet, bien que les travaux de [Klettke et al., 2015], [Izquierdo et al., 2016], [Gallinucci et al., 2018], [Maity et al., 2018], [Baazizi et al., 2017], [Baazizi et al., 2019] et [Sevilla et al., 2015] proposent des mécanismes d'extraction de la structure physique des collections composant la BD, ils n'étudient pas les liens entre ces collections. Ainsi, ils ignorent un aspect important pour manipuler une BD en général et particulièrement une BD NoSQL.

Dans ce chapitre, nous avons proposé le processus *ToPhysicalModel* qui extrait et met à jour automatiquement le MPD d'une BD NoSQL de type documents. Ce processus, basé sur l'architecture MDA (Model Driven Architecture), vise à extraire un modèle physique de données (MPD) décrivant les collections qui composent la BD d'entrée et permettant aux utilisateurs d'exprimer des requêtes de type SQL. Le MPD généré est mis à jour ensuite au fur et à mesure de l'exploitation de la base (expression de requêtes de mise à jour par l'utilisateur). Il est généré en appliquant une séquence de transformations formalisées avec le standard QVT. L'apport majeur de notre solution se situe dans :

- la considération de tout type de champs (atomique, multivalué et structuré) quels que soient leurs niveaux dans les collections de la BD,

- la prise en compte des liens entre collections,

- l'extraction dynamique du modèle au fur et à mesure de l'exploitation de la base. En effet, le volume des données d'une BD NoSQL pouvant atteindre plusieurs téraoctets et la génération du modèle physique nécessite le balayage de la totalité de la BD. Il n'est pas donc envisageable de relancer le processus d'extraction du modèle après chaque mise à jour de la BD. Pour cela, nous avons préconisé un sous-processus qui consiste à mettre à jour le modèle physique au fur et à mesure que l'expression des requêtes de mise à jour sur la BD ; le modèle physique est ainsi disponible à tout moment.

# Chapitre 5 : Transformation du modèle physique de données en un modèle conceptuel de données

Les systèmes NoSQL ont prouvé leur efficacité dans la gestion de données de type Big Data. La plupart de ces systèmes sont connus par la caractéristique schema-less (sans schéma) qui signifie absence du modèle de données lors de la création de la BD. Cette caractéristique apporte une souplesse indéniable qui facilite l'évolution du modèle de données et permet aux utilisateurs d'ajouter de nouvelles informations sans avoir besoin d'un administrateur de BD. Cependant, cette absence de modèle rend la manipulation de la BD plus difficile. En effet, dans un contexte Big Data, les besoins des utilisateurs évoluent d'une manière exponentielle. Cette évolution nécessite donc une connaissance des modèles nécessaires pour la manipulation la BD. Ces modèles sont : (1) le modèle physique qui décrit la manière dont les données sont organisées dans la BD et (2) le modèle conceptuel qui exprime la sémantique de ces données. Le chapitre 3 a été consacré à l'extraction du modèle physique. Dans le présent chapitre, nous complétons ces travaux en transformant le modèle physique déjà extrait (cf. Chapitre 4) en un modèle conceptuel. Pour cela, nous proposons le processus *ToConceptualModel.* Il s'agit d'une approche dirigée par les modèles qui établit une correspondance entre le modèle physique et le modèle conceptuel. Celui-ci est exprimé en utilisant le formalisme UML.

Le chapitre 5 est structuré comme suit. La section 5.1 introduit notre processus de génération du modèle conceptuel de données. Les sections 5.2, 5.3 et 5.4 détaillent respectivement l'entrée, la sortie et les règles de transformation du processus. Enfin, la section 5.5 conclut ce chapitre en rappelant nos principales contributions en termes d'extraction du modèle conceptuel.

# 5.1 Aperçu de notre processus

Dans les systèmes d'information en général et dans un contexte Big Data en particulier, l'importance et la nécessité du modèle conceptuel de données (MCD) sont largement reconnues. D'abord, le MCD fait abstraction des aspects techniques et fournit une connaissance sémantique proche de la compréhension humaine ; ceci garantit une gestion efficace des données [Abelló, 2015], [Herrero et al., 2016] et [Daniel et al., 2016]. De plus, le MCD est un élément d'échange entre les utilisateurs (décideurs ou développeurs) et les concepteurs de la BD. En outre, le modèle conceptuel est utilisé comme un support lors de la maintenance et de l'évolution du système qui peuvent affecter les besoins de l'entreprise et/ou la plateforme de déploiement.

L'objectif de ce chapitre est d'automatiser la génération du MCD d'une BD gérée par un système NoSQL orientée-documents. Pour ce faire, nous proposons le processus *ToConceptualModel.* Comme le montre le cadre rouge de la figure 5.1, ce processus transforme le modèle physique de données (MPD) déjà extrait par le processus *ToPhysicalModel* en un modèle conceptuel. Celui-ci est exprimé avec le formalisme UML.

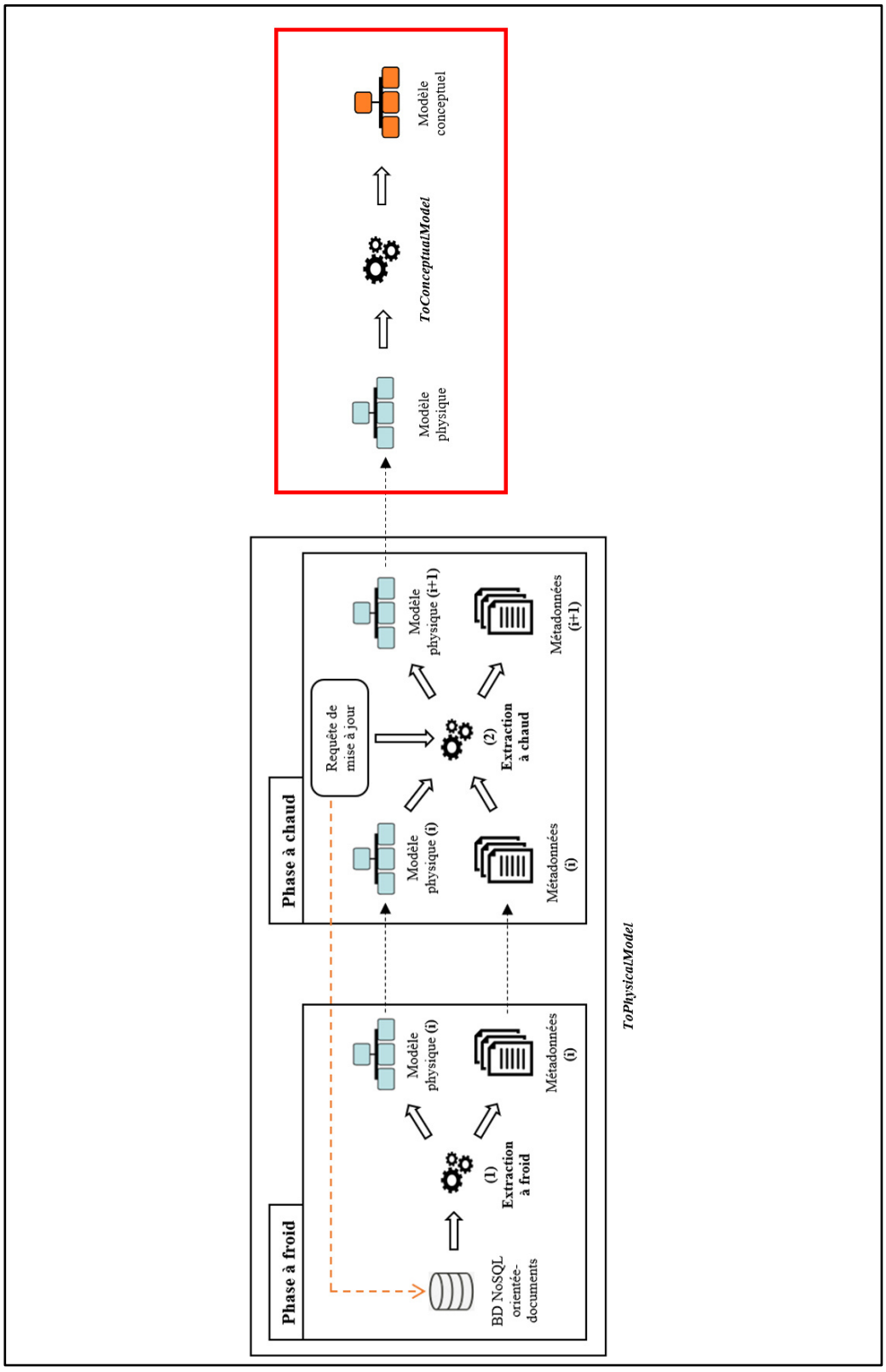



**Figure 5.1 -** Processus d'extraction du MCD d'une BD NoSQL orientée-documents

Le processus *ToConceptualModel* est défini comme suit :

- Source : le MPD de la BD NoSQL déjà extrait par le processus *ToPhysicalModel.*

- Cible : le MCD décrivant la sémantique des données de la BD NoSQL,

- Nature du processus : des transformations Model-To-Model ; ces transformations sont assurées par un ensemble de règles formalisées en QVT.

- Description synthétique du processus : ce sous-processus permet de générer un MCD à partir du MPD NoSQL décrivant une BD de type orienté-documents. Son principe consiste à trouver les classes que contient la BD ainsi que les différents liens entres elles.

A l'instar du processus *ToPhysicalModel*, le processus *ToConceptualModel* suit une architecture MDA. Comme nous l'avons indiqué précédemment (cf. Chapitre 2 – Section 2.3.2), une démarche MDA préconise l'utilisation de trois modèles CIM, PIM et PSM ainsi que deux types de transformations : Model-To-Model (M2M) ou Model-To-Text (M2T). Cependant, le nombre et le type des modèles utilisés ainsi que les transformations mises en œuvre sont propres à chaque démarche. Etant donné que dans le processus *ToConceptualModel* nous manipulons les deux éléments : (1) le MPD NoSQL et (2) le MCD, nous retenons dans ce chapitre les niveaux PSM (Platform Specific Model) et PIM (Platform Independent Model). Comme le montre la figure 5.2, nous distinguons donc les deux modèles suivants :

- PSM-MPD : c'est le modèle physique de données (MPD) de la BD NoSQL.

- PIM-MCD : c'est le modèle conceptuel de données (MCD) qui décrit la sémantique des données de la BD NoSQL.

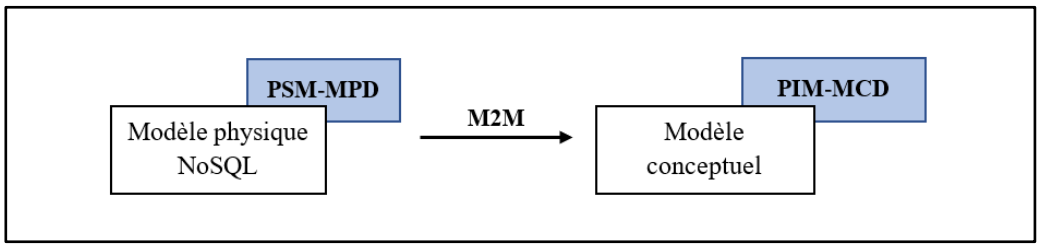



**Figure 5.2 -** Niveau de modélisation du processus *ToConceptualModel*

Le passage du PSM-MPD vers le PIM-MCD se fait en utilisant des transformations de type M2M (Model-To-Model). Nous allons formaliser par la suite ces transformations en utilisant le standard QVT (Query View Transformation) défini par l'OMG pour la transformation de modèles (cf. Section 5.4).

Dans ce qui suit, nous présentons la définition de l'entrée et de la sortie ainsi que les règles de transformation du processus *ToConceptualModel.*

## 5.2 La source : PSM-MPD

L'entrée du processus *ToConceptualModel* correspond à la sortie du processus *ToPhysicalModel* présenté dans le chapitre 4. Autrement dit, l'entrée du processus de *ToConceptualModel* est le PSM-MPD d'une BD orientée documents. Rappelons que le PSM-MPD est le modèle physique de données de la BD NoSQL. Il fait apparaître l'organisation interne des données pour permettre aux utilisateurs d'exprimer des requêtes sur cette BD. Nous avons choisi de stocker le PSM-MPD sous la forme de collections. Chacune d'elles décrit le modèle d'une collection en entrée. Une collection est composée d'un ensemble de champs ; Chacun d'eux peut être soit atomique soit structuré. Un champ atomique est présenté sous la forme d'un triplet (Nom, Type, M) ; Type correspond à un type de données prédéfini comme Integer, Boolean et String, Date, etc. ; M est un booléen qui indique si le champ est multivalué ou pas. Un champ structuré est composé d'un ensemble de champs qui, à leurs tours, peuvent être soit atomiques soit structurés.

## 5.3 La Cible : PIM-MCD

La sortie du processus *ToConceptualModel* correspond au modèle conceptuel de données (PIM-MCD) qui décrit les données de la base selon une vision sémantique et sans mentionner les techniques de stockage. Pour cela, nous utilisons le modèle proposé par la norme UML version 2.5.1 [UML, 2017] qui permet d'élaborer un diagramme de classes (DCL). Le choix obéit à des considérations professionnelles. En effet, UML est devenu un standard de modélisation incontournable pour décrire les données complexes [Piechocki, 2007] ; son modèle conceptuel des données est largement reconnu comme un modèle sémantique performant [Abelló, 2015].

Un diagramme de classes (DCL) est défini par un ensemble de classes d'objets et un ensemble de liens inter-classes.

Formellement,

**Définition 17.** Un DCL est défini par un triplet *(N, C, L)* où :

- *N* désigne le nom du diagramme,
- $CL = \{cl_1, \dots, cl_n\}$ est un ensemble de classes,
- $L = LA \cup LC \cup LAG \cup LH$ est respectivement l'ensemble de liens d'association, de composition, d'agrégation et d'héritage.

Nous rappelons qu'une classe définit la structure (attributs) et le comportement (opérations) d'un ensemble d'objets ayant une sémantique et des propriétés communes. Comme le montre la définition 18, notre approche prend en compte uniquement la partie structurelle d'une classe.

Formellement,

**Définition 18.** ∀ i ∈ [1..n], une classe $cl_i$ ∈ CL est définie par un couple ($N$, $A$) où :

- $cl_i.N$ est le nom de la classe,
- $cl_i.A = AA \cup SA \cup MA$ est l'ensemble des attributs de la classe $c_i$, où :
  - $AA = \{aa_1, \dots, aa_m\}$ est l'ensemble des attributs atomiques de $c_i$,
  - $SA = \{sa_1, \dots, sa_l\}$ est l'ensemble des attributs structurés de $c_i$,
  - $MA = \{ma_1, \dots, ma_r\}$ est l'ensemble des attributs multivalués de $c_i$.

Un attribut est élément qui sert à caractériser une classe. Notons qu'un attribut peut être atomique, structuré ou multivalué. Un attribut atomique est associé à un type de données prédéfini tel que Number, String, Boolean ou Date. Un attribut structuré est composé d'autres attributs qui, à leur tour, peuvent être atomiques, structurés ou multivalués. Un champ multivalué est composé d'un ensemble de valeurs. Celles-ci sont toutes atomiques, structurées ou multivaluées.

Formellement :

**Définition 19.** ∀ j ∈ [1..m], un attribut atomique $aa_j \in AA$ est défini par un couple (N, $T$) où :

- $aa_j$.N est le nom de l'attribut $aa_j$,
- $aa_j$.T est le type de l'attribut $aa_j$ ; T peut être String, Integer, Boolean, etc.

**Définition 20.** ∀ k ∈ [1..l], un attribut structuré $sa_k \in SA$ est défini par un couple ($N$, $A'$) où :

- $sa_k$.N est le nom de l'attribut $sa_k$ ,
- $sa_k . A' = AA' \cup SA' \cup MA'$ est l'ensemble des attributs atomiques, structurés et multivalués qui composent $sa_k$. Les sous-ensembles $AA'$, $SA'$ et $MA'$ sont respectivement définis de la même manière que $c_i.AA$, $c_i.SA$ et $c_i.MA$ avec ∀ i ∈ [1..n] (cf. Définition 18).

**Définition 21.** ∀ z ∈ [1..r], un attribut multivalué $ma_z \in MA$ est défini par un triplet ($N$, $T$, $A'$) où :

- $ma_z.N$ est le nom de l'attribut $ma_z$,
- $ma_z.T$ est le type de l'attribut $ma_z$ dans le cas ou celui-ci est multivalué de valeurs atomiques ; $ma_z.T$ est *Null* sinon,
- $ma_z . A' = AA' \cup SA' \cup MA'$ est l'ensemble des attributs atomiques, structurés et multivalués qui composent $ma_z$. Si celui-ci est multivalué de valeurs atomiques, $ma_z.A' = \emptyset$.

Après avoir décrit les classes d'objets, nous présentons les liens. Un lien est une relation entre deux ou plusieurs objets. La norme UML considère plusieurs catégories de liens ; les plus couramment utilisés (notamment dans notre application médicale décrite dans le chapitre 1) sont les liens d'association, de composition, d'agrégation et d'héritage.

Un lien d'association est une relation sémantique entre deux ou plusieurs classes. Il est défini par un nom, des extrémités (LinkEnd) et d'éventuels attributs. Une extrémité est définie par le nom de classe reliée et par une cardinalité. Celle-ci est un couple représentant le nombre minimum et maximum d'objets de la classe qui interviennent dans le lien. Les éventuels attributs d'un lien d'association figurent uniquement dans le cas d'une classe d'association. Nous rappelons que celle-ci est un lien d'association qui contient des informations supplémentaires (attributs).

Formellement :

**Définition 22.** $LA = \{la_1, \dots, la_m\}$ est un ensemble de liens d'association.

$\forall$ i $\in$ [1..m], un lien d'association $la_i \in LA$ est défini par $(N, CP, A)$ où :

- $la_i.N$ est le nom du lien.
- $la_i.CP = \{cp_1, \dots, cp_f\}$ contient l'ensemble des collections liées avec pour degré f $\geq$ 2. $\forall$ j $\in$ [1..f], $cp_f$ est défini par $(c, cr)$ où :
  - $cp_j$.c est une classe liée,
  - $cp_j.cr$ est la cardinalité correspondante à $c$.
- $la_i.A = AA \cup SA \cup MA$ est un ensemble d'attributs atomiques, structurés et multivalués du lien $la_i$. Les sous-ensembles $la_i.AA$, $la_i.SA$ et $la_i.MA$ sont respectivement définis de la même manière que $c_i.AA$, $c_i.SA$ et $c_i.MA$ avec $\forall$ i $\in$ [1..n] (cf. Définition 18).

  Notons que si $la_i.A \neq \emptyset$ alors il s'agit d'une classe d'associations.

Un lien de composition est un lien d'association particulier sur lequel est définie une contrainte d'intégrité. Il associe une classe composite et une classe composante, où chaque objet de la classe composante est associé exclusivement à un objet de la classe composite. De plus, l'existence d'un objet de la classe composante est dépendante de l'existence de l'objet composite. En effet, si celui-ci est supprimé, tous les objets composants qui lui sont associés seront aussi supprimés. Ainsi, un lien de composition est défini par deux extrémités : composite et composante. Chacune d'elles est définie par le nom de la classe associée et une cardinalité.

Formellement :

**Définition 23.** $LC = \{lc_1, \dots, lc_m\}$ est un ensemble de liens de composition.

$\forall$ i $\in$ [1..m], un lien de composition $lc_i \in LC$ est défini par $(cp^{composite}, cp^{composante})$ où :

- $lc_i . cp^{composite}$ est un couple définissant la classe composite de la forme $(c, cr)$ où :
  - $cp^{composite}$.c est le nom de la classe composite.
  - $cp^{composite} . cr$ est la cardinalité correspondante à la classe composite. Cette cardinalité contient généralement la valeur 1..1 (ou 1 pour la forme contractée).
- $lc_i . cp^{composante}$ est un couple définissant la classe composante de la forme $(c, cr)$ où:
  - $cp^{composante}$.c est le nom de la classe composante.
  - $cp^{composante} . cr$ est la cardinalité correspondante à la classe composante.

A l'instar d'un lien de composition, un lien d'agrégation est un type particulier de lien d'association. Il associe une classe agrégat et une classe agrégée, où chaque objet de la classe agrégée appartient à un ou plusieurs objets de la classe composite. Cependant, l'existence de l'objet agrégé et celle de l'objet agrégat sont indépendantes comme dans le cas d'une association classique. Un lien d'agrégation est donc défini par deux extrémités : agrégat et agrégée. Chacune d'elles est définie par le nom de la classe associée et une cardinalité.

Formellement :

**Définition 24.** $LAG = \{lag_1, \dots, lag_m\}$ est un ensemble de liens d'agrégation.

$\forall$ i $\in$ [1..m], un lien d'agrégation $lag_i \in LAG$ est défini par $(cp^{agrégat}, cp^{agrégé})$ où :

- $lag_i . cp^{agrégat}$ est un couple définissant la classe agrégat de la forme $(c, cr)$ où :
  - $cp^{agrégat}$.c est le nom de la classe agrégat.
  - $cp^{agrégat} . cr$ est la cardinalité correspondante à la classe agrégat.
- $lag_i . cp^{agrégé}$ est un couple définissant la classe agrégée de la forme $(c, cr)$ où:
  - $cp^{agrégé}$.c est le nom de la classe agrégée.
  - $cp^{agrégé} . cr$ est la cardinalité correspondante à la classe agrégée.

Un lien d'héritage consiste à définir une ou plusieurs sous-classes dans une super-classe. Les sous-classes héritent de toutes les caractéristiques (attributs et liens) de la super-classe. De plus, elles possèdent leurs propres caractéristiques. Ainsi,

un lien d'héritage est défini par un ensemble d'extrémités ; l'une de ces extrémités représente la super-classe et les autres représentent les sous-classes.

Formellement :

> **Définition 25.** $LH = \{lh_1, \dots, lh_m\}$ est un ensemble de liens d'héritage. ∀ i ∈ [1..m], $lh_i \in LH$ est défini par (sc, $SSC$) où :
>
> - $lh_i$.sc est la super-classe.
> - $lh_i . SSC = \{ssc_1, \dots, ssc_k\}$ où ∀ j ∈ [1..k], avec k ≥ 1, $ssc_1$ est une sous-classe.

Nous présentons les différents concepts du modèle PMI-MCD au travers le métamodèle de la figure 5.3. Ce métamodèle ainsi que tous les métamodèles présentés dans ce mémoire sont implantés et validés en utilisant la plateforme Eclipse Modeling Framework (EMF) présentée dans le chapitre 6.

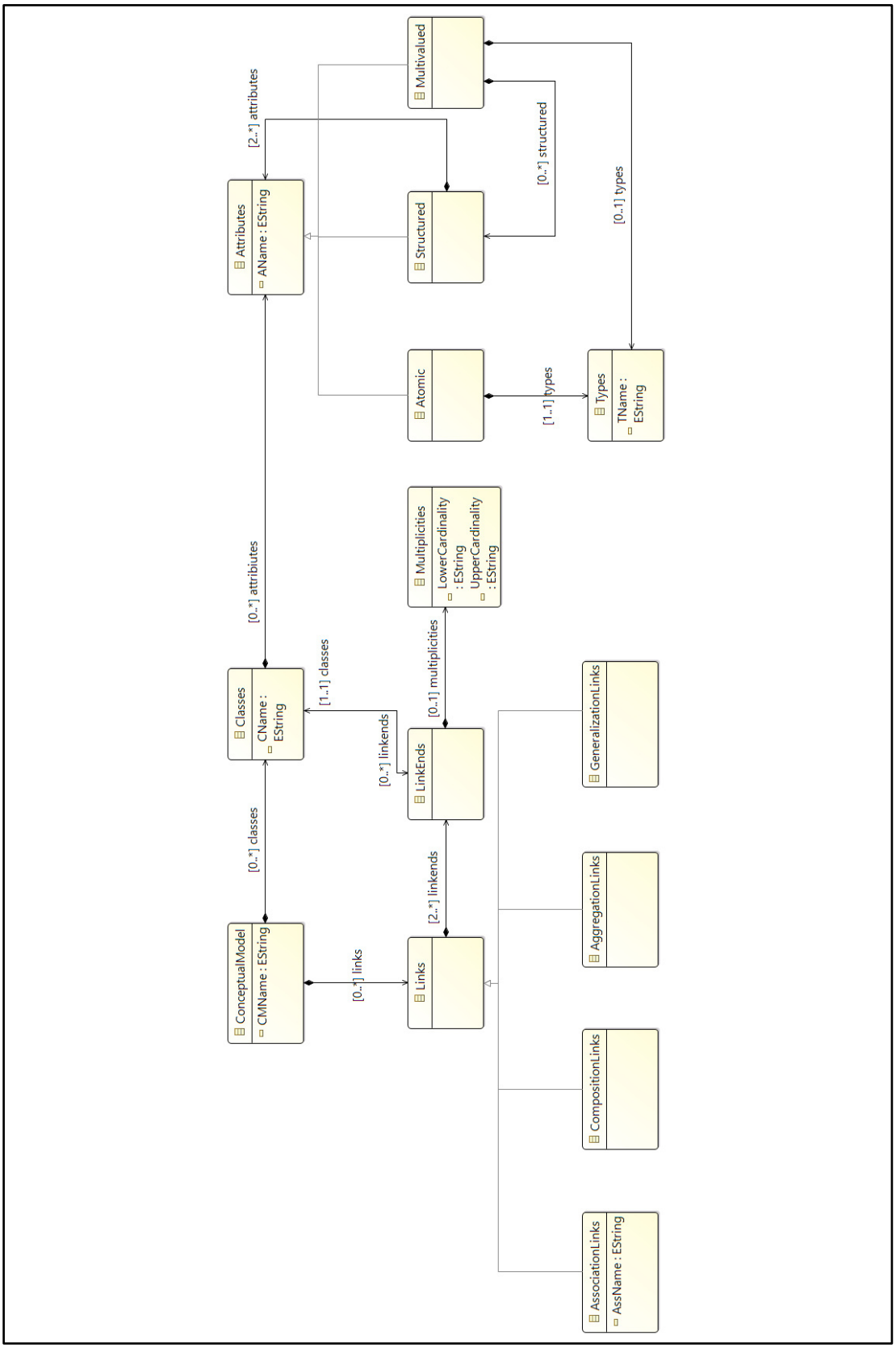



**Figure 5.3 -** Métamodèle du PIM-MCD

## 5.4 Règles de transformation

Après avoir formalisé les concepts présents dans le modèle source (modèle physique NoSQL) et dans le modèle cible (modèle conceptuel UML), nous allons décrire le passage automatique entre les deux modèles sous forme d'une séquence de règles de transformation.

**R1 :** Un modèle physique *MP* est transformé en un diagramme de classe UML *DCL*, où *DCL*.N = *MP*.N.

**R2 :** $\forall\, i \in [1..n]$, Une collection $c_i \in MP.C$ est transformée en une classe $cl_i \in DCL.CL$ , où *cl*.N = *c*.N.

**R3 :** $\forall$ k ∈ [1..p], un champ atomique $af_k \in c_i.AF$ est transformé en un attribut atomique $aa_k \in cl_i.AA$ où :

- $aa_k$.N = $af_k$.N,
- $aa_k$.T = $af_k$.T.

**R4 :** $\forall$ l ∈ [1..q], un champ structuré $sf_l \in c_i.SF$ est transformé en attribut structuré $sa_l \in cl_i.SA$ où :

- $sa_l$.N = $sf_l$.N,
- $sa_l.A'$ est généré en transformant $sf'_l.F''$ comme suit :
  - Générer $sa_l.AA'$ en appliquant R3 sur chaque champ atomique appartenant à $sf'_l.AF''$,
  - Générer $sa_l.SA'$ en appliquant R4 sur chaque champ structuré appartenant à $sf'_l.SF''$,
  - Générer $sa_l.MA'$ en appliquant R5 sur chaque champ multivalué appartenant à $sf'_l.MF''$.

**R5 :** $\forall$ z ∈ [1..r], un champ multivalué $mf_z \in c_i.MF$ est transformé en attribut multivalué $ma_z \in cl_i.MA$ où :

- $ma_z$.N = $mf_z$.N,
- $ma_z$.T = $mf_z$.T,
- $ma_z.A'$ est généré en transformant $mf'_z.F''$ comme suit :
  - Générer $ma_z.AA'$ en appliquant R3 sur chaque champ atomique appartenant à $mf'_z.AF''$,
  - Générer $ma_z.SA'$ en appliquant R4 sur chaque champ structuré appartenant à $mf'_z.SF''$,
  - Générer $ma_z.MA'$ en appliquant R5 sur chaque champ multivalué appartenant à $mf'_z.MF''$.

Dans l'exemple de la figure 5.4, notre processus applique les règles R1, R2, R3 et R5 comme suit :

- La classe Patients est obtenue en appliquant la règle R2 sur la collection Patients,
- L'attribut multivalué First-NameP est obtenu en appliquant la règle R5,
- L'attribut atomique Last-NameP est obtenu en appliquant la règle R3,
- L'attribut multivalué Addresses est obtenu en appliquant la règle R5.

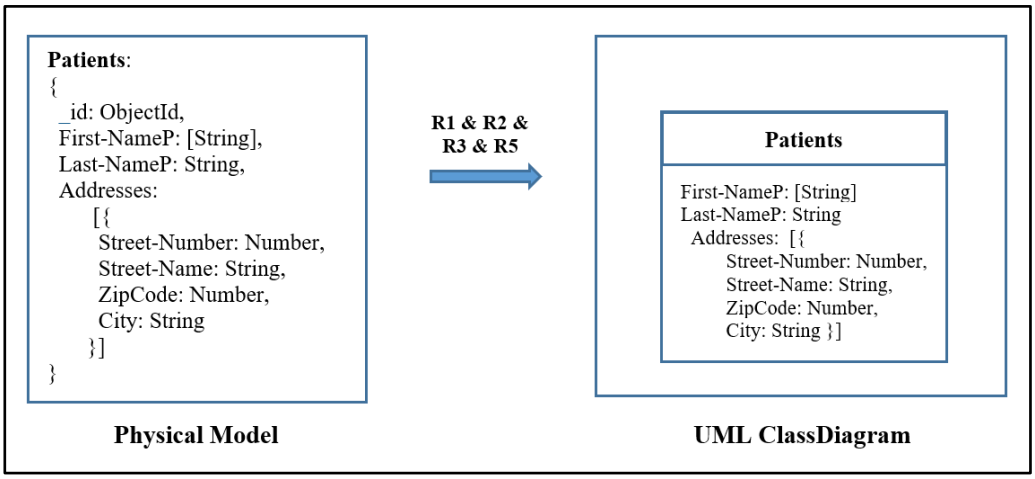



**Figure 5.4 -** Exemple de transformation d'une collection et de champs

**R6 :** Un champ DBRef $dbref$, déclaré dans la collection $c_0$ et défini par un nom N, un ensemble de champs $F'$ (constitué de *n* paires ***($id: ObjectId, $Ref: $C_i$)*** avec i ∈ [1..n] et éventuellement *m* champs atomiques et *l* champs structurés), est transformé en un lien d'association $la \in DCL.LA$ d'ordre (n+1) où :

- $la.N = dbref.N$,
- Si $n = 1$, alors il s'agit d'un lien binaire :
  - $cp_0$.c = $c_0$.N,
  - $cp_1$.c = $c_1$.N,
  - $cp_0$.cr = *0..**,
  - Si le champ *dbref* est monovalué, alors $cp_1$.cr = *0..1*
  - Si le champ *dbref* est multivalué, alors $cp_1$.cr = *0..**
  - $la.CP = cp_0 \cup cp_1$
- Si $n > 1$ alors il s'agit d'un lien d'ordre n+1 :
  - $cp_0$.c = $c_0$.N
  - Si le champ *dbref* est monovalué, alors $cp_0$.cr = *0..1*
  - Si le champ *dbref* est multivalué, alors $cp_0$.cr = *0..**
  - $la.CP = cp_0$
  - ∀ i ∈ [1..n], avec n>1 :
    - $cp_i.c = c_i.N$
    - $cp_i.cr = 0..*$
    - $la.CP = la.CP \cup cp_i$

- Chaque champ atomique $af_j \in dbref.AF'$, avec j ∈ [1..m], est transformé en un attribut atomique $aa_j$ en appliquant R3,
  - $la.AA = la.AA \cup aa_j$
- Chaque champ structuré $sf_l \in dbref.SF'$, avec l ∈ [1..q], est transformé en un attribut structuré $sa_j$ en appliquant R4,
  - $la.SA = la.SA \cup sa_l$
- Chaque champ multivalué $mf_z \in dbref.MF'$, avec z ∈ [1..r], est transformé en un attribut multivalué $ma_z$ en appliquant R5,
  - $la.MA = la.MA \cup ma_z$

Dans l'exemple de la figure 5.5, notre processus applique les règles R1, R2 et R6 comme suit :

- Les classes *Patients* et *Doctors* sont obtenues en appliquant la règle R2 sur les collections *Patients* et *Doctors* respectivement.
- Le lien d'association Treating-Doctor est obtenu en appliquant la règle R6 sur le champ DBRef *Treating-Doctor*.

La figure 5.6 montre une autre utilisation de la règle R6. Il s'agit de la génération d'une classe d'associations.

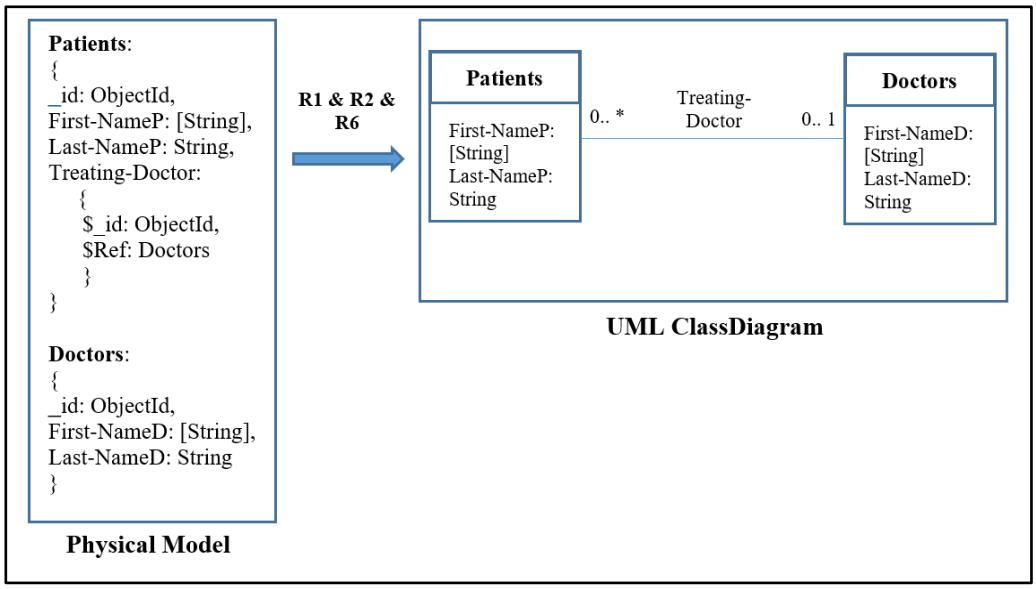



**Figure 5.5 -** Exemple de génération d'un lien d'association

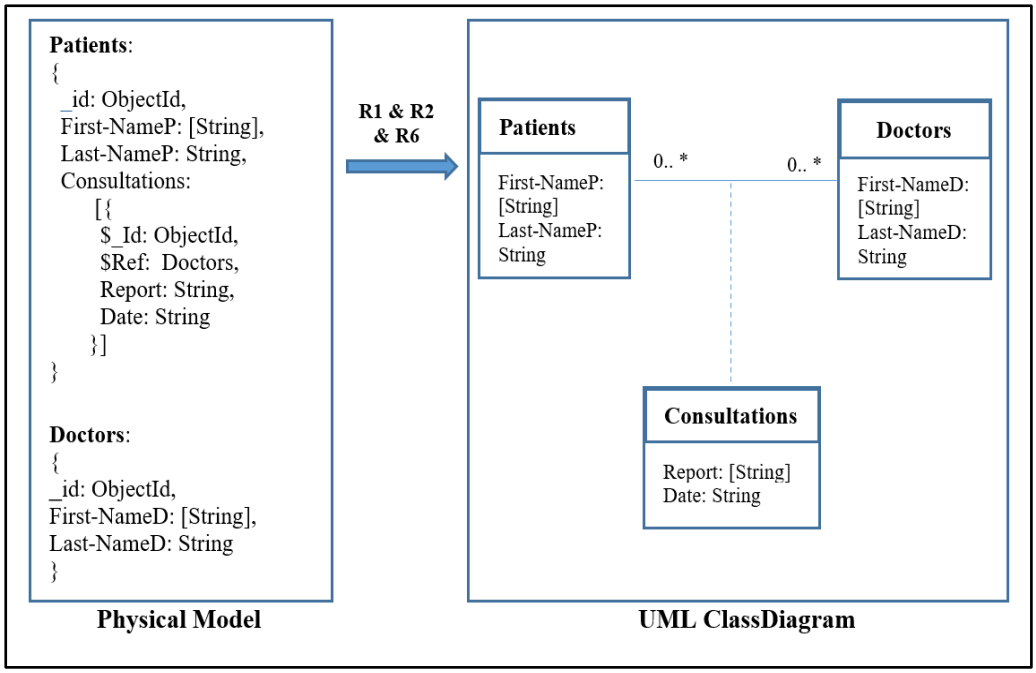



**Figure 5.6 -** Exemple de génération d'une classe d'associations

**R7 :** Un champ DBComp $dbcomp$, déclaré dans la collection $c_0$ et qui pointe la collection $c_1$, est transformé en un lien de composition $lc \in DCL.LC$ où :

- $lc.cp^{composite}.c = c_0.N$
- $lc.cp^{composite}.cr = 0..1$
- $lc.cp^{composante}.c = c_1.N$
- Si *dbcomp* est monovalué, alors $lc.cp^{composante}.cr = 0..1$
- Si *dbcomp* est multivalué, alors $lc.cp^{composante}.cr = 0..*$

Dans l'exemple de la figure 5.7, nous montrons un exemple de génération d'un lien de composition.

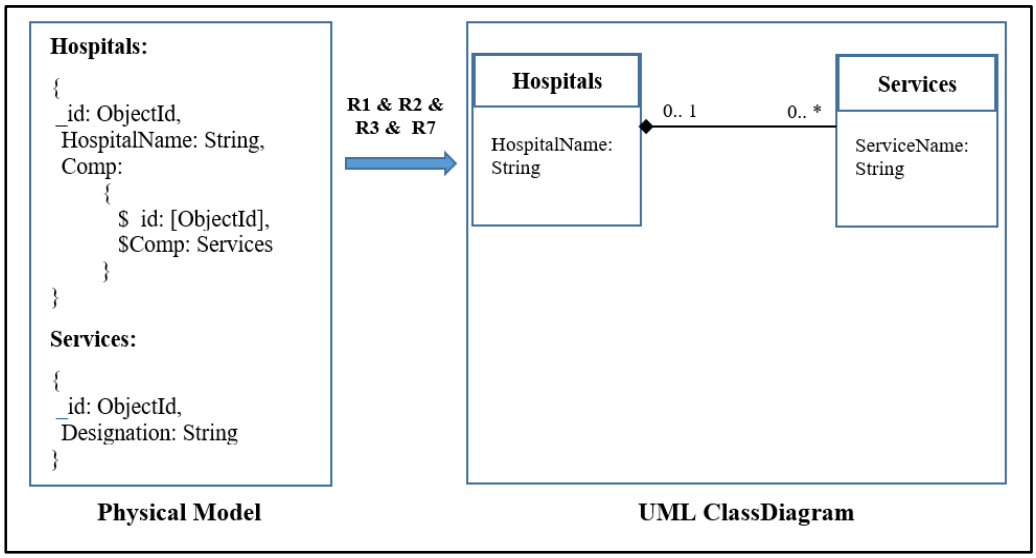



**Figure 5.7 -** Exemple de génération d'un lien de composition

**R8 :** Un champ DBAgg $dbagg$, déclaré dans la collection $c_0$ et qui pointe la collection $c_1$, est transformé en un lien d'agrégation $lag \in DCL.LAG$ où :

- $lag.cp^{agrégat}.c = c_1.N$
- $lag.cp^{agrégé}.c = c_0$
- $lag.cp^{agrégé}.cr = 0..*$
- Si *dbagg* est monovalué, alors $lag.cp^{agrégat}.cr = 1..1,$
- Si *dbagg* est multivalué, alors $lag.cp^{agrégat}.cr = 1..*$

La figure 5.8 montre la génération d'un lien d'agrégation entres les équipes et les médecins.

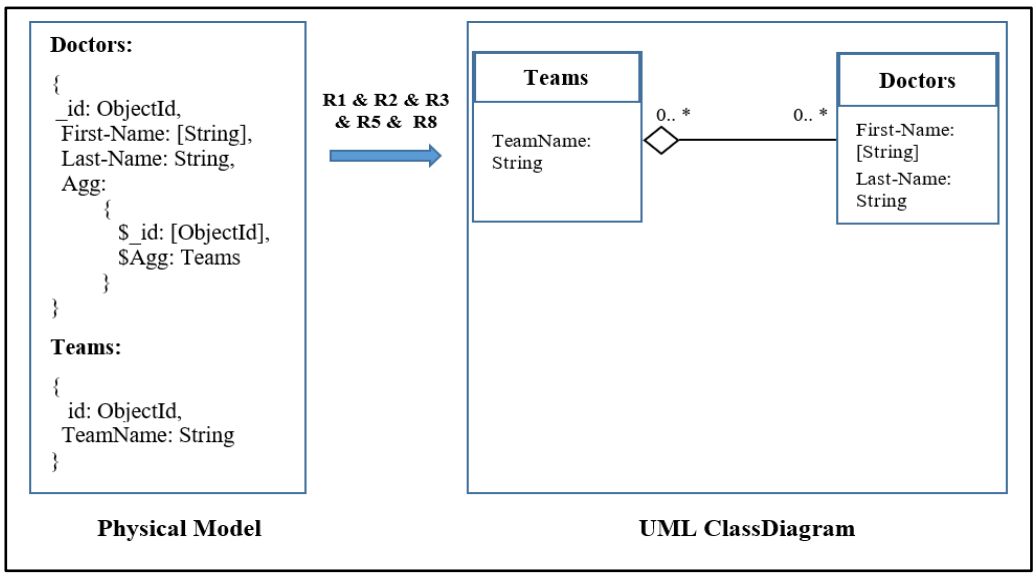



**Figure 5.8 -** Exemple de génération d'un lien d'agrégation

**R9 :** Un champ DBSub $dbsub$, déclaré dans la collection $c_0$ et qui ponte la collection $c_1$, est transformé en un lien d'héritage $lh \in DCL.LH$ où :

- $lh.sc = c_1.N$,
- $lh.ssc = c_0.N$

La figure 5.9 montre la génération d'un lien d'héritage entres les spécialistes et les médecins.

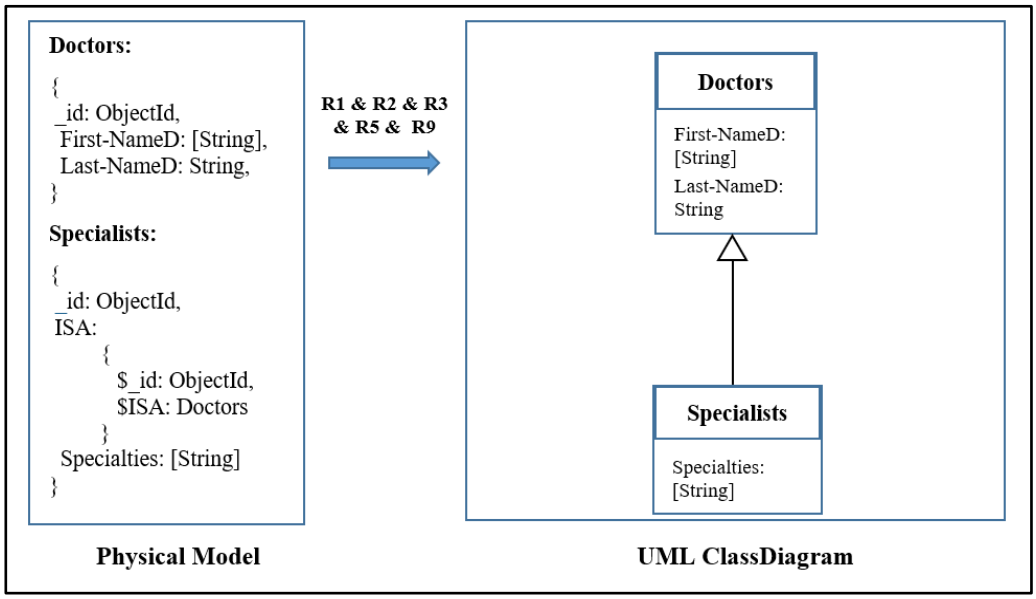



**Figure 5.9 -** Exemple de génération d'un lien d'héritage

Les règles de transformation décrites ci-dessus sont synthétisées dans le tableau 5.1.

| Règle de trasnformation | Description |
|---|---|
| **R1** | transforme un modèle physique en un diagramme de classe UML. |
| **R2** | transforme une collection en une classe UML. |
| **R3** | transforme un champ atomique en un attribut atomique de même nom et de même type dans la classe correspondante. |
| **R4** | transforme un champ structuré en un lien de composition ou en un attribut structuré. |
| **R5** | transforme un champ multivalué en un attribut multivalué de meme nom dans la classe correspondante. |
| **R6** | transforme un champ DBRef en un lien d'association de même nom |
| **R7** | transforme un champ DBComp en un lien de composition |
| **R8** | transforme un champ DBAgg en un lien d'agrégation |
| **R9** | transforme un champ DBSub en un lien d'héritage |

**Tableau 5.1 -** Tableau synthétique des règles de transformation

Nous avons aussi formalisé ces règles de transformation en utilisant le formalisme graphique du langage QVT (Query/View/Transformation), qui est un standard défini par l'OMG pour exprimer des transformations de modèles. Une transformation QVT entre deux modèles candidats (source et cible) est spécifiée grâce à un ensemble de relations. Chaque relation est composée des éléments suivants :

- « Domains » : chaque domaine désigne un modèle candidat et un ensemble d'éléments à relier.
- « Relation Domain » : permet de spécifier le type de relation entre les domaines, elle peut être marquée comme « Checkonly » (C) ou « Enforced »

(E). Un domaine « Chekonly » permet de vérifier s'il existe une correspondance valide qui satisfait la relation ; alors qu'un domaine « Enforced » permet de créer un élément dans le modèle si le lien de correspondance n'est pas vérifié. Pour chaque domaine, le nom de son métamodèle sous-jacent doit être spécifié.

- La clause « When » : décrit les pré-conditions qui doivent être remplies pour réaliser la transformation.
- La clause « Where »: détermine les post-conditions qui doivent être remplies par tous les éléments du modèle participant à la relation.

Dans la suite, nous détaillons certaines règles de transformations appliquées dans le passage du modèle physique vers le modèle conceptuel en QVT graphique.

**Relation « Main ».** Cette relation est le point d'entrée du processus de transformation. La partie gauche de la figure 5.10 montre les éléments du modèle physique source (mp : ModèlePhysique) transformés en éléments du modèle conceptuel cible (dcl : DCL) présenté par la partie droite de la figure. Le modèle physique est transformé en un DCL de même nom. La clause "where" spécifie que les collections sont transformées en classes UML.

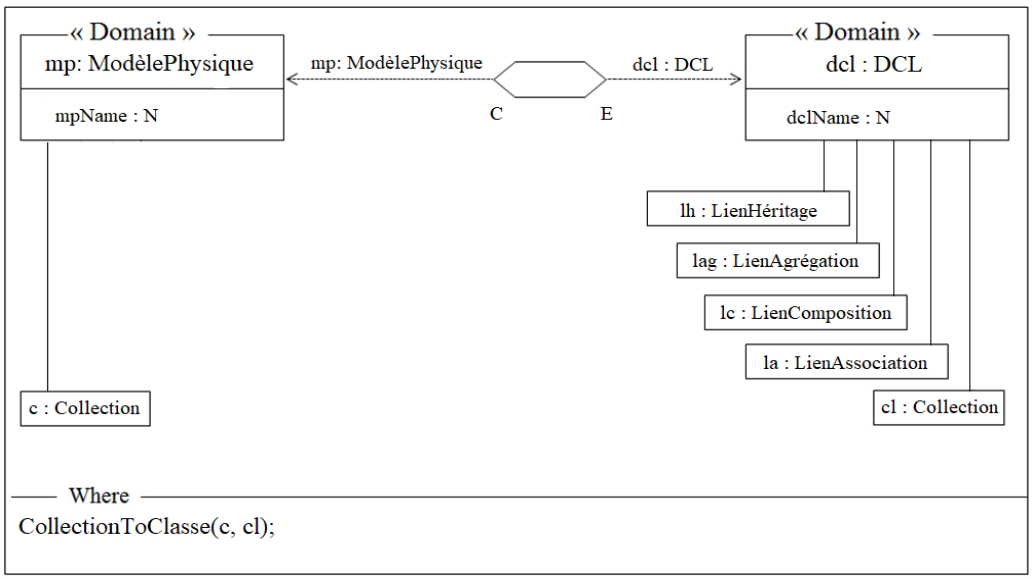



**Figure 5.10 -** Relation Main de la transformation du modèle physique en un modèle conceptuel

**Relation « CollectionToClasse ».** Une collection est transformée en une classe UML dont le nom correspond au nom de la collection d'entrée. Les champs atomiques et les champs structurés (à l'exception des champs DBRef, DBSub et DBAgg) sont transformés respectivement en des attributs atomiques et structurés dans les classes correspondantes, ceci en appliquant la relation « ChampToAttribut» indiquée dans la clause "where" de la figure 5.11.

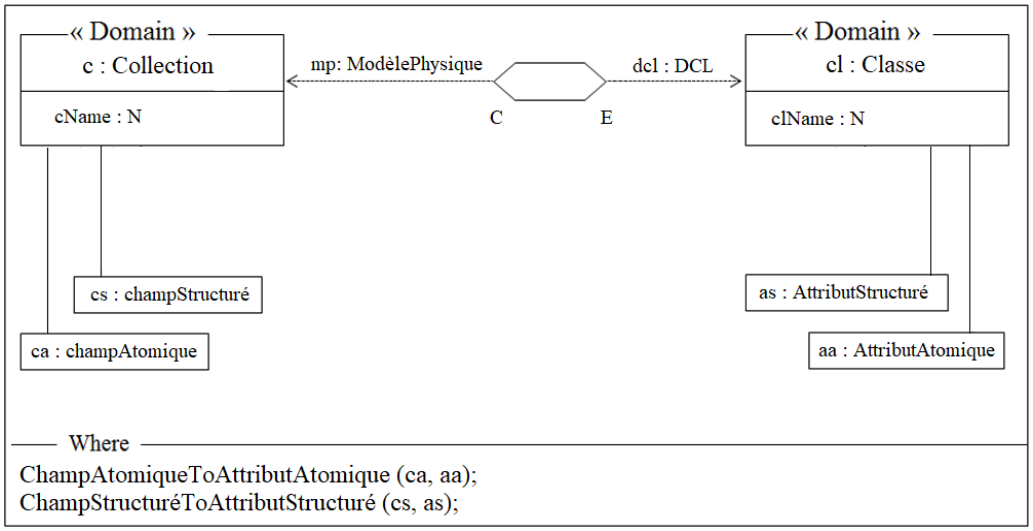



**Figure 5.11 -** Relation de la transformation des collections en classes

**Relation « ChampToLien ».** Nous traitons dans cette relation la transformation des champs structurés spéciaux DBRef, DBSub et DBAgg comme le montre la figure 5.12. Pour cela, nous appliquons respectivement trois relations : dbrefToLienAssociation, dbsubToLienHéritage et dbaggToLienAgrégation. Ainsi, les champs DBRef sont transformés en des liens d'association. Les champs DBSub sont transformés en des liens d'héritage. Enfin, les champs DBAgg sont transformés en des liens d'agrégation.

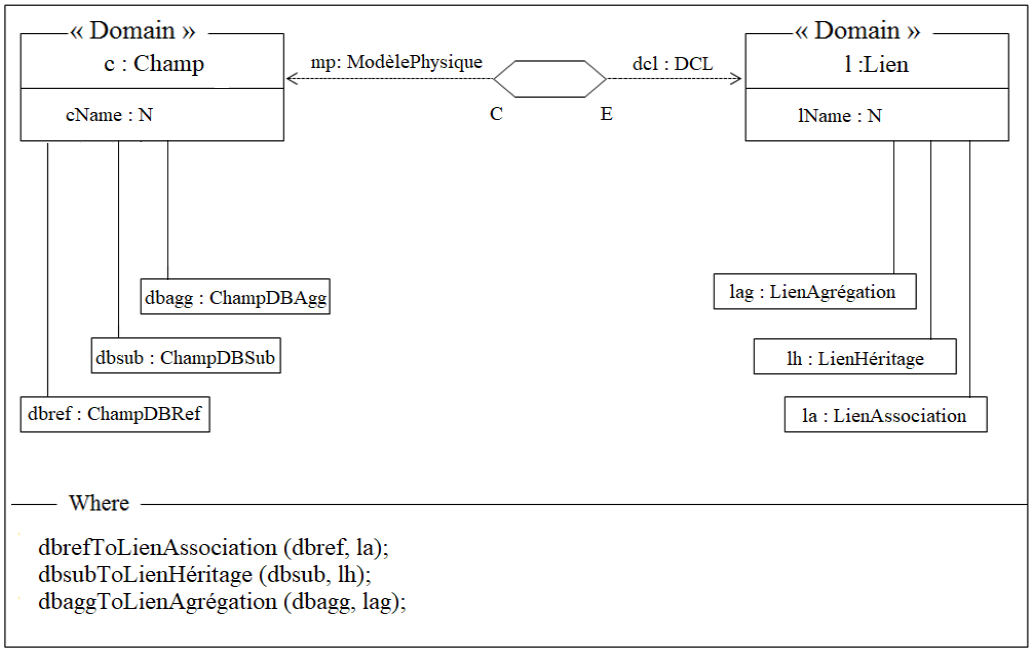



**Figure 5.12 -** Relation pour la génération des liens d'association, d'héritage et d'agrégation

## 5.5 Bilan du processus d'extraction du modèle conceptuel

### 5.5.1 Synthèse

La plupart des SGBD NoSQL sont caractérisés par la propriété schema less (sans schéma) qui signifie absence du modèle de données lors de la création d'une BD. Cette propriété apporte une souplesse indéniable en permettant l'évolution du modèle de données au fur et à mesure de l'exploitation de la BD. Cependant, l'absence du modèle ne permet pas de connaître comment les données sont stockées et reliées dans la BD. Cette méconnaissance du modèle de données rend difficile l'expression des requêtes sur cette BD. Pour surmonter cette difficulté, nous sommes convaincus qu'il est important de fournir aux utilisateurs (développeurs ou décideurs) les modèles nécessaires pour manipuler des BD NoSQL. Il s'agit de deux modèles complémentaires décrivant la BD : (1) le modèle physique qui décrit l'organisation interne des données et permet d'exprimer des requêtes et (2) le modèle conceptuel qui fournit un haut niveau d'abstraction et permet de comprendre la sémantique de la BD NoSQL. Dans le chapitre précèdent, nous avons proposé le processus *ToPhysicalModel* qui extrait du modèle physique à partir de la BD. Dans ce chapitre, nous étudions la génération du modèle conceptuel. Pour cela, nous avons proposé le processus *ToConceptualModel*. Il s'agit un processus dirigé par les modèles qui consiste à transformer le modèle physique déjà extrait en un modèle conceptuel. Celui-ci est exprimé en utilisant le formalisme UML. Le passage entre le modèle physique et conceptuel est assuré par la définition d'un ensemble de règles de transformation de type Model-To-Model (M2M) formalisées avec le standard QVT.

Les travaux de ce chapitre ont été présentés dans les publications suivantes :

[Ait Brahim et al., 2019d], [Abdelhedi et al., 2020a] et [Abdelhedi et al., 2020b]

### 5.5.2 Positionnement

Dans la littérature, de nombreux travaux proposent des solutions pour extraire les modèles de données des BD NoSQL schema-less. La plupart de ces travaux se focalisent uniquement sur le niveau physique. Cependant, peu de travaux ont abouti jusqu'à la génération du modèle conceptuel de données.

Dans cette section, nous effectuons le positionnement de notre solution au regard des travaux portant sur l'extraction du modèle conceptuel de données (MCD) d'une BD NoSQL « schema less », notamment les travaux de Comyn-Wattiau et al., 2018], [Izquierdo et al., 2016], [Chillón et al., 2019] et [Munoz-Sanchez et al., 2020].

L'article [Comyn-Wattiau et al., 2018] traite de l'extraction d'un modèle Entité/Association à partir d'une BD NoSQL orientée-graphes. Or ce type de BD ne considère pas les attributs structurés et ne permet pas de distinguer les différents types de liens (association et composition) que l'on rencontre dans notre cas d'étude. D'autre part, les travaux présentés dans [Izquierdo et al., 2016] sont basés sur une BD NoSQL orientée documents. Ils font apparaître les liens de composition mais ne prennent pas en compte les liens d'association entre les collections. Or ce dernier type de liens constitue une pièce maîtresse dans les relations entre objets pour notre cas d'étude présenté dans le chapitre 1. Les travaux de [Chillón et al., 2019] présentent un processus d'extraction d'un modèle conceptuel pour une BD orientée-documents. Ils prennent en compte les liens d'association et les liens de composition. De plus, ils considèrent les attributs atomiques et multivalués d'éléments atomiques. Cependant, ces travaux n'étudient pas les attributs structurés ainsi que les classes d'associations. Enfin, les travaux de [Munoz-Sanchez et al., 2020] présentent une approche d'extraction d'un modèle Entité/Association à partir d'une BD MongoDB. Ces travaux prennent en considération les liens d'association et d'agrégation. Cependant, ils ignorent les autres types de liens, notamment les liens de composition, d'héritage ainsi que les classes d'associations. De plus, ils ne considèrent pas les attributs structurés.

Nous avons proposé dans ce chapitre le processus *ToConceptualModel*. Il s'agit d'un processus dirigé par les modèles qui consiste à transformer le modèle physique déjà extrait par le processus *ToPhysicalModel* présenté dans le chapitre précédent (cf. chapitre 4) en un modèle conceptuel. Celui-ci est exprimé en utilisant le formalisme UML. Le passage entre le modèle physique et conceptuel est assuré par la définition d'un ensemble de règles de transformation de type Model-To-Model (M2M) formalisées avec le standard QVT. L'originalité du processus *ToConceptualModel* réside dans la prise en compte : (1) de différents types d'attributs : atomiques, multivalués et structurés et (2) de différents types de liens : association, composition, héritage, agrégation et classes d'associations.

# Chapitre 6 : Expérimentation et Validation

Le présent chapitre décrit un prototype qui a permis de vérifier et d'expérimenter nos propositions présentées dans les chapitres 4 et 5. Ce prototype est composé de deux modules : (1) le module qui extrait le modèle physique de données (MPD) à partir d'une BD NoSQL orientée-documents et (2) le module qui transforme le MPD en un modèle conceptuel de données (MCD). L'entrée de ce prototype, i.e. la BD NoSQL orientée-documents, est fournie par l'utilisateur.

Ce chapitre présente également une évaluation que nous avons effectuée dans une entreprise afin de montrer la pertinence de nos propositions. Cette évaluation porte essentiellement sur la comparaison entre les temps de rédaction des requêtes sur un BD MongoDB avec ou sans la présence des modèles physique et conceptuel.

Le chapitre 6 est structuré de la manière suivante. La section 6.1 décrit les outils techniques utilisés pour implanter notre prototype. La section 6.2 présente une description du prototype et comporte deux sous-section 6.2.1 et 6.2.2. Celles-ci détaillent les modules composant le prototype. Pour chaque module, nous décrivons les métamodèles, les modèles et les règles de transformation. La section 6.3 compare nos transformations automatiques avec celles mises en œuvre manuellement. Enfin, La section 6.4 conclut le chapitre.

# 6.1 Outils d'implantation

Dans cette section, nous présentons succinctement les techniques que nous avons utilisées pour mettre en place l'environnement expérimental. Etant donné que notre approche est basée sur MDA, nous avons besoin d'une infrastructure adaptée à la méta-modélisation, la modélisation et les transformations M2M (Model-To-Model) ou M2T (Model-To-Text). C'est ainsi que nous avons retenu EMF (Eclipse Modeling Framework) [Budinsky et al., 2004] comme plateforme d'implantation.

EMF fournit un ensemble d'outils destinés à introduire une approche de développement dirigée par les modèles au sein de l'environnement Eclipse[21]. Ces outils apportent trois fonctionnalités principales. La première est la définition d'un métamodèle représentant les concepts utilisés par l'utilisateur, la deuxième est la création des modèles instanciant ce métamodèle et la troisième fonctionnalité correspond à la transformation de modèle vers modèle ou de modèle vers texte.

Dans ce qui suit, nous présentons les outils d'EMF utilisés pour mettre en œuvre notre prototype.

[21] Projets EMF et SOA Tools Platform de Eclipse. Disponible www.eclipse.org/stp/ and www.eclipse.org/emf/

### 6.1.1 Ecore

EMF utilise le langage de méta-modélisation Ecore[22] pour la définition et la vérification des métamodèles. Ce langage se concentre sur la spécification structurelle des métamodèles [Combemale, 2008]. Il s'agit d'une implantation d'un sous ensemble du standard MOF (Meta Object Facility) de l'OMG[23]. La figure 6.1 présente les principaux concepts de méta-modélisation utilisés dans Ecore.

S'inspirant de l'approche orientée-objet, le langage Ecore repose sur la notion de package (EPackage), de classe (EClass), d'attribut (EAttribute), de lien de référence (EReference), de type de données (EDataType), et d'énumération EEnum.

A la racine de la hiérarchie de ces concepts, se trouve EPackage. Il est composé de EClass et de EDataType. EClass représente les caractéristiques structurelles d'un objet à travers un ensemble de EAttribute et de EReference. EReference permet de définir une association entre deux classes du métamodèle. EDataType représente le type d'un attribut qui peut être soit un type de données prédéfini tels que String, Integer et Boolean, soit un type défini par l'usager en utilisant le type énumération de valeurs EEnum.

[22] ECore. The eclipse modeling framework project home page. http ://www.eclipse.org/emf
[23] Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Core Specification, January 2006. Final Adopted Specification

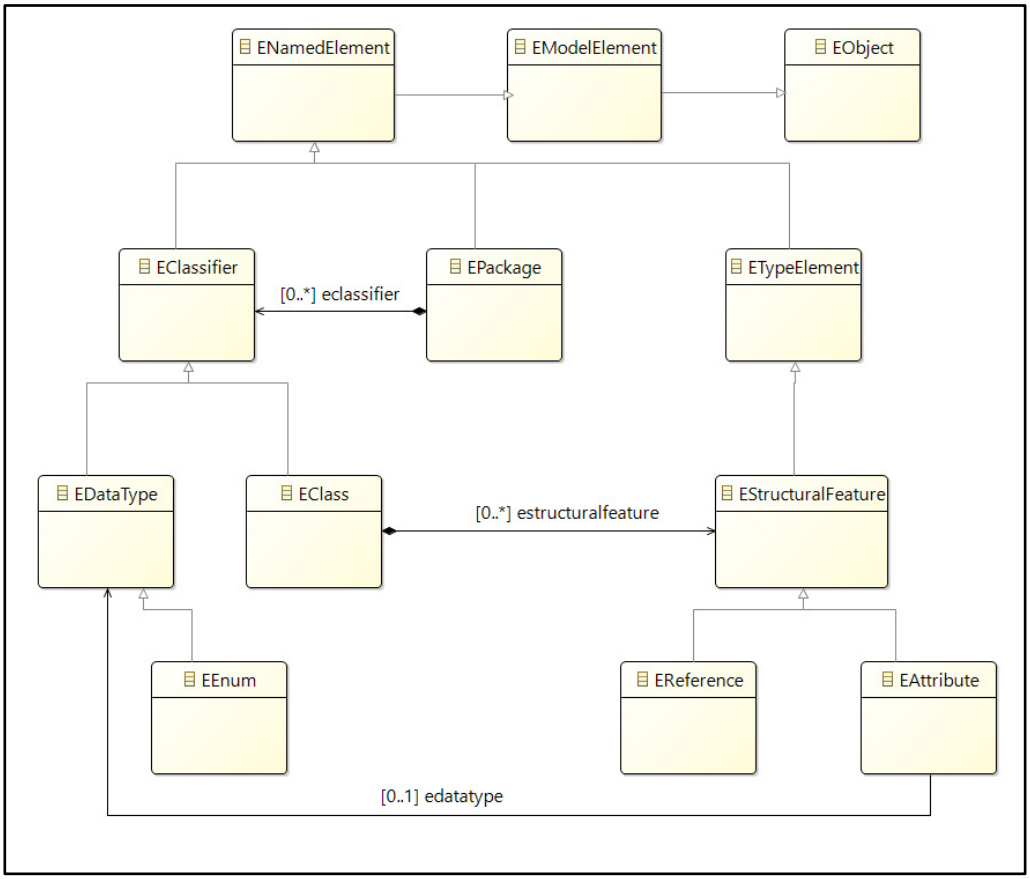



**Figure 6.1 –** Extrait simplifié du métamodèle Ecore [Budinsky et al., 2004]

### 6.1.2 XMI

EMF intègre la norme XMI (XML Metadata Interchange)[24] pour représenter des modèles en XML. Il s'agit d'un format de description qui indique comment les modèles peuvent être traduits sous la forme de documents XML. Le principe de fonctionnement de XMI est de générer automatiquement des grammaires XML (des DTD) à partir d'un métamodèle (dans notre cas il s'agit d'un métamodèle Ecore) ; ceci permet de représenter les modèles instances sous forme de documents XML. Pour ceci, XMI s'appuie sur l'analogie entre les modèles et leur métamodèle d'une part, et les documents XML et leur DTD d'autre part. Cette analogie réside dans le fait qu'un métamodèle en une DTD sont identiques en ce qu'ils définissent respectivement les structures des modèles et des documents XML. La figure 6.2 illustre le principe de fonctionnement de XMI.

[24] OMG, XML Metadata Interchange (XMI) Specification. (http://www.omg.org/technology/documents/formal/xmi.htm)

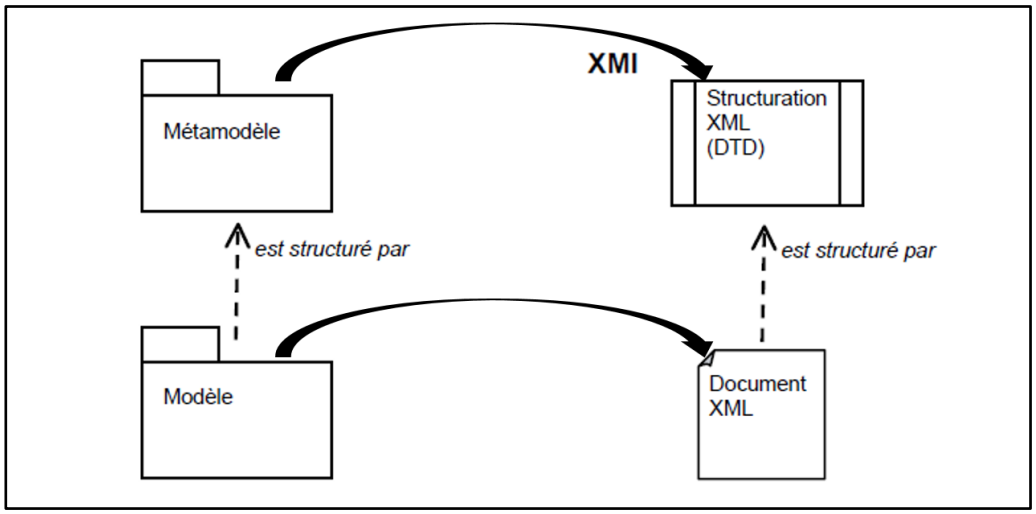



**Figure 6.2 –** Principe de fonctionnement de XMI [Blanc and Salvatori, 2011]

### 6.1.3 QVT

Les transformations de modèles sont au centre d'une approche MDA. Il est donc nécessaire de disposer d'un langage permettant d'accomplir les transformations de type Model-To-Model (M2M). Pour ce faire, l'OMG a proposé le standard QVT[25] (Query, View, Transformation). Il s'agit d'un langage de haut niveau intégré dans l'environnement EMF pour exprimer les transformations de modèles. Il permet d'établir le passage automatique entre les différentes phases d'une approche MDA.

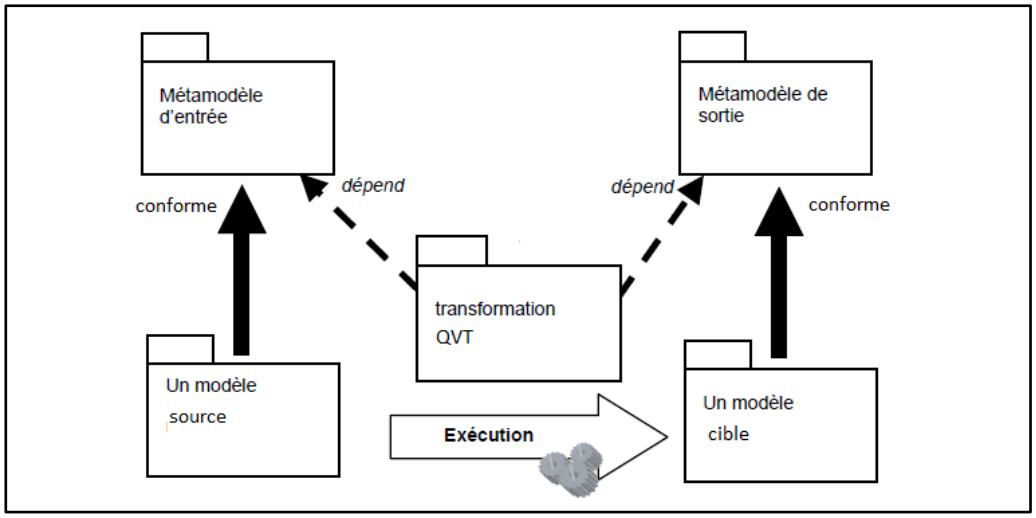



**Figure 6.3 –** Principe d'une transformation QVT [Blanc and Salvatori, 2011]

[25] OMG, Q. (2009). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, OMG (2008).

Le principe d'une transformation QVT consiste à exprimer des règles de correspondance structurelles entre les métamodèles source et cible de cette transformation ; ces règles seront ensuite interprétées par un moteur qui réalisera la transformation. Autrement dit, une transformation QVT prend en entrée un modèle source et renvoie en sortie un modèle cible. Chacun des modèles source et cible est conforme à un métamodèle. La figure 6.3 illustre ce principe.

Notre choix du langage QVT repose sur les deux points suivants :

- Il s'agit d'une norme édictée par l'OMG,
- La plateforme d'implantation EMF que nous avons utilisée, dispose de ce formalisme.

# 6.2 Description du prototype

L'objectif du prototype réalisé est de :

- Générer un modèle physique de données (MPD) à partir d'une BD NoSQL orientée-documents et le mettre à jour au fur et à mesure de l'exécution des requêtes de mise à jour sur la BD. Le MPD résultant décrit l'organisation interne des données de la base et permet d'exprimer des requêtes.
- Transformer le MPD en un modèle conceptuel de données (MCD). Celui-ci fait abstraction des aspects techniques et se concentre sur la sémantique de données.

Notons que notre prototype ne produit pas la BD NoSQL d'entrée. Celle-ci est fournie par un utilisateur (entreprise, décideur ou informaticien).

Comme illustré dans la figure 6.4, le prototype se compose des deux modules ; le module de génération du MPD à partir de la BD NoSQL orientée-documents : *ToPhysicalModel* et le module de transformation du MPD en un MCD : *ToConceptuelModel*.

Le premier permet d'avoir un point de vue global sur l'ensemble de la BD ; il consiste à extraire le MPD et le mettre à jour tout au long de l'exploitation de la BD. Le second module a pour rôle de compléter le premier en intégrant la sémantique des données pour aboutir à un modèle facile à comprendre. Les sections 6.2.1 et 6.2.2 ci-dessous présentent respectivement l'architecture de chacun de ces modules.

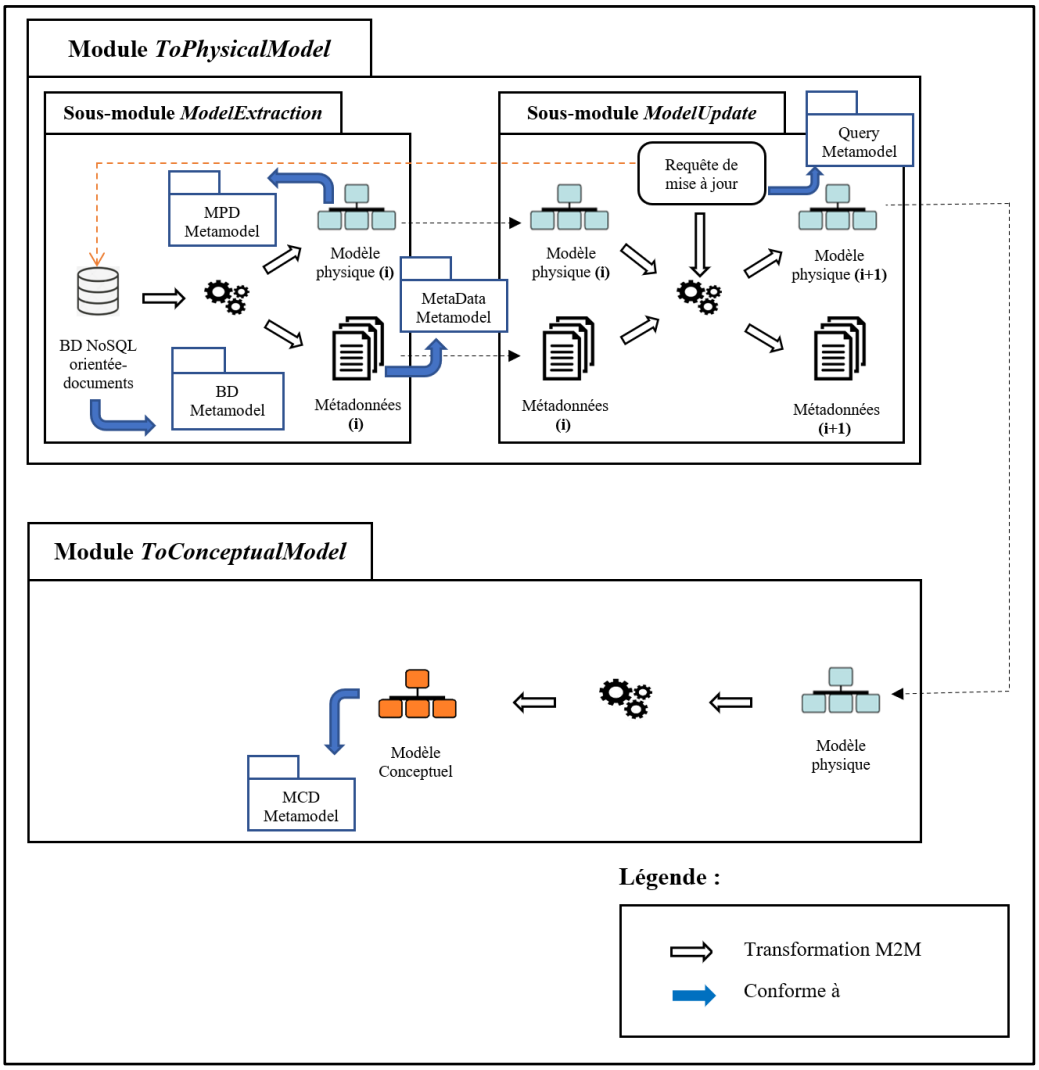



**Figure 6.4 –** Architecture de notre prototype

## 6.2.1 Module *ToPhysicalModel*

Ce module a pour objectif d'automatiser la génération du modèle physique de données (MPD) à partir d'une BD NoSQL orientée-documents. Comme le montre la figure 6.5, ce module est composé de deux sous-modules qui s'appliquent successivement :

- *ModelExtraction* : qui consiste à extraire le MPD à partir de la BD d'entrée ; il s'agit de l'extraction « à froid ».
- *ModelUpdate* : qui met à jour le MPD au fur et à mesure de l'exécution des requêtes de mise à jour sur la BD ; il s'agit de l'extraction « à chaud ».

Pour les sous-modules, le passage d'un modèle à un autre se fait en utilisant des transformations de type M2M formalisées en QVT.

Nous détaillons par la suite les deux sous-modules en précisant l'entrée, la sortie, les transformations et l'implantation.

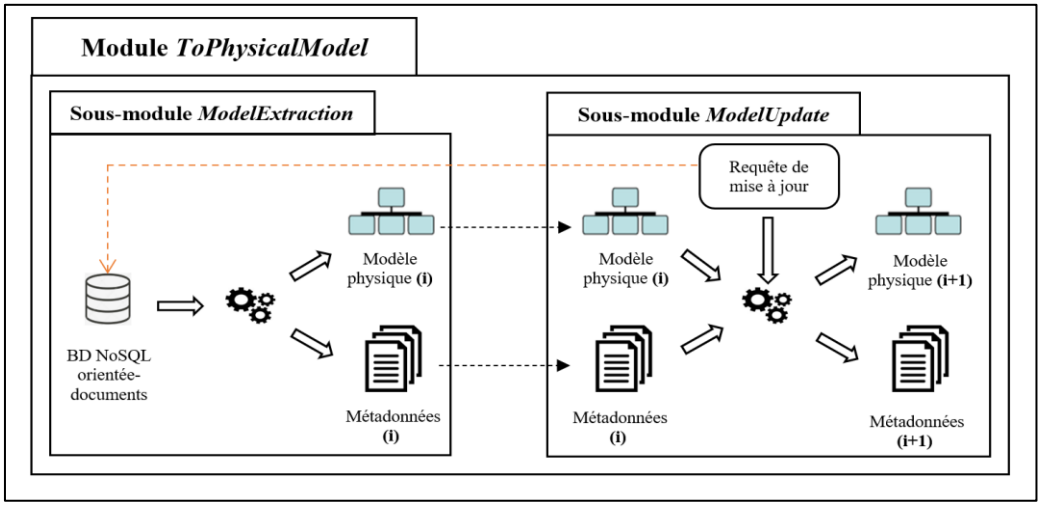



**Figure 6.5 –** Module *ToPhysicalModel*

### 6.2.1.1 Sous-module *ModelExtraction* :

**Entrée**

L'entrée du sous-module *ModelExtraction* est une BD NoSQL orientée-documents MongoDB. L'utilisateur fournit la BD d'entrée en instanciant le métamodèle du PSM-BD que nous avons proposé dans le chapitre 4 (cf. figure 4.10). Ce métamodèle montre les principaux éléments composant une BD MongoDB ainsi que leurs caractéristiques structurelles.

**Sortie**

La sortie du sous-module *ModelExtraction* est composée de deux éléments : (1) Le MPD qui fait apparaître l'organisation interne des données de la BD MongoDB et (2) les métadonnées qui seront utilisées dans le traitement des requêtes de mise à jour dans le sous-module *ModelUpdate*.

**Transformation**

Le sous-module applique deux transformations *DBToPhysicalModel* et *DBToMetaData.* La première traduit une BD NoSQL orientée-documents MongoDB en un modèle physique (MPD) conforme au métamodèle du PSM-MPD que nous avons proposé dans le chapitre 4 (cf. figure 4.11). La deuxième alimente les métadonnées à partir de la BD MongoDB. Ces métadonnées sont conformes au métamodèle du PSM-MetaData (cf. figure 4.12). Ces deux transformations sont réalisées par un ensemble de règles M2M formalisées en QVT.

**Implantation**

La figure 6.6 montre les différentes étapes d'implantation du sous-module *ModelExtraction*. Cette implantation nécessite la définition préalable d'un ensemble de métamodèles et de règles de transformation de type M2M.

Tout d'abord, nous avons créé les métamodèles ECORE qui sont illustrés par la figure 6.7. Il s'agit des métamodèles PSM-BD, PSM-MPD et PSM-MetaData. Ces métamodèles décrivent respectivement la structure d'une BD MongoDB, du modèle physique de données et des métadonnées. Une description détaillée de ces métamodèles est donnée dans le chapitre 4 (cf. figures 4.10, 4.11 et 4.12).

Ensuite, nous avons utilisé le langage QVT pour implanter les règles de transformation assurant les deux passages : *DBToPhysicalModel* (BD MongoDB vers le MPD) et *DBToMetaData* (BD MongoDB vers métadonnées). Les figures 6.8 et 6.9 montrent des extraits du script QVT correspondant à chaque passage ; les commentaires figurant dans les scripts indiquent les règles utilisées.

Finalement, nous avons testé le sous-module en suivant les étapes suivantes :

- **Etape 1 :** Instancier le métamodèle du PSM-BD pour produire la BD MongoDB (cf. figure 6.10) de l'application médicale décrite dans le chapitre 1. Cette BD est stockée sous la forme d'un fichier XMI,

- **Etape 2 :** Exécuter le script *DBToPhysicalModel* sur la BD MongoDB d'entrée. Le résultat est illustré dans la figure 6.11. Il s'agit d'un modèle physique de données (MPD) contenant principalement des collections reliées entre elles par des liens.

- **Etape 3 :** Exécuter le script *DBToMetaData* sur la BD MongoDB d'entrée. Le résultat de cette étape est illustré dans la figure 6.12. Il s'agit des métadonnées de la BD.

Notons que nous présentons uniquement un extrait des métamodèles et des scripts QVT utilisés par le sous-module *ModelExtraction*.

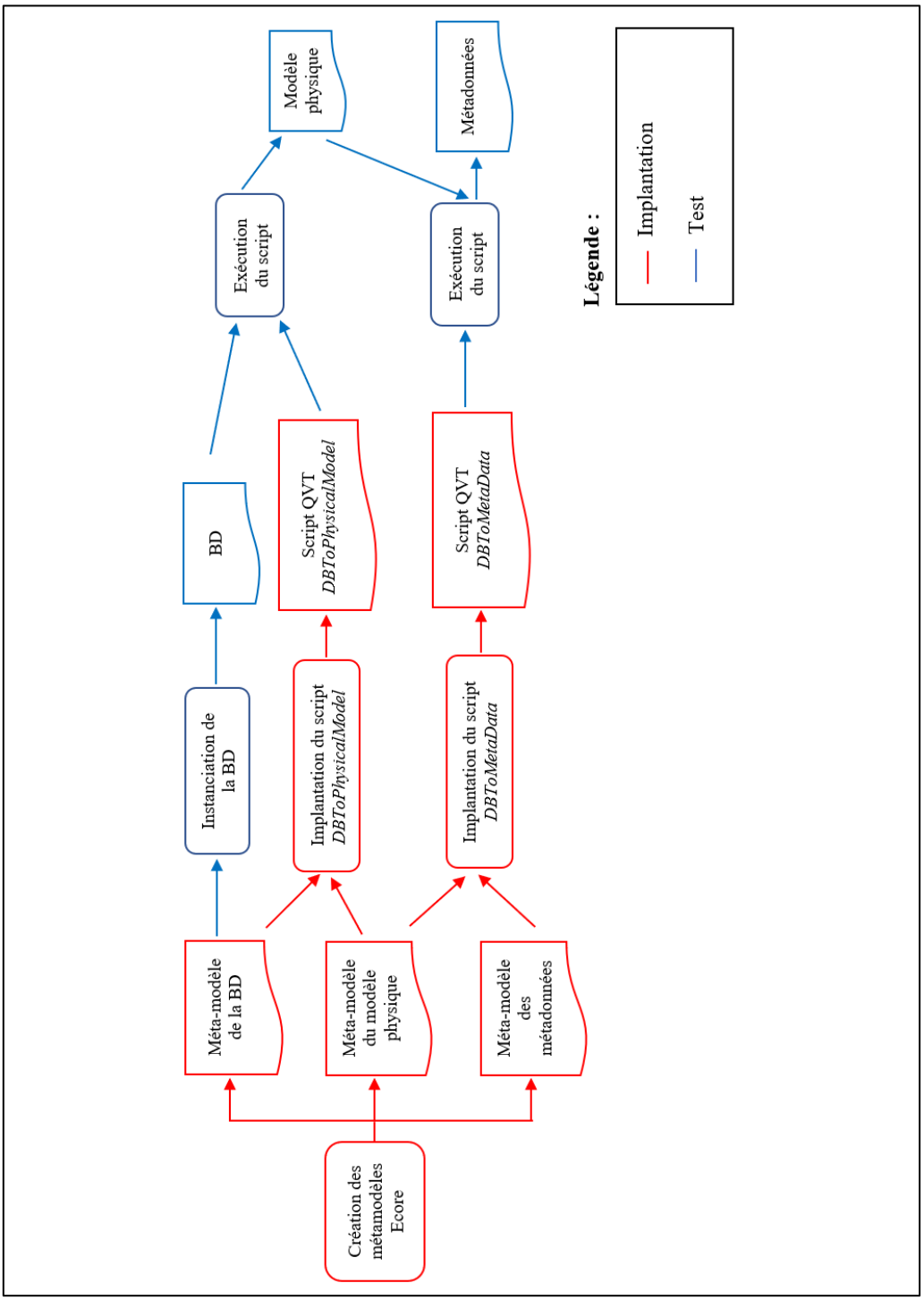



**Figure 6.6 –** Etapes d'implantation du sous-module *ModelExtraction*

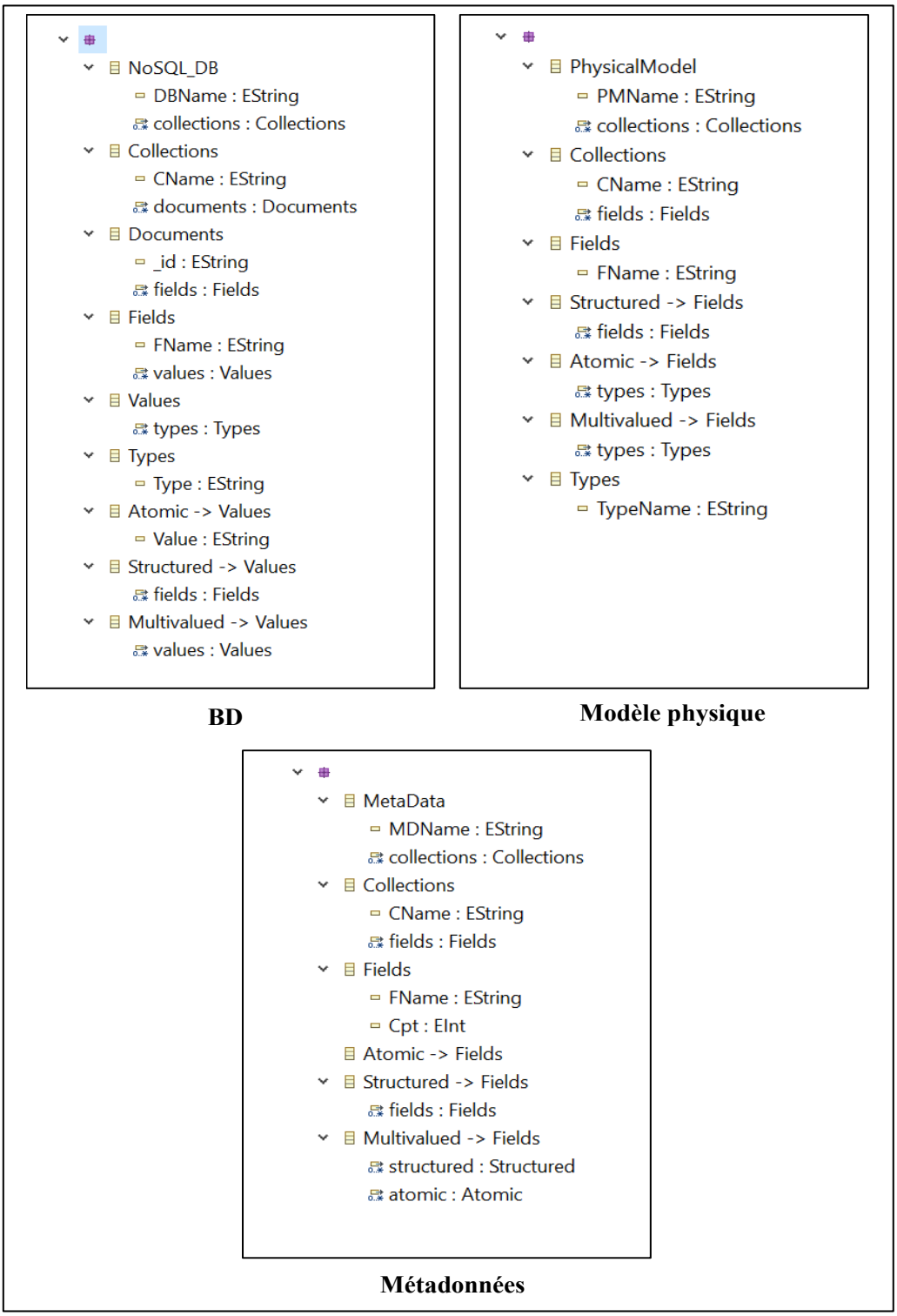



**Figure 6.7 –** Métamodèles Ecore du sous-module *ModelExtraction*

```
modeltype NoSQL_DB uses "http://NoSQLDBMetaModel.com";
modeltype PhysicalModel uses "http://PhysicalModelMetaModel.com";
transformation DBToPhysicalModel (in Source: NoSQL_DB, out Target:
PhysicalModel);
main() {Source.rootObjects()[NoSQL_DB] -> map toPhysicaModel();}
mapping NoSQL_DB:: NoSQL_DB toPhysicaModel ():PhysicalModel:: PhysicalModel {
PMName := self.DBName;
colections := self.collections -> map toCollections();
//Transformation des collections de la BD en collections du modèle physique

mapping NoSQL_DB:: Collections toCollections ():PhysicalModel::Collections {
CName := self.CName;
atomic := self.atomic -> map toAtomicFields ();
structured := self.structured -> map toStructuredFields ();
multivalued := self.multivalued -> map toMultivaluedFields ();
//Transformation des champs atomiques

mapping NoSQL_DB:: Atomic toCollections ():PhysicalModel::Atomic {
FName := self.CName;
FType := self.CValue -> map toTypeFields ();
//Déduction des types de champs

mapping NoSQL_DB:: Value toCollections ():PhysicalModel::Type {
// Type « boolean »

if ((self.Value=”True”) or (self.Value=”False”)
{Type:=”Boolean”}endif ;
// Type « Number »
…
```

**Figure 6.8 –** Script QVT de la transformation *DBToPhysicalModel*

```
modeltype NoSQL_DB uses "http://NoSQLDBMetaModel.com";
modeltype MetaData uses "http://MetaDataMetaModel.com";
transformation DBToMetaData (in Source: NoSQL_DB, out Target: MetaData);
main() {Source.rootObjects()[NoSQL_DB] -> map toMetaData();}
mapping NoSQL_DB:: NoSQL_DB toMetaData ():MetaData:: MetaData {
MDName := self.DBName;
colections := self.collections -> map toCollections();

//Transformation de collections de la BD en collections des métadonnées

mapping NoSQL_DB:: Collections toCollections ():MetaData::Collections {
CName := self.CName;
field := self.field -> map toFields ();
//Calculer le nombre d’apparitions des champs dans la collection

mapping NoSQL_DB:: Fields toCollections ():MetaData::Fields {
FName := self.CName;
Cpt := Cpt+1 ;
…
```

**Figure 6.9 –** Script QVT de la transformation *DBToMetaData*


**Figure 6.10 –** PSM-BD

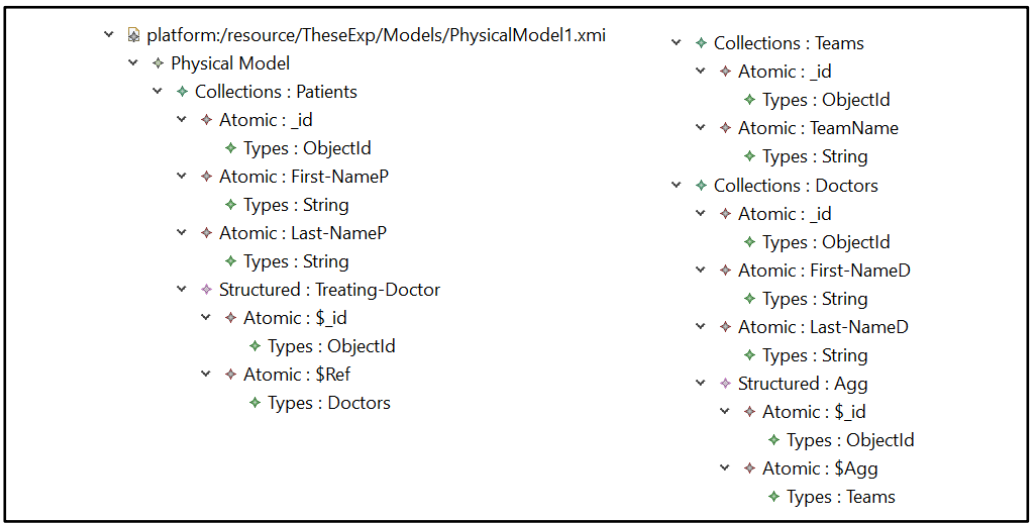



**Figure 6.11 –** PSM-MPD

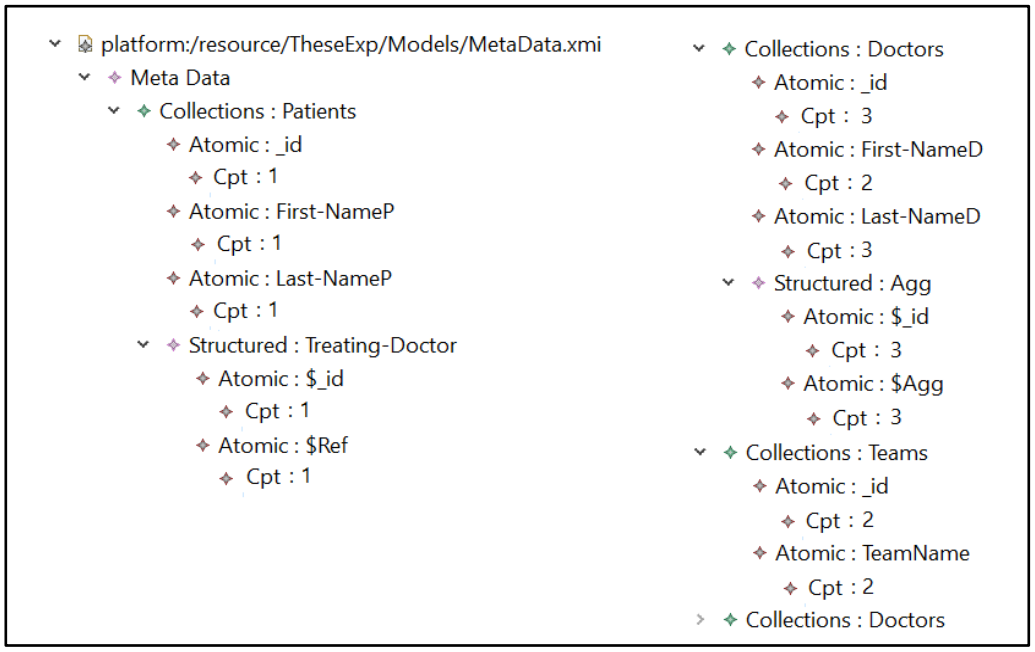



**Figure 6.12 –** PSM-MetaData

#### 6.2.1.2 Sous-module *ModelUpdate* :

**Entrée**

L'entrée du sous-module *ModelUpdate* est constituée de trois éléments : (1) la requête de mise à jour, (2) le modèle physique de données (MPD) et (3) les métadonnées de la BD. Notons que la première version du MPD et des métadonnées de ce sous-module correspondent respectivement au MPD et aux métadonnées résultant du sous-module *ModelExtraction*. La requête de mise à jour est fournie par l'utilisateur en instanciant le métamodèle du PSM-Query que nous avons proposé dans le chapitre 4 (cf. figure 4.20). Rappelons qu'une requête de mise à jour peut être de type adjonction, suppression ou modification.

**Sortie**

La sortie du sous-module *ModelUpdate* est constituée de deux éléments : (1) une version modifiée du MPD et une version modifiée des métadonnées.

**Transformation**

Le sous-module *ModelUpdate* analyse la requête de mise à jour pour déterminer la collection sur laquelle elle porte ainsi que ses attributs. Ensuite, il compare ces éléments avec la version actuelle du MPD. Si celle-ci ne correspond pas aux éléments de la requête, alors il met à jour le MPD. Sinon, il garde la version précédente. *ModelUpdate* procède de la même manière à la mise à jour des métadonnées.

**Implantation**

La figure 6.13 montre les différentes étapes d'implantation du sous-module *ModelUpdate*.

En plus des métamodèles PSM-MPD et PSM-MetaData (utilisés pour l'implantation du sous-module *ModelExtraction*), nous avons créé le métamodèle PSM-Query comme le montre la figure 6.14. PSM-Query décrit les requêtes de mise à jour, de type adjonction modification et suppression, appliquées à la BD. Une description détaillée de ce métamodèle est donnée dans le chapitre 4 (cf. figures 4.20).

Ensuite, nous avons utilisé le langage QVT pour implanter les règles de transformation du sous-module *ModelUpdate*. La figure 6.15 montre un extrait du script QVT correspondant à ces règles.

Finalement, nous avons testé le sous-module en suivant les étapes suivantes :

- **Etape 1 :** Instancier le métamodèle du PSM-Query pour produire une requête de mise à jour. Celle-ci est stockée sous la forme d'un fichier XMI comme le montre la figure 6.16. De plus, reprendre les fichiers XMI contenant le MPD et les métadonnées résultant de l'exécution du sous-module *ModelExtraction* (cf. figures 6.11 et 6.12).

- **Etape 2 :** Exécuter le script QVT du sous-module *ModelUpdate* sur les fichiers XMI de l'étape 1. Le résultat de l'exécution de ce script est une version modifiée du MPD et des métadonnées comme le montrent respectivement les figures 6.17 et 6.18.

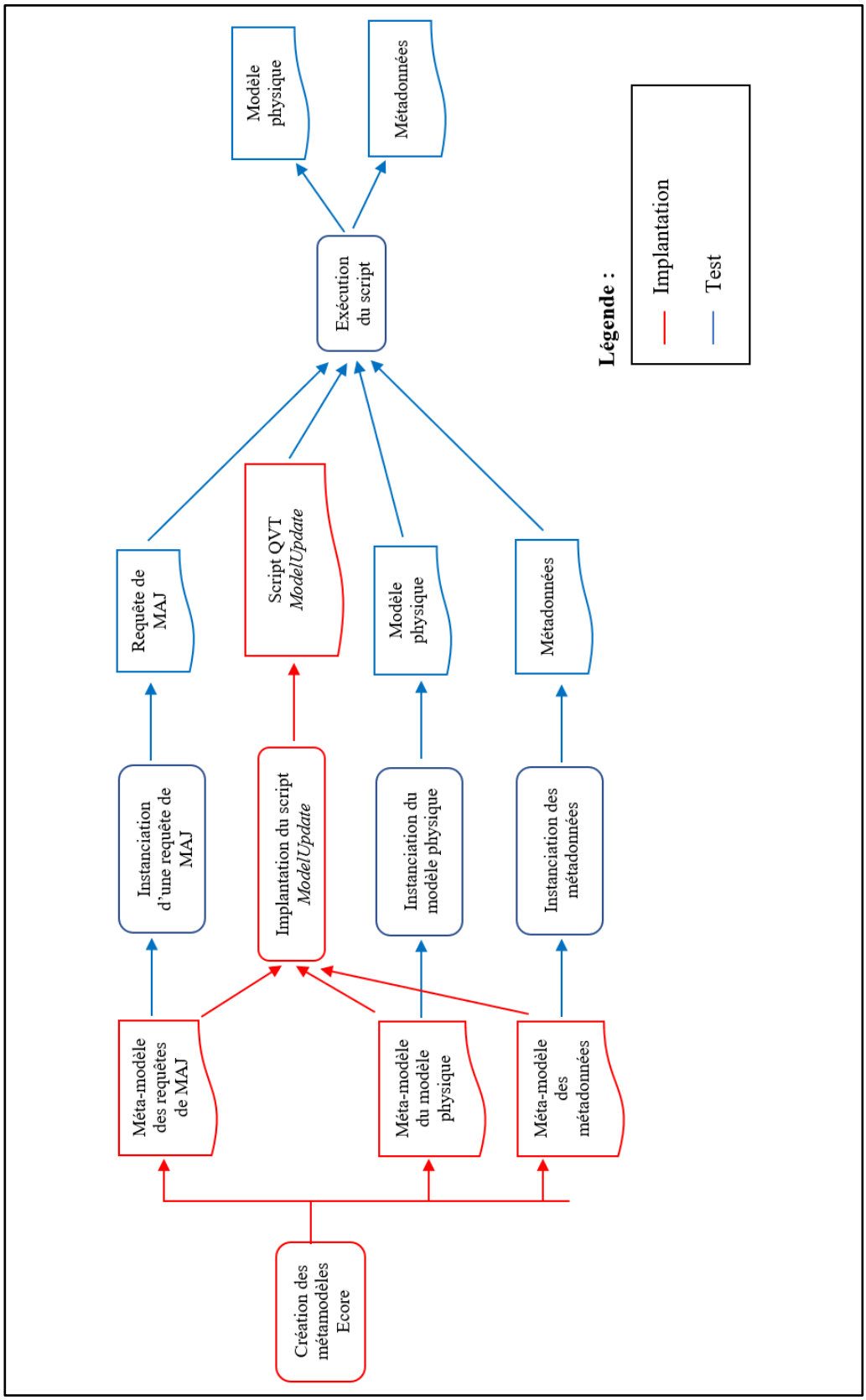



**Figure 6.13 –** Etapes d'implantation du sous-module *ModelUpdate*

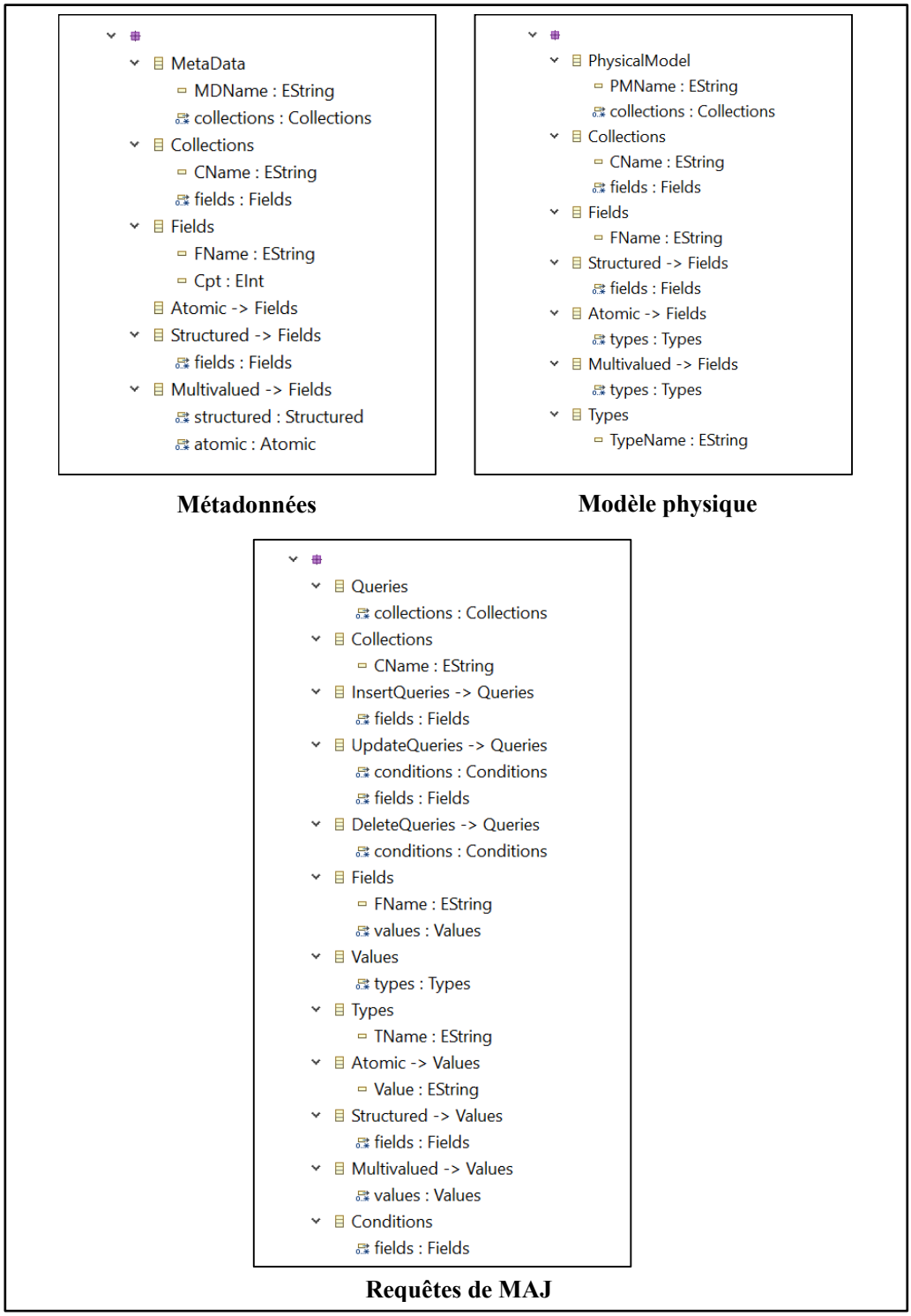



**Figure 6.14 –** Métamodèles Ecore du sous-module *ModelUpdate*

```
modeltype Queries uses "http://QueriesMetaModel.com";
modeltype PhysicalModel uses "http://PhysicalModelMetaModel.com";
transformation DBToPhysicalModel (in Source: NoSQL_DB, out Target:
PhysicalModel);
main() {Source.rootObjects()[NoSQL_DB] -> map toPhysicaModel();}
mapping Queries:: Queries toPhysicaModel ():PhysicalModel:: PhysicalModel {
colections := self.collections -> map toCollections();

//Traitement de la requête de MAJ

mapping Queries:: Query toCollections ():PhysicalModel::Collections {
CName := self.Collections.CName;
Fields := self. Collections.atomic -> map toFields ();

//Cas d'une requête d'adjonction

mapping Queries:: InsertQueries traitementInsertQueries ():PhysicalModel::
PhysicalModel {
Collection.FName := self. Collections.CName;
Collection.FType := self. Collections.CValue -> map toInsertFields ();

//Cas d'une requête de suppression

mapping Queries:: DeleteQueries traitementDeleteQueries ():PhysicalModel::
PhysicalModel {
Collection.FName := self. Collections.CName;
Collection.FType := self. Collections.CValue -> map toDeleteFields ();

//Cas d'une requête de modification

mapping Queries:: UpdateQueries traitementUpdateQueries ():PhysicalModel::
PhysicalModel {
Collection.FName := self. Collections.CName;
Collection.FType := self. Collections.CValue -> map toUpdateFields ();
…
```

**Figure 6.15 –** Script QVT du sous-module *ModelUpdate*

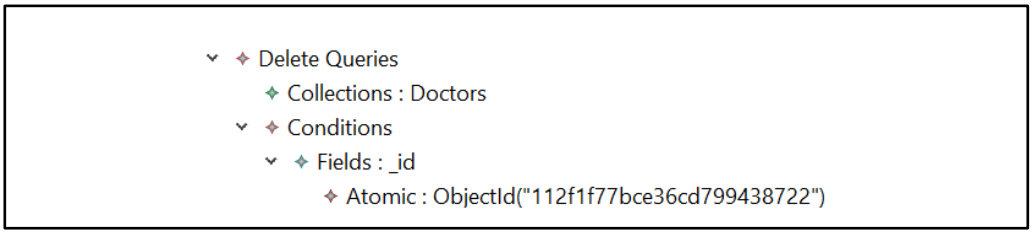



**Figure 6.16 –** PSM-Query

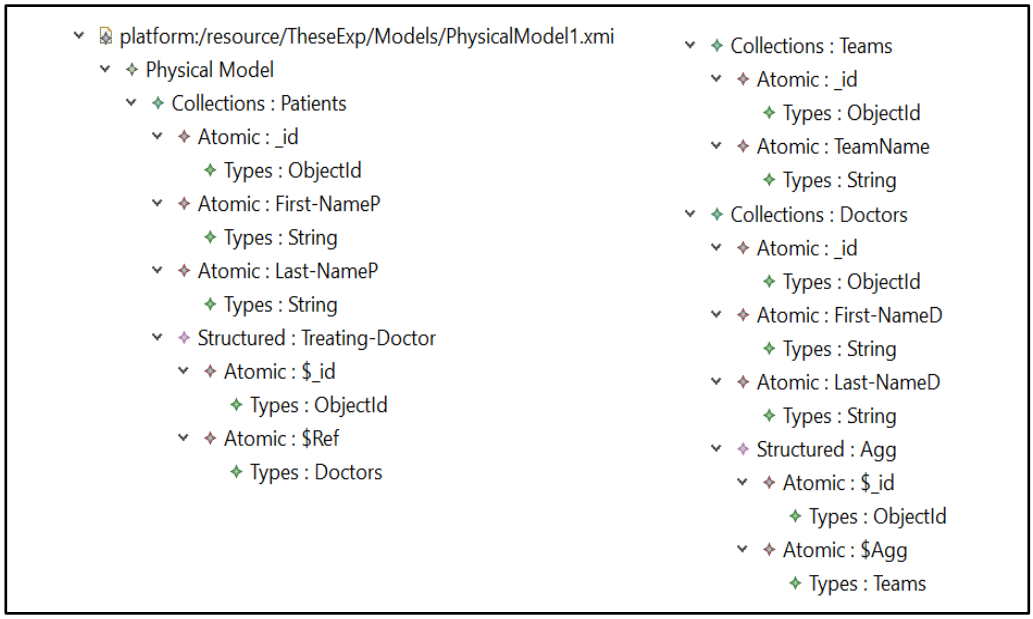



**Figure 6.17 –** Version évoluée du PSM-MPD

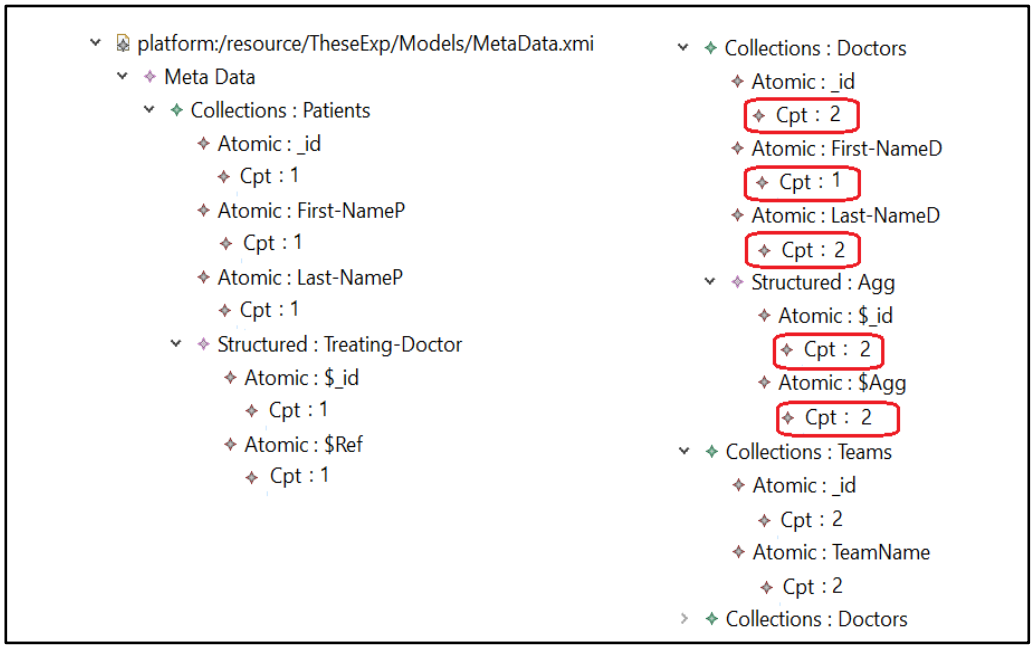



**Figure 6.18 –** Version évoluée du PSM-MetaData

## 6.2.2 Module *ToConceptualModel*

Ce module a pour objectif d'automatiser la transformation du PSM-MPD (déjà extrait par le module *ToPhysicalModel*) en un modèle conceptuel PIM-MCD. Celui-ci est exprimé en utilisant le formalisme UML. Le passage d'un modèle à

un autre se fait via des transformations de type M2M formalisées en QVT. La figure 6.19 montre les composants du module *ToConceptualModel*.

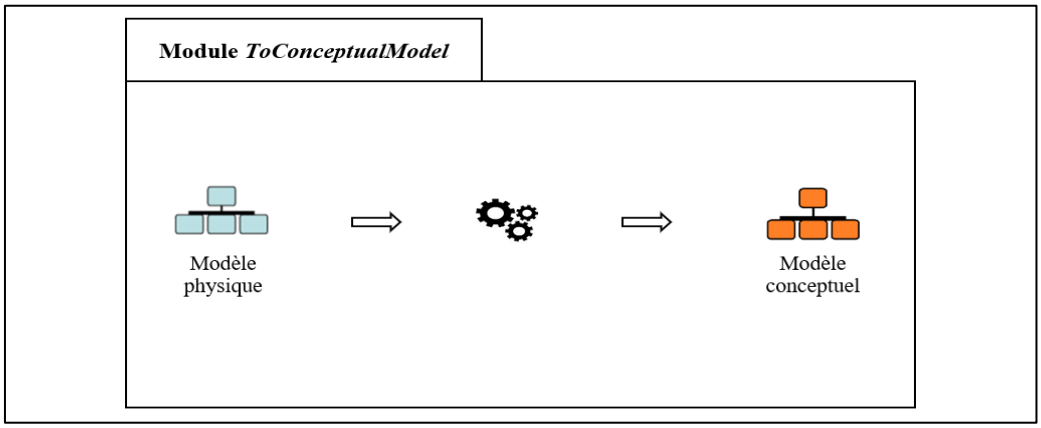



**Figure 6.19 –** Module *ToConceptualModel*

**Entrée**

L'entrée du module *ToConceptualModel* correspond à la sortie du module *ToPhysicalModel* présenté dans la section 6.2.1. Autrement dit, l'entrée du module *ToConceptualModel* est le modèle physique (MPD) d'une BD MongoDB (cf. figures 6.17).

**Sortie**

La sortie du module *ToConceptualModel* correspond au modèle conceptuel de données (PIM-MCD). Celui-ci est exprimé en utilisant le formalisme UML. Le PIM-MCD a pour objectif de décrire les données de la BD selon une vision sémantique et sans mentionner les techniques de stockage. Il fait apparaitre les classes ainsi que les liens entre elles.

**Transformation**

Le module *ToConceptualModel* applique des règles de transformation pour assurer le passage entre le modèle PSM-MPD et le PIM-MCD. Celui-ci est conforme au métamodèle que nous avons proposé dans le chapitre 5 (cf. figure 5.3). Les règles de transformations sont réalisées par un ensemble de règles M2M formalisées en QVT.

**Implantation**

La figure 6.20 montre les différentes étapes d'implantation du module *ToConceptualModel*. Cette implantation nécessite la définition préalable de deux métamodèles : PSM-MPD et PSM-MCD ainsi qu'un ensemble de règles de transformation de type M2M.

Le métamodèle PSM-MPD est déjà créé lors de l'implantation du module *ToPhysicalModel.* Ainsi, nous allons créer uniquement le métamodèle PIM-

MCD comme le montre la figure 6.21. Une description détaillée de ce métamodèle est donnée dans le chapitre 5 (cf. figures 5.3).

Ensuite, nous avons utilisé le langage QVT pour implanter les règles de transformation du module *ToConceptualModel*. La figure 6.22 montre un extrait du script QVT correspondant à ces règles.

Finalement, nous avons testé le module en suivant les étapes suivantes :

- **Etape 1 :** Reprendre le fichiers XMI contenant le PSM-MPD résultant de l'exécution du module *ToPhysicalModel* (cf. figures 6.23).
- **Etape 2 :** Exécuter le script QVT de la figure 6.22 sur le fichier XMI de l'étape 1. Le résultat de l'exécution de ce script est un fichier XMI contenant le PIM-MCD comme le montre la figure 6.24.

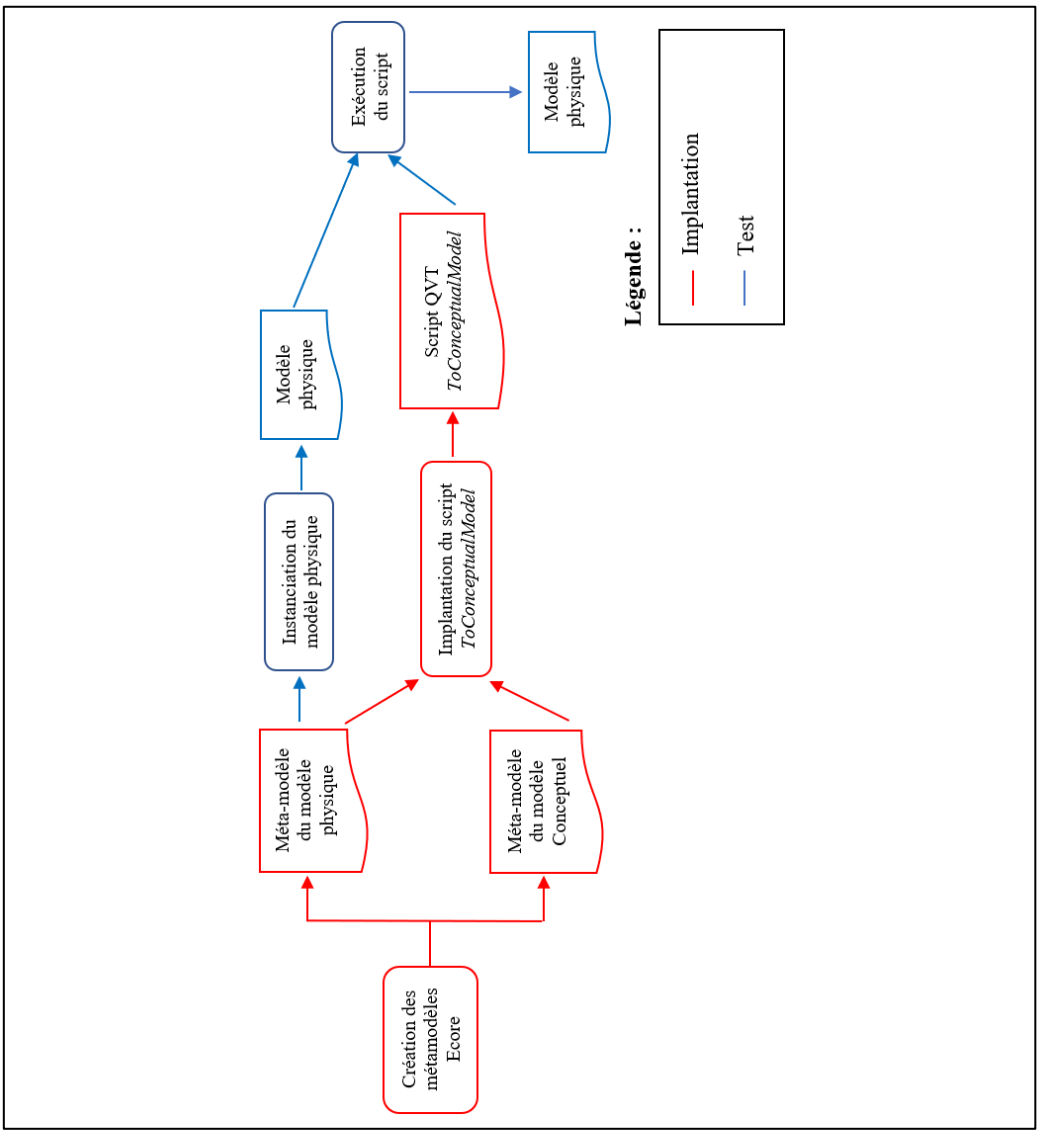



**Figure 6.20 –** Etapes d'implantation du module *ToConceptualModel*

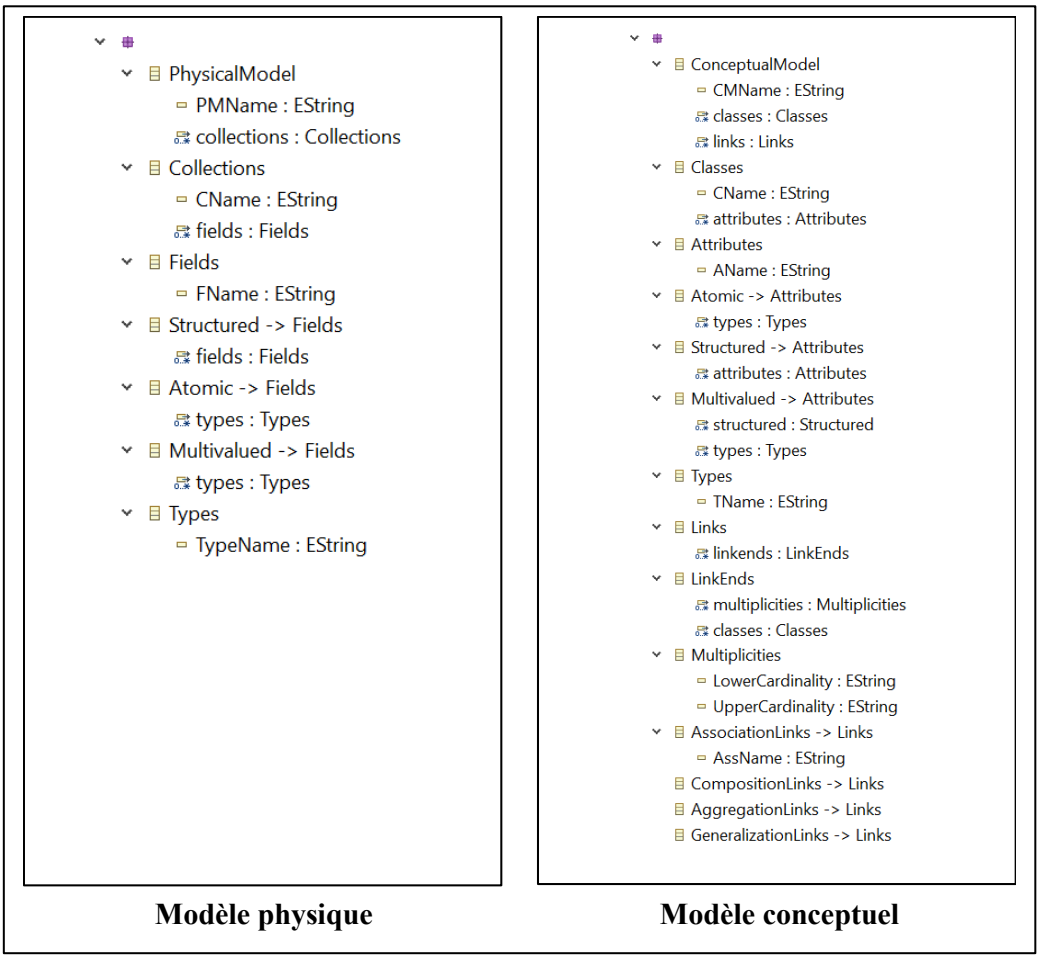



**Figure 6.21 –** Métamodèles Ecore du module *ToConceptualModel*

```
modeltype PhysicalModel uses "http:// PhysicalModelMetaModel.com";
modeltype ConceptualModel uses "http://ConceptualModelMetaModel.com";
transformation ToConceptualModel (in Source: PhysicalModel, out Target:
ConceptualModel);
main() {Source.rootObjects()[PhysicalModel] -> map ToConceptualModel();}
mapping PhysicalModel:: PhysicalModel toConceptModel ():ConceptualModel::
ConceptualModel {
CMName := self.PMName;
classes := self.collections -> map toClasses();
//Transformation des collections du modèle physique en classes du modèle conceptuel
mapping PhysicalModel:: Collections toCollections ():ConceptualModel::Classes {
CName := self.CName;
attributes := self.fields -> map toAttributes ();
//Transformation des champs atomiques en attributs atomiques
mapping PhysicalModel:: AtomicField toAttributes ():ConceptualModel::AtomicAttrubute
{
AName := self.FName;
AType := self.FType;
//Transformation des champs structurés en attributs structurés
…
```

**Figure 6.22 –** Script QVT du module *ToConceptualModel*

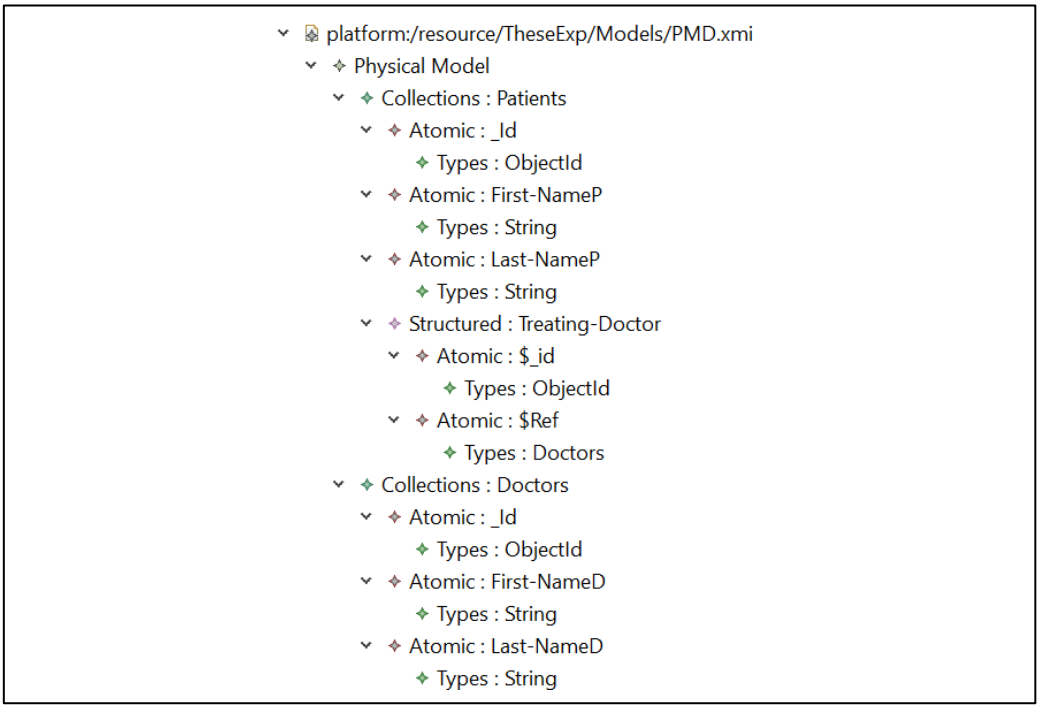



**Figure 6.23** – PSM-MPD

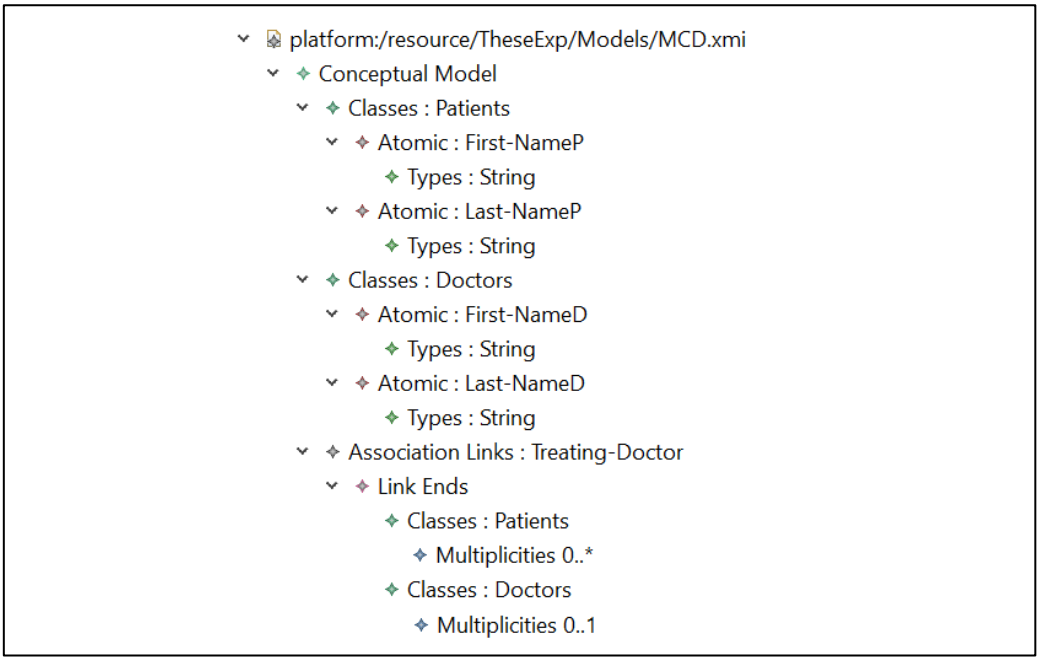



**Figure 6.24** – PIM-MCD

## 6.3 Validation

L'expérimentation présentée dans la section précédente a porté sur notre application médicale présentée dans le chapitre 2 (cf. section 2.4). Elle a permis de montrer la faisabilité de nos propositions. Ainsi, grâce au développement logiciel de notre processus de transformations, des modèles physique et conceptuel peuvent être automatiquement extraits à partir d'une BD NoSQL MongoDB. A présent, il convient d'évaluer l'apport de notre solution dans une démarche professionnelle.

L'objectif de nos travaux est bien de simplifier certaines tâches dans l'exploitation des BD NoSQL de type schema-less ; plus précisément, il s'agit de réduire les ressources nécessaires (principalement temporelles) à la compréhension de la sémantique des données et à l'écriture des requêtes.

Travaillant en liaison avec une entreprise de développement d'applications dans les domaines de la Business Intelligence et du Big Data, la société TRIMANE[26], située dans la région parisienne, nous avons pu obtenir la mise en œuvre de notre logiciel sur trois applications réelles (pour des raisons de confidentialité, les noms des fichiers et des objets manipulés ont été changés). Ceci nous a permis de réaliser la validation de nos propositions. La première application « Juris-Dépôt » traite de la mise à disposition des décisions des tribunaux. La seconde « Disponibilités Des Agents » concerne l'étude de la disponibilité d'agents d'intervention. Enfin, la troisième « GMAO » traite de la gestion de la maintenance assistée par ordinateur (GMAO) pour un groupe pharmaceutique privé.

Il s'agit de trois applications décisionnelles développées par TRIMANE ; elles manipulent du texte et des photos et présentent des volumes de données importants (plusieurs téraoctets). Ces caractéristiques ont justifié le choix d'une implantation des données sur des systèmes NoSQL.

Afin d'évaluer la pertinence de l'approche, notre prototype a été mis en œuvre par dix développeurs de Trimane. Bien que nous ayons mis en œuvre cette expérimentation dans un environnement professionnel sur des données réelles, les opérations présentées ci-dessous ont été simulées pour nous permettre d'obtenir des éléments de comparaisons fiables.

Les dix développeurs expérimentés (ingénieurs-consultants en informatique) ont été chargés d'assurer des opérations de maintenance pour les trois applications.

Aucun des développeurs ne connaissait préalablement les données manipulées par les applications. Chacun d'eux a écrit dix requêtes de complexité croissante sur chacune des applications, mais dans trois situations différentes : (1) sans modèle de données, (2) avec le modèle physique ou bien (3) avec les modèles conceptuel et physique des données. Les figures 6.25 et 6.26 montrent un exemple de modèles conceptuel et physique de l'application « Juris-Dépôt » ;

[26] http://www.trimane.fr/

pour notre évaluation, les modèles fournis aux développeurs correspondent à des vues d'une dizaine de classes, ceci pour permettre une comparaison des résultats.

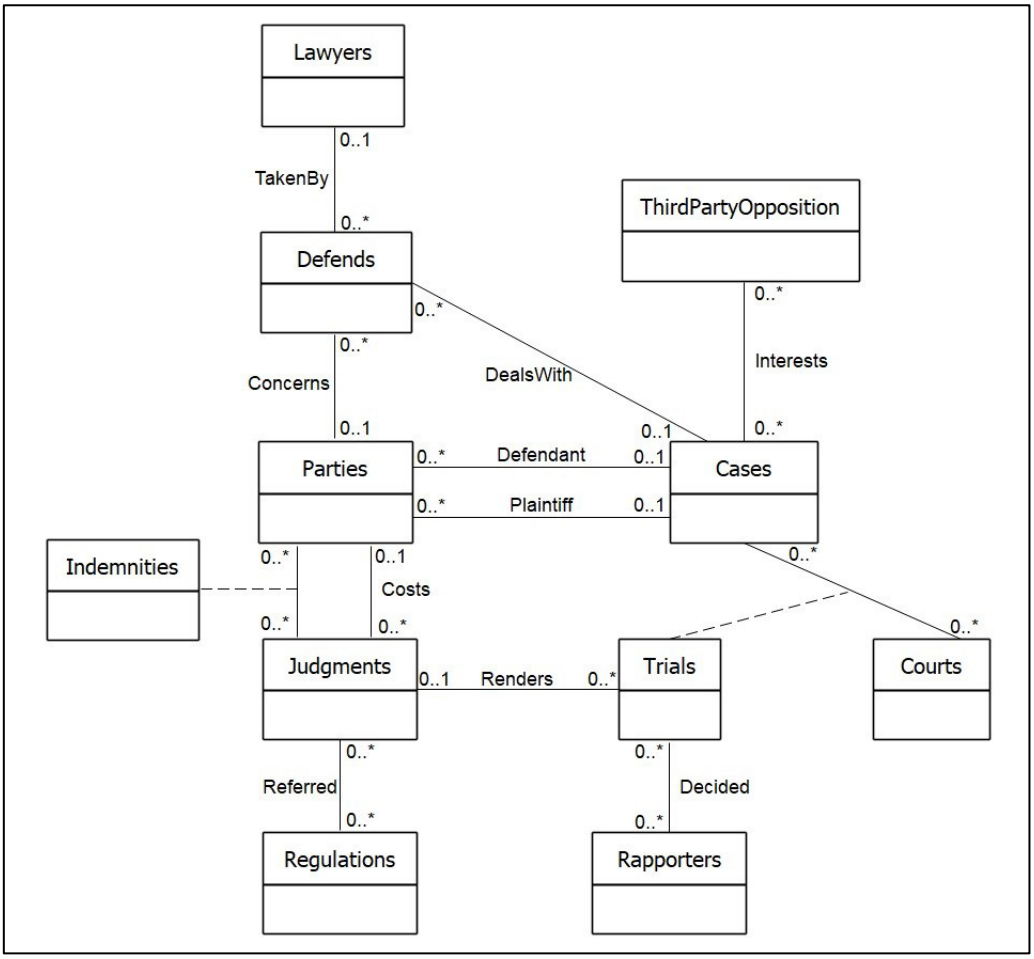



**Figure 6.25 –** Le modèle conceptuel de la BD de l'application « Juris-Dépôt »

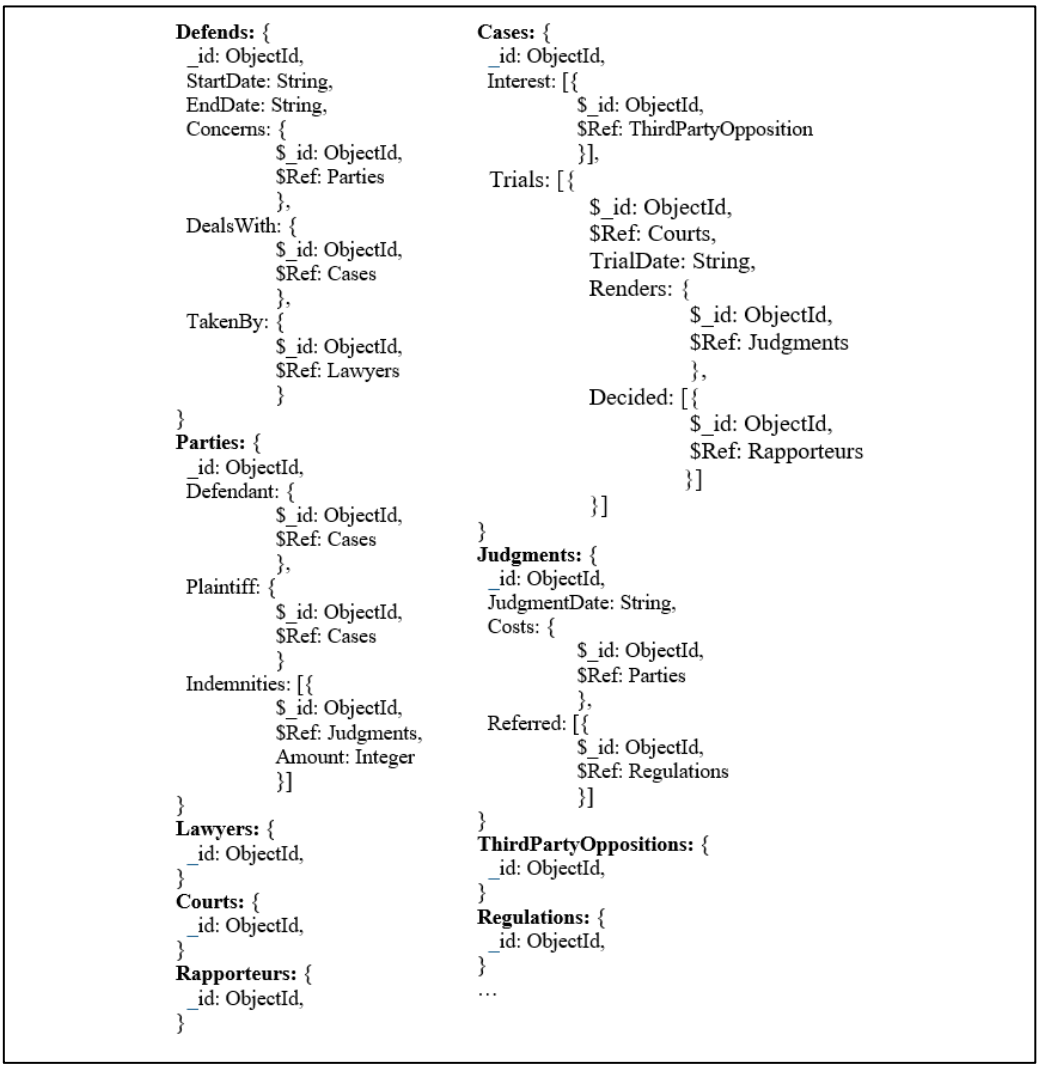



**Figure 6.26 –** Le modèle physique de la BD de l'application « Juris-Dépôt »

Chaque BD est associée à un jeu unique de requêtes dont les énoncés en langage naturel sont fournis aux dix développeurs. L'affectation a été faite comme le montre la figure 6.27 :

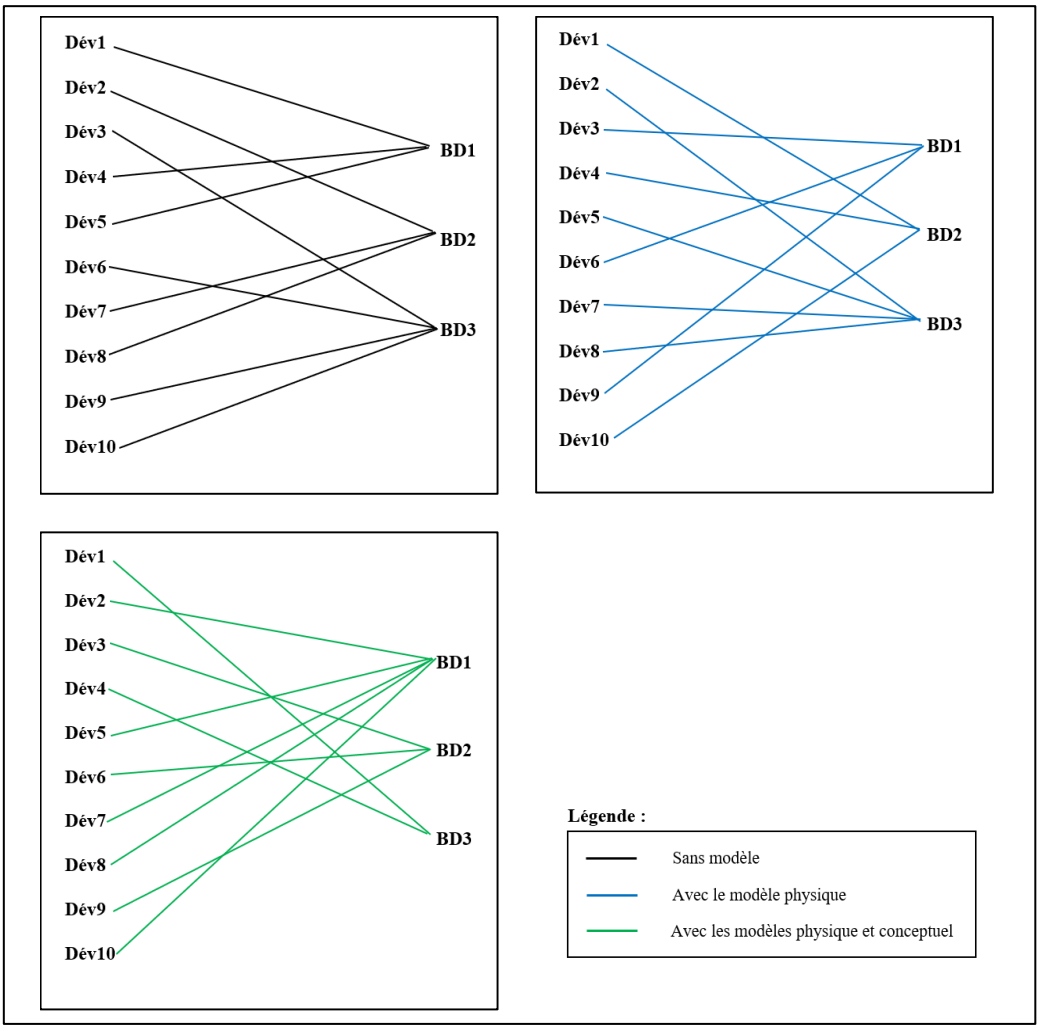



**Figure 6.27 –** Affectation des développeurs sur les applications

Ainsi, le développeur n°1 (Dév1) a écrit trois jeux de 10 requêtes :

- Un jeu appliqué à la base de données de l'application « Juris-Dépôt » (BD1) ; Dév1 n'ayant à sa disposition aucun modèle de données, il a donc basé l'écriture des nouvelles requêtes sur les requêtes déjà écrites,
- Un jeu de requêtes sur la BD de l'application « Disponibilités Des Agents » (BD2) en s'appuyant sur le seul modèle physique de cette BD obtenu grâce à notre prototype
- Un jeu de requêtes sur la BD « GMAO » (BD3) en utilisant le modèle conceptuel et le modèle physique produits par notre prototype.

Nous avons calculé la moyenne des temps de rédaction des requêtes obtenus par les dix développeurs dans chaque situation : sans modèle, avec un modèle ou avec deux modèles comme le montre la tableau 6.1.

| | Sans modèle | Modèle physique seul | Modèles conceptuel et physique |
|---|---|---|---|
| **Dév1** | BD1 : 50mn | BD2 : 23 mn | BD3 : 18mn |
| **Dév2** | BD2 : 40mn | BD3 : 25mn | BD1 : 16mn |
| **Dév3** | BD3 : 48mn | BD1 : 20mn | BD2 : 16mn |
| **Dév4** | BD1: 45mn | BD2 : 22mn | BD3 : 15mn |
| **Dév5** | BD2 : 43mn | BD3 : 24mn | BD1 : 18mn |
| **Dév6** | BD 3: 48mn | BD1 : 21mn | BD2 : 14mn |
| **Dév7** | BD2 : 46mn | BD3 :22 mn | BD1 : 16mn |
| **Dév8** | BD2: 51mn | BD3 : 24mn | BD1 : 14mn |
| **Dév9** | BD3 : 49mn | BD1 : 25mn | BD2 : 16mn |
| **Dév10** | BD3 : 47mn | BD2 :22 mn | BD1 : 16mn |
| Moyenne | 46,7 mn | 22,8 mn | 15,9 mn |

**Tableau 6.1 –** Temps de rédaction des requêtes sur les BD des applications

Malgré un nombre de cas limité, notre hypothèse de départ a été vérifiée dans les situations envisagées. Ceci établit qu'une connaissance de la sémantique et de la structure des données permet au développeur de rédiger plus rapidement les requêtes sur une BD NoSQL « schema less ».

La faible différence constatée entre l'usage du seul modèle physique et l'usage des deux modèles (conceptuel et physique), est probablement due à l'expérience des dix développeurs.

# 6.4 Conclusion

Dans ce chapitre, nous avons décrit la réalisation de notre prototype d'extraction des modèles physique et conceptuel à partir d'une BD MongoDB. Ce prototype a permis de valider l'intérêt de nos travaux. Il est composé de deux modules :

- Module d'extraction du modèle physique : *ToPhysicalModel*,
- Module de génération du modèle conceptuel : *ToConceptuelModel*.

Le premier est chargé de générer un modèle physique de données (MPD) à partir d'une DB NoSQL orientée-documents et le mettre à jour au fur et à mesure de l'exécution des requêtes de mise à jour sur la BD. Le MPD résultant décrit l'organisation interne des données de la base et permet d'exprimer des requêtes. Le second module a pour objectif de transformer le MPD en un modèle

conceptuel de données (MCD). Celui-ci fait abstraction des aspects techniques et se concentre sur la sémantique de données.

Pour chaque module, nous avons présenté : l'entrée, la sortie, les transformations et les différentes étapes de son implantation.

Pour tester le prototype, nous avons mené des expériences qui portent sur l'extraction des modèles physique et conceptuel de l'application médicale (présentée au chapitre 2). Pour ceci, nous avons instancié les métamodèles proposés dans les chapitres précédents pour générer des exemples d'instances (une BD MongoDB, un MPD, des métadonnées, des requêtes de mise à jour ainsi qu'un MCD).

Ce chapitre montre également la pertinence de nos propositions à travers une évaluation. Celle-ci a été effectuée dans le cadre de projets développés par la société TRIMANE. L'objectif est de comparer le temps de rédaction de requêtes sur une BD MongoDB avec ou sans la présence des modèles physique et conceptuel.

# Chapitre 7 : Conclusion générale

## 7.1 Synthèse de nos travaux

Les travaux présentés dans ce mémoire de thèse se situent dans le contexte des données massives (Big Data). Ces données sont caractérisées par la règle dite des (3V). Il s'agit du Volume (des masses des données considérables à gérer), de la Variété (des données complexes) et de la Vélocité (en référence à la collecte et au traitement en temps-réel de ces données). Les approches classiques basées principalement sur le paradigme relationnel, ne peuvent pas répondre à ces caractéristiques. Ainsi, sont apparus de nouveaux systèmes de gestion des données : les systèmes NoSQL. Ceux-ci sont plus adaptés que les systèmes relationnels pour gérer de gros volumes de données présentant des types et des formats variés.

La majorité des BD gérées par des SGBD NoSQL sont connues par la propriété schema-less. Celle-ci signifie que le modèle de données de la BD n'est pas défini à priori. Cette caractéristique apporte une souplesse indéniable qui facilite l'évolution du modèle de données et permet aux utilisateurs d'ajouter de nouvelles informations sans avoir recours à l'administrateur de BD. En contrepartie, la manipulation de ce type de BD, plus précisément l'expression des requêtes, est devenue complexe. En effet, dans un contexte Big Data, les besoins des utilisateurs évoluent d'une manière exponentielle. Cette évolution nécessite donc une connaissance des modèles pour connaitre la manière dont les données sont stockées et reliées dans la BD. Par exemple, pour écrire des requêtes, l'utilisateur doit pouvoir visualiser la structure de la BD où sont mentionnés les noms des tables, les noms des attributs ainsi que leurs types.

Dans ce mémoire, nous nous sommes intéressés à la génération de deux modèles de données pour la manipulation des SGBD NoSQL de type schema-less. Il s'agit du modèle physique de données (MPD) qui décrit l'organisation interne des données et permet d'exprimer des requêtes ; et du modèle conceptuel de données (MCD) qui fait abstraction des aspects techniques et favorise la compréhension des données.

Pour ce faire, nous proposons d'adopter une approche dirigée par les modèles composée de deux processus : *ToPhysicalModel* et *ToConceptuelaModel*. Ces processus se résument comme suit :

- Le processus *ToPhysicalModel* produit et met à jour le MPD d'une BD NoSQL schema-less. Il est composé de deux sous-processus qui s'appliquent successivement : *ModelExtraction* et *ModelUpdate*. Le sous-processus *ModelExtraction* permet de générer un MPD NoSQL à partir d'une BD de type orienté-documents. Il s'agit du sous-processus « à froid ». Son principe consiste à balayer la totalité de la BD à un instant *t* et à fournir une structure interne conforme à l'organisation des données à l'instant *t*. Le sous-processus *ModelUpdate* recalcule le MPD au fur et à mesure de l'exécution des requêtes de mise à jour sur la BD. Il s'agit du sous-processus « à chaud ». Son principe consiste à utiliser des métadonnées pour le traitement des requêtes de mise à

jour. Selon l'architecture MDA, les modèles utilisés dans les deux sous-processus sont conformes à des métamodèles. Pour assurer le passage automatique entre les modèles sources et cibles, nous avons défini des règles de transformations de type Model-To-Model (M2M) formalisées avec le standard QVT.

- Le processus *ToConceptualModel* consiste à transformer le modèle physique déjà extrait par le processus *ToPhysicalModel* en un modèle conceptuel. Celui-ci est exprimé en utilisant le formalisme UML. Le passage entre le modèle physique et conceptuel est assuré par la définition d'un ensemble de règles de transformation de type Model-To-Model (M2M) formalisées avec le standard QVT.

Nous pouvons résumer les avantages de notre comme suit :

- L'utilisation de MDA offre un cadre formel aux mécanismes de transformation des modèles,
- La formalisation de la sortie de notre approche repose sur le standard UML de l'OMG,
- Les modèles nécessaires pour la manipulation des BD NoSQL schema-less sont les modèles physique et conceptuel ; notre solution prend en compte cette dualité,
- L'utilisation du principe MDA de méta-modélisation permet de vérifier la conformité d'un modèle par rapport à un référentiel ;
- Les règles de transformation sont basées sur la norme QVT de l'OMG.

Afin de vérifier la faisabilité de notre solution, nous avons développé un prototype en utilisant la plateforme EMF (Eclipse Modeling Framework) [Budinsky et al., 2004]. Cette plateforme d'implantation présente un environnement complet et adapté à la méta-modélisation, la modélisation et la transformation des modèles. Elle dispose de l'ensemble d'outils nécessaires à la mise en œuvre de notre solution. Nous avons ensuite effectué des expérimentations sur l'extraction des modèles physique et conceptuel d'une application médicale afin de tester le fonctionnement des modules composant notre prototype.

En outre, notre solution a été validée dans un cadre professionnel par des ingénieurs de la Société TRIMANE spécialisée en Big Data et en décisionnel, située à Saint Germain en Laye.

## 7.2 Perspectives

Les perspectives de nos travaux portent principalement sur trois points.

Le premier concerne l'enrichissement du processus d'extraction du modèle physique de données (MPD). En effet, Au cours de l'implantation de la BD MongoDB d'entrée, un ensemble d'hypothèses ont été imposées aux développeurs. Ces hypothèses concernent l'expression des liens entre collections. Comme MongoDB ne permet pas d'exprimer des liens d'héritage, de composition et d'agrégation entre collections, nous avons proposé des modes d'implantation pour simuler l'utilisation de ces liens. Pour cela, nous avons suggéré trois syntaxes : DBSub, DBComp et DBAgg en s'appuyant sur le principe de DBRef préconisé dans la documentation de MongoDB [MongoDB, 2018]. Il serait donc intéressant de rendre la détection des liens automatique plutôt que d'imposer aux développeurs des hypothèses sans lesquelles le processus ne fonctionnera pas correctement. Pour cela, nous prévoyons d'utiliser des techniques d'apprentissage automatique.

Le deuxième point vise à enrichir sémantiquement le modèle conceptuel UML généré par notre approche. Actuellement, ce modèle est limité aux liens de base qui sont fréquemment utilisés. Nous souhaitons donc proposer des extensions qui portent sur ces liens entre classes. En effet, dans son état actuel, notre approche considère uniquement les types de liens suivants : l'association, l'agrégation, la composition et l'héritage. Il serait alors possible de prendre en compte d'autres types de liens, notamment le lien de dépendance [OMG, 2017]. Ceci nécessite typiquement (1) l'extension de la source et de la cible des deux processus constituant notre approche en formalisant les concepts liés aux nouveaux éléments que nous souhaitons ajouter, puis (2) la mise à jour de l'ensemble des métamodèles et des règles de transformation utilisés.

La troisième perspective concerne la généralisation de notre approche à tous les types de SGBD NoSQL. En effet, à la suite de la justification donnée dans le chapitre 2 (cf. Section 2.2.4), nous avons limité notre étude aux BD gérées par des systèmes NoSQL de type orientés documents. Il serait donc judicieux d'étendre notre approche pour prendre aussi en considération les BD gérées par des systèmes NoSQL orientés clé-valeur, orientés colonnes et orientés graphes. Pour cela, nous envisageons d'introduire un niveau intermédiaire entre les niveaux physique et conceptuel, où nous faisons apparaître un modèle logique générique qui : (1) est compatible avec les quatre familles de SGBD NoSQL (colonnes, documents, graphes et clé-valeur) et (2) fait abstraction des caractéristiques techniques propres aux SGBD NoSQL. L'intérêt d'introduire un niveau logique entre les niveaux physique et conceptuel est de garantir l'indépendance de la description des données vis-à-vis des spécificités techniques des plateformes NoSQL et de leurs évolutions.

# Bibliographie

Abadi, D. J., Madden, S. R., & Hachem, N. (2008). Column-stores vs. row-stores: How different are they really? In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 967-980). ACM.

Abelló, A. (2015, October). Big data design. In Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP (pp. 35-38). ACM.

Ait Brahim, A. (2018). Approche dirigée par les modèles pour l'implantation de bases de données massives sur des SGBD NoSQL (Doctoral dissertation).

Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2018, October). Extraction du schéma d'une BD NoSQL orientée documents. In 14ème Journées Francophones sur les Entrepôts de Données et l'Analyse en ligne-EDA 2018 (pp. 313-320). Editions RNTI.

Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2019a). MDA process to extract the data model from a document-oriented NoSQL database.

Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2019b). Incremental extraction of a NoSQL database model using an MDA-based process.

Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2019c). Model Driven Extraction of NoSQL Databases Schema: Case of MongoDB. In KDIR (pp. 145-154).

Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2019d). Extraction process of conceptual model from a document-oriented NoSQL database. In 2019 11th International Conference on Knowledge and Systems Engineering (KSE) (pp. 1-5). IEEE.

Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2020, February). Approche dirigée par les modèles pour l'extraction automatique du modèle NoSQL. In Business Intelligence & Big Data: 15ème Edition de la conférence EDA, Montpellier France 2019. BoD-Books on Demand.

Abdelhédi, F., Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2020a). Discovering of a Conceptual Model from a NoSQL Database. In ICEIS (1) (pp. 61-72).

Abdelhédi, F., Ait Brahim, A., Tighilt Ferhat, R., & Zurfluh, G. (2020b). Reverse Engineering Approach for NoSQL Databases.

In International Conference on Big Data Analytics and Knowledge Discovery (pp. 60-69). Springer, Cham.

Angadi, A. B., Angadi, A. B., & Gull, K. C. (2013). Growth of New Databases & Analysis of NOSQL Datastores. International Journal of Advanced Research in Computer Science and Software Engineering, 3, 1307-1319.

[ANSI/SPARC, 1975] "ANSI/X3/SPARC Study Group on Data Base Management Systems, 'Interim Report,'", FDT (ACM SIGMOD bulletin), Vol. 7, No. 2, 1975.

Arora, R., & Aggarwal, R. R. (2013). Modeling and querying data in mongodb. International Journal of Scientific and Engineering Research, 4(7), 141-144.

Baazizi, M.-A., Lahmar, H. B., Colazzo, D., Ghelli, G., & Sartiani, C. (2017). Schema inference for massive JSON datasets.

Baazizi, M.-A., Colazzo, D., Ghelli, G., & Sartiani, C. (2019). Parametric schema inference for massive JSON datasets. The VLDB Journal, 28(4), 497–521.

Bézivin, J., & Gerbé, O. (2001, November). Towards a precise definition of the OMG/MDA framework. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)* (pp. 273-280). IEEE.

Bondiombouy, C. (2015, September). Query processing in cloud multistore systems. In BDA: Gestion de Données—Principes, Technologies et Applications.

Budinsky, F., Ellersick, R., Steinberg, D., Grose, T. J., & Merks, E. (2004). Eclipse modeling framework: a developer's guide. Addison-Wesley Professional.

Combemale, B. (2008). Ingénierie Dirigée par les Modèles (IDM)--État de l'art.

Comyn-Wattiau, I., & Akoka, J. (2017). Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j. 2017 IEEE International Conference on Big Data (Big Data), 453–458.

Daniel, G., Sunyé, G., & Cabot, J. (2016, November). UMLtoGraphDB: mapping conceptual schemas to graph databases. In International Conference on Conceptual Modeling (pp. 430-444). Springer, Cham.

DB-Engines Ranking. (n.d.). DB-Engines. Retrieved December 10, 2020, from https://db-engines.com/en/ranking.

Demchenko, Y., Ngo, C., & Membrey, P. (2013). Architecture framework and components for the big data ecosystem. Journal of System and Network Engineering, 1-31.

Diaw, S., Lbath, R., & Coulette, B. (2010). Etat de l'art sur le développement logiciel basé sur les transformations de modèles.

Douglas, L., 2001. 3d data management: Controlling data volume, velocity and variety. Gartner. Retrieved, 6, 2001.

Gallinucci, E., Golfarelli, M., & Rizzi, S. (2018). Schema profiling of document-oriented databases. Information Systems, 75, 13–25.

Gantz, J., & Reinsel, D. (2011). Extracting value from chaos. IDC iview, 1142(2011), 1-12.

Han, J., Haihong, E., Le, G., & Du, J. (2011, October). Survey on NoSQL database. In Pervasive computing and applications (ICPCA), 2011 6th international conference on (pp. 363-366). IEEE.

Herrero, V., Abelló, A., & Romero, O. (2016, November). NOSQL design for analytical workloads: variability matters. In International Conference on Conceptual Modeling (pp. 50-64). Springer, Cham.

Hutchinson, J., Rouncefield, M., & Whittle, J. (2011, May). Model-driven engineering practices in industry. In Proceedings of the 33rd International Conference on Software Engineering (pp. 633-642). ACM.

Izquierdo, J. L. C., & Cabot, J. (2016). JSONDiscoverer: Visualizing the schema lurking behind JSON documents. Knowledge-Based Systems, 103, 52–55.

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

Klettke, M., Störl, U., & Scherzinger, S. (2015). Schema extraction and structural outlier detection for JSON-based NoSQL data stores. Datenbanksysteme Für Business, Technologie Und Web.

Kumar, R., Charu, S., & Bansal, S. (2015). Effective way to handling big data problems using NoSQL Database (MongoDB). Journal of Advanced Database Management & Systems, 2(2), 42-48.

Maity, B., Acharya, A., Goto, T., & Sen, S. (2018). A Framework to Convert NoSQL to Relational Model. Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology, 1–6.

MongoDB (2018). Mongodb atlas database as a service. https://www.mongodb.com/.Online; 5 July 2018.

Muñoz-Sánchez, P. D., Candel, C. J. F., Molina, J. G., & Ruiz, D. S. (2020, November). Managing Physical Schemas in MongoDB Stores. In International Conference on Conceptual Modeling (pp. 162-172). Springer, Cham.

Object Management Group OMG. MDA Guide Version 1.0.1, June 2003. URL http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf.

Object Management Group OMG. Meta Object Facility (MOF) Core Specification, , Version 2.4.1, August 2011. URL http://www.omg.org/spec/MOF/2.4.1/PDF/

OMG, OMG Unified Modeling Language (OMG UML) Version 2.5.1, 2017. (http://www.omg.org/spec/UML/2.5.1/)

Piechocki, L. (2007). UML, le langage de modélisation objet unifié. Laurentpiechnocki. developpez. com.

Sevilla Ruiz, D., Morales, S. F., & Molina, J. G. (2015). Inferring versioned schemas from NoSQL databases and its applications. International Conference on Conceptual Modeling, 467–480.

UML, O. (2017). OMG (2017) Unified Modeling Language®(OMG UML®) Version 2.5. 1 https://www. omg. org/spec. UML.