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

L’université n’entend ni approuver ni désapprouver les opinions particulières du candidat.

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE
Délivré par l'Université Toulouse 1 Capitole

Présentée et soutenue par

Hela RAJHI

Le 14 décembre 2023

Modélisation de bases de données NoSQL : processus d'extraction des schémas logique et conceptuel

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
M. Chihab HANACHI, Professeur à l'Université Toulouse Capitole, Président
Mme Elisabeth METAIS, Professeure au CNAM de Paris, Rapporteure
M. Allel HADJALI, Professeur à l'ENSMA de Poitiers, Rapporteur
Mme Faten ATIGUI, Maître de Conférences, CNAM, Examinatrice
M. Gilles ZURFLUH, professeur émérite à l'Université Toulouse Capitole, Directeur de thèse
Mme Fatma ABDELHEDI, Directrice du laboratoire CBI^2, TRIMANE, Co-directrice de thèse

Remerciements

Je tiens à remercier Monsieur Gilles ZURFLUH, professeur émérite à l'Université Toulouse Capitole, pour avoir dirigé mes travaux de recherche avec un professionnalisme remarquable. Je suis profondément reconnaissante non seulement pour ses critiques constructives, ses conseils précieux, son assistance, ses qualités d'écoute et sa disponibilité totale qui ont considérablement contribué à l'avancement de mes travaux, mais également pour m'avoir encadré lors de mon premier cours. J'ai eu l'opportunité d'apprendre la pédagogie d'enseignement à ses côtés, bénéficiant ainsi de son expérience précieuse dans le domaine de l'enseignement. C'est avec un grand respect que je le remercie pour son rôle déterminant dans la réalisation de mes travaux. Je suis très honorée d'avoir eu l'opportunité de bénéficier de son encadrement exceptionnel. Monsieur Gilles ZURFLUH représente pour moi une source d'inspiration inégalée.

Je remercie Madame Fatma ABDELHEDI, Directrice du laboratoire CBI^2 au sein de la société TRIMANE, pour avoir encadré cette thèse. Ses critiques et conseils pertinents ont grandement enrichi la qualité de mon travail. Je suis particulièrement reconnaissante de l'opportunité qu'elle m'a offerte permettant la validation de mes propositions à partir d'applications réelles. Je tiens également à la remercier pour sa participation au jury.

Je tiens à exprimer le plaisir et l'honneur que m'a accordés Monsieur Chihab HANACHI professeur à l'Université Toulouse Capitole en acceptant d'examiner ce mémoire et d'être membre de mon jury.

Je remercie Madame Elisabeth METAIS professeure au Conservatoire National des Arts et Métiers (CNAM) de Paris et Monsieur Allel HADJALI professeur à l'école nationale supérieure de mécanique et d'aérotechnique (ENSMA) de Poitiers, pour avoir accepté d'être rapporteurs de mes travaux de thèse. Je les remercie également d'avoir pris le temps de lire attentivement mon mémoire, pour leurs remarques pertinentes et d'avoir participé au jury.

Je remercie également Madame Faten ATIGUI, Maître de conférences au Conservatoire National des Arts et Métiers (CNAM) de Paris, pour avoir accepté d'examiner ce mémoire et d'être membre de mon jury.

Je tiens à remercier Madame Michèle CUESTA, gestionnaire à la faculté d'informatique, pour son aide généreuse tout au long de la préparation de la soutenance de thèse. Sa bienveillance et sa disponibilité exceptionnelles ont grandement facilité cette étape cruciale. Je la remercie également d'avoir fait preuve d'une gentillesse extrême à mon égard.

J'adresse mes remerciements à Messieurs Laurent PERRUSSEL et Frédéric

AMBLARD professeurs à l’Université Toulouse Capitole, avec qui j’ai collaboré pour dispenser des cours. Cette collaboration m’a offert l’opportunité d’échanger sur la pédagogie.

Je tiens à exprimer mes plus sincères remerciements envers ma famille et tout particulièrement envers mes chers parents Hedi et Sabiha, mon frère Achref ainsi que ma sœur Ilhem, à qui je dédie cette thèse. Leur amour inconditionnel et leurs encouragements constants ont été une source inestimable de force tout au long de mon parcours académique. Je leur suis profondément reconnaissante de m'avoir inculqué l'ambition nécessaire pour atteindre mes objectifs. Leur soutien constant a joué un rôle essentiel dans la concrétisation de ce parcours académique. C'est grâce à eux que je suis la personne que je suis aujourd'hui. Mes remerciements les plus sincères vont à chacun d'entre eux, pour leur influence positive et leur impact inestimable dans ma vie.

Résumé

Nos travaux de recherche s'inscrivent dans le cadre d'une application médicale décisionnelle utilisant des bases de données complexes NoSQL. La plupart des systèmes NoSQL sont caractérisés par la propriété « Schemaless » qui signifie qu'un schéma n'est pas défini au moment de la création de la base de données. L'absence de schéma de données rend difficile d'une part la compréhension de la sémantique des données et d'autre part la rédaction des requêtes. Pour des décideurs, cette difficulté représente un obstacle majeur pour manipuler des objets à structure complexe.

Nos travaux portent sur l'extraction de schémas à partir d'une BD NoSQL « Schemaless ». Nous proposons un processus qui permet de générer automatiquement, à partir d'une base de données NoSQL, deux schémas de données :

- un schéma logique contenant la description technique des données stockées dans la BD telles que les collections et les relations entre ces collections ; ce schéma permet d'écrire des requêtes.
- un schéma conceptuel décrivant des classes d'objets exemptes des aspects techniques et permettant de comprendre la sémantique des données.

Pour manipuler une base NoSQL, les décideurs disposent donc d'un double schéma dont les deux facettes (logique et conceptuelle) peuvent être consultées de façon concomitante.

Le processus que nous proposons est basé sur la plateforme EMF qui supporte l'architecture dirigée par les modèles (MDA). Cette technique de développement nous a permis de modéliser une base de données NoSQL ainsi que les schémas logique et conceptuel. L'application d'une suite de règles de transformations écrites dans un langage formel (QVT) assure le passage entre les modèles et permet de générer le double schéma.

Abstract

Our research work is part of a medical decision-making application using complex NoSQL databases. Most NoSQL systems are characterized by the schemaless property, which means that a schema is not defined when the database is created. The absence of a data schema makes it difficult to understand data semantics and to write queries. For decision-makers, this difficulty represents a major obstacle to manipulating objects with complex structures.

Our work focuses on extracting schemas from a schemaless NoSQL database. We propose a process that automatically generates two data schemas from a NoSQL database:

- a logical schema containing the technical description of the data stored in the database such as the collections and the relationships between these collections; this schema allows to write queries.
- a conceptual schema describing classes of objects free from technical aspects and allowing to understand the semantics of the data.

To manipulate a NoSQL database, decision-makers therefore have a dual schema, with both facets (logical and conceptual) that can be consulted simultaneously.

The process we propose is based on the EMF platform which supports model-driven architecture (MDA). This development technique allowed us to model a NoSQL database as well as the logical and conceptual diagrams. The application of a series of transformation rules written in a formal language (QVT) ensures the transition between the models and allows to generate the double schema.

Table des matières

Liste des figures

Liste des tableaux

Chapitre 1
Introduction générale

1.1 Contexte général

Nos travaux se concentrent sur le domaine de l'informatique décisionnelle. Ils concernent la gestion et l'utilisation des données pour soutenir les processus de prise de décision. L'émergence des données massives, appelées Big Data, a entraîné des changements significatifs dans le domaine de l'informatique décisionnelle. Ces changements ont conduit à l'intégration de bases de données (BD) massives dans les sources utilisées par les systèmes de prise de décision. Notre cas d'étude, portant sur une application médicale, reflète cette évolution. Elle représente la principale source de motivation pour nos travaux et est présentée en détail dans le chapitre 2. Dans le cadre d'un programme de suivi de pathologies, les médecins se trouvent dans la nécessité d'interroger et d'analyser quotidiennement, sur une période prolongée, des BD de type Big Data. Cette tâche régulière et continue est essentielle pour obtenir des informations pertinentes, suivre l'évolution des pathologies et adapter les traitements au fil du temps. Grâce à l'utilisation de ces BD massives, les médecins ont la capacité d'explorer une quantité considérable de données médicales, ce qui leur permet d'approfondir leurs connaissances sur les pathologies étudiées et d'améliorer les protocoles de soins pour leurs patients. Les systèmes de gestion de BD relationnelles ont été confrontés à des limites en termes de capacité de stockage des volumes de données massifs et variés. C'est là que les BD NoSQL entrent en jeu. Les BD NoSQL sont conçues pour répondre aux exigences spécifiques des Big Data. Contrairement aux BD relationnelles où le schéma de données est défini lors de la création de la table, les BD NoSQL sont généralement "Schemaless", ce qui signifie qu'elles ne disposent pas de schémas définis au préalable. En d'autres termes, les noms des champs sont déterminés au moment où les données sont ajoutées à la BD. La propriété "Schemaless offre une grande flexibilité permettant au schéma de données d'évoluer facilement au fil de l'exploitation de la BD. De plus, elle permet aux utilisateurs d'ajouter de nouvelles données sans aucune obligation de respecter un schéma précis. Mais, la formulation de requêtes par des décideurs, sans connaître les structures sous-jacentes, peut entraîner des erreurs et nuire à l'interrogation des données massives [1].

Nos travaux s'inscrivent dans le contexte des BD NoSQL "Schemaless".

Les sections suivantes abordent les problématiques et les contributions de notre recherche ainsi que l'organisation de ce mémoire.

1.2 Problématiques de recherche

Comme nous l'avons introduit précédemment, la propriété "Schemaless" de la plupart des BD NoSQL présente plusieurs avantages. Elle offre une flexibilité précieuse pour faire évoluer le schéma de données et s'adapter aux besoins

évolutifs des applications. Par exemple, elle permet d'ajouter, de modifier ou de supprimer des champs de manière dynamique, sans aucun impact sur les documents déjà enregistrés. De plus, elle est particulièrement adaptée aux données non structurées ou semi-structurées, ce qui facilite le stockage et le traitement de divers formats et structures de données. Cependant, la propriété “Schemaless” présente quelques inconvénients. Tout d'abord, l'absence de schéma rend difficile le maintien de l'intégrité des données. De plus, la compréhension des données peut être difficile. En effet, l’absence de schéma nuit à la connaissance de la structure des données (les noms et les types de champs) et des relations entre les objets. Par conséquent, un schéma séparé des données est primordial pour l’expression des requêtes. Ainsi, ce schéma peut décrire la BD à un des trois niveaux : le niveau conceptuel (indépendant de tout aspect technique lié à l’implantation), logique (introduit certaines caractéristiques d’implantation selon le type de plateforme choisie) et physique (spécifique à une plateforme donnée). Nos travaux se concentrent sur les niveaux logique et conceptuel.

Afin d’interroger efficacement une BD NoSQL, il est essentiel de répondre à deux problématiques :

- la première concerne l'extraction du schéma logique qui décrit la structure de données stockées dans cette BD ; ce schéma apporte une assistance précieuse aux décideurs en termes d’expression des requêtes sur la BD
- la seconde concerne l’extraction du schéma conceptuel qui permet aux utilisateurs de comprendre la sémantique des données de la BD en faisant abstraction des aspects techniques

1.3 Contributions

En vue de répondre aux problématiques citées ci-dessus, nous proposons le système ProdSchemas, qui extrait des schémas. Ce processus se compose de deux modules : ToLogSchema et ToConceptSchema. D’abord ToLogSchema génère, à partir d’une BD NoSQL orientée document, un schéma logique décrivant l’organisation interne des données. Ensuite, ToConceptSchema transforme le schéma logique en un schéma conceptuel. Ce dernier est exprimé d’une part en format XMI[1] et d’autre part en utilisant le formalisme UML[2]. Il fait apparaître les classes d’objets avec les noms et les types de leurs attributs ainsi que les liens sémantiques entre les objets (association, composition et héritage). Ces deux processus s’appuient sur l’architecture MDA pour offrir une grande flexibilité face aux changements des versions des systèmes NoSQL.

[1] https://www.omg.org/spec/XMI/
[2] omg.org/spec/UML

Les travaux de cette thèse ont été présentés dans les publications suivantes : [2], [3], [4] et [5].

1.4 Organisation du mémoire

Ce mémoire est structuré en sept chapitres : le premier chapitre est une introduction générale, suivi de six autres chapitres. Dans le chapitre 2, nous abordons le contexte technologique dans lequel s'inscrivent nos travaux. Ceci englobe le Big Data, les systèmes NoSQL et l'ingénierie dirigée par les modèles. De plus, nous présentons le cas d'étude qui a motivé nos travaux.

Le chapitre 3 passe en revue l'état de l'art concernant l'extraction de schéma à partir d'une BD NoSQL "Schemaless". Dans ce chapitre, nous présentons à la fois les logiciels et les travaux de recherche qui traitent de l'extraction du schéma de données. Enfin, nous résumons les travaux de recherche et nous positionnons nos travaux par rapport à l'état de l'art.

Les chapitres 4 et 5 sont consacrés à nos propositions en matière d'extraction de schéma de BD NoSQL comme suit. Le chapitre 4 décrit le processus ToLogSchema qui vise à extraire automatiquement le schéma logique d'une BD NoSQL "Schemaless". Nous détaillons ainsi les composants de ce processus, notamment la source, la cible et les règles de transformation. Dans le chapitre 5, nous abordons le processus ToConceptSchema qui se focalise sur la transformation du schéma logique (préalablement extrait dans le chapitre 4) en schéma conceptuel. Ce dernier est exprimé en format XMI et en utilisant le formalisme UML. Etant donné que la source de ce processus correspond à la cible du processus ToLogSchema, nous nous concentrons uniquement sur la cible et les règles de transformation qui permettent de la générer.

Le chapitre 6 décrit la réalisation de notre processus global ProdSchemas de production des schémas logique et conceptuel à partir d'une BD NoSQL orientée document implanté sous le système OrientDB[3]. Ce processus a été expérimenté et évalué sur une BD réelle.

Finalement, le chapitre 7 conclut ce mémoire en mettant en avant nos contributions, leurs originalités mais aussi leurs limites. Nous évoquons également les perspectives de recherche envisagées.

[3] http://orientdb.org/

Chapitre 2
Contexte technologique et cas d'étude

Dans la dernière décennie, la quantité de données générées par les dispositifs informatiques a explosé. Ces données désignées par le terme “Big Data” suivent la règle des 3V [6] et sont fréquemment stockées sur des plateformes NoSQL dont les fonctionnalités surpassent celles des SGBD relationnels. Ces systèmes NoSQL sont apparus pour faire face aux limitations des SGBD relationnels et ont été conçus spécifiquement pour la gestion de BD massives avec des schémas flexibles. Ils se distinguent également par leur capacité à s'adapter à l'augmentation du volume de données et offrent de bonnes performances en termes de temps de réponse [6]. Face à ces données massives, des solutions dédiées ont été développées pour permettre à des utilisateurs, occasionnels ou experts, d’appréhender les données. L’utilisation d’ontologies a notamment permis d’appréhender des connaissances complexes relevant d’un domaine particulier, la linguistique par exemple [7]. Bien que nos travaux s’inscrivent dans un cas d’étude médical, notre problématique n’est pas dédiée à un domaine précis.

Ce chapitre est structuré comme suit. Les sections 2.1 et 2.2 définissent respectivement le Big Data et les modèles NoSQL. La section 2.3 présente les différents niveaux de modélisation des données. La section 2.4 présente l'ingénierie dirigée par les modèles (MDE). La section 2.5 expose le cas d'étude qui a inspiré nos travaux. La section 2.6 donne un aperçu de notre solution.

2.1 Big Data

Dans cette section, nous donnons une définition du Big Data. Nous présentons les 3V (volume, vélocité et variété) et nous examinons quelques cas d’utilisation du Big Data.

2.1.1 Définition

Le Big Data désigne un ensemble de données volumineux et complexes qui nécessite des méthodes spécifiques de stockage, de traitement et d’analyse. Ces données se caractérisent par leur volume massif, leur variété de formats et de sources tels que les réseaux sociaux, les capteurs, les appareils connectés, les transactions en ligne, les données géospatiales, les fichiers multimédias, ainsi que leur vitesse de génération. Les technologies et les outils spécialement conçus pour gérer et analyser le Big Data sont nécessaires pour extraire des connaissances exploitables à partir de ces ensembles de données.

2.1.2 Principe des 3V

Le Big Data se caractérise par la règle des 3V. Il s’agit du volume, de la variété et de la vélocité. Ces caractéristiques sont détaillées ci-après.

- Volume : les données présentent une taille importante, généralement plusieurs téraoctets. Elles peuvent être générées par différentes sources telles que les capteurs, l'Internet des objets (IoT), les réseaux sociaux et les transactions financières. Par exemple, les capteurs sont de plus en plus utilisés pour collecter des données d'un environnement particulier, les produits et les personnes. L'IoT est un réseau d'objets connectés qui génèrent des données en temps réel. Les réseaux sociaux sont une autre source importante de données ; les utilisateurs y partagent des informations sur leurs activités, leurs intérêts et leurs opinions.
- Variété : les données peuvent être structurées (BD relationnelle, fichier Excel, etc), semi-structurées (XML, JSON, HTML, etc) ou non structurées (image, vidéo, audio, etc) et nécessitent des méthodes d'analyse adaptées à leur format spécifique.
- Vélocité : les données peuvent être générées et traitées à grande vitesse. Elles sont alors produites en temps réel ou à un rythme très élevé, exigeant des capacités de collecte, de stockage et d'analyse en temps réel pour prendre des décisions rapides.

D'autres V sont apparus pour représenter certaines caractéristiques du Big Data telles que la valeur et la véracité [8]. Celles-ci sont souvent considérées comme des aspects importants à prendre en compte lors de la gestion et de l'analyse du Big Data. Leur explication est présentée ci-dessous :

- Valeur : c'est la mesure du profit ou de l'avantage potentiel que les organisations peuvent obtenir du Big Data
- Véracité : elle se réfère à la qualité et la fiabilité des données. Le Big Data est sujet à des erreurs, des duplications, des incohérences et des biais. Il est essentiel d'évaluer et de maintenir la qualité des données pour obtenir des résultats fiables lors de l'analyse.

2.1.3 Cas d'utilisation

Le Big Data est utilisé dans de nombreux secteurs d'activités. Nous donnons quelques exemples des applications fréquentes du Big Data :

- Analyse des données clients : les entreprises utilisent le Big Data pour analyser les comportements et les préférences des clients afin de personnaliser les offres, d'améliorer l'expérience client et de prendre des décisions marketing.
- Prévisions et optimisation : les modèles d'analyse prédictive basés sur le Big Data permettent de prévoir les tendances, les demandes du marché, les

pannes d'équipement, etc. Ces prévisions aident les entreprises à prendre des décisions stratégiques, à optimiser les opérations et à réduire les coûts.

- Médecine : le Big Data est utilisé dans le domaine de la santé pour l'analyse des données médicales, la recherche pharmaceutique, la détection de maladies, le suivi des patients, la gestion des dossiers médicaux, etc. Il permet une meilleure prise de décision clinique et contribue à l'avancement de la médecine.

- Analyse financière : les institutions financières (banques, compagnies d'assurance, …) utilisent le Big Data pour l'analyse des marchés, la détection de fraudes, l'évaluation des risques, etc. Ceci leur permet de prendre des décisions en matière d'investissement et de gestion des risques.

- Transport et logistique : le Big Data est utilisé pour la gestion de la chaîne d'approvisionnement, la planification des itinéraires, l'optimisation des opérations logistiques, la réduction des temps d'attente, etc. Cela aide à réduire les coûts et à améliorer la satisfaction des clients.

- Recherche scientifique : les chercheurs utilisent le Big Data pour l'analyse de données dans des domaines tels que la génomique, l'astronomie, la climatologie et la physique des particules. Ceci permet d'assister les chercheurs, de modéliser des phénomènes et de résoudre des problèmes complexes.

Nos travaux s'inscrivent dans le contexte du Big Data appliqué au domaine médical. Nous détaillerons notre cas d'étude médical dans la section 2.5.

2.2 NoSQL

Les systèmes relationnels (SGBDR) gèrent des données conformes à un schéma prédéfini. Ce schéma décrit les tables, les attributs, les clés primaires et étrangères et les contraintes d'intégrité. En respectant ce schéma, les données sont organisées de manière cohérente, ce qui facilite les opérations de requêtes. Cependant, les SGBDR peuvent atteindre leurs limites face aux caractéristiques du Big Data. Le volume massif de données, la vitesse de collecte et de traitement, ainsi que la diversité des formats et des sources de données peuvent dépasser les capacités des SGBDR. Ceci peut entraîner des problèmes de mise à l'échelle, des temps de réponse lents et des coûts élevés pour le stockage et le traitement des données. C'est pourquoi les systèmes NoSQL ont émergé [9]. Ils sont conçus pour faire face aux limites des SGBDR en permettant une mise à l'échelle, une flexibilité des structures et une gestion efficace de grandes quantités de données. Les SGBD NoSQL offrent différents modèles, tels que les modèles clé-valeur, orientés colonnes, graphes et documents. Chacun de ces

modèles présente des caractéristiques spécifiques qui les rendent adaptés à des cas d'utilisation particuliers. Dans le paragraphe suivant, nous détaillons chaque modèle.

2.2.1 Modèle orienté clé-valeur

Le modèle clé-valeur est le modèle initial et le plus basique dans les SGBD NoSQL. Il a servi de point de départ à l'évolution des autres modèles. Ces autres modèles, tels que les modèles orientés colonnes, graphes et documents, ont été développés en prenant le modèle clé-valeur comme base, mais en y ajoutant des fonctionnalités spécifiques en fonction de leurs propres structures et objectifs. Il stocke les données sous forme de paires clé-valeur (voir figure 2.1), où la clé permet de désigner la valeur qui lui est associée. L'utilisation de ce modèle est avantageuse dans des scénarios où la récupération rapide des données est essentielle et où la structure des données est relativement simple. Il est souvent utilisé dans des cas d'utilisation tels que la mise en cache, la gestion de sessions utilisateur et la gestion de listes d'objets ou de paramètres de configuration. Parmi les SGBD clé-valeur les plus connus, nous pouvons citer **:**

- Redis[4], qui est un système open-source de stockage en mémoire. Il se distingue par sa capacité à stocker et à accéder rapidement aux données.
- Amazon DynamoDB[5], un système de stockage distribué et évolutif. Les données sont stockées sous forme d'objets identifiés par des clés uniques. Seules les opérations de lecture et d'écritures avec la clé d'un objet sont supportées. Ce modèle de données très simple a été adopté pour les besoins particuliers d'Amazon.

Comme mentionné dans l'article [6], bien que les SGBD clé-valeur offrent des fonctionnalités de requêtage simples, ils peuvent être moins adaptés lorsqu'il s'agit d'effectuer des requêtes complexes. Dans de tels cas, d'autres modèles de SGBD NoSQL peuvent être plus appropriés pour répondre aux besoins spécifiques de l'application. Ces modèles, qui sont des évolutions du modèle clé-valeur, sont présentés dans la suite.

[4] https://redis.io/
[5] https://aws.amazon.com/fr/dynamodb/

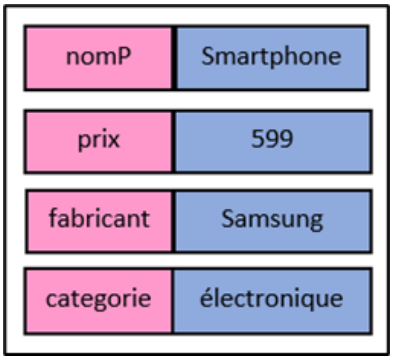


Figure 2. 1 - Organisation de données dans une BD clé-valeur

2.2.2 Modèle orienté colonne

Dans ce modèle, les données sont organisées en familles de colonnes. Chaque ligne est identifiée par une clé et composée d'un ensemble de valeurs. Chaque valeur est associée à une colonne spécifique. Ainsi, pour rechercher une valeur, on parcourt la séquence suivante : clé de ligne -> famille de colonnes -> colonne. Contrairement à une BD relationnelle où les colonnes sont statiques et présentes dans chaque ligne, dans une BD orientée-colonne, les colonnes sont dynamiques et n'apparaissent que dans les lignes concernées. En d'autres termes, chaque ligne peut avoir un nombre différent de colonnes (voir figure 2.2) et de nouvelles colonnes peuvent être ajoutées à tout moment. Parmi les SGBD orientés colonnes, on peut citer Apache Cassandra[6], Apache HBase[7] et Google Bigtable[8].

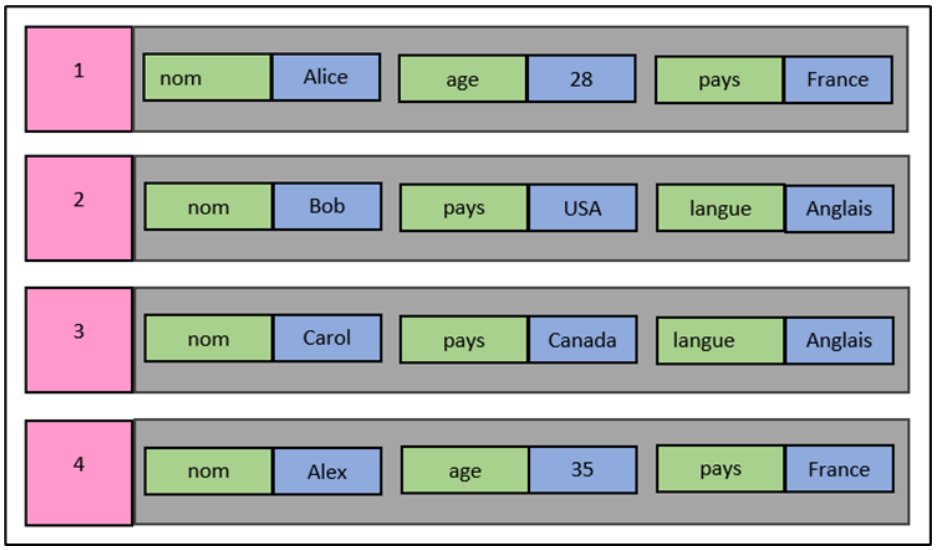


Figure 2. 2 - Organisation de données dans une BD orientée colonne

[6] https://cassandra.apache.org/_/index.html
[7] https://hbase.apache.org/
[8] https://cloud.google.com/bigtable

2.2.3 Modèle orienté graphe

Dans ce modèle, les données sont organisées sous forme de nœuds (représentant des entités) reliés par des arêtes (représentant des relations entre ces entités). Les nœuds et les arêtes peuvent porter des propriétés exprimées sous la forme de paires clé-valeur (voir figure 2.3). Les SGBD Neo4j[9], Amazon Neptune[10], ArangoDB[11] et OrientDB suivent le modèle orienté graphe. Ils sont couramment utilisés dans divers domaines, tels que la gestion des réseaux sociaux, l'analyse des relations entre les entités, la recommandation de produits et la détection de fraudes.

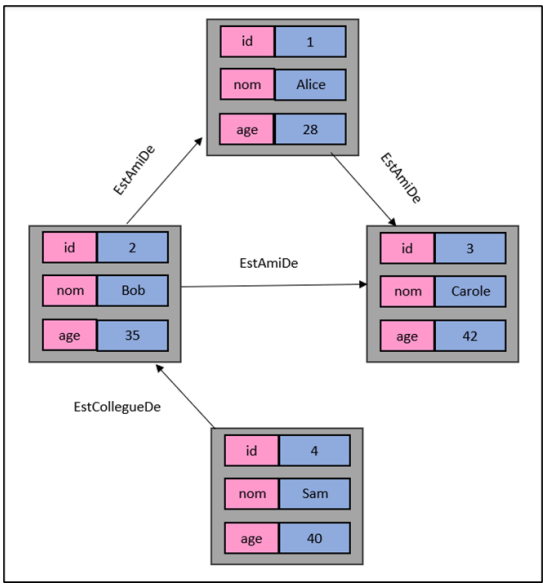


Figure 2. 3 - Organisation de données dans une BD orientée graphe

2.2.4 Modèle orienté document

Ce modèle se base sur la notion de collection de documents pour stocker et organiser les données. Chaque document est constitué d'un identifiant (clé) et d'une valeur composée de paires clé-valeur (voir figure 2.4). Chaque document peut avoir une structure flexible, ce qui signifie que les documents d'une même collection peuvent avoir des structures différentes. OrientDB est un SGBD multimodèle, ce qui signifie qu'il peut être utilisé de deux manières : en tant que SGBD orienté graphe, comme nous l'avons mentionné dans la section précédente, ou orienté document. MongoDB[12] et CouchDB[13] sont aussi des SGBD orientés document qui offrent des fonctionnalités avancées pour la

[9] https://neo4j.com/
[10] https://aws.amazon.com/neptune/
[11] https://www.arangodb.com/
[12] https://www.mongodb.com/
[13] https://couchdb.apache.org/

manipulation des documents. Ils sont couramment utilisés dans divers domaines, tels que le développement d'applications web, la gestion de contenu et l'Internet des objets.

Ce mémoire se focalise sur les BD NoSQL orientées-document, reconnues pour leur capacité avancée à exprimer les liens entre les données. Pour représenter un lien, les utilisateurs peuvent utiliser un champ contenant l'identifiant d'un document ou imbriquer un champ multivalué structuré dans un document. Dans notre étude, nous avons adopté les deux options, ce qui justifie notre choix de BD orientée-document pour le stockage des données de notre application.

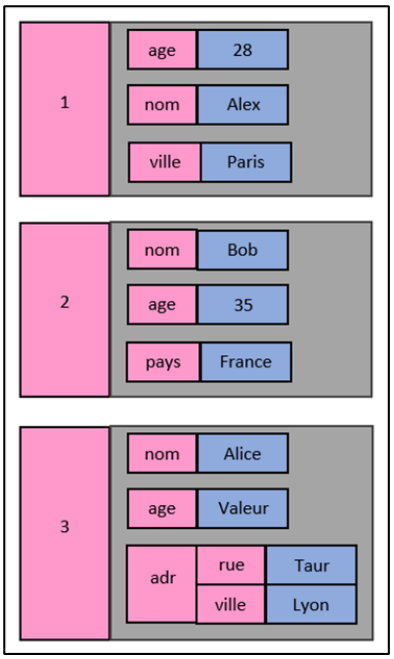


Figure 2. 4 - Organisation de données dans une BD orientée document

2.3 Modélisation de données

La description des données dans les BD NoSQL revêt une importance cruciale, notamment dans les SGBD qui ne permettent pas la création d'un schéma prédéfini. Contrairement aux BD relationnelles, où un schéma rigide est établi à l'avance, les BD NoSQL offrent une approche plus souple et adaptative aux Big Data. Dans un tel environnement, un schéma préétabli peut s'avérer inadapté parce qu'il ne permettrait pas de prendre en compte toutes les variations des données. Grâce à cette approche flexible, les BD NoSQL peuvent facilement évoluer et s'adapter aux besoins évolutifs des utilisateurs et des applications. Dans ce contexte, on peut distinguer trois niveaux de modélisation de données : les niveaux conceptuel, logique et physique, comme illustré dans la figure 2.5. Ces niveaux seront abordés en détails dans le paragraphe suivant.

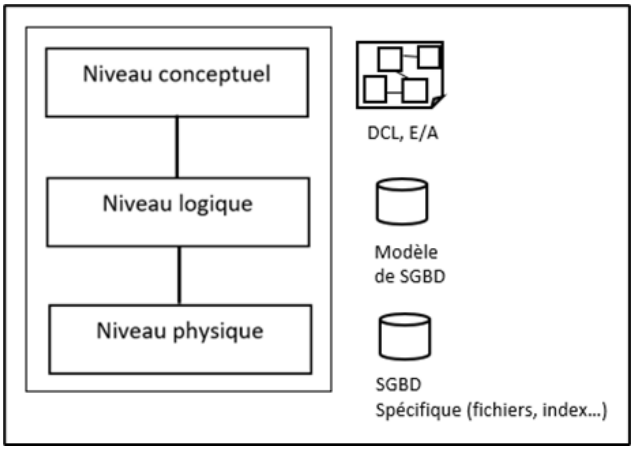


Figure 2. 5 - Niveaux de modélisation de données

2.3.1 Niveau conceptuel

Ce niveau représente les données indépendamment de la manière dont elles sont physiquement stockées. A ce niveau, la description des données se fait généralement à l'aide de concepts tels que classes, attributs et relations. Les classes représentent des objets du monde réel, les attributs décrivent les caractéristiques des objets et les relations représentent les liens entre les classes. La modélisation conceptuelle permet de créer un schéma conceptuel indépendant des détails techniques de stockage. Ce schéma facilite la compréhension de la sémantique des données, en se concentrant sur les relations entre les données plutôt que sur la façon dont elles sont stockées. Ainsi, l'importance du schéma conceptuel est reconnue dans la gestion et l'analyse des Big Data [10] et [11]. Dans ce domaine, les informations peuvent provenir de multiples sources et être sous formats variés ; le schéma conceptuel devient alors un outil essentiel pour organiser et structurer les données de manière cohérente et compréhensible pour faciliter leurs analyses. Par ailleurs, le schéma conceptuel peut être adapté aux évolutions constantes des Big Data. Dans un environnement où de nouvelles sources de données émergent fréquemment et où les besoins métier évoluent rapidement, il permet une intégration flexible de nouvelles informations sans nécessiter de modifications majeures dans l'infrastructure de la BD.

2.3.2 Niveau logique

Le niveau logique se situe entre le niveau conceptuel et le niveau physique. A ce niveau, les concepts du modèle conceptuel sont transformés en structures logiques concrètes, telles que des collections, des champs et des relations. La modélisation logique permet de créer un schéma logique décrivant la structure d'une BD liée à une famille de SGBD mais indépendamment de toute implantation de SGBD spécifique. Le schéma logique permet de faciliter les

opérations de requêtage. En fournissant une structure cohérente, il rend les données facilement accessibles et compréhensibles. Le schéma logique permet également de séparer la structure des données de l'implantation physique propre à un SGBD spécifique. Cette séparation permet aux développeurs de se concentrer sur la logique des requêtes sans avoir à se préoccuper des détails techniques du stockage des données. Ainsi, les développeurs peuvent concevoir des requêtes sans être directement liés aux particularités de la BD sous-jacente.

2.3.3 Niveau physique

Ce niveau permet une représentation concrète des données sur les supports de stockage physiques, prenant en compte les particularités du modèle de données utilisé. Il peut inclure la description d'éléments tels que les index et la pagination du support de stockage. Par exemple, les tables d'index permettent d'accélérer les opérations de recherche et de requêtage des données en créant des structures d'accès rapides basées sur certaines clés ou champs. Il peut également impliquer la fragmentation des données en les répartissant sur plusieurs nœuds du système. Ceci permet d'améliorer les performances de lecture et d'écriture. Pour assurer la disponibilité et la tolérance aux pannes, le niveau physique peut inclure des stratégies de réplication qui permettent de conserver des copies des données sur plusieurs serveurs.

Nos travaux se concentrent principalement sur les niveaux conceptuel et logique de modélisation de données.

2.4 Ingénierie dirigée par les modèles

L'Ingénierie Dirigée par les Modèles (ou en anglais Model Driven Engineering : MDE) est un domaine de l'informatique qui a émergé pour répondre aux défis de la complexité des logiciels. Elle repose sur l'utilisation de modèles abstraits pour décrire les différentes facettes d'un logiciel, allant de ses aspects fonctionnels à ses aspects techniques. MDE a été développée en réponse aux limites des approches traditionnelles de développement logiciel qui étaient souvent basées sur le développement direct du code source. Ces approches ont rendu difficile la maintenance des logiciels face aux changements constants des exigences métier et des technologies sous-jacentes. MDE a permis de remédier à ces problèmes en introduisant une séparation claire entre la logique métier et les implantations techniques. Elle a permis d'améliorer la productivité du développement logiciel en réduisant la complexité des tâches de programmation et en favorisant la réutilisation des modèles pour de multiples applications [12].

L'Architecture Dirigée par les Modèles (ou en anglais Model Driven Architecture : MDA) est une approche de MDE développée par l'Object Management Group

(OMG). Elle repose sur trois types de modèles : “Computation Independent Model” (CIM), “Platform Independent Model” (PIM) et “Platform Specific Model” (PSM). Une description plus détaillée de ces différents modèles est présentée dans la section 2.4.2.

Dans cette section, nous présentons les principes de MDE ainsi que MDA sur laquelle repose notre mémoire. Nous mettons en évidence l’objectif et les modèles de MDA.

2.4.1 Principes généraux

MDE repose sur la notion de modèle. Un modèle est une représentation abstraite d'un système complexe conçu dans un but spécifique [13]. Il permet de capturer les informations essentielles et les aspects pertinents du système tout en ignorant les détails de mise en œuvre et les spécificités techniques. Un modèle dans MDE est utilisé pour générer automatiquement du code. Il constitue une vue plus compréhensible et abstraite du système, facilitant la communication entre les différents acteurs impliqués dans le processus de développement logiciel. Dans MDE, la notion de modèle est étroitement liée au langage dans lequel il est exprimé. Ainsi, un langage de modélisation prend généralement la forme d'un métamodèle. Un métamodèle décrit d'autres modèles dans un domaine spécifique [13]. Il fournit une représentation abstraite d’un langage de modélisation et sert de base pour la création et la compréhension des modèles. Cette définition conduit à l'identification d'une relation appelée "conformeA" qui lie un modèle à son métamodèle. Selon cette relation, l'ensemble des éléments du modèle doit respecter les définitions énoncées par le métamodèle. Cette relation de conformité est essentielle dans la modélisation, car elle garantit l'adéquation et la validité des modèles par rapport à leur métamodèle.

L’OMG a défini une architecture de modélisation qui organise les modèles utilisés dans le développement logiciel en hiérarchie à quatre niveaux : M0, M1, M2 et M3 (voir figure 2.6).

- M0 : est le niveau le plus bas de l'architecture de modélisation. À ce niveau, on trouve l'implantation réelle du système, c'est-à-dire le code source et les données utilisées par l'application. Les caractéristiques principales du niveau M0 incluent une mise en œuvre spécifique à la plateforme où le code est adapté pour fonctionner sur un environnement logiciel et matériel particulier. Les données réelles manipulées par le système sont présentes à ce niveau, ce qui inclut les BD, les fichiers et les tableaux.
- M1 : à ce stade, on représente le système sous forme de modèles conceptuels indépendants de toute technologie. Les modèles du niveau M1

sont utilisés comme base pour décrire l'implantation du système au niveau M0 et sont définis à l'aide des métamodèles appartenant au niveau M2.

- M2 : à ce niveau sont décrits des métamodèles. Ceux-ci décrivent les concepts que les modèles du niveau M1 doivent suivre.
- M3 : est le niveau des méta-métamodèles. Ceux-ci définissent les concepts que les métamodèles du niveau M2 doivent respecter. Les méta-métamodèles sont généralement définis à l'aide d'un langage de modélisation formel tel que MOF (Meta-Object Facility) [14]. MOF permet de créer des métamodèles ou de modifier des métamodèles existants.

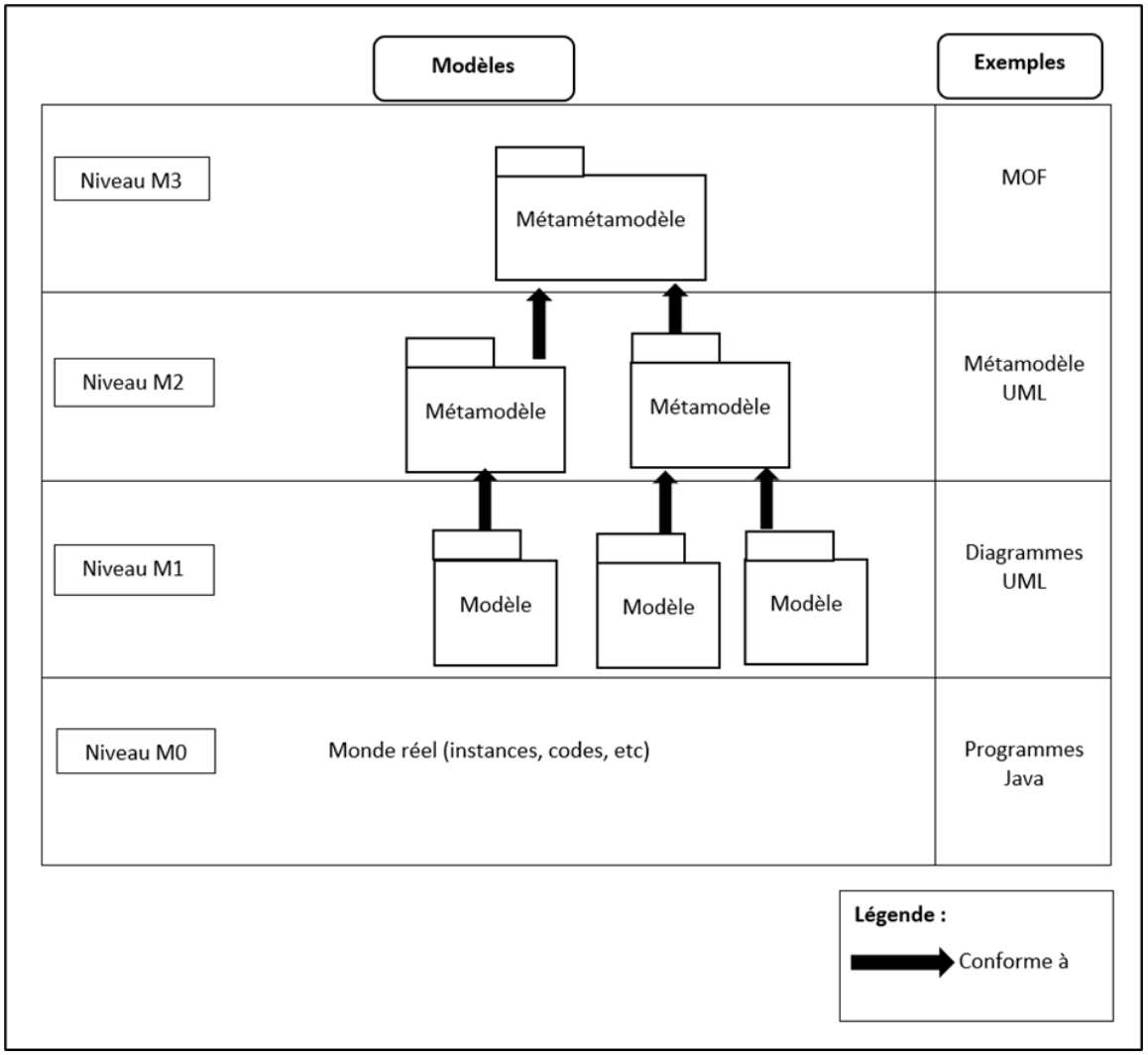


Figure 2. 6 - Architecture de modélisation à quatre niveaux [15]

2.4.2 MDA

MDA est une architecture de développement logiciel basée sur l'utilisation de modèles comme éléments clés du processus de conception et de mise en œuvre

des systèmes [16]. Dans les sections 2.4.2.1 et 2.4.2.2, nous présentons l'objectif et les différents types de modèles de MDA.

2.4.2.1 Objectif

L'objectif de MDA est de séparer les spécifications fonctionnelles et techniques, en mettant l'accent sur des modèles qui sont indépendants de la plateforme et de la technologie. Ceci facilite le développement, la compréhension, la portabilité, la réutilisation et la maintenabilité des systèmes logiciels [16] :

- Compréhension du système : MDA propose différents types de modèles préalables à la génération du code source, facilitant la compréhension du système.
- Portabilité : MDA favorise la portabilité du logiciel entre différentes plateformes matérielles et logicielles.
- Réutilisation : les modèles MDA peuvent être réutilisés dans d'autres applications, ce qui permet de réduire les coûts de développement.
- Génération automatique de code : MDA repose sur le principe de la génération automatique de code à partir des modèles. Ceci permet de réduire les erreurs humaines et de faciliter la mise en œuvre des concepts définis dans les modèles.
- Cohérence et maintenabilité : en utilisant des modèles, MDA permet de renforcer la cohérence du système et de faciliter la maintenance du code.

2.4.2.2 Modèles de MDA

Dans le contexte de MDA, différents types de modèles sont utilisés pour représenter les facettes d'un système. La figure 2.7 présente ces types de modèles, que nous décrivons ci-dessous :

- CIM : est le premier niveau des modèles utilisés dans MDA. Il ne se focalise pas sur la structure du système mais plutôt sur la représentation des exigences fonctionnelles et non fonctionnelles du système, indépendamment des détails de sa mise en œuvre technique.
- PIM : vise à décrire les détails du système d'une manière indépendante d'une plateforme spécifique. Il met l'accent sur la modélisation des aspects métier du système, en se concentrant sur les concepts et les règles du domaine.
- PSM : décrit les détails et les caractéristiques du système dans une plateforme technologique spécifique.

- Code : représente le logiciel généré par le PSM.

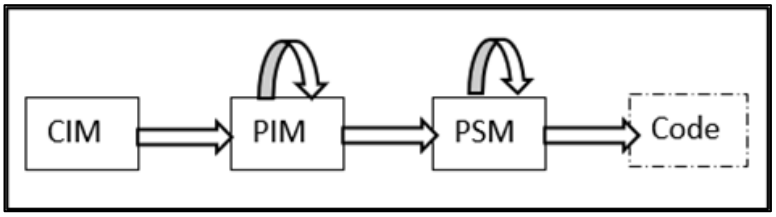


Figure 2. 7 - Modèles de MDA [15]

Le passage d'un modèle à un autre s'effectue à travers des transformations de modèles. Il s'agit de processus qui permettent de générer un modèle cible à partir d'un modèle source en appliquant des règles de transformation. Il existe deux types de transformations de modèle couramment utilisés dans l'approche MDA : 1) la transformation modèle vers modèle (M2M) qui est utilisée pour convertir un modèle en un autre modèle et 2) la transformation modèle vers texte (M2T) : qui est utilisée pour produire des artefacts textuels (modèles interprétables par une machine) tels que du code source à partir d'un modèle. Les transformations de modèles sont utilisées pour automatiser une grande partie du processus de développement logiciel. Elles consistent à appliquer un ensemble de règles décrivant le passage d'un modèle source à un modèle cible. L'OMG a établi le langage "Query/View/Transformation" (QVT) comme norme pour exprimer ces règles. La figure 2.8 montre un exemple de transformation M2M. Dans ce cas, des règles de transformation exprimées en QVT sont appliquées sur des données conformes à un modèle source pour générer des données conformes à un modèle cible.

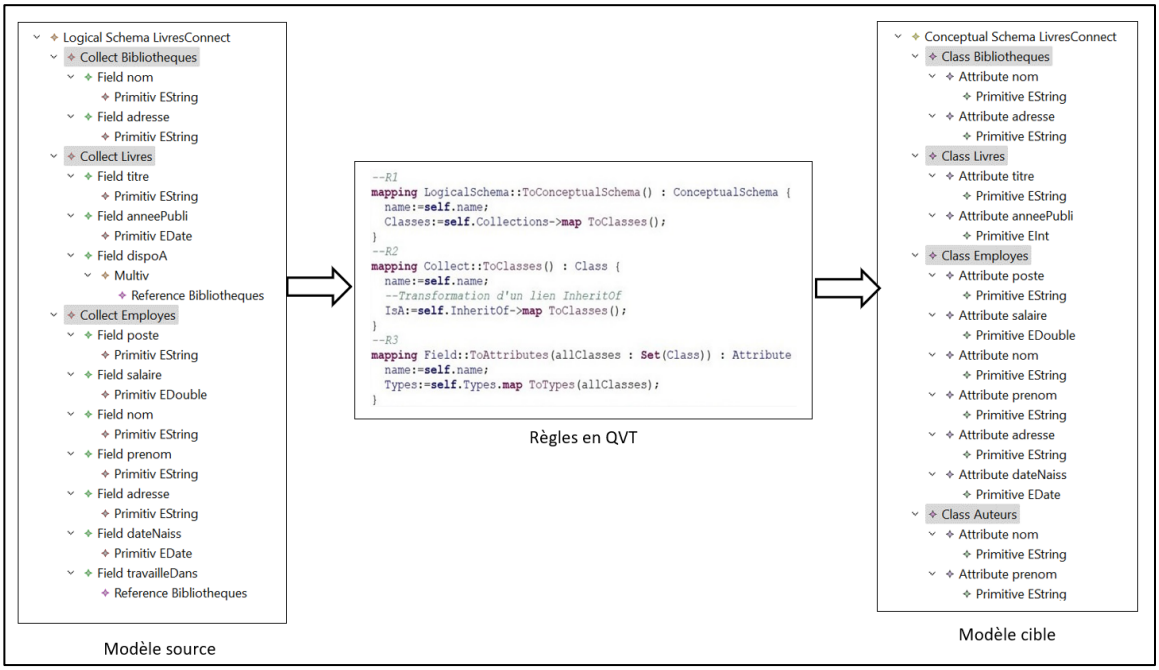


Figure 2. 8 - Exemple de transformation M2M.

2.5 Motivation : le cas d'étude

Nos travaux s'inscrivent dans le cadre d'une application médicale développée pour un projet industriel. Les sections suivantes décrivent les données médicales utilisées dans notre étude et présentent l'objectif de l'application médicale.

2.5.1 Des données médicales

Des programmes scientifiques ont été mis en œuvre pour le suivi de pathologies rares sur des patients hospitalisés. Chaque programme peut regrouper une cinquantaine d'établissements européens (hôpitaux, cliniques et centres de soins spécialisés). Au sein de chaque programme, des données significatives sur l'évolution de la maladie permettent d'étudier ses interactions avec des maladies opportunes et d'évaluer l'influence de ses traitements à court et moyen terme. Un programme a une durée déterminée lors de son lancement et s'étale généralement sur trois ans (il peut atteindre dix ans dans certains cas).

Au lancement du programme, un médecin-coordonnateur est désigné dans chaque établissement participant. Son rôle est de :

- Constituer un groupe de patients volontaires pour participer au programme (une trentaine de patients en moyenne).
- Établir les protocoles de soins en conformité avec les objectifs du programme.
- Désigner des praticiens de l'établissement chargés de réaliser les mesures, prodiguer les soins, administrer les traitements et effectuer les consultations nécessaires.

Avant le lancement du programme, chaque coordinateur doit saisir l'ensemble les données initiales, incluant :

- Les informations personnelles de chaque patient : identité, date de naissance, coordonnées de l'organisme social, médecin traitant, personne(s) de confiance.
- Les éventuels antécédents médicaux du patient, tels que des maladies graves, des opérations chirurgicales ou des accidents. Dans ce cas, des documents sous forme d'images comme des comptes rendus, radiographies ou séquences vidéo, peuvent être enregistrés.
- Les informations de chaque professionnel de santé affecté au programme, comprenant leur identité et leur spécialité.

- Les protocoles de soin trimestriels spécifiques à chaque patient, incluant les mesures à collecter, les traitements à administrer et les contrôles à effectuer. Chacun de ces éléments est associé à une fréquence ou à une date de réalisation et le cas échéant, il est affecté à un professionnel de santé.

Dès le lancement d'un programme, des données sont collectées grâce à des capteurs tels que des thermomètres ou des tensiomètres mis à la disposition des patients, ou en utilisant des tablettes mobiles et des équipements spécialisés tels que des scanners ou des IRM par l'intermédiaire des personnels médicaux. La consultation d'un patient par un médecin peut être réalisée soit en chambre lorsque le patient est hospitalisé, soit en consultation ambulatoire. Elle est systématiquement programmée, soit dans le protocole trimestriel, soit ajoutée par un médecin. Pendant la consultation, le médecin rédige un compte rendu et enregistre toutes les mesures et tous les documents fournis par le patient. De plus, le médecin peut prescrire une ou plusieurs ordonnances comme une radiographie, des médicaments non prévus dans le protocole, ou décider de suspendre temporairement ou définitivement un traitement. Ces données sont ensuite regroupées pour être analysées. Elles présentent les caractéristiques généralement admises pour le Big Data (les 3 V). D'une part, le volume des données médicales collectées quotidiennement auprès des patients peut atteindre plusieurs téraoctets sur trois ans pour l'ensemble des établissements. D'autre part, les données saisies sont multiples (mesures de constantes, radiographie, scintigraphie, etc.) et varient en fonction de l'état de santé de chaque patient. Enfin, certaines données sont produites de façon continue par des capteurs ; elles doivent être traitées en temps réel (mesures franchissant un seuil qui impliquerait l'intervention urgente d'un praticien, par exemple). En raison de la nature et de la diversité des données à stocker, nous avons utilisé le modèle orienté document du système NoSQL OrientDB. Ce modèle offre une diversité de liens (association et héritage) entre les objets. Ce qui le rend proche des concepts proposés par l'ISO et l'ODMG. De plus, il dispose de la fonctionnalité "Schemaless" permettant une grande souplesse en matière de description des données.

2.5.2 Développement de l'application

Notre application vise à élaborer un environnement logiciel destiné à des personnels médicaux pour (1) collecter des données sur les patients et (2) interroger et analyser ces données. Ainsi, comme mentionné dans la section précédente, ces données ont été stockées dans une BD gérée par OrientDB. Mais l'absence de schéma des données dans ce système représente un obstacle majeur pour rédiger des requêtes précises sur la BD. En effet, l'expression d'une requête nécessite de connaître les noms des collections ainsi que les noms des champs et leurs types. Par exemple, considérons la requête posée par un

médecin participant à un programme : Obtenir pour chaque médecin la référence des médicaments prescrits à des patients atteints de la maladie de Creutzfeldt-Jakob. La traduction de cette requête avec un langage de type SQL pourrait être la suivante (extrait) :

Select X.name, M.ref

From Diseases AS D, Doctors AS X, Prescribe AS P, Medicines AS M

Where D.name = "Creutzfeldt-Jakob"

Group By X.name;

Sur cet exemple de requête exprimée par un médecin (au travers d'une interface graphique appropriée), on peut percevoir la difficulté à laquelle est confronté un non informaticien pour rédiger des requêtes précises portant sur des données complexes. Cette difficulté repose sur deux verrous :

- la nécessité de comprendre la sémantique des données complexes et de leurs relations
- l'utilité de connaître les structures de données pour rédiger des requêtes répondant à des besoins précis

Au regard de ces exigences, nous avons donc développé un système de production de schémas de données à partir d'une BD existante.

2.6 Survol de notre solution

Nos travaux Pour interroger et analyser une BD de type Big Data, les non-informaticiens ont besoin de connaître à la fois la sémantique et la structure des données contenues dans la BD. Cette connaissance est généralement présente dans un schéma de données. Mais, comme nous l'avons vu, les systèmes NoSQL n'offrent pas une telle fonctionnalité. Notre solution consiste donc à proposer un système de production d'un double schéma contenant tous les éléments nécessaires pour d'une part appréhender la signification de données et d'autre part connaître leurs structures. Ce système comporte deux modules : l'extraction du schéma logique et la transformation du schéma logique en un schéma conceptuel. Il permet donc de visualiser simultanément le schéma conceptuel pour percevoir la sémantique des données et le schéma logique pour exprimer les requêtes. Ainsi, il permet à un non-informaticien (un décideur) de sélectionner, par exemple, un lien d'association dans le schéma conceptuel et de consulter de façon concomitante sa traduction dans le schéma logique.

Le premier module d'extraction de schéma logique repose sur l'architecture MDA. Celle-ci permet de créer deux métamodèles : source représentant la BD

et cible représentant son schéma logique. Ainsi, elle permet de proposer un ensemble de règles de transformations écrites en langage QVT. En appliquant ces règles sur une BD d'entrée, un schéma logique sera généré automatiquement dans le format XMI. Le deuxième module transforme le schéma logique généré par le premier module en un schéma conceptuel. Son développement repose également sur MDA selon laquelle la source (schéma logique) et la cible (schéma conceptuel) sont métamodélisées. Des transformations successives appliquées à un schéma logique d'entrée permettent de produire un schéma conceptuel dans le format XMI et sous forme de DCL UML.

La figure 2.9 montre un exemple de visualisation d'un double schéma pour une BD NoSQL "LivresConnect" conçue pour stocker des informations sur des livres, des auteurs, des utilisateurs, des bibliothèques et d'autres entités liées à la gestion d'une bibliothèque. Ainsi, la partie gauche de la figure présente le DCL du schéma conceptuel de la BD. Ce schéma contient des classes telles que Livres et Bibliotheques et des liens tels que DispoA et Oeuvres. Quand le lien DispoA est sélectionné dans le schéma conceptuel, alors ce même lien (sous une forme différente) est grisé dans la partie droite de la figure.

Nous verrons dans le chapitre suivant "Etat de l'art" que les travaux les plus avancés portant sur les BD NoSQL, proposent un schéma logique unique. Dans ce cas, cet unique schéma décrit des structures de données souvent très complexes accompagnées d'une représentation technique des liens. Or, si le schéma logique est bien adapté pour écrire des requêtes (la forme des liens est prise en compte dans les requêtes), il s'avère parfois abscons pour comprendre la sémantique des données.

Ce problème n'est pas nouveau si l'on compare deux modèles connus de longue date et largement utilisés dans les entreprises : les modèles Relationnel (niveau logique) et Entité-Relation (niveau conceptuel). Considérons par exemple une BD contenant les associations Posséder et Conduire entre des Personnes et des Voitures. Ces deux associations, décrites de façon similaire dans le schéma Entité-Relation, peuvent être représentées respectivement par une clé étrangère et une table de clés dans le schéma Relationnel correspondant. Le schéma conceptuel facilite donc la compréhension des données et le schéma logique permet d'exprimer des requêtes.

Dans le cas où une BD n'est pas complexe, le recours à un double schéma n'est pas vraiment nécessaire. Mais nous avons pu constater que, dans notre cas d'étude et face à un Big Data médical, les décideurs éprouvent de réelles difficultés à appréhender la sémantique des données. Dans ce contexte, la consultation concomitante des deux schémas représente un atout majeur pour les décideurs.

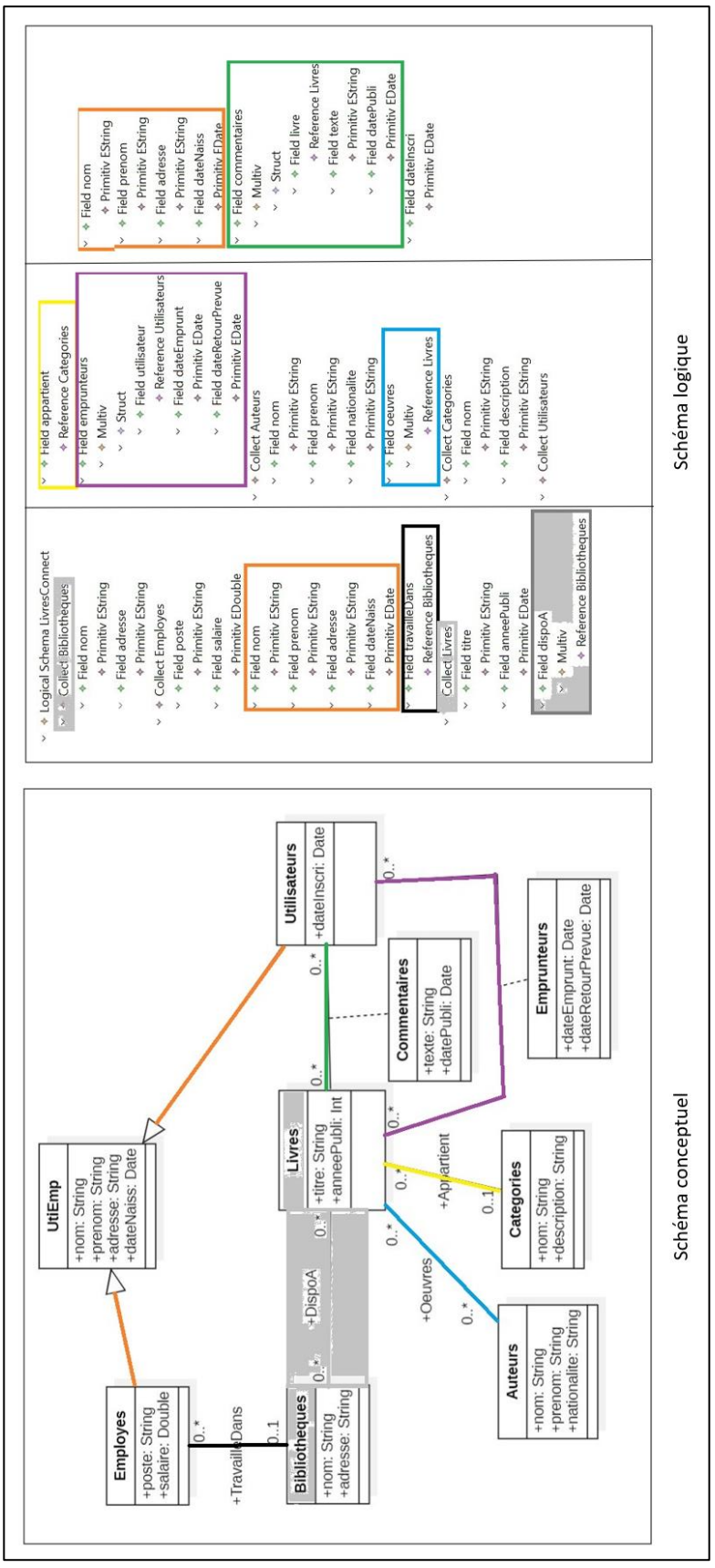


Figure 2. 9 – Visualisation concomitante des deux schémas.

Chapitre 3
Etat de l’art

Dans ce chapitre, nous présentons des travaux qui portent sur l'extraction de schéma de BD NoSQL orientée document. L'objectif de ces travaux est de fournir aux utilisateurs un schéma de données pour manipuler efficacement une BD "Schemaless". La section 3.1 passe en revue les travaux de recherche sur l'extraction de schéma. La section 3.2 étudie les limites de ces travaux et la section 3.3 dresse un bilan de ce chapitre.

3.1 Extraction de schéma

Certains systèmes NoSQL "Schemaless" proposent une fonctionnalité de visualisation des structures de données. OrientDB est un SGBD NoSQL "open-source". Il propose une fonctionnalité facultative de création de schéma qui permet de définir la structure et les contraintes des données stockées dans la BD ; il offre donc trois modes de fonctionnement différents en ce qui concerne le schéma des données : le mode sans schéma, le mode avec schéma complet et le mode avec schéma mixte :

- mode sans schéma ("Schemaless") : dans ce mode, OrientDB ne nécessite pas la définition préalable d'un schéma pour les données stockées. On peut donc ajouter de nouveaux enregistrements avec des propriétés arbitraires sans avoir à définir une structure de données fixe au préalable. Cela offre une grande flexibilité, car on peut ajouter, modifier ou supprimer des propriétés lors de l'exploitation de la BD.
- mode avec schéma complet ("Full Schema") : dans ce mode, on doit définir un schéma complet pour les données avant de pouvoir les insérer dans la BD. Cela signifie qu'on doit définir les classes, les propriétés, les relations et les contraintes de manière explicite.
- mode avec schéma mixte ("Mixed Schema") : ce mode combine des aspects des deux modes précédents. Dans ce mode, on peut définir un schéma partiel pour certaines parties de données tout en conservant la flexibilité du mode sans schéma pour d'autres parties.

Compass[14] et Studio 3T[15] sont deux interfaces dédiées aux BD MongoDB qui permettent d'extraire le schéma d'une collection de documents sous forme de tableau. Ce schéma affiche les noms et les types des champs. Dans Compass, une structure DBRef est de type DBRef, tandis que dans Studio3T, elle est de type Reference. Hackolade[16] est un logiciel d'extraction de schéma graphique représentant les collections sous la forme de classes d'entités avec leurs

[14] https://www.mongodb.com/docs/compass/current/
[15] https://studio3t.com/fr/
[16] https://hackolade.com/

attributs. De plus, le schéma généré contient des liens d'association monovalués et multivalués créés en repérant des structures DBRef.

Des travaux de recherche ont complété ces fonctionnalités pour extraire un schéma pour une BD NoSQL. Il convient de noter que dans la suite de ce chapitre, nous utilisons le terme “logique” pour désigner un schéma qui contient des éléments techniques et le terme “conceptuel” pour désigner un schéma sémantique exempt de toute technique. Ainsi, un schéma conceptuel désigné par un DCL contient au moins les liens d’association les plus couramment utilisés.

Des travaux ont étudié l’extraction de schéma logique pour une BD NoSQL, notamment une BD orientée document. Dans une telle BD, les données sont stockées sous forme de collections de documents. Un document est un objet qui contient des champs avec leurs valeurs de type quelconque : des chaînes, des nombres, des tableaux et des documents.

L'article [17] propose un processus d'extraction de schéma logique sous forme d’un arbre à partir d'une collection de documents. Le processus commence par analyser chaque document de la collection. Pour chaque document, un schéma est extrait et présenté sous forme d'un arbre. Dans cet arbre, les nœuds correspondent aux champs et les arêtes représentent les niveaux, particulièrement utiles pour les champs structurés. Cette approche permet une représentation hiérarchique des informations contenues dans les documents. Une fois que les schémas des documents ont été extraits, le processus regroupe les schémas équivalents en catégories. Chaque catégorie représente un type d'objet. Au sein de chaque catégorie, le processus unifie les champs de même signification mais ayant des noms différents. Pour ce faire, il remplace le nom d'un champ par celui ayant la fréquence d'apparition la plus élevée dans les documents de la collection. Par conséquent, le processus d'extraction obtient un schéma unique pour chaque catégorie d'objets présents dans la collection de documents. Enfin, le processus fusionne les schémas des différentes catégories pour générer un schéma unique pour la collection de documents.

Dans [18] et [19], les auteurs présentent un processus d'extraction de schéma logique en format Json pour une collection de documents. Ce processus se base sur un programme Map-Reduce qui est utilisé pour le traitement parallèle de grands ensembles de données. Le processus présenté est composé de deux étapes complémentaires. La première étape, intégrée à la phase “Map”, joue un rôle essentiel dans l'extraction de schéma à partir de chaque document de la collection. Elle consiste à analyser chaque document pour extraire les noms des champs et déduire leurs types en se basant sur les valeurs présentes dans les documents. À la fin de cette étape, nous obtenons plusieurs schémas, chacun décrivant un document. La deuxième étape est intégrée à la phase “Reduce” du

programme Map-Reduce. Elle consiste à fusionner les schémas extraits lors de l'étape précédente afin de générer un schéma unique pour la collection. Pour effectuer cette fusion, les auteurs proposent une fonction spécifique qui prend en entrée deux types, notés T et U et les compare pour les fusionner. Si les types T et U sont différents, la fonction retourne un résultat de la forme T + U. Ces travaux ont été étendus dans un second temps pour proposer un processus paramétrique où la deuxième étape utilise une relation d'équivalence comme paramètre pour déterminer si deux types sont suffisamment similaires ou trop différents pour être fusionnés.

Un autre processus d'extraction de schéma logique en format JSON d'une collection de documents implantée sous MongoDB est proposé dans [20]. Les documents intègrent d'autres types tels que les DBRef. Le processus comprend trois étapes. Dans la première étape, le schéma de chaque document est déduit en format JSON, incluant les noms et les types des champs. Cette étape initiale permet d'obtenir plusieurs schémas. Dans la deuxième étape, les schémas distincts sont fusionnés en utilisant une structure de données basée sur un arbre. Chaque nœud dans cet arbre peut représenter soit un champ, soit un type. Les champs sont reliés à leur type par des arêtes. Lorsque le type est complexe, par exemple structuré, il est relié aux champs qui le composent. Ces champs sont à leur tour liés à leurs types. La dernière étape du processus consiste à générer un schéma unique en format JSON à partir de l'arbre construit précédemment.

L'article [21] décrit un processus de gestion de schéma pour une BD implantée sous MongoDB. Tout d'abord, le processus extrait le schéma logique en format Json d'une collection de la BD en utilisant deux outils d'extraction de schéma. Ce dernier contient les noms des champs ainsi que les types qui leurs sont associés. L'objectif de cette étape est de garantir la qualité des données stockées dans la BD. Le SGBD peut être configurée pour rejeter toute insertion de donnée qui ne respecte pas le schéma extrait. Ensuite, le processus permet de comparer, au niveau sémantique, le schéma extrait avec un autre schéma. Cette comparaison permet de déterminer si le schéma extrait est un sous-ensemble ou un sur-ensemble, équivalent ou différent d'un autre schéma.

Dans [22], un processus pour l'extraction d'un schéma logique d'une collection est proposé. Un objet contenant un ensemble de champs sous la forme de paires (clé, valeur), est représenté par un rectangle contenant les noms des champs. Chaque champ est généralement associé à un type atomique (nombre, chaîne de caractères, date, etc.). Il est possible qu'un champ ait plusieurs types. Par exemple, un numéro de téléphone peut être soit un nombre, soit une chaîne de caractères. Ceci est représenté dans le schéma par un opérateur logique "OU" entre les types possibles. Le schéma prend en compte également les champs multivalués et multivalués structurés.

Nous citons aussi les travaux de [23] qui proposent un processus pour extraire des versions de schéma logique, sous forme d'un arbre, pour une collection de documents. Les documents sont classifiés grâce à un arbre de décision dont les nœuds représentent les champs des documents et les arêtes précisent les conditions sur lesquelles la classification est basée. Ces conditions sont liées soit à la présence, soit à l'absence, soit à la valeur d'un champ dans un document. Le résultat retourné par ce processus n'est pas un schéma unique mais plutôt un ensemble de schémas dont chacun correspond à un groupe de documents.

L'article [24] propose un processus d'extraction de schéma logique en format Json d'une collection de BD MongoDB. Ce processus comprend trois étapes. La première consiste à extraire le schéma sous forme d'un arbre dont les nœuds représentent des champs atomiques ou des tableaux, et les arêtes représentent les niveaux dans le cas des champs structurés. Le type de chaque champ est stocké dans le nœud. Si un champ a plusieurs types différents, ceux-ci sont enregistrés en leur associant le mot clé "OneOf". La deuxième étape consiste à générer le schéma de la collection en format JSON en se basant sur l'arbre construit. Dans la troisième étape, le processus utilise des mesures de similarité pour capturer le degré d'hétérogénéité des documents d'une collection.

L'article [25] présente un processus d'extraction de variantes de schéma logique en format Json à partir d'une collection de documents MongoDB. Le processus consiste en plusieurs étapes. La première consiste à extraire pour chaque document son schéma. Ce dernier contient le chemin de chaque champ (comprenant le nom de la collection à laquelle il appartient ainsi que le nom de la structure s'il est imbriquée dans cette structure) concaténé son type. Le résultat retourné par cette étape est un ensemble de schémas en format JSON. La deuxième étape consiste à regrouper les schémas similaires. La similarité entre deux schémas de documents est calculée en utilisant TF-IDF (Term Frequency - Inverse Document Frequency) qui est une mesure utilisée pour évaluer l'importance d'un terme (champ) dans un document par rapport à une collection de documents. La troisième étape consiste à extraire, à partir des groupes construits dans l'étape précédente, des variantes de schéma. Une variante de schéma est un ensemble de chemins de champs qui sont compatibles les uns avec les autres. Dans cette étape, FCA (Formal Concept Analysis) est utilisée pour identifier les variantes de schéma. FCA est une technique qui permet de découvrir des structures dans des données. Dans le contexte de l'extraction des variantes de schéma, FCA est utilisée pour identifier les concepts communs aux chemins de champs dans chaque groupe. Les variantes de schéma sont dérivées des concepts communs. Une variante de schéma est donc composée de tous les chemins de champs qui appartiennent à un même concept.

Dans [26], un processus d'extraction de schéma logique en format Json d'une BD orientée document est proposé. Ce processus repose sur un programme Map-Reduce et comporte trois étapes. La première étape est intégrée dans la phase "Map" et consiste à extraire, pour chaque collection, les schémas de tous les documents composant cette collection. Le schéma d'un document contient les noms des champs. Le résultat de cette étape est un ensemble de schémas. La deuxième étape est intégrée dans la phase "Reduce" et consiste à fusionner les schémas extraits à partir des documents qui composent une collection en vue d'obtenir un schéma unique pour chaque collection. La troisième étape consiste à construire à partir des schémas de collections un graphe contenant tous les champs. Dans ce graphe, un nœud contient le nom d'un champ et une arête représente un niveau (par exemple dans le cas d'un champ structuré). Le but de la dernière étape est d'obtenir un seul schéma pour toute la BD.

Dans [27], [28] et [29], il est proposé un processus d'extraction de schéma logique Json pour une BD (MongoDB ou CouchDB) comportant plusieurs collections. Ce processus basé sur MDE repose sur la métamodélisation de la source et de la cible et l'utilisation de règles de transformation permettant de générer la cible à partir de la source. Selon MDE, ce processus vise à générer un schéma conforme à un métamodèle cible. Le schéma contient des entités. Une entité est composée de plusieurs variations structurelles (versions d'entités : correspond à différentes structures qu'une entité peut prendre en fonction des champs contenus dans les objets). Les champs peuvent être des attributs, des références ou bien des agrégations. Ainsi une référence est extraite à partir des liens DBRef. Une agrégation est déduite lorsqu'un champ a pour valeur un objet. Ces travaux ont été étendus, dans un premier temps, dans le but d'automatiser l'utilisation de "Mappeurs", c'est-à-dire des suites logicielles qui assistent les développeurs pour définir et manipuler des objets orientés document (par exemple Mangoose basé sur MongoDB). Les auteurs montrent l'utilisation des schémas extraits d'une BD existante pour générer du code applicatif. Ces travaux ont été étendus, dans un second temps, pour extraire les liens d'héritage entre les entités en se basant sur leurs variations structurelles. Le processus d'extraction de ce type de lien comprend quatre étapes. Tout d'abord, une matrice de variation est établie pour chaque entité, représentant la présence d'un champ dans une variation structurelle. Ensuite, l'ensemble des sous-types de l'entité est identifié. Puis, un raffinement est appliqué à cet ensemble afin de déterminer si certains champs optionnels doivent être considérés comme obligatoires pour certains sous-types. Enfin, le champ discriminant est déduit. Un champ discriminant est un champ qui doit être présent dans toutes les variations et doit posséder une valeur unique dans chaque sous-type. Une fois le processus d'extraction des liens d'héritage terminé, le schéma initial est mis à jour en intégrant les sous-types de l'entité.

D'autres recherches ont exploré l'extraction de schémas logiques pour des BD NoSQL de type différent de celles orientées documents**.** Par exemple, l'article [30] propose un processus d'extraction de schéma logique en format JSON d'une BD orientée colonne (HBase). Rappelons que dans ce type de BD, les données sont organisées en familles de colonnes. Chaque ligne est caractérisée par une clé et est composée d'un ensemble de valeurs. Ce processus se compose de trois étapes. La première étape consiste à identifier les tables et les familles de colonnes dans la BD. La deuxième étape consiste à déterminer le type de données de chaque colonne en fonction de sa valeur. La dernière étape consiste à générer le schéma JSON. Pour ceci, chaque famille de colonnes implique la génération d'un document JSON. L'identifiant de ce document est la concaténation du nom de la famille de colonnes et des clés de chaque colonne. Chaque colonne appartenant à une famille de colonne entraîne la génération d'un champ de même nom dans le document et associé à un type de données qui peut être atomique ou objet.

Dans [31], il est présenté un processus d'extraction de schéma logique en format JSON d'une BD orientée graphe. Dans ce contexte, une BD graphe est composée de nœuds et d'arêtes. Chaque nœud est étiqueté et regroupe un ensemble de propriétés. De même, une arête reliant deux nœuds est aussi étiquetée et comporte un ensemble de propriétés. Une propriété est caractérisée par un nom et une valeur atomique ou complexe. L'extraction de schéma est réalisée en trois étapes. Dans la première étape, des groupes de nœuds et d'arêtes ayant les mêmes étiquettes sont formés. A cette étape, chaque groupe de nœuds contient des propriétés avec pour chacune son nom et son type de donnée. La deuxième étape consiste à déduire les relations d'héritages entre les nœuds. Pour ceci, des groupes de nœuds ayant plus qu'une étiquette sont identifiés. L'intersection entre les étiquettes des groupes est utilisée pour générer un nouveau groupe incluant les étiquettes, les propriétés et les arêtes communes. Ceux-ci sont ensuite supprimés des groupes spécifiques en utilisant un marqueur "super-type" pour indiquer l'étiquette du super-type. La troisième étape vise à générer le schéma au format JSON pour chaque groupe.

Dans [2], il est proposé un processus d'extraction de schéma en format XMI pour une BD gérée par le système MongoDB. Ce système ne prend pas en compte tous les liens sémantiques qu'on trouve dans un schéma de type UML. Le processus proposé repose sur MDA qui permet de définir un métamodèle source représentant la BD, un métamodèle cible représentant son schéma et un ensemble de règles de transformation qui assurent le passage de la BD vers son schéma.

Peu de travaux permettent d'extraire un schéma conceptuel pour une BD orientée document. Par exemple dans l'article [32]**,** les auteurs proposent d'extraire, à partir d'une collection, un schéma conceptuel graphique en se

basant sur un langage de modélisation conceptuelle : IDEF1X (“Integrated DEFinition for Information Modelling”). Un diagramme IDEF1X est composé d’entités et de relations. Une entité contient des attributs. Une relation décrit comment les entités sont liées. Pour créer le schéma, le processus parcourt les documents et regroupe les noms distincts des champs qui les composent dans un seul document. Il utilise des techniques de similarité pour identifier les champs ayant la même signification mais écrits différemment. Les champs identifiés sont remplacés par leurs équivalents. Ensuite, le schéma graphique est généré à partir du document en utilisant un ensemble de règles ; lorsque le séparateur "{" est détecté (champ structuré), une nouvelle entité est générée et reliée par une relation monovaluée (0:1). Quand le séparateur "[{" est détecté (champ multivalué structuré), alors une nouvelle entité est générée et reliée par une relation (0:N). Les champs sont transformés en attributs dans leur entité correspondante. Dans le cas d’un champ multivalué de valeurs atomiques, une nouvelle entité contenant un seul attribut nommé “att” est générée et reliée par une relation (0:N). Les relations (0:N) et (0:1) sont identifiées par “EMBED” et “REFI” respectivement.

De même, l’article [33] présente un processus d’extraction d’un schéma conceptuel, limité aux liens de composition, à partir d’un ensemble de documents Json. Ce processus se déroule en deux étapes : la première étape consiste à extraire le schéma de chaque document. La deuxième étape porte sur la fusion de tous les schémas pour obtenir un schéma unique. Ce dernier affiche les classes ainsi que les attributs et leurs types. Il mentionne également les liens de composition. Tout champ complexe (structuré ou multivalué structuré) appartenant à une collection X, est transformé en une nouvelle classe de même nom. Cette classe est reliée par un lien de composition à la classe X. Si le champ est structuré, le lien est monovalué, s’il est multivalué alors le lien est multivalué.

Dans [34], les auteurs proposent un processus d’extraction d’un schéma conceptuel sous forme de diagramme entité/association (E/A) pour une BD orientée-graphe (Neo4J). Le processus comporte deux étapes. La première étape consiste à construire un graphe à partir d’un ensemble de requêtes d’insertion de nœuds et d'arêtes. La deuxième étape repose sur MDA et consiste à extraire de ce graphe un schéma (E/A). Pour ce faire, deux métamodèles source et cible sont créés : l’un représente le graphe et l’autre son schéma E/A. Le métamodèle cible est composé d’entités, de relations et de relations “ISA”. Une entité contient des attributs. Une relation représente une association dans un diagramme E/A : elle contient des attributs et est reliée à l’entité source et cible de cette relation. Une relation “ISA” représente un héritage entre deux entités : elle est reliée à la super-entité et la sous-entité. De plus, pour extraire le schéma E/A, un ensemble de règles de transformation est créé. Ces règles

décrivent le passage du métamodèle source vers le métamodèle cible. Ainsi, les nœuds portant la même étiquette dans un graphe sont transformés en une entité dans un schéma E/A. Les propriétés de ces nœuds sont transformées en attributs dans l'entité correspondante. Une arête entre deux nœuds est transformée en une relation reliant les entités correspondantes. Une arête étiquetée "ISA" dans le graphe est transformée en une relation "ISA" entre les entités correspondantes dans le schéma E/A.

D'autres travaux se concentrent sur la transformation d'un schéma d'un niveau vers un autre. Par exemple, l'article [35] présente une approche qui comporte deux processus : 1) un processus d'extraction de schéma physique pour une BD MongoDB et 2) un processus de transformation bidirectionnelle entre le schéma physique et le schéma logique. Selon les auteurs, cette transformation est utile pour créer des vues de BD pour divers utilisateurs. Le processus d'extraction de schéma physique se base sur un programme Map-Reduce. Ce processus est composé de quatre étapes. La première étape consiste à appliquer la fonction "Map" sur chaque document composant une collection pour extraire son schéma. Le résultat de cette étape est un ensemble de schémas en format Json. La deuxième étape consiste à appliquer la fonction "Reduce" sur l'ensemble des schémas obtenus dans la première étape. Les schémas de même structure sont réduits à un seul schéma appelé variation de document, qui sera ajoutée à la Collection correspondante. Cette étape retourne, pour chaque collection, un ensemble de variations de document dont chacune contient des champs avec leurs types qui peuvent être soit atomiques, soit des tableaux soit structurés (représentés par des variations de document). La troisième étape consiste à extraire les champs de références exprimant des relations entre les collections. Ces champs sont ajoutés ensuite aux variations de document. La dernière étape consiste à récupérer les statistiques de chaque champ tels que le nombre d'occurrence, la taille min et max et la cardinalité min et max pour les tableaux. Le processus de transformation du schéma physique déjà extrait en un schéma logique est réalisé à l'aide d'un ensemble de cinq règles de transformation. La première règle transforme chaque collection en une entité de même nom. La deuxième transforme chaque variation de document en une variation d'entité. Dans la troisième règle, les champs de types atomiques et tableaux sont transformés en des attributs de types atomiques et tuples respectivement. La quatrième transforme un tableau de variation de document en une agrégation. Enfin, la cinquième consiste à transformer un champ de référence en une référence.

La transformation d'un schéma conceptuel vers un schéma logique ou physique a fait objet de certains travaux de recherche. Le schéma conceptuel est une représentation abstraite des données, tandis que les schémas logique et physique sont des représentations plus concrètes des données. La

transformation du schéma conceptuel vers le schéma logique consiste à traduire les classes, les attributs et les liens en collections (ou tables), champs (ou colonnes) et relations entre les collections (références ou clés étrangères). La transformation du schéma conceptuel vers le schéma physique consiste à tenir compte des contraintes de stockage et de performance de la BD telles que la taille des collections, le nombre de champs et la répartition des données sur les disques. Parmi les travaux qui ont examiné la transformation du schéma conceptuel vers le schéma logique ou physique, on peut citer : [36], dans lequel les auteurs proposent un processus qui transforme un schéma conceptuel, élaboré manuellement sous forme de DCL, en un schéma logique pour une BD documents. Cette transformation suit plusieurs étapes : d'abord, chaque classe est transformée en une collection avec les mêmes attributs. Ensuite, un lien d'association monovalué entre deux classes X et Y est transformé en insérant une référence par exemple vers X dans la collection Y et en imbriquant X dans Y. Puis, un lien d'association multivalué est transformé en une nouvelle collection contenant deux références vers les deux collections reliées ainsi que ces deux collections imbriquées. Enfin, un lien d'héritage est transformé en une nouvelle collection contenant les attributs de la super-classe et les sous-collections imbriquées. Chaque sous-collection contient ses attributs spécifiques ainsi que deux références (vers la sous-collection et la super-collection).

L'article [37] étudie la transformation des liens contenus dans un schéma conceptuel présenté sous forme de DCL en un schéma logique compatible avec une BD documents. Cette transformation est réalisée à l'aide de trois règles de transformation. La première règle consiste à transformer un lien d'héritage en se basant sur deux solutions : la première consiste à supprimer les sous-collections et à déplacer leurs champs dans la super-collection. Un nouveau champ type est ajouté également à la super-collection pour distinguer entre les sous-collections. La deuxième solution implique la suppression de la super-collection et la migration de ses champs dans les sous-collections. La deuxième règle consiste à transformer un lien d'association (0:1) ou (1:1) en se basant sur la cardinalité minimale des deux classes reliées. Si la cardinalité minimale des deux classes est de 0, alors une collection est imbriquée dans l'autre et une référence est insérée dans chaque collection. Si la cardinalité minimale est de 0 pour l'une et de 1 pour l'autre, alors la collection correspondante à la classe ayant une cardinalité minimale de 0 est imbriquée dans l'autre. Si les deux cardinalités minimales sont de 1, alors une collection est imbriquée dans l'autre. Finalement, la troisième règle consiste à transformer un lien d'association (N:N) en insérant une référence dans chaque collection reliée.

L'article [38] propose un processus de transformation d'un schéma conceptuel représenté sous la forme d'un diagramme entité-association (E/A) en un schéma

logique en format JSON adapté à une BD orientée document. Pour réaliser la transformation, un ensemble de règles sont définies en langage naturel pour établir la correspondance entre les éléments d'un diagramme E/A et d'un schéma logique. Ainsi, une entité est transformée en une collection marquée par deux accolades. Les noms d'attributs d'une entité sont également mentionnés et séparés par des virgules dans la collection correspondante. Dans le cas d'un attribut multivalué, son nom est suivi par deux crochets représentant les valeurs possibles, dans le schéma logique. Pour un attribut structuré, son nom est suivi par deux accolades listant tous les attributs composants. Une relation (0:1) ou (0:N) dans le diagramme E/A est transformée en imbriquant un document ou en insérant une référence dans la collection reliée dans le cas où la taille des données dépasse la taille d'un document. Une relation (N:N) est transformée en une référence dans une des deux collections reliées.

Dans [39] et [40], il est proposé un processus de transformation d'un schéma conceptuel exprimé sous forme de DCL vers un schéma physique en format XMI. Le processus est composé de deux sous-processus : le premier permet de transformer un DCL en un schéma logique XMI compatible avec les BD orientées colonnes, documents et graphes et le deuxième permet de transformer le schéma logique en un schéma physique (Cassandra, MongoDB et Neo4J). Ces deux processus sont basés sur MDA qui consiste à proposer des métamodèles source et cible et un ensemble de règles de transformation qui assure le passage de la source vers la cible.

Dans [41], il est proposé un processus de transformation d'un schéma conceptuel sous forme de DCL en un schéma physique compatible avec une BD orientée graphes. Ce processus, basé sur MDA, consiste à appliquer des règles de transformation pour générer un schéma physique conforme à un métamodèle cible. Ainsi, une classe est transformée en un nœud. Un attribut est transformé en une propriété de même nom et de même type dans le nœud correspondant. Un lien d'association entre deux classes est transformé en une arête reliant les deux nœuds correspondants. Un lien d'association qui relie plusieurs classes est transformé en un nœud. Une classe d'association est transformée en un nœud contenant des propriétés.

3.2 Bilan

Dans cette section, nous abordons les limites des principaux travaux de recherche qui sont proches de la problématique que nous traitons : extraction de schéma de BD NoSQL. Pour cela nous dressons un tableau comparatif de leurs processus en se basant sur les critères suivants :

- Modèle de BD NoSQL

- Nombre de collections en entrée
- Niveau de modélisation
- Extraction de différents liens (association, composition et héritage).

D'après le tableau 3.1, il s'avère que les solutions proposées dans l'état de l'art présentent certaines limites. En effet, certains des travaux portant sur l'extraction de schéma logique de BD NoSQL « Schemaless » ne prennent pas en compte aucun liens entre les objets ou bien se limitent à un seul type de lien. Ceci provient du fait que la plupart des plateformes NoSQL disponibles sur le marché ne permettent pas de déclarer explicitement les liens entre les objets. Par ailleurs, d'autres travaux extraient des liens simulés ; des liens créés par l'utilisateur sans que le système en assure le contrôle (et donc la cohérence). Par exemple, dans MongoDB, les liens d'association sont simulés en utilisant une convention d'écriture qui ne permet pas de vérifier l'intégrité référentielle. Cette convention est représentée sous forme d'un champ structuré appelé DBref.

D'autre part, peu de travaux ont étudié l'extraction de schéma conceptuel. Les schémas extraits dans ces travaux sont sémantiquement pauvres ; ils se limitent à certains types de liens et négligent d'autres. Ces travaux ne permettent pas d'extraire un schéma conceptuel sous forme de DCL contenant tous les liens sémantiques qu'on trouve dans UML.

Dans le contexte de notre cas d'étude, notre besoin est spécifique. Il consiste à utiliser un SGBD qui se distingue par sa capacité à créer et gérer efficacement les liens entre les objets. OrientDB est un SGBD qui permet de créer des liens d'association et d'héritage tels que définis par l'ODMG. Ceci en fait un SGBD adapté à notre cas d'étude, où la gestion des liens est cruciale pour atteindre nos objectifs.

3.3 Conclusion

Dans ce chapitre, nous avons examiné les travaux de recherche concernant notre problématique : extraction d'un schéma logique et d'un schéma conceptuel pour une BD NoSQL « Schemaless ». Nous avons identifié divers travaux, abordant notamment : (i) l'extraction de schéma logique, (ii) l'extraction de schéma conceptuel de différents modèles de BD NoSQL (iii) et la transformation d'un schéma d'un niveau vers un autre. Dans les chapitres suivants, nous présentons notre solution pour extraire un double schéma à partir d'une BD NoSQL orientée document « Schemaless ».

Processus / Critères	Modèle NoSQL	Collections	Niveau	Association	Agrégation/ Composition	Héritage
[30]	Colonne	n	Logique	Non	Non	Non
[31]	Graphe	n	Logique	Non	Non	Oui
[34]	Graphe	n	Conceptuel	Oui	Non	Oui
[17], [19], [20], [22], [23], [24], [25]	Document	1	Logique	Non	Non	Non
[21], [26]	Document	n	Logique	Non	Non	Non
[29]	Document	n	Logique	Oui (simulés)	Oui (agrégations simulées)	Oui (simulés)
[32]	Document	1	Conceptuel	Oui	Non	Non
[33]	Document	1	Conceptuel	Non	Oui (composition)	Non

Tableau 3. 1 - Tableau synthétisant les travaux de recherche sur l'extraction de schéma

Chapitre 4
Extraction du schéma logique

Depuis une dizaine d'années, le volume de données s'est accru de façon considérable en raison de la multiplicité des dispositifs informatiques présents dans tous les domaines de nos vies professionnelles, publiques et personnelles. Pour exploiter ces données massives, les entreprises ont besoin de stocker de gros volumes de données (plusieurs téraoctets) et de les gérer avec des dispositifs à la fois souples et puissants.

Actuellement, les BD relationnelles dominent le marché mais elles présentent plusieurs limites principalement liées à leurs structures de données limitées (tables plates) et figées (schéma préétabli). En effet, les BD relationnelles stockent des données sous la forme de tables à deux dimensions. Ceci est inadapté en cas de stockage de données semi-structurées ou non structurées. De plus, la structure d'une table peut être modifiée après sa création. Ce qui peut être contraignant pour les données qui évoluent rapidement, car il n'est pas possible de faire évoluer le schéma de la BD sans modifier les données existantes. Par exemple, si on doit ajouter une nouvelle colonne à une table, on devra également mettre à jour toutes les lignes de la table pour inclure la nouvelle colonne. Ainsi, les BD relationnelles présentent des inconvénients significatifs lorsqu'il s'agit de gérer de gros volumes de données. Leurs structures rigides et leurs schémas préétablis entraînent des coûts de stockage élevés, ce qui devient particulièrement problématique quand le volume de données augmente. Pour faire face à ces limitations, les BD NoSQL ont été proposées. Contrairement aux BD relationnelles, les BD NoSQL sont généralement "Schemaless". Ainsi, il n'est pas nécessaire de déclarer un schéma au préalable ; on peut donc stocker des données en précisant le schéma au fur et à mesure de leur insertion dans la BD. La propriété "Schemaless" apporte un gain important de souplesse dans la manipulation de données, car le schéma peut être adapté à l'évolution des besoins utilisateurs. Par exemple, dans notre cas d'étude médicale présenté dans le chapitre 2, les données enregistrées dans la BD doivent être adaptées en permanence aux évolutions d'une même maladie pour chaque patient. La propriété "Schemaless" répond bien à ce besoin d'évolution des données enregistrées. Cependant, l'absence du schéma rend difficile la tâche d'interrogation de données puisque l'expression de requêtes de type SQL nécessite de connaître les noms de collections, les noms et types des champs ainsi que les relations entre les collections. Pour pallier cette difficulté, l'élaboration des requêtes sur une BD NoSQL peut être confiée à un développeur qui soit a participé à la création de la BD (et donc connaît implicitement son schéma), soit va extraire manuellement le schéma par une consultation, souvent aléatoire, du contenu. C'est à ce niveau que l'on voit l'importance de disposer d'un schéma logique de la BD. Celui-ci définit la structure des données et les relations entre les collections, offrant ainsi une compréhension cohérente de la façon dont les données sont organisées. Mais, il ajoute aux concepts certaines caractéristiques d'implantation selon le type de

plateforme choisi ; par exemple le choix des modes de traduction des liens entre objets. Le schéma logique aide donc à élaborer les requêtes. L'objectif de ce chapitre est de proposer un processus d'extraction dynamique du schéma logique à partir d'une BD NoSQL "Schemaless" afin d'assister les utilisateurs lors du requêtage. Dans notre cas d'étude, nous travaillons avec une BD NoSQL orientée document implémentée dans le système OrientDB. Par conséquent, nous nous concentrons spécifiquement sur ce type de BD.

Ce chapitre est organisé en quatre sections. La section 4.1 présente un aperçu de notre processus d'extraction de schéma logique. Les sections 4.2 et 4.3 présentent successivement la source (BD NoSQL) et la cible (schéma logique) de notre processus. Dans la section 4.4, nous décrivons comment le passage de la source vers la cible est assuré par un ensemble de règles de transformation. Enfin, la section 4.5 conclut ce chapitre en résumant les principaux points abordés et en positionnant notre processus d'extraction de schéma logique par rapport à d'autres processus similaires.

4.1 Survol de notre processus

Dans cette section, nous présentons un aperçu de notre processus d'extraction de schéma logique. Il part d'une BD NoSQL orientée document et génère son schéma logique. Celui-ci permet à un utilisateur de disposer de tous les éléments nécessaires à la rédaction des requêtes de type SQL. Il contient les noms des collections et, pour chacune d'elles, les noms et types des champs ; il doit aussi indiquer les relations entre les collections. Nous avons élaboré le processus ToLogSchema pour les BD NoSQL orientées document. Comme certains systèmes de gestion de BD n'offrent pas tous les éléments structurels nécessaires à notre cas d'étude, par exemple, le système MongoDB ne permettait pas d'exprimer des relations en "mode natif", nous avons mis en œuvre notre processus pour les BD gérées par le système OrientDB. Ce dernier présente des caractéristiques du modèle objet proches des spécifications de l'ODMG, notamment en matière d'expression des liens. Néanmoins, le processus proposé se veut plus large, c'est-à-dire qu'il peut s'appliquer à d'autres BD NoSQL orientées document qui ne comportent pas les caractéristiques mentionnées ci-dessus ; c'est le cas pour le système orienté document MongoDB. Notre processus repose sur MDA que nous avons présentée dans la section 2.4.2. Cette architecture est basée sur la métamodélisation et l'application de règles de transformation. Selon MDA, il convient de spécifier un métamodèle source décrivant la BD NoSQL en entrée du processus, un métamodèle cible décrivant le schéma résultant et enfin des règles de transformations utilisées pour générer la cible souhaitée à partir de la source. Les métamodèles et les règles de transformation peuvent être aisément modifiés si l'on traite un type de BD différent (BD NoSQL orientée graphe, BD

relationnelle, BD multidimensionnelle). Notre cas d'étude n'étant pas limité à une plateforme NoSQL particulière, c'est cette propriété qui nous a conduit à retenir MDA.

La figure 4.1 illustre la source et la cible de notre processus ; à partir d'une BD NoSQL instanciée d'un métamodèle source, notre processus applique des règles de transformation pour générer un schéma logique conforme au métamodèle cible. Ce schéma, généré au format XMI, décrit la structure des données stockées dans cette BD, ce qui facilite la rédaction des requêtes.

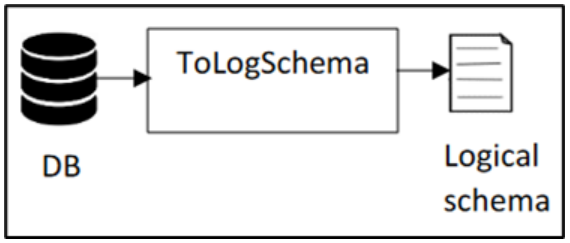


Figure 4. 1 - Le processus ToLogSchema pour l'extraction d'un schéma logique

La source, la cible ainsi que les règles de transformation seront détaillées dans les sections suivantes. Cependant, il est crucial de faire un choix de vocabulaire approprié afin de garantir une bonne compréhension des concepts présents dans la source et la cible. Pour cela, nous avons identifié les vocabulaires spécifiques aux deux SGBD OrientDB et MongoDB. Ce double choix est motivé par deux raisons principales : premièrement, notre cas d'étude concerne une BD NoSQL implémentée sous OrientDB et, deuxièmement, MongoDB est le système le plus largement utilisé dans l'industrie[17]. Dans le tableau 4.1, nous présentons les vocabulaires spécifiques à OrientDB et MongoDB.

OrientDB	**MongoDB**
Classe	Collection
Enregistrement	Document
Propriété	Champ
RID	DBref
Héritage	-

Tableau 4. 1 - Vocabulaire des deux systèmes NoSQL

En se basant sur le tableau 4.1, nous avons pu fixer le vocabulaire pour définir la source et la cible de notre processus dans la suite de l'exposé. La source représente la BD dont on souhaite extraire le schéma, alors que la cible désigne le schéma logique. Les données dans la source et la cible sont organisées en

[17] https://db-engines.com/en/ranking

Collections. Nous avons opté pour le terme Collection de MongoDB, car le terme Classe d'OrientDB est couramment utilisé dans le schéma conceptuel, ce qui peut entraîner une confusion et rendre la compréhension plus complexe. Dans la source, chaque collection regroupe un ensemble de documents et chaque document est composé de champs. Chaque champ a un nom et une valeur. La valeur d'un champ peut être spécifique ; contenant un RID (Record ID) d'un document tel que l'on trouve dans OrientDB ou bien une structure DBRef comme dans MongoDB. Dans la cible, une collection est composée d'un ensemble de champs, où chaque champ est caractérisé par un nom et un type. Une valeur RID ou DBRef dans la source correspond à une référence vers une collection dans la cible. De plus, la cible peut également contenir des liens d'héritage qui sont reportés depuis la source. Le tableau 4.2 résume le vocabulaire décrit ci-dessus.

Source	Cible
Collection	Collection
Document	Absent dans le schéma
Champ	Champ
Valeur	Type
RID ou DBRef	Référence
Héritage	Héritage

Tableau 4. 2 - Vocabulaire de la source et de la cible de notre processus

4.2 Source : BD NoSQL

Dans cette section, nous détaillons la source de notre processus (BD NoSQL orientée document). Pour cela, nous allons présenter successivement des hypothèses de départ que nous avons émises sur la BD, la formalisation des concepts et des cas particuliers des liens présents dans la BD.

4.2.1 Hypothèses de départ

La flexibilité et la permissivité des BD NoSQL orientées document nous a conduits à ne pas aborder ici certains problèmes de conception qui ont été traités par des travaux antérieurs. Ainsi, des "dérives" de conception, telles que la présence de différents types pour un même champ ou de noms de champs synonymes dans les documents d'une même collection, ont été résolues ont pu être résolues [20], [27], [42].

Pour éviter de reprendre ces problèmes et leurs solutions connues, nous avons émis des hypothèses basées sur le fait que notre BD est correctement conçue et organisée (le cas échéant, les contrôles de conformité ont été effectués en amont du processus d'extraction du schéma).

- H1 : une collection de la BD contient des documents sémantiquement homogènes ; par exemple des employés et des voitures sont stockés dans des collections distinctes. Mais des documents peuvent aussi être imbriquées dans une collection sous la forme d'un champ pour exprimer une relation (imbrication) ; il s'agit d'un choix de représentation de la réalité. Par exemple, les voitures peuvent être considérées comme des champs insérés dans des documents d'employés ; dans ce cas, seul "Employés" apparaît sous la forme d'une collection. Une collection regroupe donc des documents ayant la même signification, conformément à la notion ensembliste de relation d'équivalence. Il convient de noter que nous avons pris l'option de regrouper tous les champs apparaissant dans les différents documents d'une collection. Autrement dit, contrairement aux travaux dans [27], nous ne considérons pas différentes versions dans une même collection.
- H2 : Les documents au sein d'une même collection peuvent contenir un nombre variable de champs (propriété "Schemaless"). L'extraction d'un schéma unique pour une collection suppose le regroupement des champs ayant le même nom et le même type. Par exemple, les deux champs phone et telephone de type String ne seront pas regroupés ; ils généreront deux champs distincts.
- H3 : Dans les documents d'une collection, les champs de même nom et de types différents ont été traités au préalable par un processus spécifique appliqué aux BD sources. Ce traitement détecte les incohérences de type de valeurs puis harmonise les champs de même nom, soit automatiquement, soit avec l'intervention d'un administrateur. Ainsi, dans deux documents d'une même collection, on ne peut pas trouver un champ Client# de type Integer et un champ Client# de type String.

Ces trois hypothèses permettent d'obtenir des schémas excluant certaines anomalies de modélisation de la réalité ; en effet, celles-ci pourraient compromettre la validité des schémas extraits. Cependant, il est à noter que ces trois hypothèses réductrices n'altèrent pas l'intérêt de nos propositions puisque d'une part des solutions de traitement existent par ailleurs et d'autre part notre processus et nos schémas ne sont pas impactés. Mais le non-respect de ces hypothèses entrainerait la production d'un schéma non conforme à la réalité et pourrait faire douter de la validité de notre processus.

4.2.2 Formalisation

Avant de procéder à l'extraction automatique du schéma logique d'une BD NoSQL orientée document, nous commençons par la formalisation des concepts qui y sont présents. Pour décrire formellement une telle BD, nous adoptons le paradigme basé sur des couples (clé, valeur).

Notons B une BD NoSQL orientée document contenant n collections tel que B = (N, L) où la clé N désigne le nom de la BD et la valeur L = $\{c_1, c_2, \dots, c_n\}$ est l'ensemble de collections composant B. ∀i∈ [1, n], chaque collection $c_i \in$ L est désignée par un nom et regroupe m documents tels que c_i= (N, D) avec N est le nom de la collection et D = $\{d_1, d_2, \dots, d_m\}$ est l'ensemble des documents de c_i. ∀j∈ [1, m], tout document $d_j \in$ D est associé à un identifiant unique et contient des champs tel que d_j est de la forme (Id, CH) où Id est un identifiant et CH = $\{ch_1, ch_2, \dots, ch_p\}$ représente l'ensemble de p champs composant le document d_j. Tout champ est caractérisé par un nom et une valeur. Dans une collection, le nombre et les noms des champs peuvent être différents d'un document à l'autre. $\forall k \in [1, p]$, un champ $ch_k \in CH$ est un couple (N, V) où N est le nom du champ et V sa valeur. La valeur d'un champ peut être atomique par exemple un entier, une chaîne de caractère ou une valeur booléenne. Elle peut être aussi complexe (structurée ou multivaluée), c'est-à-dire élaborée à l'aide de constructeurs ; par exemple une liste de prénoms, une adresse ou un ensemble de spécialités. Une valeur V structurée est composée par des champs qui sont caractérisés à leurs tours par un nom et une valeur atomique ou complexe tel que V= "Structured" $\{(N_1, V_1), (N_2, V_2), (N_3, V_3), \dots, (N_l, V_l)\}$. Une valeur V multivaluée est composée par des valeurs qui peuvent être atomiques ou complexes tel que V= "Multivalued" $[V_1, V_2, V_3, \dots, V_q]$. Ci-dessous, nous montrons le contenu d'une collection "Patients" extraite d'une BD OrientDB et formalisée en XMI. Cette collection contient un document identifié par le RID "#11:0" et composé de trois champs : "lastName" (type String), "firstNames" (type liste de String), "address" composée de "streetNbr", "streetName", "postalCode" et "phones" (type liste de structures).

```
<Collections name="Patients">
    <Documents identifier="#11:0">
        <Fields name="lastName">
            <Values xsi:type="SourceDB:PrimitiveDB" value="&quot;PETIT&quot;"/>
        </Fields>

        <Fields name="firstNames">
            <Values xsi:type="SourceDB:MultivaluedDB">
                <IsMultivaluedOf xsi:type="SourceDB:PrimitiveDB" value="&quot;Léo&quot;"/>
                <IsMultivaluedOf xsi:type="SourceDB:PrimitiveDB" value="&quot;Victor&quot;"/>
            </Values>
        </Fields>

        <Fields name="address">
            <Values xsi:type="SourceDB:StructuredDB">
                <IsStructuredOf name="streetNbr">
                    <Values xsi:type="SourceDB:PrimitiveDB" value="6"/>
                </IsStructuredOf>
                <IsStructuredOf name="streetName">
                    <Values xsi:type="SourceDB:PrimitiveDB" value="&quot;rue Bellegarde&quot;"/>
                </IsStructuredOf>
                <IsStructuredOf name="postalCode">
                    <Values xsi:type="SourceDB:PrimitiveDB" value="&quot;31000&quot;"/>
                </IsStructuredOf>
            </Values>
        </Fields>

        <Fields name="phones">
            <Values xsi:type="SourceDB:MultivaluedDB">
                <IsMultivaluedOf xsi:type="SourceDB:StructuredDB">
                    <IsStructuredOf name="phoneNbr">
                        <Values xsi:type="SourceDB:PrimitiveDB" value="&quot;0751014106&quot;"/>
                    </IsStructuredOf>
                    <IsStructuredOf name="location">
                        <Values xsi:type="SourceDB:PrimitiveDB" value="&quot;Toulouse&quot;"/>
                    </IsStructuredOf>
                </IsMultivaluedOf>
            </Values>
        </Fields>
    </Documents>
</Collections>
```

Le langage XMI offre une syntaxe précise mais peu lisible. Pour faciliter la lecture, nous utiliserons des raccourcis comme le montre l'exemple suivant :

```
Collection Patients
    Document "#11:0"
        Field lastName : Primitive "PETIT"

        Field firstNames : Multivalued
            Primitive "Léo"
            Primitive "Victor"

        Field address : Structured
            Field streetNbr    : Primitive 6
            Field streetName : Primitive "rue Bellegarde"
            Field postalCode  : Primitive "31000"

        Field phones : Multivalued Structured
            Field phoneNbr : Primitive "0751014106"
            Field location    : Primitive "Toulouse"
```

4.2.3 Cas particuliers

Un champ peut contenir une valeur atomique qui diffère selon le type de SGBD utilisé. Dans le cas d'OrientDB, cette valeur correspond à un RID, qui est un identifiant d'un autre document. Dans MongoDB, cette valeur prend la forme d'une structure DBRef. Dans ce qui suit, nous présentons les deux cas de figure.

- RID (Cas OrientDB)

Un champ dans une collection peut contenir l'identifiant d'un document, sa valeur Vest donc de la forme "#ab:c" tel que : "ab" est l'identifiant du cluster et "c" est la position du document dans le cluster. Cette représentation correspond à une relation entre deux documents. Il convient de noter que lors de la création d'une relation dans une BD implantée sous OrientDB, des paramètres sont enregistrés par le système dans le dictionnaire de la BD (notamment le nom de la collection visée). Cependant, le dictionnaire ne contient pas les noms et les types des champs de chaque collection. La figure 4.2 présente un extrait du dictionnaire de données d'une BD créée sous OrientDB. Le dictionnaire est stocké en format JSON. Pour le visualiser d'une manière plus claire, nous utilisons l'interface "Online JSON Viewer"[18] qui affiche les données sous la forme d'un arbre. Dans cet exemple, le nom d'une collection "Patients" contient le champ "familyDoc"qui exprime une relation vers la collection "Doctors".

Nous présentons ci-dessous un document de la collection "Patients" identifié par le RID "#12:0". Ce document contient un champ nommé "familyDoc" dont la valeur est un RID ("#41:0") pointant vers un autre document de la collection "Doctors"(comme indiqué dans "ReferTo").

Collection Patients
    Document "#12:0"
        Field familyDoc ReferTo Doctors : Primitive "#41:00"

- DBRef (Cas MongoDB)

Contrairement à OrientDB, il convient de noter que la plupart des systèmes NoSQL (MongoDB par exemple) ne supportent pas la création des relations. En revanche, ces relations peuvent être simulées en respectant une structure bien définie. Ainsi, pour exprimer une relation entre deux documents sous MongoDB, la documentation préconise d'utiliser un champ de valeur structurée DBRef[19]. Cette structure contient deux informations : l'identifiant du document et le nom de la collection pointée telle que V = "Structured" $\{(N_1, V_1), (N_2, V_2)\}$

[18] http://jsonviewer.stack.hu/
[19] https://www.mongodb.com/docs/manual/reference/database-references/

avec N_1= “$id”, V_1 de la forme “ObjectId(“..”)"représentant l’OId (ObjectId) du document, N_2 = ”$ref” et V_2 est le nom de la collection pointée. Il faut cependant noter que le système MongoDB ne contrôle pas l'intégrité référentielle pendant la saisie de données dans la BD ; ceci revient à la tâche du logiciel de saisie et de mise à jour.

Ci-dessous, nous montrons une collection "Doctors" contenant un document identifié par l’OId “507f…ea”. Ce document comprend un champ "hisHead" dont la valeur structurée DBRef exprime une relation réflexive.

```
Collection Doctors
    Document "ObjectId(“507f191e810c19729de860ea”)"
        Field hisHead : Structured
            Field $id   : Primitive “ObjectId(“507f1f77bcf86cd799439011”)”
            Field $ref  : Primitive “Doctors”
```

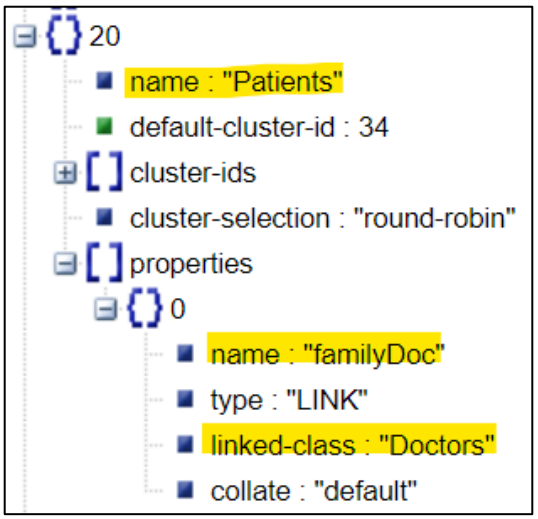


Figure 4. 2 - Extrait d’un dictionnaire de données de BD OrientDB

- Héritage (Cas OrientDB)

OrientDB offre la possibilité d'établir des relations d'héritage entre les collections. Ceci signifie qu'une collection peut hériter de plusieurs autres collections. Comme illustré dans la figure 4.3, les noms des super-collections sont ajoutés au dictionnaire. Par exemple, la collection "Doctors" hérite de la collection "Employees". Ainsi, le nom de la super-collection "Employees" est inclus dans le dictionnaire.

Nous présentons ci-dessous le même exemple implanté sous OrientDB : la collection “Doctors”, caractérisée par le champ “proNbr”, hérite de “Employees”.

Collection Doctors InheritOf Employees
  Document "#40:0"
    Field proNbr : Primitive "XD5577"

Soulignons que MongoDB n'autorise pas la création des relations d'héritage.

Les concepts présents dans une BD sont exprimés dans un métamodèle source. Ce métamodèle est illustré dans la figure 4.4 sous la forme graphique Ecore[20], qui est proche du formalisme UML. Une BD est constituée de collections de documents. Chaque document est composé d'un ou plusieurs champs. Chaque champ peut contenir une valeur atomique, structurée ou multivaluée. En outre, une BD peut contenir des relations telles que "ReferTo" dans le métamodèle, qui indique qu'un champ pointe vers une collection et "InheritOf", qui représente qu'une collection peut hériter de plusieurs autres collections.

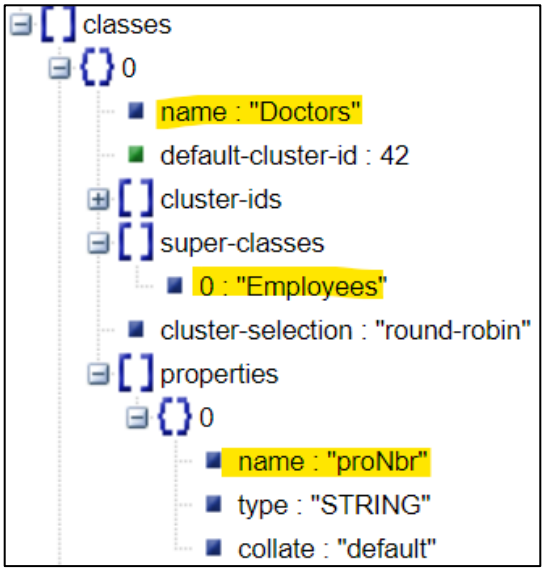


Figure 4. 3 - Extrait d'un dictionnaire de données de BD OrientDB.

4.3 Cible : Schéma logique

L'objectif du processus ToLogSchema est de générer le schéma logique d'une BD NoSQL orientée document. Contrairement aux BD relationnelles où le schéma est distinct de l'extension (c'est-à-dire des données), le schéma d'une BD NoSQL est partiellement intégré dans les données. Partiellement parce que la BD contient les noms des collections et des champs mais on n'y trouve pas leurs types ; or, cette information est déterminante pour élaborer le schéma ainsi que pour connaître la nature des relations entre les collections. Pour formaliser le schéma d'une BD NoSQL, nous reprenons les notations sous la forme (clé, valeur) vues dans la section précédente. Le schéma d'une base B est

[20] https://www.eclipse.org/ecoretools/doc/index.html

un couple (N, S) où N désigne le nom du schéma et S est un ensemble de schémas de collections ; la cardinalité de S est égale au nombre de collections composant B. Le schéma d'une collection $sch \in S$ est un couple (N, CHT) où N désigne le nom de la collection et CHT contient un ensemble de champs typés.

Tout champ typé $cht \in CHT$ est un couple (N, T) où N est le nom du champ et T est son type. Le type d'un champ peut être soit atomique (entier, chaîne de caractères, booléen, etc.), soit structuré (c'est-à-dire constitué d'autres champs), soit multivalué (associé à un type unique), soit une référence (exprimant une relation entre deux collections). Le type d'un champ n'est pas présent dans la BD ; il sera déduit par notre processus. Un type structuré est composé par des champs qui sont caractérisés à leurs tours par un nom et un type atomique, complexe (structuré ou multivalué) ou référence tel que T = "Structured" $\{(N_1, T_1), (N_2, T_2), (N_3, T_3), \dots, (N_l, T_l)\}$. Un type multivalué est construit sur un type qui peut être soit atomique, soit complexe, soit une référence telle que T = "Multivalued" T_1. L'exemple ci-dessous montre le schéma logique d'une collection "Hospitals" selon le formalisme XMI. Ce schéma est composé de deux champs : "hospitalName" de type atomique et "address" de type structuré.

```
<Collections name="Hospitals">
 <Fields name="hospitalName">
  <Types xsi:type="TargetMetamodel:Primitiv" type="EString"/>
 </Fields>
 <Fields name="address">
  <Types xsi:type=" TargetMetamodel:Struct">
   <IsStructuredOf name="streetNbr">
    <Types xsi:type="TargetMetamodel:Primitiv" type="EInt"/>
   </IsStructuredOf>
   <IsStructuredOf name="streetName">
    <Types xsi:type="TargetMetamodel:Primitiv" type="EString"/>
   </IsStructuredOf>
   <IsStructuredOf name="postalCode">
    <Types xsi:type="TargetMetamodel:Primitiv" type="EString"/>
   </IsStructuredOf>
  </Types>
 </Fields>
</Collections>
```

Nous utilisons une syntaxe simplifiée pour représenter un schéma logique :

```
Collection Hospitals
  Field hospitalName : Primitive EString
  Field address : Structured
    Field streetNbr    : Primitive EInt
    Field streetName : Primitive EString
    Field postalCode : Primitive EString
```

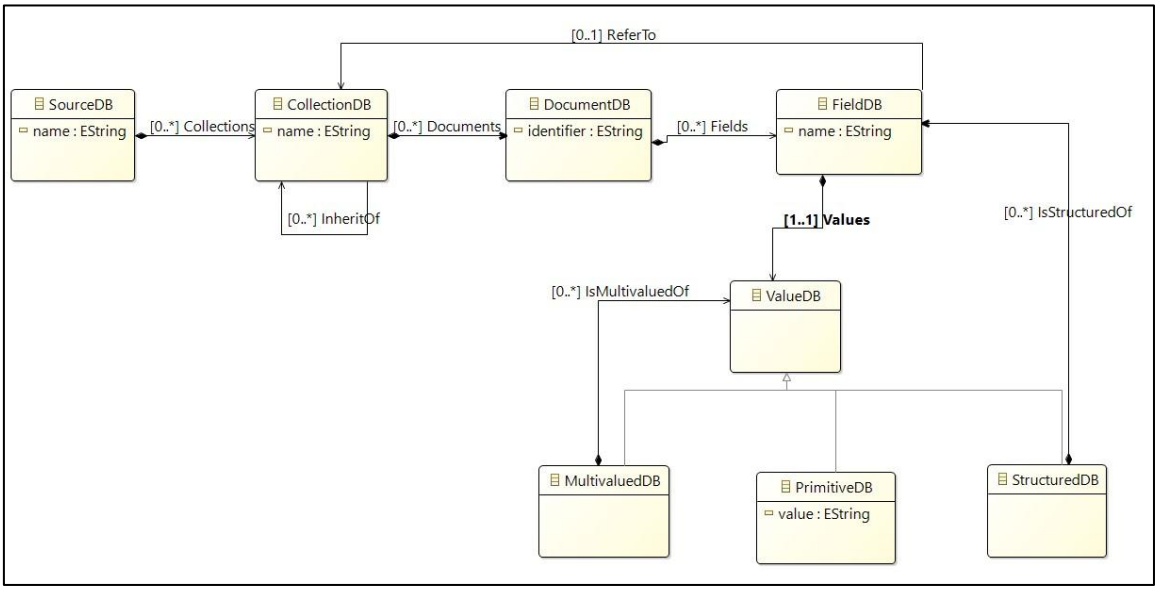


Figure 4. 4 - Métamodèle décrivant la source de notre processus.

- Référence

Dans le schéma logique, une référence est un type qui contient le nom d'une collection référencée. Elle représente une relation entre deux collections et se situe généralement dans l'une des deux collections liées ; elle peut être mono ou multivaluée tel que son type T= "Reference" N avec N est le nom d'une collection référencée. Nous montrons, ci-dessous, une collection "Doctors" contenant deux champs : un champ monovalué "exercise" et un champ multivalué "manageActivities" indiquant l'hôpital où chaque médecin exerce son activité et l'ensemble des tâches qui lui sont attribuées.

```
Collection Doctors
  Field exercise : Reference Hospitals
  Field manageActivities : Multivalued Reference Tasks
```

- Héritage

Le schéma logique mentionne d'autres relations entre les collections notamment l'héritage qui lie une collection à une sous-collection. L'exemple suivant décrit une sous-collection de médecins-spécialistes parmi les médecins ; le champ "speciality" caractérise la sous-collection.

Collection Specialists : InheritOf Doctors
Field speciality : Primitive EString

Le schéma résultant du processus ToLogSchema doit être conforme au métamodèle cible présenté dans la figure 4.5. Ce métamodèle décrit le schéma de chaque collection en spécifiant le nom et le type de chaque champ qu'elle contient. De plus, il mentionne les références et les héritages entre les collections.

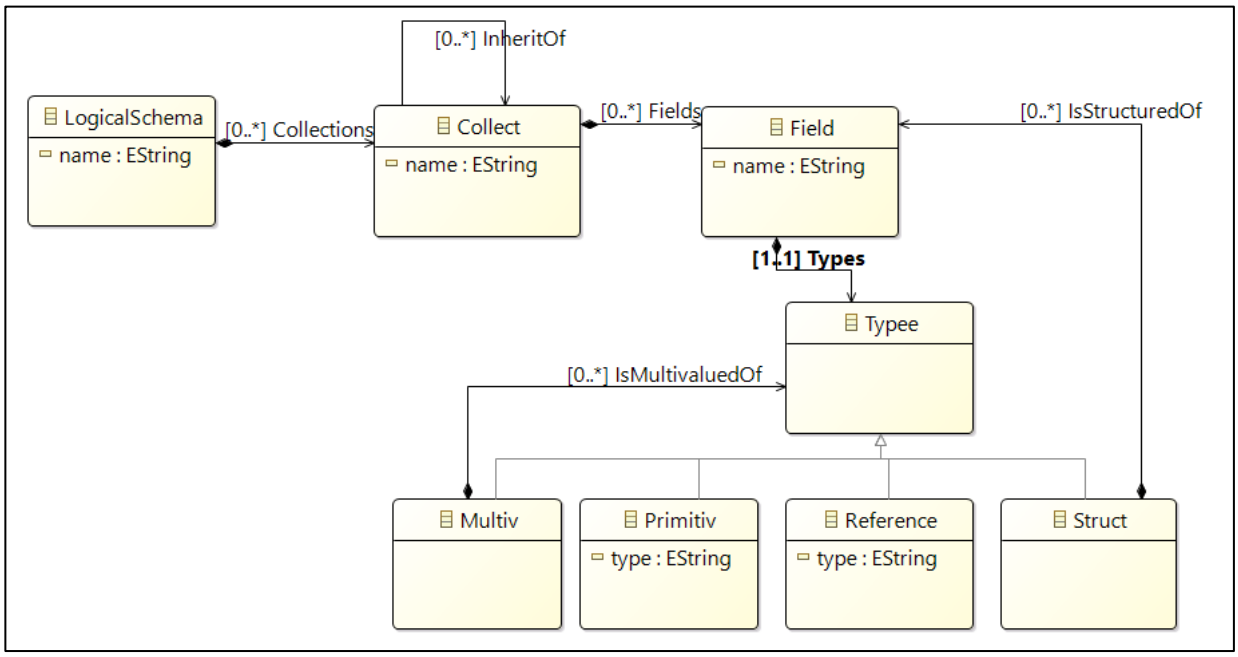


Figure 4. 5 - Métamodèle décrivant le schéma cible de notre processus.

4.4 Application des règles de transformation

Le processus ToLogSchema vise à obtenir le schéma logique d'une BD NoSQL. Il applique une suite de règles de transformation sur la BD après que celle-ci a été mise en conformité avec le métamodèle source en introduisant des balises XMI spécifiques telles que "CollectionDB", "DocumentDB", "FieldDB", "ValueDB". Ces règles peuvent s'appliquer à une BD orientée document créée avec un système tel qu'OrientDB ou MongoDB. Etant donné que le traitement des relations diffère dans ces deux systèmes, nous considérons des règles spécifiques. Le traitement doit donc déterminer (1) la description de chaque collection, c'est-à-dire les champs (nom et type) communs à une collection et (2) les références et les héritages entre les collections.

Cette section présente les règles de transformation exprimées en langage courant et se divise en deux sections : la section 4.4.1 pour présenter les règles d'extraction de schéma de chaque collection et la section 4.4.2 pour présenter les règles d'extraction des références et des héritages.

4.4.1 Collections

Chaque collection regroupe des documents de même sémantique. Ainsi, Patients, Médecins et Examens sont des collections distinctes (conformément à l'Hypothèse H1 de la section 4.2.1). Notre processus ne génère pas des versions de schémas mais plutôt un schéma unique pour chaque collection.

Pour définir formellement ces concepts, nous rappelons les définitions données dans les sections précédentes. Une base notée B est constituée de n collections telle que B = (N, L) où L = $\{c_1, c_2, c_3, \dots, c_n\}$. $\forall i \epsilon [1, p]$, chaque collection quelconque c_i est définie par le couple (N, D) avec D est un ensemble de m documents de c_i. Le schéma d'une collection est défini par la fonction C-Sch. Cette fonction prend en entrée c_i et retourne son schéma défini par le couple (N, CHT) tel que N est le nom de c_i et CHT est un ensemble de champs typés ; C-Sch(c_i) = (N, CHT). En appliquant la fonction Type sur un champ ch = (N, V) avec N son nom et V sa valeur, nous obtenons un champ typé cht∈ CHT tel que cht = (N, T), N son nom avec ch.N = cht.N et T son type qui peut être atomique (entier, chaîne de caractères…) ou complexe. Lorsque des types complexes sont présents dans le schéma de la BD, ils contiennent des constructeurs tels que "Multivalued" et "Strcutured".

Dans la suite, nous donnons les règles de transformation permettant d'extraire le schéma d'une collection y compris les noms et les types de ses champs. L'expression de ces règles est relativement triviale ; seules les transformations de valeurs en types présentent des difficultés de formulation.

R1 : Une BD NoSQL caractérisée par un nom est transformée en un schéma de même nom dans la cible.

R2 : Chaque collection en entrée est transformée en un schéma de collection de même nom.

Pour l'ensemble des documents appartenant à une collection, chaque champ de la forme (N, V) est transformé en un couple (N, T).

R3 : Pour une valeur V atomique (autre qu'un RID**)**, on associe le constructeur "Primitive" au type correspond. Notons que R3 est constituée d'un nombre de sous-règles égal au nombre de types atomiques. Nous présentons ci-dessous trois sous-règles pour extraire le type d'une valeur atomique.

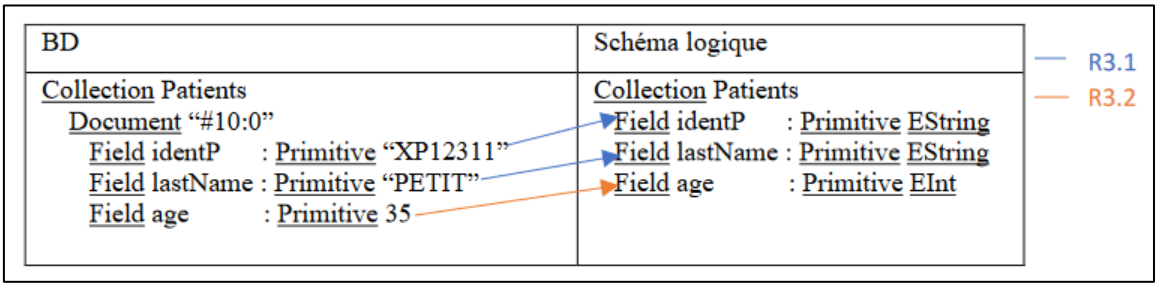
R3.1 : Si V est une chaîne encadrée par les caractères " ", alors son type T= "Primitive EString".

R3.2 : Si V est un entier, alors T= "Primitive EInt".

R3.3 : Si V ∈ {True, False}, alors T= "Primitive EBoolean".

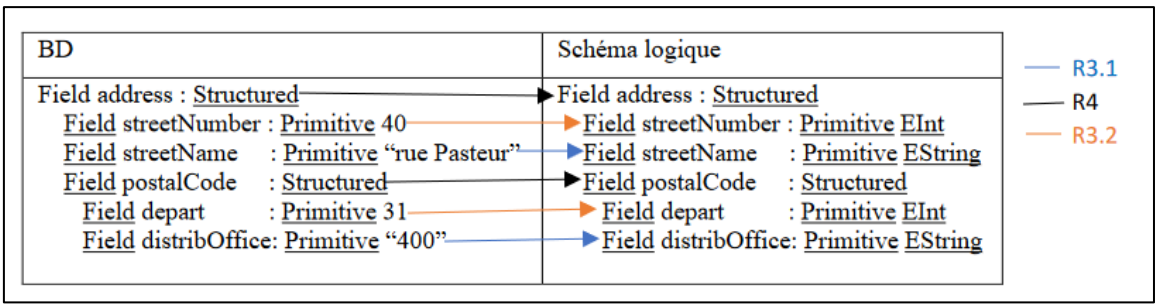
L'exemple suivant présente la transformation de trois champs "identP", "lastName" et "age" de valeurs atomiques. Notre processus applique la règle

R3.1 sur les valeurs des champs “identP” et “lastName” pour extraire le type “Primitive EString” et la règle R3.2 sur la valeur du champ “age” pour extraire le type “Primitive EInt”.

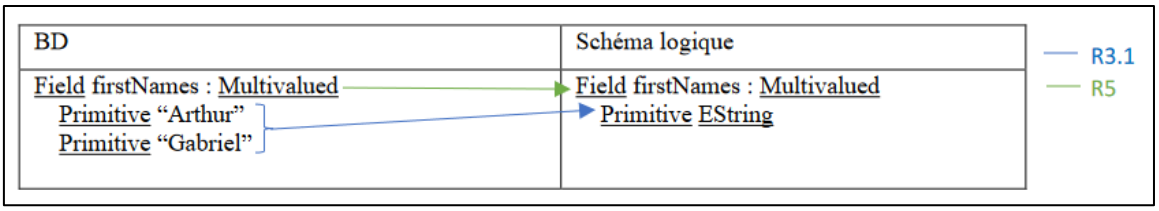


R4 : Une valeur V structurée (autre qu’un DBRef) est transformée en un type structuré indiqué par le constructeur “Structured”. Les règles précédentes sont appliquées de manière récursive sur chaque sous-champ en vue d’extraire son type. Si V= “Structured” $\{(N_1, V_1), (N_2, V_2), (N_3, V_3), \dots, (N_l, V_l)\}$ alors son type est T= “Structured” $\{(N_1, T_1), (N_2, T_2), (N_3, T_3), \dots, (N_l, T_l)\}$.

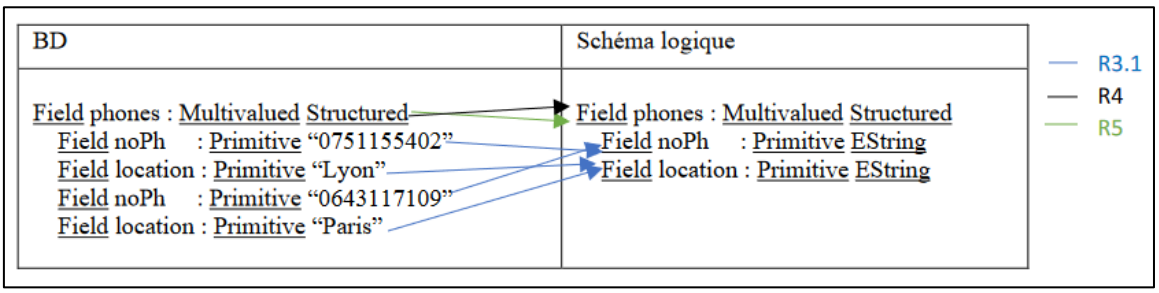
Ci-dessous, nous montrons un exemple de transformation d’une valeur structurée d’un champ “address”. Cette valeur est constituée des sous-champs “streetNumber”, “streetName et “postalCode”. Notre processus applique la règle R4 sur la valeur du champ “address”. Ainsi la règle R4 fait appel à une des règles précédentes pour extraire les types des sous-champs composant “address”. Par exemple, la règle R3.2, appliquée sur la valeur du champ “streetNumber”, génère le type “Primitive EInt”.



R5 : Une valeur V multivaluée est transformée en un type multivalué en l’associant le constructeur “Multivalued” au type composant. La détermination de celui-ci résulte de l’application d’une des règles précédentes. Si V = “Multivalued” $[V_1, V_2, V_3, \dots, V_l]$ alors son type est T= “Multivalued” T_1 avec T_1 est un type composant T. Nous présentons ci-dessous un exemple de transformation d’une valeur multivaluée d’un champ “firstNames”. Cette valeur est composée de deux valeurs “Arthur” et “Gabriel”. Notre processus applique la règle R5 sur la valeur du champ “firstNames”. R5 fait appel à la règle R3.1 pour générer le type “Primitive EString”.



Une valeur V peut être aussi multivaluée de structures telle que V = "Multivalued" $[VS_1, VS_2]$ avec VS_1 et VS_2 deux valeurs structurées, VS_1 = "Structured" $\{(N_1, V_1), (N_2, V_2), \ldots, (N_l, V_l)\}$ et VS_2 = "Structured" $\{(N_1, V_1'), (N_2, V_2'), \ldots, (N_l, V_l')\}$. Le type de V est T= "Multivalued Structured" $[\{(N_1, T_1), (N_2, T_2), \ldots, (N_l, T_l)\}]$. Nous donnons ci-dessous un exemple de transformation d'une valeur multivaluée structurée d'un champ "phones". Cette valeur se compose de deux structures, chacune contenant deux sous-champs : "noPh" et "location". Dans notre processus, la règle R5 est appliquée sur la valeur du champ "phones". R5 utilise la règle R4 sur la première structure de données. À son tour, R4 fait appel à la règle R3.1 pour extraire les types "Primitive EString" à partir des valeurs des champ "noPh" et "location".



4.4.2 Références et héritages

Dans une BD, les relations sont représentées soit par des champs contenant des identifiants de documents (RID ou structure DBRef), soit par des déclarations explicites. Ces deux représentations correspondent respectivement à des références et des héritages dans le schéma logique. Dans les sections 4.4.2.1 et 4.4.2.2, nous décrivons successivement les règles d'extraction des références et des héritages.

4.4.2.1 Références

Il s'agit de déterminer les relations entre les collections d'une BD. Conformément au modèle objet de l'ODMG, une relation est un champ d'un document dont la valeur contient un identifiant pointant vers un autre document. Une telle valeur est marquée par le système. Par exemple, dans OrientDB, elle correspond à un RID et elle est préfixée par le caractère « # ». Tandis que dans MongoDB, elle est enregistrée dans une structure DBRef. La fonction Type appliquée sur un champ ch = (N, V) avec V est un RID ou une

structure DBRef pointant vers un document d'une collection ci, retourne un champ typé cht = (N, T), N son nom avec ch.N = cht.N et T son type (une référence associée au nom de la collection référencée) avec T = "Reference c_i.N".

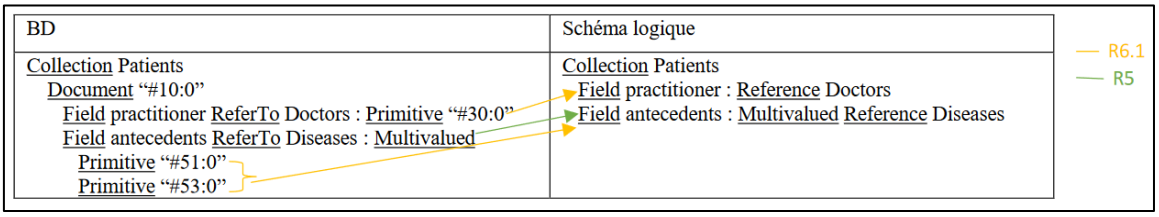
Dans les paragraphes suivants, nous présentons les règles de transformation en langage naturel permettant d'extraire les références.

R6 : Un document peut contenir un champ dont la valeur est un RID ou une structure DBRef, c'est-à-dire une relation vers un autre document. Pour extraire le type d'un tel champ, deux règles (R6.1 et R6.2) sont considérées selon le SGBD utilisé.

La règle R6.1 est appliquée dans le cas du système OrientDB. Une relation entre deux documents se présente sous la forme d'un champ dont la valeur est l'identifiant du document lié. Comme indiqué dans la section 4.2.3, le nom de la collection visée est stocké dans le dictionnaire par le système lors de la première création de ce champ.

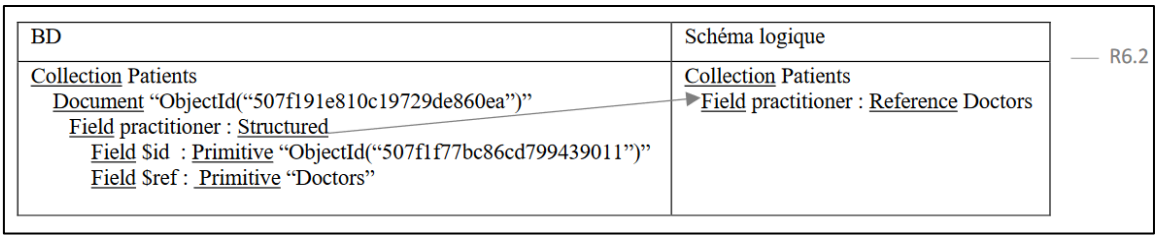
R6.1 : Une valeur de la forme "#ab:c" est transformée en un type « Reference » associé au nom de la collection référencée.

Dans l'exemple ci-dessous, la collection "Patients" contient deux champs, "practitioner" et "antecedents" dont les valeurs sont des RID. Notre processus applique les règles : R6.1 sur la valeur "#30:0" du champ "practitioner" pour extraire le type "Reference Doctors" et R5 sur la valeur du champ "antecedents". R5 utilise R6.1 pour l'appliquer sur la valeur "#51:0" et extraire le type "Multivalued Reference Diseases".



La règle R6.2 s'applique pour MongoDB qui exprime une relation dans un champ structuré DBRef ; celui-ci inclut le nom de la collection ("$ref") et l'identifiant du document pointé ("$id").

R6.2 : Une valeur structurée DBRef est transformée en un type «Reference» associé au nom de la collection référencée. Nous montrons un exemple d'application dans lequel la collection "Patients" contient un champ "practitioner" de valeur structurée DBref exprimant une référence monovalué vers la collection "Doctors". Notre processus applique la règle R6.2 sur la valeur du champ "practitioner" pour extraire le format "Reference Doctors".

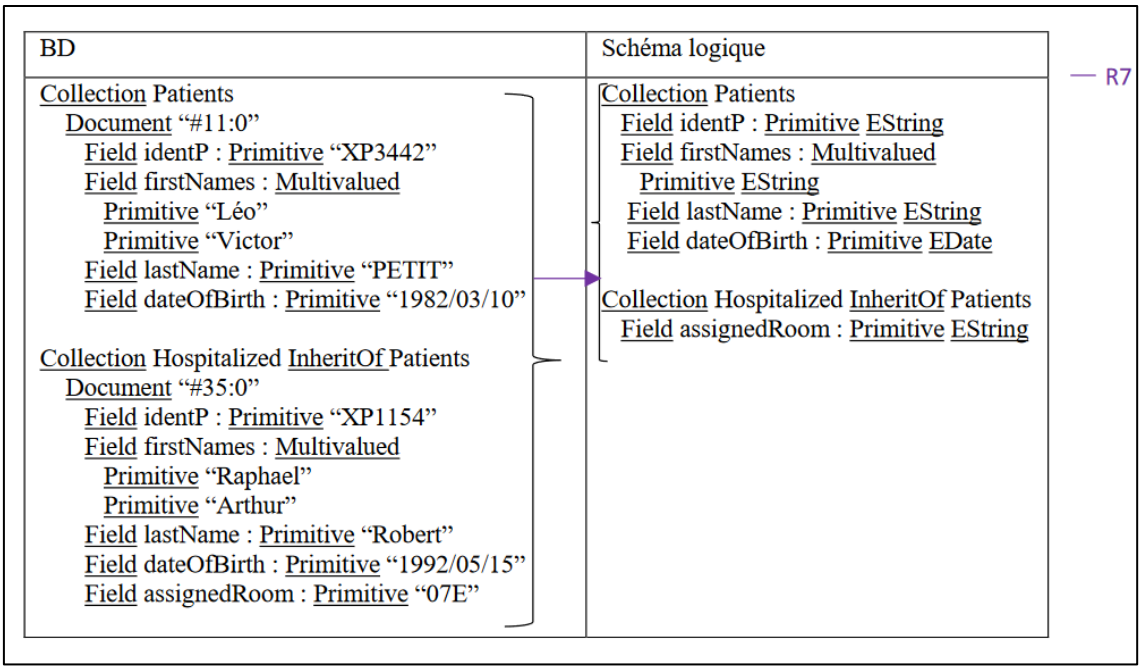


4.4.2.2 Héritages

Toute sous-collection c_2 reliée par un héritage à une super-collection c_1 implique que $c_2 \subseteq c_1$; autrement dit, tout document de c_2 appartient aussi à c_1 et hérite de ses champs. Par exemple, dans OrientDB, la déclaration "Create class c_2 extends c_1 ;" est un préalable à toute insertion d'un document dans une sous-collection. Comme indiqué dans la section 4.2.3, les métadonnées correspondantes sont enregistrées dans le dictionnaire par le système.

R7 : Dans OrientDB, une collection peut hériter de plusieurs autres collections. Les noms des super-collections sont précisés dans le paramètre source/cible "InheritOf".

Nous illustrons l'application de la règle R7 par l'exemple suivant ; la collection "Hospitalized" hérite de "Patients" via le paramètre "InheritOf". Dans la partie droite de la figure, nous présentons le résultat obtenu après l'application de la règle R7. Le schéma de la super-collection "Patients" contient les champs communs "identP", "firstNames", "lastName" et "dateOfBirth", chacun avec son type. Le schéma de la sous-collection "Hospitalized", héritant de "Patients", est caractérisé par le champ "assignedRoom". Notre processus permet de simplifier la conception du schéma logique en utilisant l'héritage pour éviter la duplication des champs communs entre les collections.



4.5 Bilan

Cette section comporte deux sections. La section 4.5.1 souligne les points majeurs qui définissent notre processus. La section 4.5.2 situe notre proposition par rapport aux travaux de l'état de l'art.

4.5.1 Synthèse

La propriété "Schemaless" des BD NoSQL est présentée comme un atout car elle permet une grande flexibilité dans l'évolution de la structure des données. Toutefois, cette absence de schéma peut également constituer un obstacle majeur pour les développeurs et les décideurs. En effet, pour exprimer des requêtes de type SQL, il est nécessaire de connaître précisément le schéma de la BD. Dans ce chapitre, nous avons proposé le processus ToLogSchema qui génère un schéma logique à partir d'une BD NoSQL. Ce dernier affiche le schéma de chaque collection, les champs qu'elle contient avec leurs types atomiques ou complexes, ainsi que les références et les héritages entre les collections. ToLogSchema repose sur MDA où la BD et son schéma doivent être conformes à deux métamodèles source et cible. Le passage de la source à la cible est réalisé grâce à un ensemble de règles de transformation.

4.5.2 Positionnement

Nos travaux se concentrent sur l'extraction du schéma logique d'une BD NoSQL sans schéma. Dans cette section, nous positionnons notre processus par rapport aux travaux cités dans l'état de l'art et traitant la même problématique. À cet

effet, nous présentons le tableau 4.3 qui compare les différents processus, y compris le nôtre (ToLogSchema), en se basant sur plusieurs critères. D'après le tableau 4.3, les processus présentés dans [20], [24] et [25] s'appliquent sur un seul système NoSQL (MongoDB). A la différence de ces processus, celui proposé dans [29] supporte deux systèmes : MongoDB et CouchDB. Or ces systèmes ne supportent pas la création des relations entre les collections de manière native ; les relations sont exprimées en se basant sur une convention d'écriture. ToLogShema supporte deux systèmes NoSQL : OrientDB qui permet de créer des relations entre les collections et MongoDB où les relations sont simulés par l'utilisateur.

En termes d'extraction de relations, dans [17], [19], [20], [21], [22], [23], [24], [25] et [26] l'extraction des relations entre les collections n'est pas prise en compte. Concernant les travaux de [29], le processus proposé extrait les relations entre les collections : les références et les héritages. Ces relations sont simulées par l'utilisateur ; elles ne sont pas nativement supportées par le SGBD. De plus, le processus extrait uniquement les références situées au premier niveau d'une collection. ToLogSchema extrait les références, quelque soit leurs niveaux, dans une collection et les relations d'héritages explicites tels que définis par l'ODM.

A la différence des travaux [17], [19], [20], [21], [22], [23], [24], [25] et [26], ToLogSchema et le processus décrit dans [29] reposent sur l'architecture MDA. Cette approche présente une grande souplesse pour suivre les évolutions techniques des SGBD ainsi que pour prendre en compte de futurs modèles de systèmes NoSQL.

Critères / **Processus**	**[20], [24] et [25]**	**[17], [19], [22] et [23]**	**[21] et [26]**	**[29]**	**ToLogSchema**
Systèmes NoSQL	MongoDB	-	-	MongoDB et CouchDB	MongoDB et OrientDB
Collections	1	1	n	n	n
Référence	Non	Non	Non	Oui (simulés)	Oui
Héritage	Non	Non	Non	Oui (simulés)	Oui
MDE/MDA	Non	Non	Non	Oui	Oui

Tableau 4. 3 - Comparaison de notre proposition avec les travaux de recherche

Chapitre 5
Extraction du schéma conceptuel

Les systèmes NoSQL sont largement reconnus pour leur efficacité dans la gestion de Big Data. Une des caractéristiques les plus notables de ces systèmes est la propriété “Schemaless”, ce qui signifie qu'il n'est pas nécessaire de définir un schéma de données lors de la création de la BD. Ceci offre une grande flexibilité, favorisant ainsi l'évolution du schéma de données et permettant aux utilisateurs d'insérer de nouvelles données de manière autonome sans se heurter aux limitations imposées par les systèmes traditionnels. Cependant, l'absence de schéma dans les systèmes NoSQL complique la manipulation de la BD. Dans le contexte du Big Data, les utilisateurs font face à des défis en raison de l'évolution de leurs besoins. Les données à gérer deviennent de plus en plus volumineuses, complexes et diversifiées. C'est pourquoi, il est primordial d'acquérir une connaissance approfondie des schémas pour gérer efficacement la BD. Le schéma logique, nécessaire pour l’écriture des requêtes, contient des aspects liés à l’organisation interne des données. Le schéma conceptuel fait abstraction des éléments techniques en décrivant uniquement la sémantique des données.

Dans ce chapitre, nous présentons le processus ToConceptSchema permettant de transformer un schéma logique déjà extrait en un schéma conceptuel exprimé en langage textuel XMI. Toutefois, pour plus de clarté, nous présentons également ce schéma sous forme d’un diagramme de classes UML. Ceci permet une représentation visuelle claire et concise du schéma, facilitant ainsi sa compréhension et son analyse. Le schéma conceptuel est composé de classes regroupant des attributs atomiques ou complexes (multivalués/structurés). Il fait apparaître des liens sémantiques entre les classes, tels que des liens d'association, de composition et d'héritage.

La section 5.1 donne un aperçu du mécanisme d'extraction de schéma conceptuel. Ensuite, les sections 5.2 et 5.3 définissent successivement la source (schéma logique) et la cible (schéma conceptuel) de notre processus. Dans la section 5.4, nous présentons les règles de transformation assurant le passage de la source à la cible. Enfin, la section 5.5 positionne notre processus par rapport aux processus d'extraction de schéma conceptuel existants.

5.1 Survol de notre processus d’extraction de schéma conceptuel

Le processus ToConceptSchema extrait le schéma conceptuel à partir du schéma logique produit par notre processus ToLogSchema présenté dans le chapitre 4. La figure 5.1 est un exemple de schéma logique généré par ToLogSchema. Elle montre une collection “Patients” avec les champs “identP”, “firstNames”, “lastName”, “dateOfBirth” et leurs types.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<TargetMetamodel:LogicalSchema name="MedicalDB">
  <Collections name="Patients">
    <Fields name="identP">
      <Types xsi:type="TargetMetamodel:Primitiv" type="EString"/>
    </Fields>
    <Fields name="firstNames">
      <Types xsi:type="TargetMetamodel:Multiv">
        <IsMultivaluedOf xsi:type="TargetMetamodel:Primitiv" type="EString"/>
      </Types>
    </Fields>
    <Fields name="lastName">
      <Types xsi:type="TargetMetamodel:Primitiv" type="EString"/>
    </Fields>
    <Fields name="dateOfBirth">
      <Types xsi:type="TargetMetamodel:Primitiv" type="EDate"/>
    </Fields>
  </Collections>
</TargetMetamodel:LogicalSchema>
```

Figure 5. 1 - Extrait d'un schéma logique XMI produit par notre système

Comme illustré dans la Figure 5.2, les processus ToLogSchema et ToConceptSchema s'insèrent dans le processus global ProdSchemas qui produit deux schémas distincts. Les utilisateurs peuvent ainsi consulter ces deux schémas complémentaires de manière concomitante. Ceci leur permet de 1) comprendre la signification des données (à travers le schéma conceptuel) et 2) d'obtenir une vue d'ensemble de la structure des données et notamment la forme des liens (à travers le schéma logique).

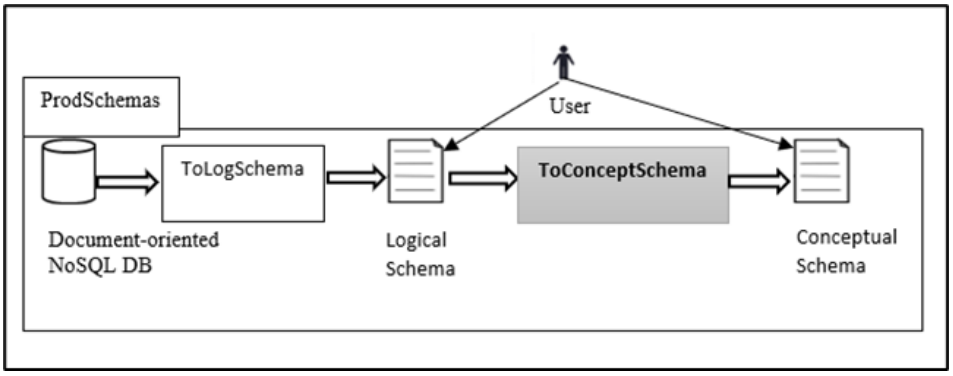


Figure 5. 2 - Architecture du système de production des schémas

ToLogSchema et ToConceptSchema sont basés sur MDA proposant trois niveaux de description des données : le CIM, le PIM et le PSM. La section 2.3 présente une étude de ces niveaux. Nos travaux se situent au niveau PIM puisque les schémas générés sont indépendants d'un système de gestion particulier (MongoDB, OrientDB, CouchDB ou autre). Comme le montre la figure 5.3, le schéma logique se trouve dans le niveau PIM logique tandis que le schéma conceptuel se trouve dans le niveau PIM conceptuel. Le passage d'un niveau à un autre s'effectue par une transformation M2M (Model to Model).

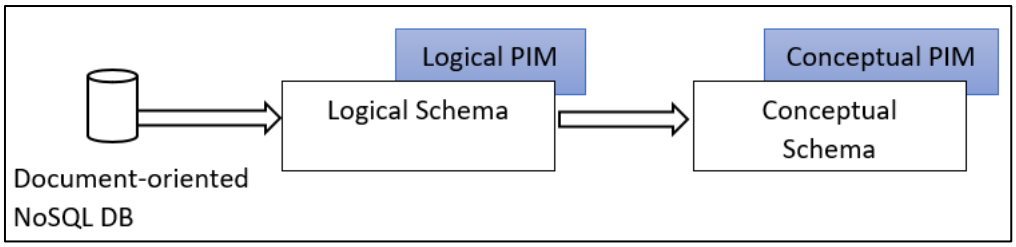


Figure 5. 3 - Les niveaux de modélisation de données

Dans ce chapitre, nous abordons le processus ToConceptSchema d'extraction du schéma conceptuel. Ainsi, un schéma conceptuel est une représentation abstraite d'une BD fournissant une description sémantique des données d'une manière claire et assez proche de la compréhension humaine, facilitant ainsi les analyses de données. Le schéma logique contient aussi les éléments sémantiques mais souvent associés à des considérations techniques ; or, celles-ci sont inutiles à la compréhension des données et nuisent à la clarté du schéma. Par exemple, une simple relation entre deux collections A et B peut être traduite techniquement de différentes façons, soit par une référence dans A, soit par une référence dans B ou encore en mentionnant deux références dans une collection C. Le choix effectué par l'administrateur de données pour traduire le lien figure dans le schéma logique puisque la forme du lien est nécessaire à l'expression des requêtes. Mais ceci ne doit pas figurer dans un schéma conceptuel qui se veut indépendant de tout choix d'organisation des données.

Pour réaliser l'extraction, il convient de définir un métamodèle décrivant la source (le schéma logique), un métamodèle définissant la cible de notre processus (le schéma conceptuel) et un ensemble de règles de transformation permettant le passage de la source à la cible. La source, la cible et les règles de transformation seront présentées en détail dans les sections suivantes.

Comme contrairement aux BD traditionnelles où le vocabulaire utilisé pour décrire les concepts clés tels que les tables, les colonnes, les relations et les clés étrangères est standardisé et normalisé, le vocabulaire dans les Big Data n'est pas standardisé et il peut y avoir plusieurs termes utilisés pour décrire un même concept. On parle ainsi de collection, de table ou de classe pour désigner un dataset typé ou bien de champ ou d'attribut pour une caractéristique. Dans notre processus, afin d'éviter toute confusion et ambiguïté dans la description de la source et de la cible, nous avons décidé d'utiliser un vocabulaire spécifique et conventionnel pour chacun. Cela permet de garantir une interprétation cohérente des concepts et des termes utilisés dans les règles de transformation. Le tableau 5.1 établit la correspondance entre les concepts des schémas logique (source) et conceptuel (cible). Par exemple, une collection du schéma logique correspond à une classe dans le schéma conceptuel, un champ du schéma logique correspond à un attribut dans le schéma conceptuel, etc. En utilisant ce vocabulaire standardisé et cette correspondance claire entre les concepts des deux schémas, nous pouvons élaborer les règles de transformation de

manière précise et cohérente, garantissant ainsi une transformation fiable et efficace du schéma logique vers le schéma conceptuel.

Schéma logique (source)	Schéma conceptuel (cible)
Collection	Classe
Champ	Attribut
Type	Type
Référence	Lien d'association
-	Classe d'associations
Héritage	Lien d'héritage
Champ multivalué structuré	Lien d'agrégation/composition

Tableau 5. 1 - Convention des termes et correspondances

5.2 Source : schéma logique

Dans la section 4.3, nous avons introduit la cible de notre processus ToLogSchema, qui correspond au schéma logique. Rappelons que ce dernier représente l'organisation interne des données et permet donc aux utilisateurs d'exprimer des requêtes sur la BD. Il affiche chaque collection y compris les champs et leurs types qui peuvent être soit atomiques (entier, chaîne de caractères, booléen, etc.), soit structurés (c'est-à-dire constitués d'autres champs dont chacun est associé à son tour à un type), soit multivalués (construit sur un type) ainsi que les références et les héritages entre les collections. Ce même schéma est également utilisé comme source pour notre processus ToConceptSchema. Comme nous l'avons déjà présenté dans le chapitre précédent, nous allons seulement examiner, dans la section suivante, le métamodèle de la cible utilisé par le processus ToConceptSchema.

5.3 Cible : schéma Conceptuel

A partir du schéma logique d'une BD NoSQL, ToConceptSchema génère le schéma conceptuel selon le formalisme XMI. Conformément au tableau 5.1, on considère les concepts de classes, d'attributs, de liens d'association, de composition et d'héritage. Le schéma conceptuel d'une base B est représenté par un couple (N, CS) où N désigne le nom du schéma et CS contient un ensemble de schémas de classes. Le schéma d'une classe cs ∈ CS est un couple (N, TA) où N désigne le nom de la classe et TA décrit un ensemble d'attributs. Tout attribut a ∈ TA est un couple (N, T) où N est le nom de l'attribut et T désigne son type. Le type d'un attribut peut être atomique (entier, chaîne de caractères, booléen, etc.), structuré (c'est-à-dire constitué d'autres attributs) ou multivalué (associé à un type unique). L'exemple suivant montre un extrait de la définition conceptuelle d'une classe "Doctors" dans le formalisme XMI. Cette classe contient trois attributs : "lastName" de type atomique, "firstNames" de

type multivalué et “address” de type structuré de deux attributs “streetName” et “city” de types atomiques.

```
<Classes name="Doctors">
      <Attributes name="lastName">
            <Types xsi:type="TargetMetamodel2:Primitive" type="EString"/>
      </Attributes>
      <Attributes name="firstNames">
            <Types xsi:type="TargetMetamodel2:Multivalued">
                  <IsMultivaluedOf xsi:type="TargetMetamodel2:Primitive" type="EString"/>
            </Types>
      </Attributes>
      <Attributes name="address">
            <Types xsi:type="TargetMetamodel2:Structured">
                  <IsStructuredOf  name="streetName">
                        <Types xsi:type="TargetMetamodel2:Primitive" type="EString"/>
                  </IsStructuredOf>
                  <IsStructuredOf name="city">
                        <Types xsi:type="TargetMetamodel2:Primitive" type="EString"/>
                  </IsStructuredOf>
            </Types>
      </Attributes>
</Classes>
```

Comme dans le cas du schéma logique (chapitre 4), nous avons recours à une syntaxe simplifiée pour représenter les descriptions conceptuelles. Nous donnons une représentation simplifiée du schéma de la classe “Doctors” défini ci-dessus. L'utilisation de cette syntaxe simplifiée permet de rendre la représentation visuelle plus claire et plus concise. Ceci est particulièrement important lorsqu'il s'agit du contenu d'une BD massive contenant des données de structures complexes.

```
Class Doctors
  Attribute lastName   : Primitive EString
  Attribute firstNames : Multivalued EString
  Attribute address    : Structured
    Attribute streetName : Primitive EString
    Attribute City       : Primitive EString
```

Un schéma conceptuel peut contenir différents types de liens entre les classes, on considère généralement les trois liens les plus utilisés dans les BD objet : les liens d'association, d’agrégation et d’héritage. Le lien d’association décrit une relation entre deux classes. Le lien d’agrégation représente un lien de type “tout ou partie”, où une classe peut être composée de plusieurs classes. Enfin, le lien d’héritage exprime une relation d’ordre d’inclusion entre deux classes ; il permet à une classe d'hériter des attributs d'une autre super-classe, ce qui peut faciliter la conception et la maintenance de la BD.

Un lien d’association noté *La* relie n classes avec n >= 2 ; il s’agit en fait du sous-ensemble du produit cartésien de ces n classes. Son schéma est un couple (N, SA) où N est le nom du lien et SA représente l’ensemble des attributs-rôles des classes

reliées par *La*. Chaque attribut-rôle r ∈ SA est décrit par un triplet (N, RelatedClass, Cardinality) avec N est le nom de l'attribut-role, RelatedClass est le nom de la classe reliée par La et Cardinality est le nombre min/max d'occurrences. Dans l'exemple suivant, nous montrons un lien binaire nommé "Belong" qui relie les classes "Prescriptions" et "Consultations". Les termes "pre" et "con" représentent les rôles de chaque classe dans ce lien, tandis que les couples de nombres près de ces termes indiquent leurs nombres d'occurrences respectives (ou cardinalités). Les cardinalités précisent combien de fois chaque classe peut apparaître dans ce lien "Belong".

```
Class Belong
  Attribute pre : Prescriptions (0..*)
  Attribute con : Consultations (0..1)
```

Un lien d'agrégation noté Lg relie une classe composite à une classe composante. Son schéma est un triplet (N, CI, CA) avec N est le nom du lien, CI et CA représentent respectivement les classes composite et composante. Chaque classe est décrite par un triplet (N, Cardinality) avec N est son nom et Cardinality est son nombre minimum/ maximum d'occurrences dans le lien Lg. Un lien de composition est un cas particulier de lien d'agrégation "fort" ; dans ce cas, la suppression d'un objet de la classe composite entraîne la suppression des objets liés dans la classe composante. Nous montrons ci-dessous un exemple de lien entre les notices de médicaments et les effets indésirables qui les composent. La syntaxe proposée ici ne permet pas de nommer explicitement le lien ; celui-ci est identifié par le couple (Notices, SideEffects).

```
Class SideEffects IsComponentOf Notices
  Attribute description : Primitive EString
```

Un lien d'héritage noté Lh exprime une généralisation/spécialisation entre une classe et une de ses sous-classes. Dans la mesure où il s'agit d'une inclusion, les objets d'une sous-classe n'ont pas d'identifiant propre ; plus concrètement, un objet d'une sous-classe appartient également à la super-classe et bénéficient du même identifiant. Par conséquent, le schéma d'un lien d'héritage correspond à la définition de la sous-classe. Il s'agit d'un triplet (N, SN, A) où N est le nom d'une super-classe tel que, SN correspond au nom donné à la sous-classe et A représente l'ensemble des attributs spécifiques de la sous-classe. Dans l'exemple suivant, nous observons que la classe "HospPatients" désignant la sous-classe des patients hospitalisés hérite de la classe "HosAmb" désignant l'ensemble des patients-ambulatoires. Ceci signifie que la sous-classe peut accéder aux attributs communs et aux attributs spécifiques tels que "from" et "to" indiquant la date de début et la date de fin d'hospitalisation.

```
Class HospPatients InheritOf HosAmb
  Attribute from : Primitive EDate
  Attribute to   : Primitive EDate
```

Le métamodèle représenté dans la figure 5.4, basé sur le langage Ecore, permet de décrire l'ensemble des éléments que nous avons définis précédemment. Ce

métamodèle est utilisé pour garantir la conformité du schéma conceptuel généré par notre processus ToConceptSchema à une structure bien définie.

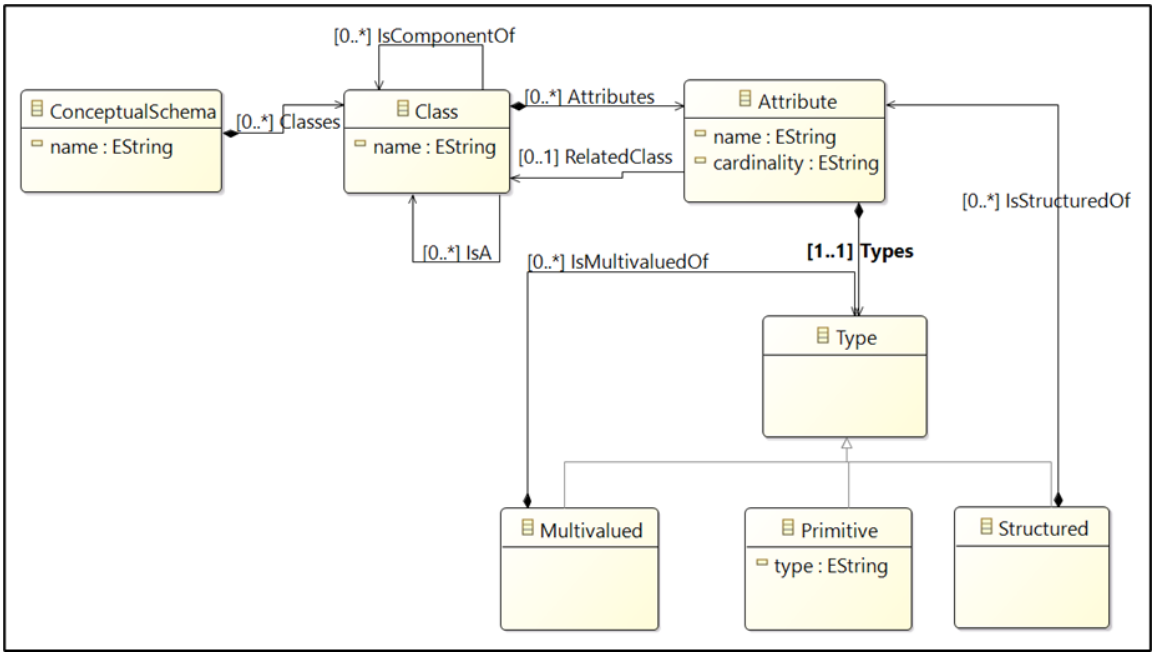


Figure 5. 4 - Métamodèle décrivant la cible de notre

ToConceptSchema est un processus qui permet de transformer automatiquement un schéma logique en un schéma conceptuel, en utilisant un ensemble de règles de transformation. Ces deux schémas sont conformes aux métamodèles sources et cibles décrits ci-dessus. Les règles de transformation sont décrites dans la section suivante. Les résultats de ces transformations sont présentés sous deux formes différentes : en langage XMI simplifié et sous la forme de diagrammes UML.

5.4 Règles de transformation

Notre processus applique un ensemble de règles de transformation sur le schéma logique en entrée et génère automatiquement un schéma conceptuel conforme au métamodèle cible. Il s'agit d'un mécanisme automatique de rétroconception qui permet de passer d'un schéma logique à un schéma conceptuel (en effet, dans une démarche classique d'analyse/conception d'une BD, on élabore les schémas dans l'ordre inverse). Le passage logique → conceptuel consiste donc à éliminer les aspects techniques présents dans le schéma logique. Mais, il convient de noter que le schéma logique est parfois plus pauvre que le conceptuel ; en effet, il ne matérialise pas toujours l'ensemble des éléments conceptuels qui ont présidé à la création de la BD. C'est par exemple le cas de certaines propriétés de relations telles que les cardinalités, qui ne sont pas mentionnées dans le schéma logique. Le schéma conceptuel généré pourra donc contenir des incomplétudes ou des ambiguïtés qui seront signalées lors de l'exposé des règles de transformation.

Dans un schéma logique, les collections peuvent présenter des structures hiérarchiques profondes. Ces structures de données sont transformées en termes identiques lors du passage du logique au conceptuel. Il en est de même pour les types de données standard qui sont reportés dans le schéma conceptuel selon les mêmes termes. Des règles de transformation permettent de transformer ces structures logiques dans le schéma conceptuel. Dans la suite, nous présentons les règles relatives à la transformation du schéma logique d'une collection, d'une référence (monovaluée et multivaluée), d'un champ multivalué structuré et d'un héritage.

5.4.1 Schéma logique d'une collection

Cette section vise à présenter les règles permettant de transformer un schéma logique de collection en un schéma conceptuel de classe. Bien que les premières règles soient triviales, nous avons souhaité donner une liste exhaustive.

Règle 1 : un schéma logique caractérisé par un nom est transformé en un schéma conceptuel de même nom.

Règle 2 : un schéma logique d'une collection, représenté par s = (N, CHT) avec N son nom et CHT un ensemble de champs typés, est transformé en un schéma conceptuel de classe, représenté, cs = (N, TA) avec N son nom et TA un ensemble d'attributs typés. Les deux schémas sont de même nom tel que s.N = cs.N.

Règle 3 : chaque champ cht= (N, T) d'une collection est transformé en un attribut a = (N, T) de même nom tel que cht.N = a.N.

Règle 4 : si le type d'un champ est atomique alors il est transformé en un type atomique tel que cht.T = a.T.

Règle 5 : si le type d'un champ est structuré alors il est transformé en un type structuré tel que cht.T = a.T = "Structured" $\{(N_1, T_1), (N_2, T_2), (N_3, T_3), \ \ldots, (N_l, T_l)\}$.

Règle 6 : si le type d'un champ est multivalué alors il est transformé en un type multivalué tel que cht.T= a.T = "Multivalued" T_1. Nous présentons ci-dessous un exemple de transformation d'une collection "Patients" contenant trois champs : "lastName" (de type "String"), "firstNames" (de type multivalué) et "address" (de type structuré). Pour transformer le schéma logique de la collection "Patients" en un schéma conceptuel, notre processus applique la règle 4 sur le champ "lastName" pour extraire le type "Primitive EString", la règle 6 sur le champ "firstNames" qui à son tour fait appel à la règle R4 pour déduire le type "Primitive EString" et la règle 5 sur le champ "address".

Schéma logique	Schéma conceptuel
Collection Patients Field lastName : Primitive EString Field firstNames : Multivalued Primitive EString Field address : Structured Field streetNumber : Primitive EInt Field streetName : Primitive EString Field postalCode : Primitive EString	Class Patients Attribute lastName : Primitive EString Attribute firstNames : Multivalued Primitive EString Attribute address : Structured Attribute streetNumber : Primitive EInt Attribute streetName : Primitive EString Attribute postalCode : Primitive EString

5.4.2 Référence

Dans le schéma logique, une référence est un champ associé au nom d'une collection référencée ; elle établit donc une relation entre deux collections. Une référence peut être située au premier niveau d'une collection ou imbriquée dans une structure ; les traitements diffèrent selon les cas. De plus, une référence peut être monovaluée ou multivaluée.

Une référence du schéma logique est traduite dans le diagramme UML par un lien d'association. Ce lien, qui exprime l'ensemble des relations entre les objets, est formalisé par une classe de relations. Dans UML, ce lien est représenté par un arc entre les deux classes. Lorsque ce lien est associé à des attributs, il est alors représenté par une classe d'associations UML.

5.4.2.1 Référence monovaluée

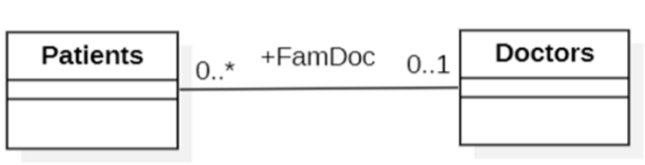
Lorsque le type d'un champ est une référence dans le schéma logique, alors la règle de transformation suivante est appliquée.

Règle 7 : une référence R située au premier niveau dans une collection CA et référençant une collection CB, est transformée en une nouvelle classe dans le schéma conceptuel. Cette classe, de même nom que R, établit un lien entre les classes CA et CB. Soit R= (N, T) une référence pointant vers CB tel que son type T= "Reference CB.N" et "La" la nouvelle classe créée tel que La= (N, SA) où La.N = R.N et SA ={role_CA, role_CB} est l'ensemble des attributs-rôles de CA et CB. L'attribut role_CA est décrit par un quadruplet (N, RelatedClass, Cardinality) avec N est le nom de l'attribut-role, RelatedClass = CA est le nom de la classe reliée et Cardinality = (0..*). L'attribut role_CB est décrit par le quadruplet (N, RelatedClass, Cardinality) avec N est le nom de l'attribut-role, RelatedClass = CB et Cardinality = (0..1).

Prenons l'exemple d'un schéma logique dans lequel la collection "Patients" contient une référence "famDoc" ; ce champ désigne le "family medecine physicians" et pointe vers la collection "Doctors". Dans le schéma conceptuel en cours d'élaboration et qui contient déjà les classes "Patients" et "Doctors", la règle 7 transforme la référence en une nouvelle classe "FamDoc".

Schéma logique	Schéma conceptuel
Collection Patients … Field famDoc : References Doctors	Class Patients … Class FamDoc Attribute pat : Patients (0..*) Attribute doc : Doctors (0..1)

Il convient de noter que les noms d'attributs "pat" et "doc" exprimant les rôles des classes dans le lien "FamDoc", ne sont pas présents dans le schéma logique ; ils sont donc générés automatiquement à partir du nom des classes reliées (minimum 3 caractères). Dans le cas d'une référence réflexive, les attributs-rôles sont distingués par les suffixes -1 et -2. La référence monovaluée "famDoc" permet de fixer la cardinalité maximale à 1 sur la classe "Doctors". Quand les cardinalités ne peuvent pas être extraites du schéma logique, elles sont fixées dans leurs termes les plus généraux, c'est-à-dire (0..*). Notre processus ToConceptSchema ne produit pas la forme graphique du schéma conceptuel. Mais, pour plus de clarté, nous donnons ci-dessous le DCL UML correspondant ; la classe "FamDoc" est représentée par un arc étiqueté et associé aux cardinalités.



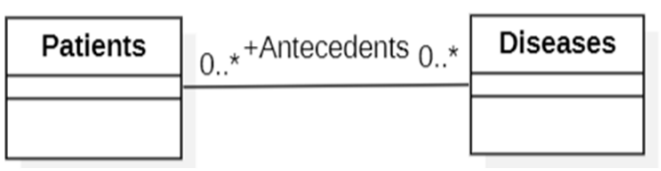
5.4.2.2 Référence multivaluée

Une référence peut être associée à un constructeur d'ensemble ; dans ce cas, elle traduit un lien binaire multivaluée.

Règle 8 : une référence R multivaluée, située au premier niveau dans une collection CA et référençant une collection CB, est transformée en une nouvelle classe de même nom que R et est associée à une cardinalité (0..*) sur CB. Soit R= (N, T) une référence pointant vers CB telle que T = "Multivalued Reference CB.N" et "La" une nouvelle classe tel que La= (N, SA) où La.N = R.N, SA = {role_CA, role_CB}, role_CA=(N, relatedClass, Cardinality) avec Cardinality = (0..*). L'attribut role_CB = (N, RelatedClass, Cardinality) avec Cardinality = (0..*). Dans l'exemple suivant, la collection "Patients" comprend une référence nommée "antecedents". Cette dernière désigne l'ensemble des maladies d'un patient et pointe vers la collection "Diseases".

Schéma logique	Schéma conceptuel
Collection Patients Field antecedents : Multivalued Reference Diseases	Class Patients … Class Antecedents Attribute pat : Patients (0..*) Attribute dis : Diseases (0..*)

Nous donnons, ci-dessous, le DCL correspondant au schéma conceptuel de l'exemple ; la classe "Antecedents" se traduit par un arc entre les classes "Patients" et "Diseases" étiqueté par le nom du lien et associé aux cardinalités.



Nous venons de voir les deux formes de relations les plus courantes dans les schémas logiques de BD ; les références sont situées au premier niveau de description d'une collection. Mais, nous allons considérer des formes particulières que l'on peut aussi rencontrer. En effet, le schéma logique peut contenir des références situées dans un niveau hiérarchique plus profond, c'est-à-dire dans des champs structurés.

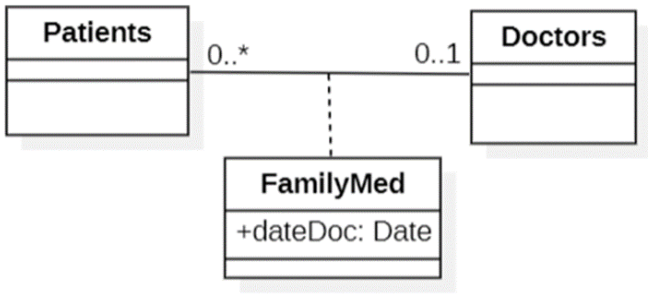
5.4.2.3 Référence caractérisée par des champs

Dans le schéma logique, une référence peut être caractérisée par des champs ; référence et champs sont regroupés au sein d'une structure. Par exemple, un "family medecine physicians" a été choisi par un patient à une certaine date. Dans ce cas, la référence vers le médecin ainsi que la date sont imbriquées dans une même structure de données ; la référence n'est donc plus située au premier niveau de la collection (voir exemple suivant).

Règle 9 : une référence R (mono ou multivaluée) située à un niveau > 1 du schéma d'une collection CA et référençant une collection CB, est extraite de CA avec sa structure immédiatement supérieure. Cette structure S et la référence sont transformées en une nouvelle classe de nom S. Soit S= (N, T) un champ structuré tel que son type T= "Structured" $\{(N_1^R, T_1^R), (N_2, T_{2)}), (N_3, T_3), \dots, (N_l, T_l)\}$ où R = (N_1^R, T_1^R) une référence monovaluée située dans S avec T_1^R= "Reference CB.N". La nouvelle classe créée est telle que La= (N, SA) où La.N = S.N, SA ={role_CA, role_CB, $(N_2, T_{2)}), (N_3, T_3), \dots, (N_l, T_l)\}$ représente l'ensemble des attributs-rôles de CA et CB ainsi que les attributs-non liens résultant de la transformation des champs situés dans S. L'attribut role_CA = (N, RelatedClass, Cardinality) avec Cardinality = (0..*). L'attribut role_CB = (N, RelatedClass, Cardinality) avec Cardinality = (0..1)

Schéma logique	Schéma conceptuel
Collection Patients Field familyMed : Structured Field famDoc : Reference Doctors Field dateDoc : Primitive EDate	Class Patients … Class FamilyMed Attribute pat : Patients (0..*) Attribute doc : Doctors (0..1) Attribute dateDoc : Primitive EDate

Le schéma conceptuel est le suivant ; la classe “FamilyMed” se traduit par une classe d’associations entre les classes “Patients” et “Doctors” contenant l’attribut “dateDoc” et associée aux cardinalités.



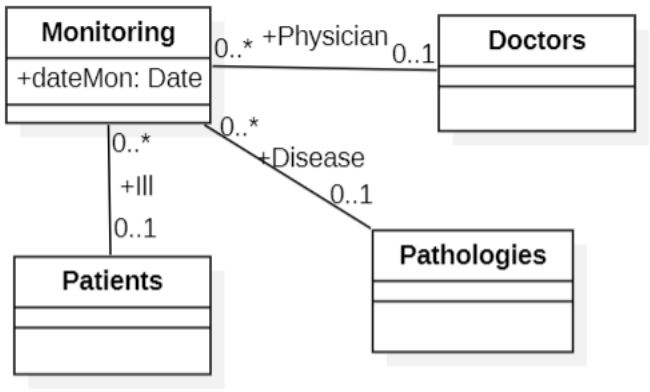
5.4.2.4 Collection de références

Nous avons vu que pour traduire une relation binaire entre deux collections, une référence peut être insérée dans une des deux collections. Pour exprimer une relation n-aire avec n >= 2, les références vers les collections reliées sont situées dans une collection distincte. C’est le cas de la relation ternaire présentée ci-dessous, où l'on observe la relation "Monitoring" qui associe un patient, une pathologie et un médecin. Les règles 7 et 8 sont appliquées sur la collection ; si la collection “Monitoring” ne contient pas de champs autres que les références, alors le schéma de la classe correspondante ne contient aucun attribut.

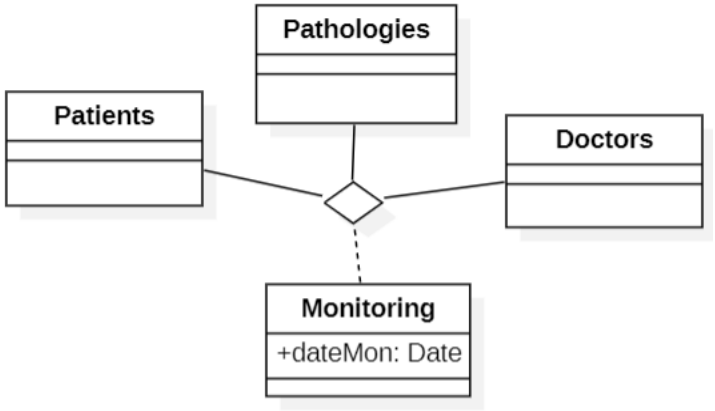
Schéma logique	Schéma conceptuel
Collection Monitoring Field ill : Reference Patients Field physician : Reference Doctors Field disease : Reference Pathologies Field dateMon : Primitive EDate	Class Monitoring Attribute dateMon : Primitive EDate Class Ill Attribute mon : Monitoring (0..*) Attribute pat : Patients (0..1) Class Physician Attribute mon : Monitoring (0..*) Attribute doc : Doctors (0..1) Class Disease Attribute mon : Monitoring (0..*) Attribute pat : Pathologies (0..1)

Le DCL d’UML correspondant au schéma conceptuel est le suivant : les classes “Ill”, "Physician" et “Disease” se traduisent respectivement par des arcs reliant la classe

“Monitoring” à “Patients”, “Monitoring” à “Doctors” et “Monitoring” à “Pathologies”.



Dans cet exemple, notre connaissance du contexte médical nous permet de déduire que la collection Monitoring correspond à un lien ternaire (classe d’associations dans le modèle UML) entre Patients, Doctors et Pathologies. Le DCL UML devrait ressembler à celui illustré ci-dessous.



Mais un tel contexte ne peut pas être systématiquement déduit du schéma logique ; cette limite est due au processus de rétroconception. En effet, si nous prenons le cas d’une collection “Patients” dont le schéma logique est décrit comme suit :

```
Collection Patients
   Field nom      : Primitive EString
   Field famDoc : Reference Doctors
   Field address : Reference Cities
```

Nous ne pouvons pas déduire de cet exemple que “Patients” est un lien (ici binaire) entre “Doctors” et “Cities”. Par conséquent, les règles 7 et 8 doivent s’appliquer sur les références “famDoc” et “address” ; le schéma conceptuel qui en résulte est le suivant :

```
Class Patients
   Attribute nom : Primitive EString

Class FamDoc
   Attribute pat : Patients (0..*)
   Attribute doc : Doctors (0..1)

Class Address
   Attribute pat : Patients (0..*)
   Attribute cit  : Cities (0..1)
```

Un schéma logique peut contenir non seulement des références mais aussi des champs complexes (multivalués structurés) et des héritages. Nous présentons dans les sections 5.4.3 et 5.4.4 les règles de transformation d'un champ multivalué structuré et d'un héritage.

5.4.3 Champ multivalué structuré

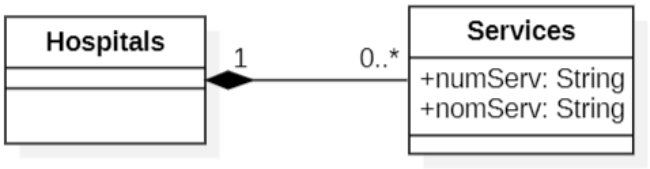
Dans le schéma logique, une collection peut contenir des champs multivalués structurés. Ces structures de données particulières correspondent généralement[21] à des liens de composition entre une classe composite et une classe "imbriquée" (composante). Par exemple, les hôpitaux sont composés de services qui, eux-mêmes, sont composés de chambres. Ces structures répondent bien à la définition des liens de composition où la suppression d'un objet composite entraîne la suppression de ses composantes.

Règle 10 : dans une collection X, un champ multivalué structuré est extrait de X et considéré comme une nouvelle collection Y ; la règle 10 s'applique sur la collection Y de manière récursive. Les collections X et Y sont transformées en classes liées par un lien de composition. Soit cht= (N, T) un champ multivalué structuré tel que N est son nom et T son type ; T = "Multivalued Structured" $\{(N_1, T_1), (N_2, T_2), (N_3, T_3),\ldots, (N_l, T_l)\}$. Ce champ devient une nouvelle classe Y telle que Y = (N, TA) où cht.N = Y.N et TA est l'ensemble d'attributs composants T. Ci-dessous un exemple qui illustre une collection appelée "Hospitals" constituée de services.

Schéma logique	Schéma conceptuel
Collection Hospitals Field services Multivalued Structured Field numServ : Primitive EString Field nomServ : Primitive EString	Class Hospitals … Class Services IsComponentOf Hospitals Attribute numServ : Primitive EString Attribute nomServ : Primitive EString

Nous présentons dans la figure ci-dessous le DCL d'UML correspondant au schéma conceptuel de l'exemple ; les classes "Hospitals" et "Services" sont reliées par un lien de composition associé aux cardinalités.

[21] Ces structures n'expriment pas toujours des liens de composition ; voir Chapitre 7



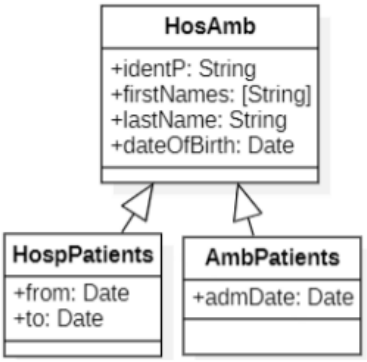
5.4.4 Héritage

Un héritage au niveau logique peut être exprimé de façons différentes selon les fonctionnalités offertes par les plateformes NoSQL. Ainsi, un schéma logique de BD OrientDB peut contenir explicitement un lien “InheritOf” ; sa traduction dans le schéma conceptuel est alors évidente. Mais il est aussi possible d’exprimer implicitement un lien d’héritage lorsque le système NoSQL n’offre pas de mécanisme idoine comme dans MongoDB. Dans notre cas, le schéma logique peut contenir des collections distinctes ayant des champs communs. Ces champs sont factorisés dans une classe générique en faisant apparaître un lien d’héritage. Cependant, un mauvais choix de noms des champs dans des collections différentes (par exemple Code pour identifier à la fois un employé et un produit), pouvait induire des liens d’héritage inappropriés. Nous avons donc introduit par paramétrage la contrainte d’avoir au moins quatre champs communs pour créer un lien d’héritage.

Règle 11 : dans deux collections X et Y, la présence d’au moins quatre champs communs conduit à la création d’une classe générique G regroupant ces champs et dont le nom est formé en concaténant les trois premières lettres des noms de ses sous-classes. Des liens d’héritage sont établis entre G et les classes X et Y dont on extrait les attributs factorisés. Par exemple, HospPatients et AmbPatients sont des sous-classes de la classe générique HosAmb.

Schéma logique	Schéma conceptuel
Collection HospPatients Field identP : Primitive EString Field lastName : Primitive EString Field firstNames : Multivalued EString Field dateOfBirth : Primitive EString Field from : Primitive EDate Field to : Primitive EDate Collection AmbPatients Field firstNames : Multivalued EString Field identP : Primitive EString Field lastName : Primitive EString Field admDate : Primitive EDate Field dateOfBirth : Primitive EString	Class HosAmb Attribute identP : Primitive EString Attribute lastName : Primitive EString Attribute firstNames : Multivalued EString Attribute dateOfBirth : Primitive EString Class HospPatients IsA HosAmb Attribute from : Primitive EDate Attribute to : Primitive EDate Class AmbPatients IsA HosAmb Attribute admDate : Primitive EDate

Le DCL UML correspondant au schéma conceptuel de l’exemple est le suivant ; le lien “IsA” se traduit par un lien d’héritage entre la classe “HosAmb” et les classes “HospPatients” et “AmbPatients”.



Les règles de transformation exposées précédemment sont résumées dans le tableau 5.2 pour une meilleure compréhension.

Règle de transformation	**Description**
Règle 1	transforme un schéma logique en un schéma conceptuel de même nom
Règle 2	transforme une collection en une classe de même nom
Règle 3	transforme un champ en un attribut de même nom
Règle 4, 5, 6	transforme un type atomique ou bien complexe
Règle 7	transforme une référence monovalué en une classe. Cette classe traduit un lien d'association monovalué dans UML
Règle 8	transforme une référence multivalué en une classe. Cette classe traduit un lien d'association multivalué dans UML
Règle 9	transforme une référence caractérisée par des champs en une classe contenant des attributs. Cette classe traduit une classe d'association dans UML

Tableau 5. 2 - Tableau récapitulatif des règles de transformation

Dans cette section, nous avons présenté onze règles de transformation. En appliquant ces règles exprimées en langage QVT sur un schéma logique, notre processus génère automatiquement le schéma conceptuel correspondant conformément aux principes MDA.

5.5 Bilan

Notre Cette section met tout d'abord l'accent sur les points majeurs de notre processus puis elle positionne notre proposition par rapport aux travaux de l'état de l'art.

5.5.1 Synthèse

Les BD NoSQL sont généralement "Schemaless", ce qui signifie qu'elles ne disposent pas d'un schéma de données préétabli qui impose une structure fixe lors du stockage des documents. Le processus ToConceptSchema, que nous avons proposé dans ce chapitre, résout ce problème en générant un schéma conceptuel à partir d'un schéma logique déjà extrait (voir le chapitre 4). Ce schéma conceptuel fournit une vision non technique des classes, de leurs attributs et des liens sémantiques (lien d'association, classe d'associations, lien de composition et d'héritage) présents dans la BD. Ainsi, il facilite grandement la tâche des non-informaticiens, car il permet à ces derniers de comprendre et d'analyser les données de manière plus intuitive, sans avoir à plonger dans les détails techniques complexes.

De plus, notre processus est basé sur l'approche MDA. Celle-ci permet de définir un métamodèle source qui est responsable de la description du schéma logique ainsi que d'un métamodèle cible qui est chargé de décrire le schéma conceptuel correspondant. Le passage automatique du schéma logique vers le schéma conceptuel est réalisé par un ensemble de règles de transformation.

5.5.2 Positionnement

Dans cette section, nous effectuons le positionnement de notre proposition par rapport aux travaux effectués par ailleurs sur la même problématique.

Le tableau 5.3 compare nos propositions avec celles des trois travaux les plus significatifs de l'état de l'art à partir de huit critères.

D'après le tableau 5.3, les travaux [32] et [33] se limitent à une seule collection de documents. En revanche, ToConceptSchema prend en compte plusieurs collections de documents. Ceci permet une analyse plus complète des données d'une BD.

ToConceptSchema génère un schéma, exempt de tout aspect technique, dans un format textuel, à terme, dans un format graphique sous forme de DCL. Tandis que [32], [33] et [34] proposent d'extraire un schéma graphique. Toutefois, comme nous le verrons dans le paragraphe suivant, ce schéma ne contient pas tous les liens sémantiques qu'on trouve dans UML.

En termes de liens, ToConcetSchema extrait les principaux liens que l'on trouve dans UML, notamment les liens d'association, les classes d'associations, les liens de composition et d'héritage. Le processus dans [32] est limité à l'extraction des liens

d'association, sans considérer les classes d'associations, les liens de composition et d'héritage. De même, les travaux [33] se focalisent sur l'extraction des liens de composition, en excluant les liens d'association entre les collections. Or ce type de liens joue un rôle central pour notre cas d'étude présenté dans le chapitre 2. Le processus présenté dans [34] se limite à l'extraction des liens d'association et d'héritage pour une BD graphe.

Contrairement aux travaux [32] et [33], le processus de [34] et ToConceptSchema sont basés sur MDA. Cette approche offre une grande souplesse d'adaptation face aux évolutions des systèmes NoSQL. L'usage de métamodèles et de règles de transformation simplifie la prise en compte des évolutions techniques des SGBD dans le temps. La maintenance du processus est donc facilitée.

Notre système est le seul qui permet de visualiser simultanément un double schéma. Ce système est destiné à des décideurs qui analysent des données complexes. Il permet de visualiser, d'une part le schéma conceptuel pour percevoir la sémantique des données et d'autre part le schéma logique pour exprimer les requêtes (les deux schémas sont liés lors de la consultation). Ceci permettra aux utilisateurs de sélectionner, par exemple, un lien d'association dans le schéma conceptuel et de consulter simultanément sa traduction dans le schéma logique.

Critères / **Processus**	**[32]**	**[33]**	**[34]**	**ToConceptSchema**
Modèle NoSQL	Document	Document	Graphe	Document
Collections	1	1	n	n
Format du schéma généré	Diagramme IDEF1X	DCL	E/A	Textuel et DCL
Association	Oui	Non	Oui	Oui
Classe d'association	Non	Non	Non	Oui
Agrégation/ Composition	Non	Oui (composition)	Non	Oui
Héritage	Non	Non	Oui	Oui
MDA	Non	Non	Oui	Oui
Double schéma	Non	Non	Non	Oui

Tableau 5. 3 - Comparaison de notre proposition avec des travaux de recherche

Chapitre 6
Expérimentation et validation

Ce chapitre présente le système ProdSchemas que nous avons développé pour expérimenter nos propositions présentées dans les chapitres 4 et 5. Ce système est constitué de deux modules : le premier, ToLogSchema, est chargé d'extraire le schéma logique à partir d'une BD NoSQL orientée document. Le second module, ToConceptSchema, est chargé de transformer ce schéma en un schéma conceptuel. Ce chapitre est organisé comme suit. La section 6.1 décrit les outils techniques utilisés pour implanter ProdSchemas. La section 6.2 décrit les deux composants de ProdSchemas. Pour chacun de ces composants, nous présentons les métamodèles, les modèles et les règles de transformation. La section 6.3 présente l'expérimentation de ProdSchemas sur deux BD-test et une BD médicale. La section 6.4 présente la validation de ProSchemas sur une BD juridique.

6.1 Outils techniques

Dans cette section, nous présentons les outils techniques que nous avons utilisés pour implanter notre système de production de schémas. Étant donné que notre système repose sur MDA, nous avons besoin d'un environnement technique adapté pour la métamodélisation et la transformation M2M.

Nous avons donc choisi d'utiliser la plateforme EMF (Eclipse Modeling Framework) pour développer notre système. Avant de décrire les fonctionnalités d'EMF, nous introduisons tout d'abord Eclipse. Il s'agit d'un un système logiciel open-source qui a été créé par IBM dans le but de fournir un environnement de développement logiciel intégré extensible et libre. Ce cadre permet de fournir et de produire des outils pour la création d'une application, y compris les étapes de modélisation, de programmation et de tests. L'architecture d'Eclipse repose sur le concept de "Plugin" qui offre un ensemble de fonctionnalités.

EMF est un ensemble d'outils Open-source permettant de créer des applications basées sur une approche dirigée par les modèles. Il s'appuie sur trois langages : UML, XML et Java. Grâce à cette combinaison, il est possible de décrire un modèle de trois manières différentes : (1) en utilisant un diagramme UML pour visualiser graphiquement les structures et les liens, (2) en définissant un schéma XML ou (3) en écrivant du code Java pour définir les classes, les attributs et les relations. EMF permet d'utiliser une des trois représentations et de générer les deux autres. Ses fonctionnalités principales sont les suivantes :

- La métamodélisation : EMF permet de définir des métamodèles en utilisant un langage spécifique ; les métamodèles définissent les concepts des modèles dans un domaine particulier.
- La modélisation : EMF permet de créer des modèles en instanciant des métamodèles.
- La transformation de modèles : ceci correspond au passage d'un modèle à un autre ou d'un modèle à du texte.

Parmi les outils que regroupe EMF, nous avons mis en œuvre : Ecore, XMI et QVT. Ceux-ci seront détaillés dans les sections suivantes.

6.1.1 Ecore

Ecore est un composant d'EMF offrant un cadre pour la définition et la manipulation des métamodèles de données. La figure 6.1 illustre un extrait du métamodèle Ecore mettant en évidence les éléments principaux utilisés dans sa création ; parmi ces éléments, nous citerons :

- EPackage : est l'élément racine d'un métamodèle Ecore. Il peut être considéré comme un conteneur logique qui rassemble un ensemble de classes (EClass) et de types de données (EDataType) que nous définissons dans les paragraphes suivants. Il agit comme un moyen de structurer et d'organiser les différents éléments constituant un modèle dans le cadre de l'EMF. L'utilisation de packages dans Ecore facilite la structuration, la modularité et la réutilisation des éléments du modèle.
- EClass : permet de décrire la structure et le comportement d'un objet. Une classe peut contenir des attributs (EAttribute) et des références (EReference) pour établir des liens d'association entre les classes.
- EAttribute : désigne une propriété utilisée pour stocker une information spécifique à un objet. Par exemple, une classe Produit peut avoir des attributs tels que nom, prix et quantité en stock.
- EDataType : représente un type de données primitif tel que String, Integer, Boolean et Date ou personnalisé tel qu'une énumération EEnum. Les types permettent de spécifier les valeurs que peuvent prendre les attributs des classes. Par exemple, une classe Personne contient un attribut dateDeNaissance de type Date pour représenter la date de naissance d'une personne en utilisant des valeurs définies dans Date.
- EEnum : est utilisé pour définir une liste finie de valeurs possibles pour un attribut d'une classe. Par exemple, une classe Jour contient un attribut nom de type EEnum pour définir les jours de la semaine avec les valeurs possibles "Lundi", "Mardi", etc. Ceci permet d'établir un ensemble restreint de choix pour les valeurs d'attributs et facilite la validation des données.
- EReference : représente un lien (une association en termes UML) entre deux classes. Par exemple, considérons deux classes Auteur et Livre reliées par une EReference appelée auteur dans la classe Livre. Cette EReference établit un lien entre un livre et son auteur.

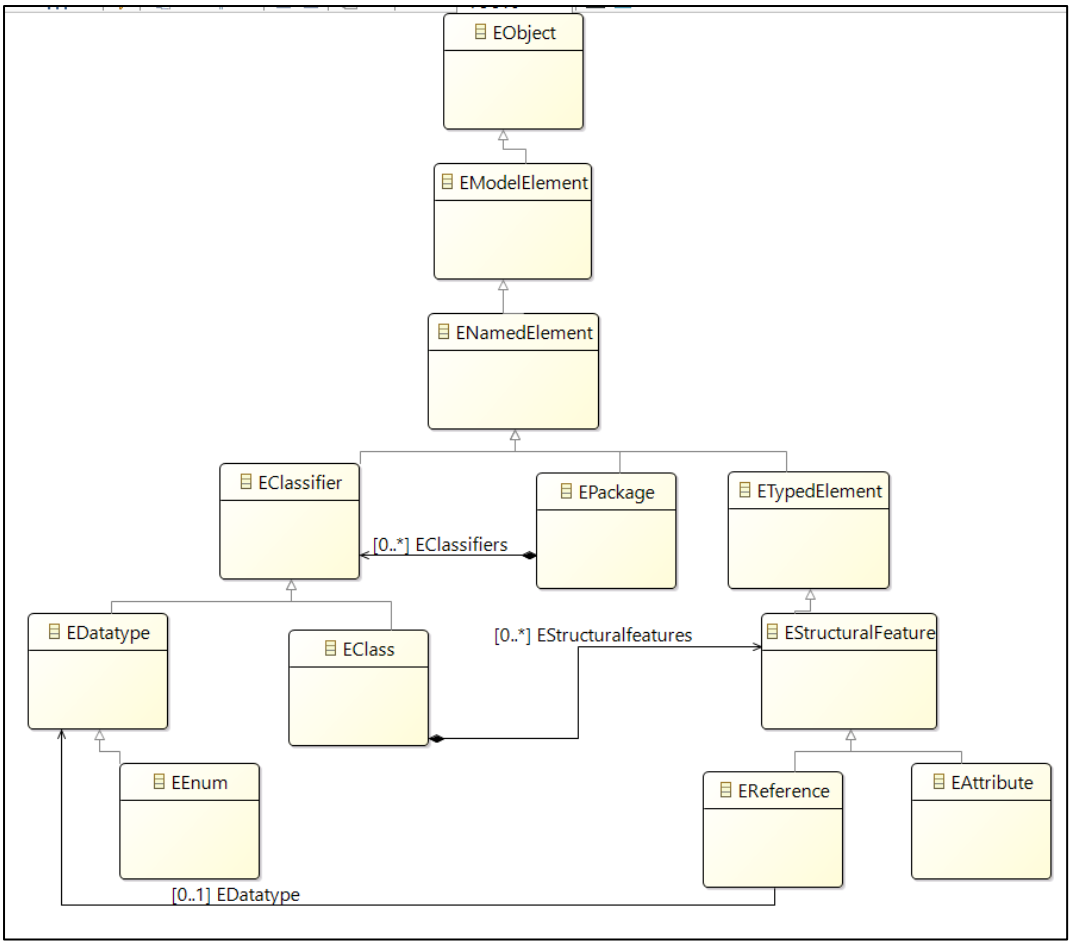


Figure 6. 1 - Extrait simplifié du métamodèle Ecore [43]

6.1.2 XMI

XMI (XML Metadata Interchange) est un format standardisé basé sur XML (eXtensible Markup Language) et utilisé pour représenter des modèles de données. Dans le contexte d'EMF, XMI est largement utilisé pour enregistrer des instances de métamodèles sous forme de fichiers XML. XMI génère automatiquement des grammaires XML, appelées DTD (Document Type Definition), à partir d'un métamodèle spécifique (dans notre cas, le métamodèle Ecore). Celui-ci permet de représenter les instances de métamodèles sous la forme de documents XML. La figure 6.2 présente le mécanisme de fonctionnement de XMI.

6.1.3 QVT

La transformation de modèles est le cœur de l'approche MDA. Il est essentiel de disposer d'un langage permettant de réaliser des transformations M2M. Nous avons donc utilisé le standard QVT qui est un langage de haut niveau proposé par l'OMG et intégré dans EMF. QVT a été proposée dans le but d'atteindre les objectifs suivants :

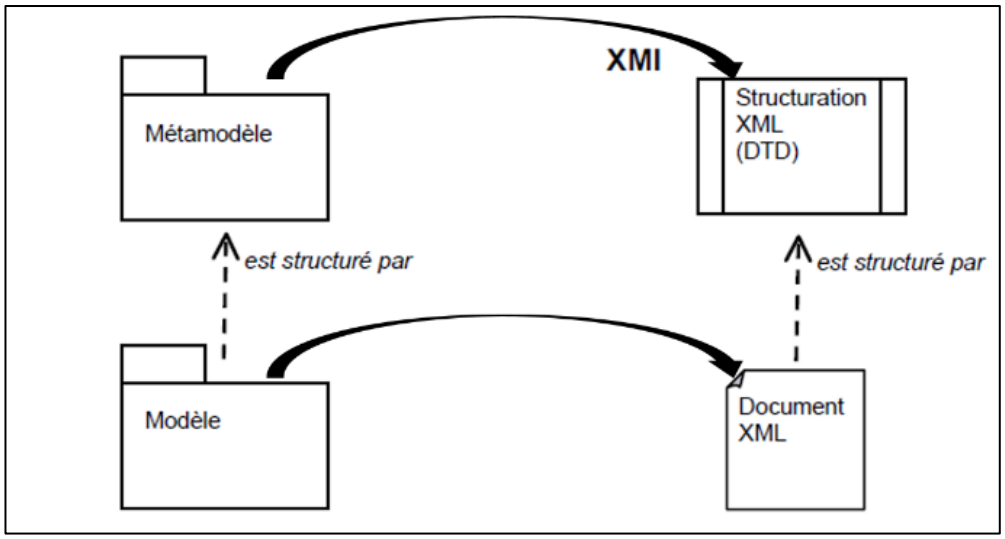


Figure 6. 2 - Mécanisme de fonctionnement de XMI [44]

- Exprimer des requêtes : cette fonctionnalité permet de sélectionner des éléments spécifiques à partir d'un modèle, ce qui facilite permettant la récupération des données précises en fonction des besoins de l'utilisateur.
- Créer des vues : l'aptitude à créer des vues facilite l'affichage des parties spécifiques d'un modèle, en mettant en évidence des aspects particuliers.
- Formaliser des règles de correspondance (Transformations) : ceci permet de définir des règles de transformation entre différents modèles. Celles-ci permettent d'établir la correspondance entre les éléments des métamodèles source et cible.

Une transformation QVT prend donc en entrée un modèle source et produit un modèle cible. Ces deux modèles sont conformes aux métamodèles source et cible respectivement. Ce principe est illustré dans la figure 6.3.

Nous avons opté pour l'utilisation du langage QVT pour deux raisons : 1) il s'agit d'une norme établie par l'OMG et 2) la plateforme EMF que nous utilisons dispose de ce formalisme.

6.2 Description de notre prototype

L'objectif du système "ProdSchemas" est :

- d'extraire un schéma logique à partir d'une BD NoSQL orientée-document. Ce schéma décrit la structure des données de la BD et permet d'exprimer des requêtes.
- de transformer le schéma logique extrait en un schéma conceptuel. Ce dernier est dépourvu des spécificités techniques et se concentre sur la sémantique de données.

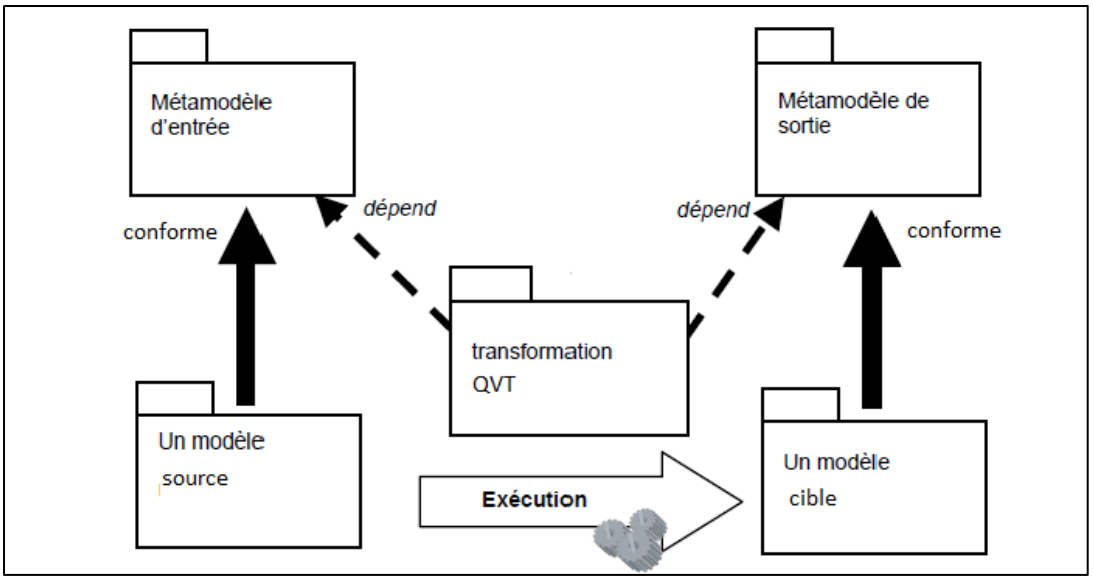


Figure 6. 3 - Principe d'une transformation QVT [44]

Comme le montre la figure 6.4, "ProdSchemas" se compose des modules "ToLogSchema" chargé de l'extraction de schéma logique à partir d'une BD fournie par un utilisateur et "ToConceptSchema" qui effectue la transformation du schéma logique en un schéma conceptuel. Les sections suivantes détaillent l'architecture de chaque module.

6.2.1 Module ToLogSchema

Les BD NoSQL sont généralement "Schemaless", ce qui signifie qu'elles ne disposent pas d'un schéma de données préétabli qui impose une structure fixe lors du stockage des documents

Ce module est chargé de l'extraction automatique du schéma logique à partir d'une BD NoSQL. Ce schéma est exprimé à l'aide du formalisme XMI. Nous détaillons ce module en précisant l'entrée, la sortie, la transformation et l'implantation.

- Entrée

L'entrée du module ToLogSchema est une BD NoSQL orientée document "Schemaless" gérée par un système tel qu'OrientDB ou MongoDB et fournie par un utilisateur ; le métamodèle source permet de lire cette BD.

- Sortie

La sortie du module ToLogSchema est un schéma logique qui fait apparaître la structure des données de la BD. Le schéma logique est conforme au métamodèle cible présenté dans la figure 4.5.

- Transformation

ToLogSchema applique une transformation pour convertir la BD NoSQL en entrée en un schéma logique. La transformation est réalisée à travers un ensemble de règles M2M formalisées en langage QVT.

- Implantation

Comme illustré dans la figure 6.5, l'implantation de ToLogSchema requiert les étapes suivantes :

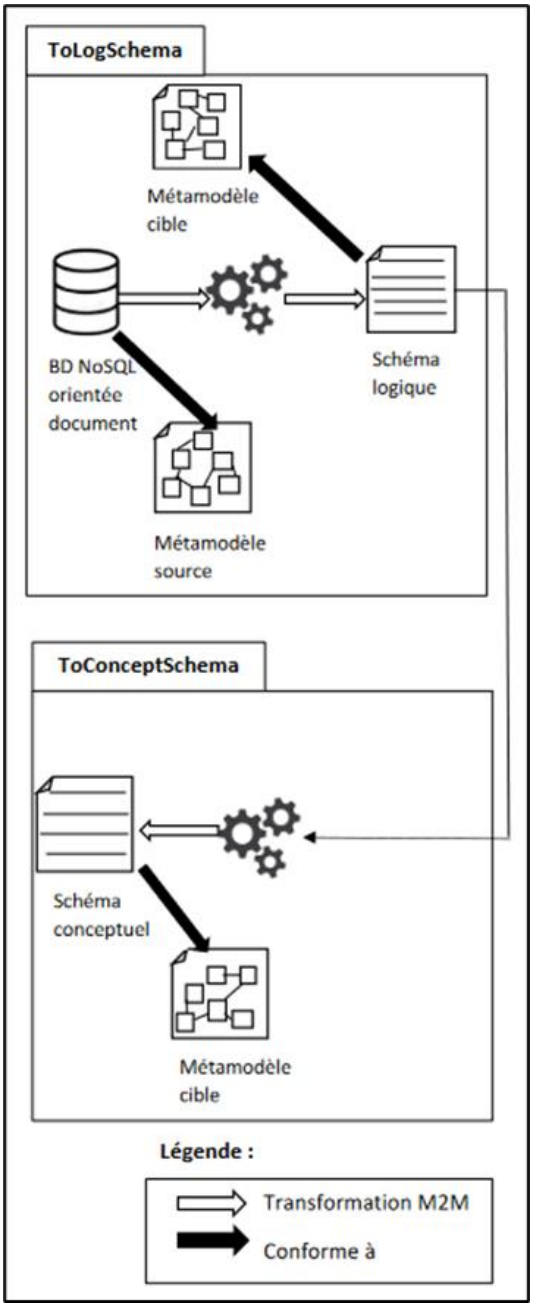


Figure 6. 4 - Architecture de notre système ProdSchemas

- La création des métamodèles source et cible sous forme de deux fichiers Ecore décrivant la BD NoSQL et son schéma logique. La figure 6.6 montre les métamodèles source et cible implantés en Ecore.
- La création des règles de transformation en utilisant le langage QVT. Elles assurent le passage d'une BD vers son schéma logique. La figure 6.7 montre un extrait du script QVT.

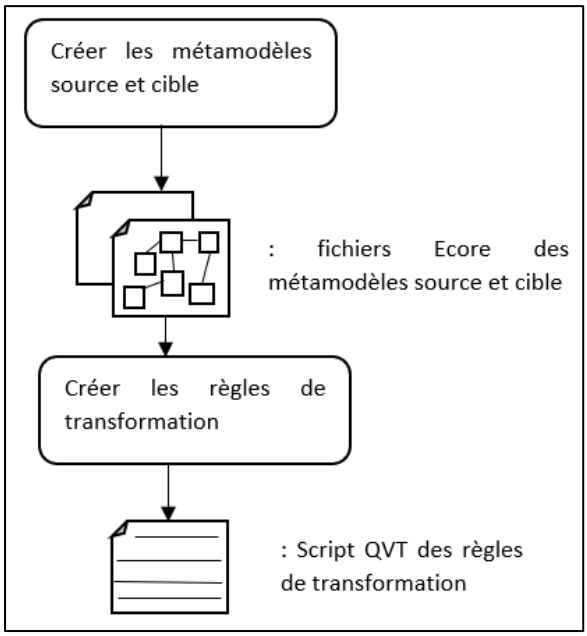


Figure 6. 5 - Étapes d'implantation du module ToLogSchema

6.2.2 Module ToConceptSchema

L'objectif de ce module est la transformation automatique du schéma logique (déjà extrait par le module ToLogSchema) en un schéma conceptuel. Ce dernier est exprimé en utilisant le formalisme XMI. Mais pour plus de clarté, nous le présentons également sous la forme d'un DCL d'UML. Le passage d'un schéma à un autre est réalisé à l'aide d'un ensemble de règles de transformation de type M2M formalisées en QVT. Dans la suite, nous présentons en détail ce module en spécifiant son entrée, sa sortie, sa transformation et son implantation.

- Entrée

L'entrée du module correspond à la sortie du module ToLogSchema présenté dans la section 6.2.1. Il s'agit donc du schéma logique d'une BD NoSQL.

- Sortie

La sortie correspond au schéma conceptuel de donnée en XMI. Ce schéma ne mentionne aucun détail technique. Il fait apparaître la description des classes ainsi que les liens entre elles.

- Transformation

La transformation effectuée par le module ToConceptSchema repose sur un ensemble de règles M2M formalisées en QVT.

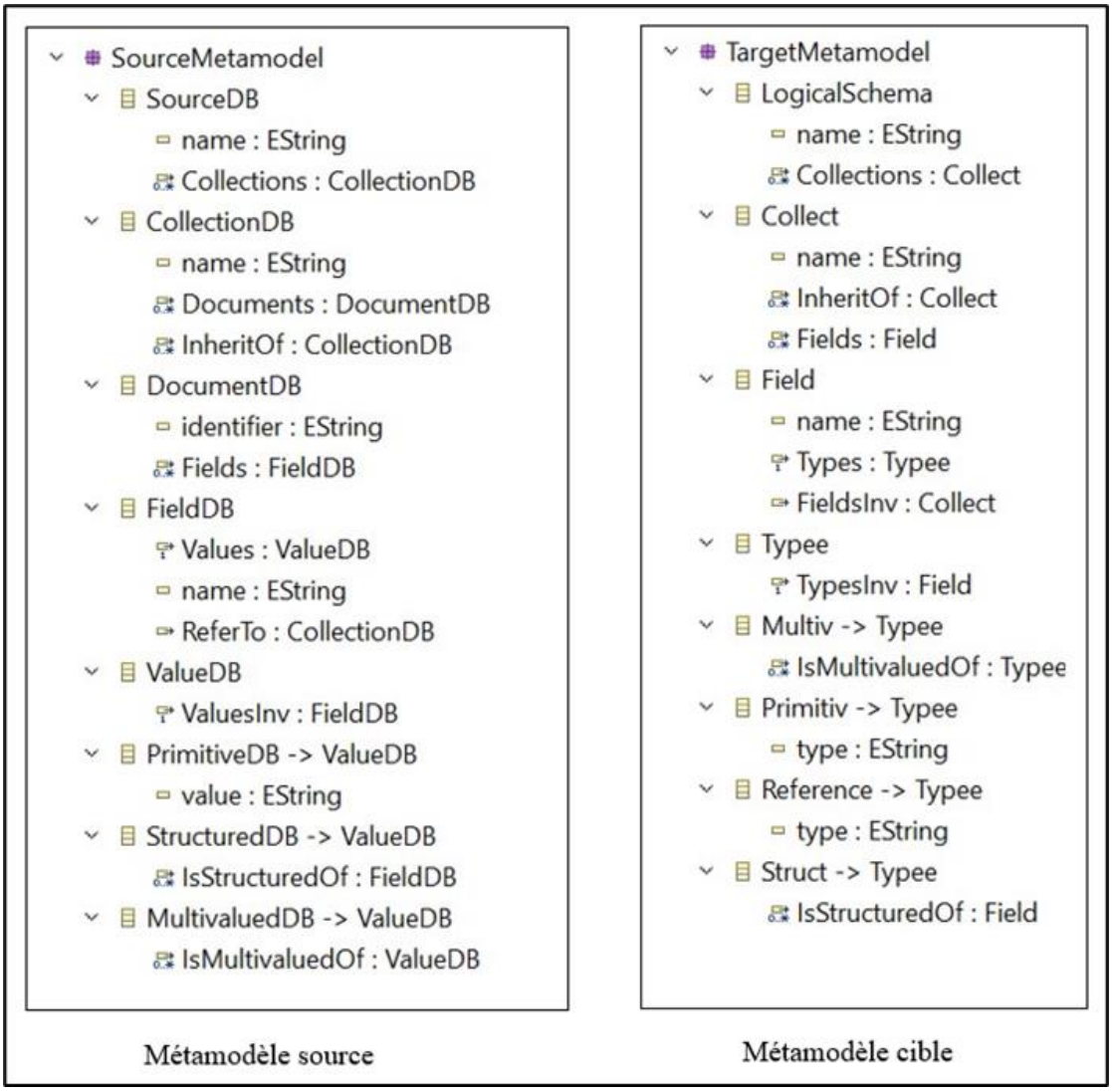


Figure 6. 6 - Métamodèles du module ToLogSchema implanté en Ecore

```
modeltype SourceMetamodel "strict" uses SourceMetamodel('http://SourceMetamodel.com');
modeltype TargetMetamodel "strict" uses TargetMetamodel('http://TargetMetamodel.com');
transformation Transformation1(in source: SourceMetamodel, out target: TargetMetamodel);

main() {
source.rootObjects()[SourceDB]-> map ToLogicalSchema();
}
--R1
mapping SourceDB::ToLogicalSchema() : LogicalSchema {
  name:=self.name;
  Collections:=self.Collections->map ToCollections();
}
--R2
mapping CollectionDB::ToCollections() : Collect {
  name:=self.name;
  Fields:=getUniqueFields(self.Documents)->map ToFields();
  --R7
  InheritOf:=self.InheritOf->map ToCollections();
}
mapping FieldDB::ToFields() : Field {
  name:=self.name;
  Types:=self.Values.map ToTypes();
}
mapping ValueDB::ToTypes() : Typee {
  init{
    var refCollect : Collect;
    var seqOfNames : Sequence (String);
    var refCollectionDB : CollectionDB;
    var seqOfFields : Sequence (FieldDB);
    var v : String;
    var firstMultiv :ValueDB;
    --R3.1
    if self.oclAsType(PrimitiveDB).value.matches('"[0-9a-zA-Z .,;:@?!]+"')then {
      result:=object Primitiv{type:='EString';}
    }endif;
    --R3.2
    if self.oclAsType(PrimitiveDB).value.matches('[0-9]+') then{
      result:=object Primitiv {type:='EInt';}
    }endif;
    --R3.3
    if self.oclAsType(PrimitiveDB).value.matches('true|false') then{
      result:=object Primitiv {type:='EBoolean';}
    }endif;
    --R3.4
    if self.oclAsType(PrimitiveDB).value.matches('[+-]?[0-9]+[.][0-9]+') then{
      result:=object Primitiv {type:='EFloat';}
    }endif;
    --R3.5
    if self.oclAsType(PrimitiveDB).value.matches('null') then{
      result:=object Primitiv {type:='ENull';}
    }endif;
    --R3.6
    if self.oclAsType(PrimitiveDB).value.matches('"[0-9][0-9]/[0-9][0-9]/[0-9][0-9][0-9][0-9]"') then{
      result:=object Primitiv{type:='EDate';}
    }endif;
    --R6.1 Cas d'OrientDB
    if self.oclAsType(PrimitiveDB).value.matches('"#[0-9][0-9]:[0-9]"') then{
      refCollectionDB:=self.ValuesInv.ReferTo;
      result:=object Reference {type:=refCollectionDB.name;}
    }endif;
     --R6.2 Cas de MongoDB
    if self.oclIsTypeOf(StructuredDB) then {
      seqOfNames:= self.oclAsType(StructuredDB).IsStructuredOf.name;
      seqOfFields:= self.oclAsType(StructuredDB).IsStructuredOf->asSequence();
      if seqOfNames->includes("$id") and seqOfNames->includes("$ref") then {
        v:=seqOfFields->last().Values.oclAsType(PrimitiveDB).value;
        result:= object Reference {type:=v;}
      --R4
      }else{
        result:=object Struct {IsStructuredOf:=self.oclAsType(StructuredDB).IsStructuredOf->map ToFields();};
      }endif;
    }endif;
```

Figure 6. 7 - Script QVT de la transformation de ToLogSchema

- <u>Implantation</u>

Comme le montre la figure 6.8, l'implantation de ToConceptSchema nécessite la réalisation des étapes les suivantes :

- Création du métamodèle cible sous la forme d'un fichier Ecore décrivant un schéma conceptuel d'une BD. La figure 6.9 montre le métamodèle cible implanté en Ecore.

- Création des règles de transformation en utilisant le langage QVT. Celles-ci assurent le passage du schéma logique vers le schéma conceptuel. La figure 6.10 montre un extrait du script QVT.

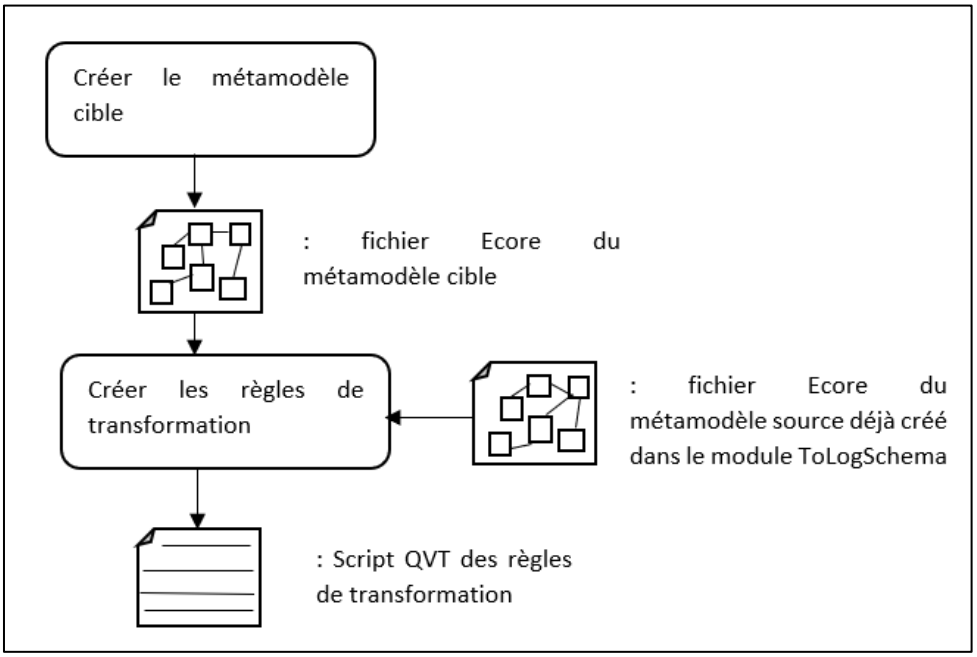


Figure 6. 8 - Étapes d'implantation du module ToConceptSchema

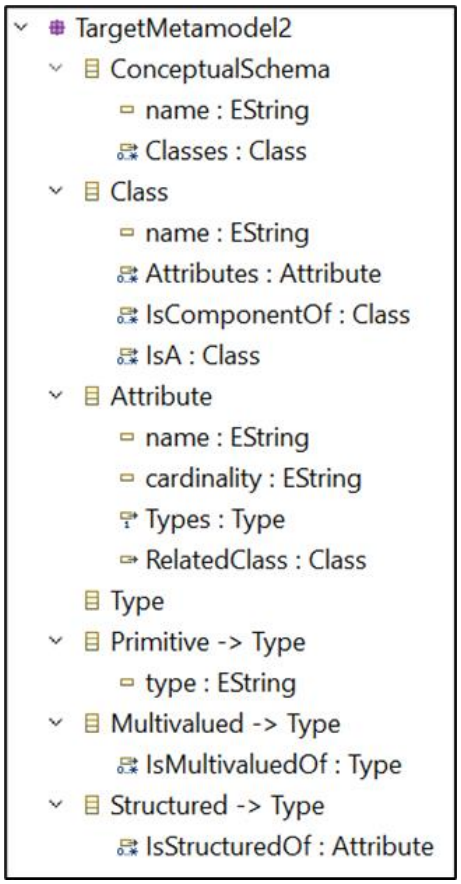


Figure 6. 9 - Métamodèle cible du module ToConceptSchema implanté en Ecore

6.3 Expérimentations

Pour vérifier le bon fonctionnement de notre système ProdSchemas, nous avons effectué deux expérimentations. La première expérimentation est réalisée sur deux BD-test dont les contenus ont été extraits de notre cas d'étude (application médicale) : l'une implantée avec MongoDB et l'autre avec OrientDB. Chaque BD contient les mêmes données et comporte douze collections. Le choix des collections, des relations et des types de champs dans cette BD-test a été fait pour montrer l'application des différentes règles de transformation ; nous avons donc introduit dans le test une grande variété de cas (types et relations notamment) bien identifiés. La deuxième expérimentation est réalisée sur une BD médicale réelle. Dans les sections suivantes, nous détaillons ces deux expérimentations.

6.3.1 Expérimentation sur deux BD-test

Pour expérimenter ProdSchemas, nous avons mis en œuvre les processus ToLogSchema et ToConceptSchema qui le composent.

Comme illustré dans la figure 6.11, l'expérimentation de ToLogSchema sur une BD-test OrientDB repose sur deux étapes successives :

- Instanciation du métamodèle source de ToLogSchema : pour créer une instance du métamodèle source, nous avons développé un logiciel en langage Java permettant de créer, à partir d'une BD fournit en entrée, un fichier XMI contenant les données de la BD avec des balises spécifiques au métamodèle source. Les paramètres utiles figurant dans le dictionnaire sont également reportés ; c'est notamment le cas des noms des collections référencées. La figure 6.12 montre un exemple d'entrée/sortie. La partie gauche de la figure montre un extrait du contenu de la BD-test implantée avec OrientDB. Cet extrait contient un document de la collection "Patients" avec deux champs "identP" et "lastName" ainsi qu'un document de la collection "Doctors" avec deux champs "identD" et "speciality". A ce stade, chaque champ est associé à une valeur. La partie droite de la figure montre la conversion des documents après ajout des balises XMI ; par exemple <Collections> pour chaque collection, <Documents> pour chaque document, <Fields> pour chaque champ, <Values> pour la valeur de chaque champ… Le logiciel parcourt la totalité de la BD source et crée une nouvelle BD au format XMI destinée à notre processus ToLogSchema.

- Exécution du script QVT de la figure 6.7 sur le fichier XMI créée dans l'étape précédente. Cette étape produit automatiquement le schéma logique de la BD-test en format XMI et en conformité avec le métamodèle cible du processus ToLogSchema.

```
modeltype TargetMetamodel "strict" uses TargetMetamodel('http://TargetMetamodel.com');
modeltype TargetMetamodel2 "strict" uses TargetMetamodel2('http://TargetMetamodel2.com');
transformation Transformation2(in source: TargetMetamodel, out target: TargetMetamodel2);

main() {
var allClasses : Set(Class);
var allCollections : Set(Collect);
var currentCollection : Collect;
source.rootObjects()[LogicalSchema]-> map ToConceptualSchema();
allClasses:=target.objectsOfType(Class);
allCollections:=source.objectsOfType(Collect);
allClasses->forEach(tt){
  currentCollection:=getCollection(tt, allCollections);
  tt.Attributes:=currentCollection.Fields->map ToAttributes(allClasses);
};
}
--R1
mapping LogicalSchema::ToConceptualSchema() : ConceptualSchema {
  name:=self.name;
  Classes:=self.Collections->map ToClasses();
}
--R2
mapping Collect::ToClasses() : Class {
  name:=self.name;
  --Transformation d'un lien InheritOf
  IsA:=self.InheritOf->map ToClasses();
}
--R3
mapping Field::ToAttributes(allClasses : Set(Class)) : Attribute {
  name:=self.name;
  Types:=self.Types.map ToTypes(allClasses);
}
mapping Typee::ToTypes(allClasses : Set(Class)) : Type {
  init{
    var refName : String;
    var collectionName : String;
    var referencedCollectionName : String;
    var class1 : Class;
    var class2 : Class;
    var role1 : String;
    var role2 : String;
    var att1 : Attribute;
    var att2 : Attribute;
    var listOfAttributes : List(Attribute);
    var classToAdd : Class;
    var fieldsOfStruct : Set(Field);
    var fieldName: String;
    var nonLinkF : List(Field);
    var firstType : Typee;
    --R4
    if self.oclIsTypeOf(Primitiv) then{
      result:=object Primitive {type:=self.oclAsType(Primitiv).type;}
    }endif;
    -- R7: Transformation d'une référence monovaluée
    if self.oclIsTypeOf(Reference) then {
      refName:=getFieldName(self.oclAsType(Reference));
      collectionName:= getCollectionName(self.oclAsType(Reference));
      referencedCollectionName:=self.oclAsType(Reference).type;
      class1:=getClass(allClasses, collectionName);
      class2:=getClass(allClasses, referencedCollectionName);
      role1:=getRole(collectionName);
      role2:=getRole(referencedCollectionName);
      if role1.equalsIgnoreCase(role2) then {
        role1:=role1+"1";
        role2:=role2+"2";
      }endif;
      att1:=object Attribute {name:=role1;
                              RelatedClass:=class1;
                              cardinality:="(0..*)";};
      att2:=object Attribute {name:=role2;
                              RelatedClass:=class2;
                              cardinality:="(0..1)";};
      listOfAttributes->add(att1);
      listOfAttributes->add(att2);
      classToAdd:=object Class {name:=toCapitalizeFirstLetter(refName);
                                Attributes:=listOfAttributes->asSet();};
    }endif;
```

Figure 6. 10 - Script QVT de la transformation de ToConceptSchema

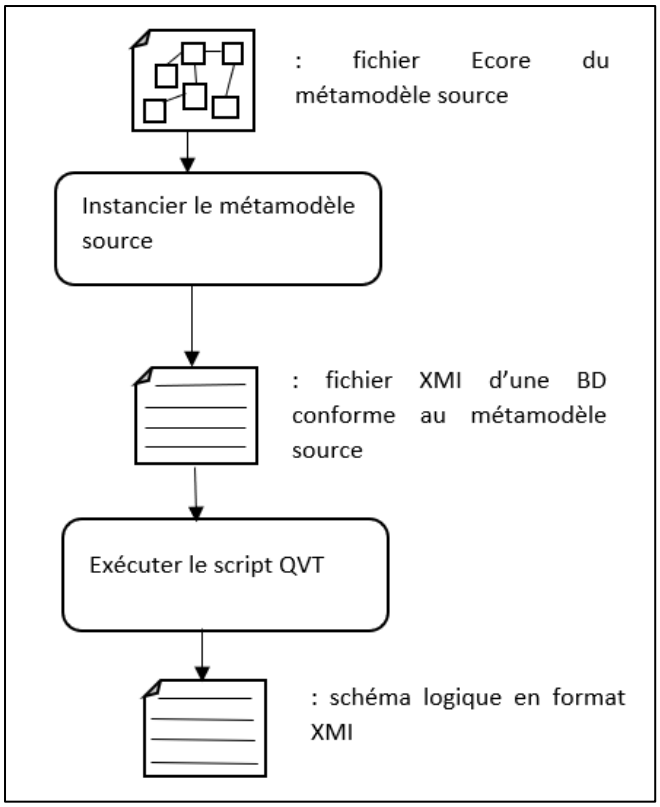


Figure 6. 11 - Étapes d'expérimentation de ToLogSchema

La figure 6.13 présente un extrait d'une BD-test implantée sous OrientDB. Cet extrait comporte cinq collections : "Patients", "Doctors", "Diseases", "HospitalizedPatients" et "Rooms". Chacune d'elles contient des champs et leurs valeurs. La figure 6.14 affiche un extrait du schéma logique généré par ToLogSchema à partir de cette BD-test. Cet extrait de schéma inclut les cinq collections. Chaque collection comporte des champs avec leurs types atomiques et complexes. De plus, elle peut comporter des références, par exemple le champ "antecedents" qui représente les antécédents d'un patient. Le schéma contient également des héritages provenant de la BD-test. Un héritage peut être visualisé dans EMF en cliquant sur la sous-collection. Un exemple d'héritage est présenté dans la figure 6.15 où la collection "HospitalizedPatients" hérite de "Hospitalized".

Nous avons également expérimenté le processus ToConceptSchema sur la BD-test implantée avec OrientDB. Cela a été réalisé en suivant les étapes suivantes présentées dans la figure 6.16.

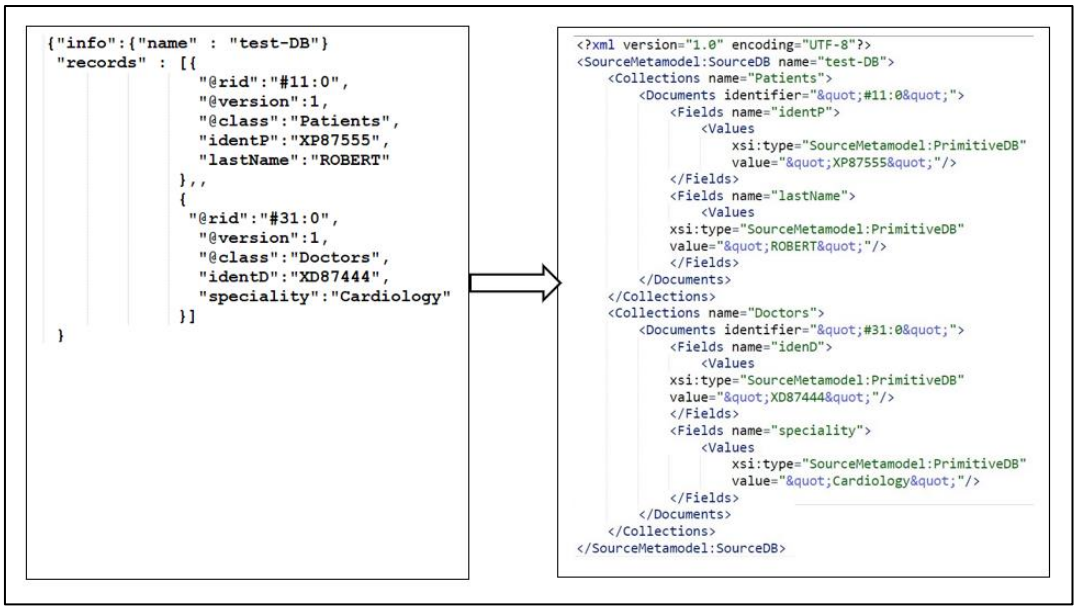


Figure 6. 12 - Conversion de données OrientDB en XMI

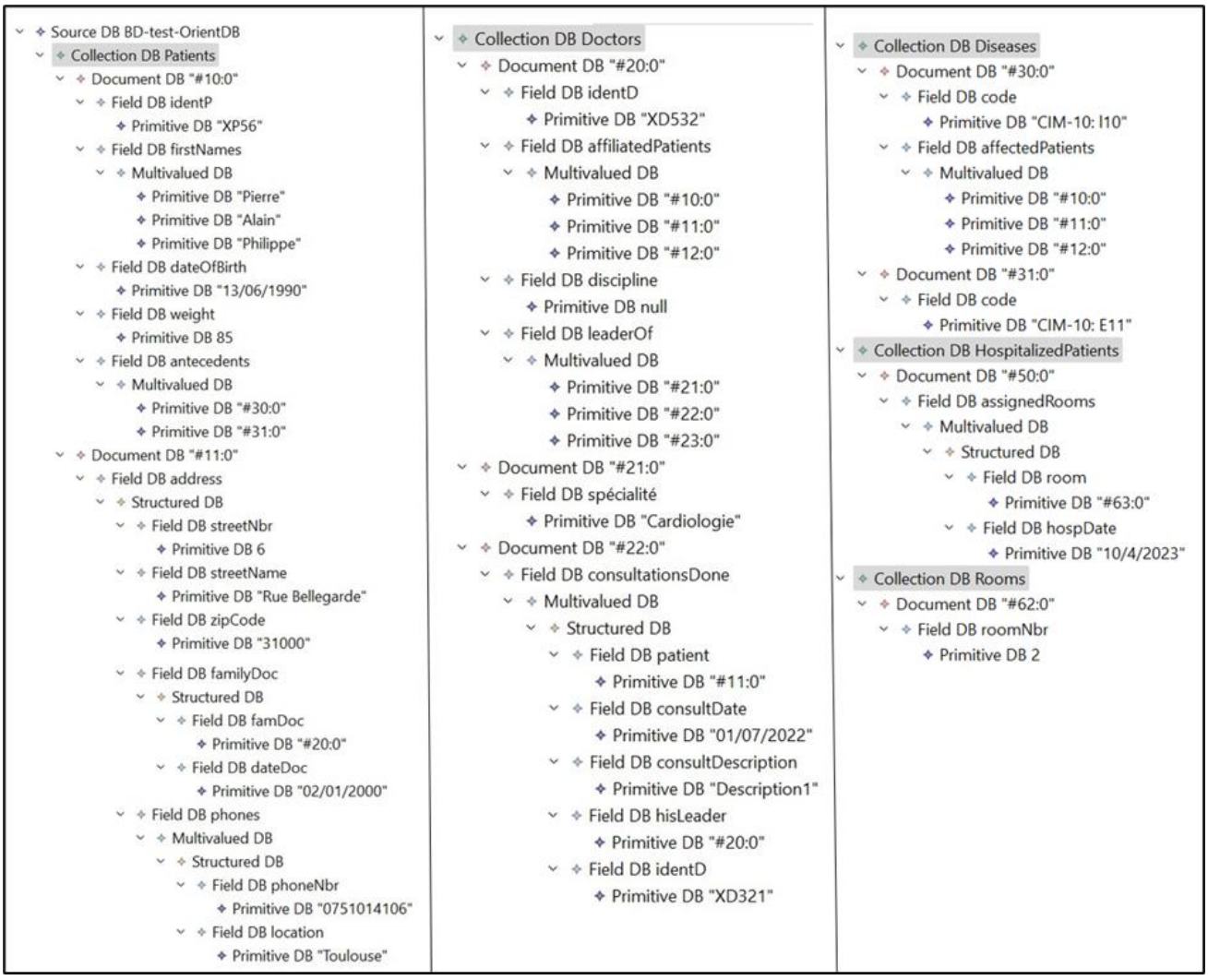


Figure 6. 13 - Extrait de BD-test OrientDB

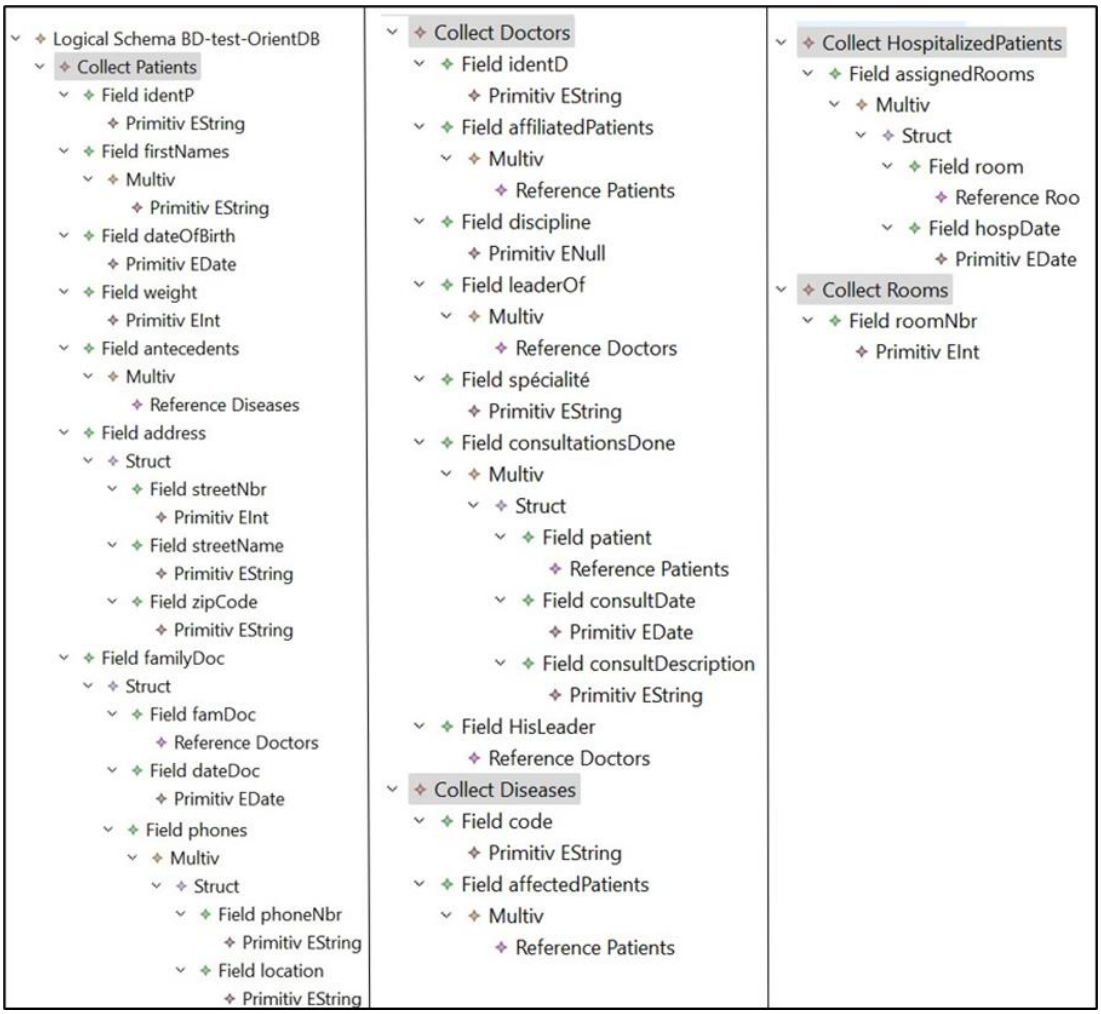


Figure 6. 14 - Extrait du schéma logique généré à partir de BD-test OrientDB

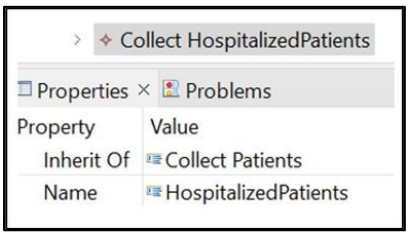


Figure 6. 15 - Exemple d'héritage dans le schéma logique de BD-test OrientDB

- Reprise du fichier XMI contenant le schéma logique résultant de l'expérimentation du processus ToLogSchema.
- Exécution du script QVT de la figure 6.10 sur le fichier XMI de l'étape précédente. Le résultat de cette étape est un schéma conceptuel en format XMI.

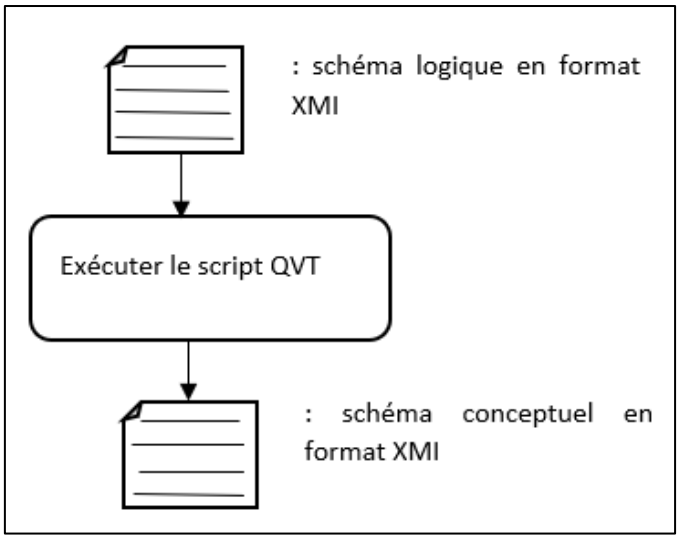


Figure 6. 16 - Étapes d'expérimentation de ToConceptSchema

La figure 6.17 présente un extrait du schéma conceptuel généré par ToConceptSchema. Ce schéma inclut les cinq classes "Patients", "Doctors", "Diseases", "HospitalizedPatients" et "Rooms". Il contient également les classes qui résultent de la transformation des références représentant des liens d'association (Par exemple "FamDoc") et des champs multivalués structurés représentant des liens de composition (Par exemple "Phones"). Dans la figure 6.18, nous montrons les caractéristiques des objets. Nous commençons par les caractéristiques des attributs-rôles dans un lien d'association. Ces attributs incluent les noms des classes reliées et les cardinalités. (Par exemple, les attributs "pat" et "doc" représentent les rôles des classes "Patients" et "Doctors" dans le lien "FamilyDoc". Ainsi, ces attributs sont associés respectivement aux cardinalités (0..*) et (0..1). Ensuite, nous présentons un exemple de lien de composition entre la classe composite "phones" et composante "Patients". Enfin, nous présentons un exemple de lien d'héritage entre la sous-classe "HospitalizedPatients" et la super-classe "Patients".

De manière similaire, pour expérimenter ProdSchemas sur la BD-test implantée avec MongoDB, nous reprenons les mêmes étapes que nous avons déployées pour OrientDB. Nous présentons dans la figure 6.19 un extrait de cette BD. Les schémas logique et conceptuel de cette BD sont les mêmes que ceux présentés dans les figures 6.14 et 6.17, respectivement.

Figure 6. 17 - Extrait du schéma conceptuel généré à partir de schéma logique de BD-test OrientDB

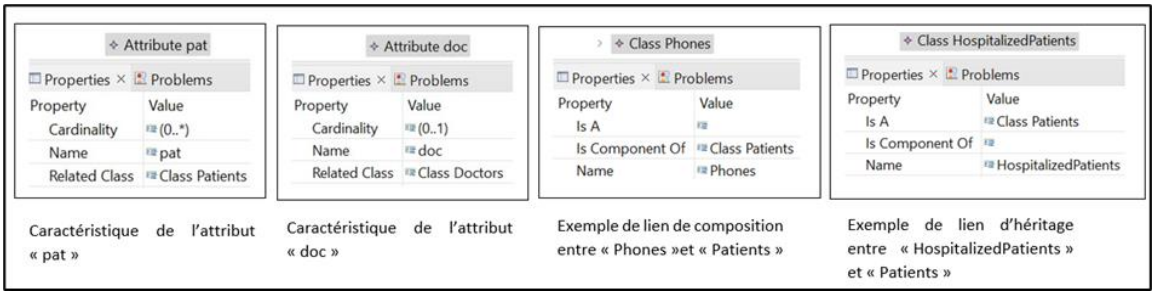


Figure 6. 18 - Caractéristique des objets présents dans le schéma conceptuel de BD-test OrientDB

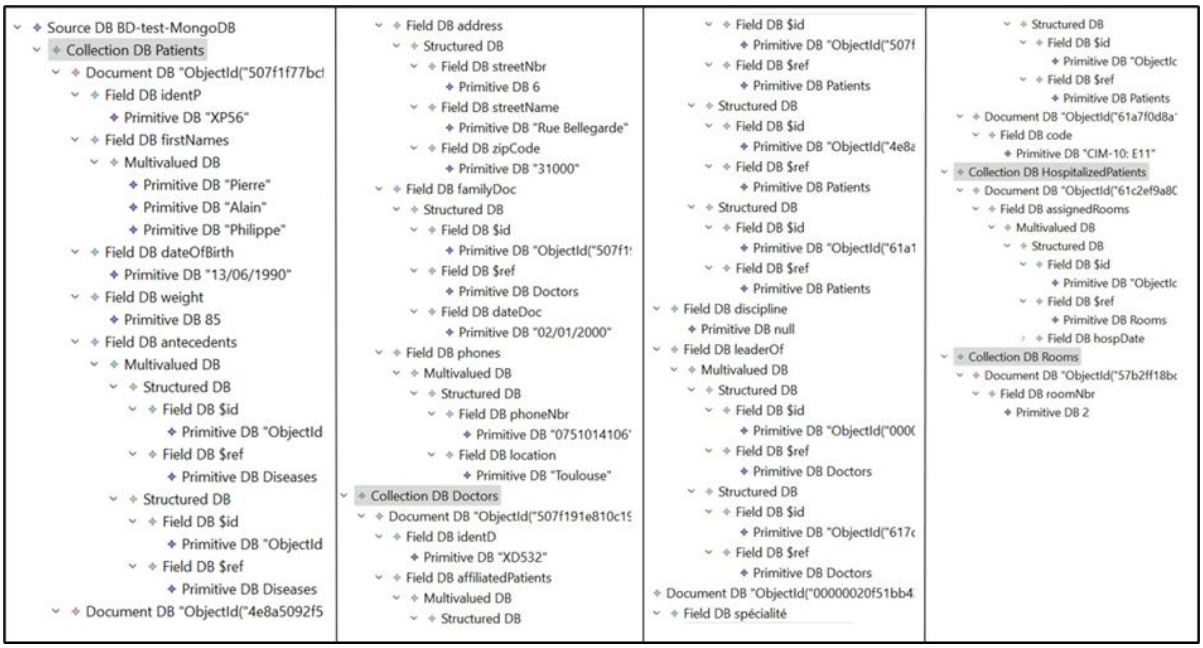


Figure 6. 19 - Extrait de BD-test MongoDB

6.3.2 Expérimentation sur une BD médicale

Après avoir vérifié sur deux BD-test que le fonctionnement de notre système ProdSchemas était correct, nous l'avons également appliqué sur une BD réelle (en cours de développement) issue de notre application médicale détaillée dans le chapitre 2 ; cette BD est implantée sur le système OrientDB et présente un volume situé entre un et deux téraoctets. La figure 6.20 montre un extrait du schéma logique généré par ToProdSchemas. Il décrit la structure de chaque collection ("Patients" par exemple) ainsi que les champs qu'elle contient avec leurs types atomiques et complexes. Il fait apparaître également des références. Par exemple, la collection "Doctors" contient "activities", une référence multivaluée vers la collection "Tasks". La figure 6.21 montre un extrait de schéma conceptuel produit par notre processus. Ce schéma affiche des classes, des attributs et leurs types atomiques, des liens d'association, de composition et d'héritage.

À ce stade, le schéma conceptuel généré est en format XMI. Cependant, compte tenu du volume important des données présentes dans la BD médicale et de leur complexité, ce schéma XMI peut se révéler difficile à appréhender par un non-informaticien. Pour résoudre ce problème, nous avons développé un logiciel en langage Java qui convertit le schéma XMI en DCL UML. Grâce à cette solution, un non-informaticien dispose d'un double schéma : un schéma conceptuel sous forme de DCL UML et un schéma logique en format XMI. Ce caractère dual permet à l'utilisateur de comprendre la sémantique des données tout en visualisant leurs structures organisationnelles. La figure 6.22 illustre un extrait du DCL UML du schéma conceptuel. Ce dernier contient des classes d'objets et des liens. Par exemple, la classe "Doctors" est reliée à "Hospitals" par un lien d'association "Exercise" représentant l'hôpital dans lequel le médecin travaille. La classe "Hospitals" est

composée de “Services”. “HosAmb” est une super-classe des sous-classes “HospPatients” et “AmbPatients”.

Figure 6. 20 - Extrait du schéma logique de la BD-médicale

Figure 6. 21 - Extrait du schéma conceptuel en format XMI

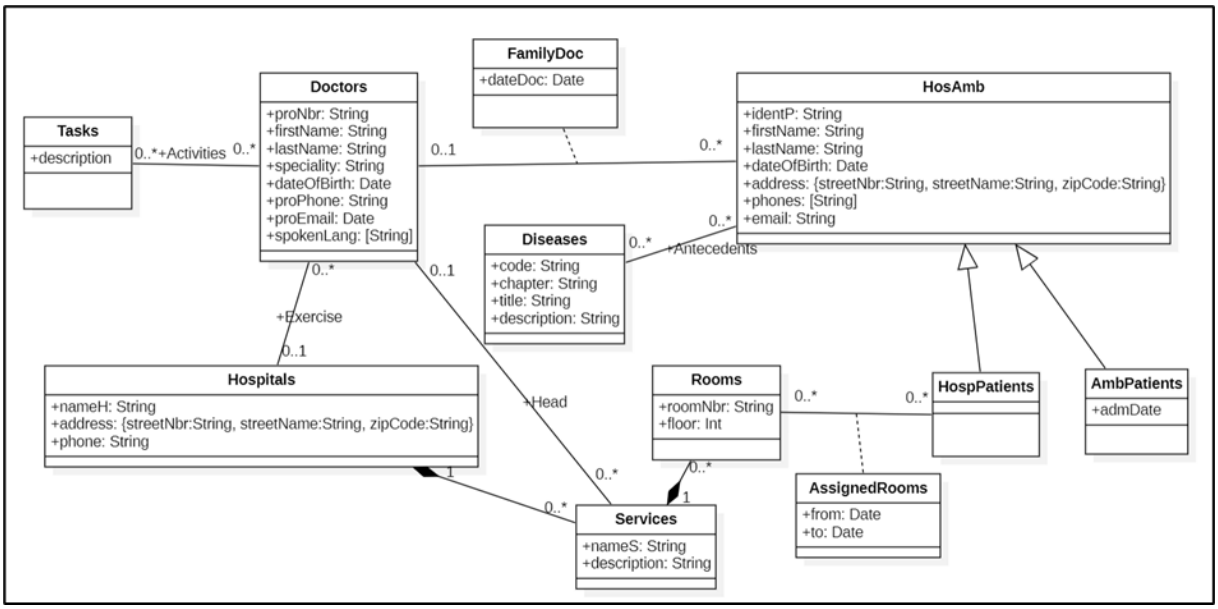


Figure 6. 22 - Extrait du DCL correspondant au schéma conceptuel XMI.

6.4 Validation

Les expérimentations présentées dans la section précédente ont été réalisées d'une part sur deux BD-test et d'autre part sur une BD médicale réelle. Ces expérimentations ont montré le caractère opérationnel de notre système. A présent, il convient d'évaluer la contribution de notre solution dans un contexte professionnel. L'objectif principal des travaux que nous avons menés est de simplifier les tâches liées à l'exploitation des BD "Schemaless" orientées document. Plus précisément, nos travaux visent à minimiser le temps nécessaire pour comprendre la signification des données et élaborer des requêtes. En collaboration avec le laboratoire de recherche CBI² de la société TRIMANE, une ESN spécialisée en Business Intelligence et Big Data, nous avons concrétisé la mise en œuvre de notre système sur trois applications décisionnelles développées par cette entreprise. La première application nommée « Juris-Dépôt », aborde la mise à disposition des décisions rendues par les tribunaux. La seconde « Disponibilités Des Agents » se focalise sur l'analyse de disponibilité des agents d'intervention. La troisième application intitulée « GMAO » se concentre sur la gestion de la maintenance assistée par ordinateur au sein d'un groupe pharmaceutique privé. Ces trois applications sont dédiées à la manipulation de contenu textuel et d'images. Elles se distinguent par des volumes de données considérables atteignant plusieurs téraoctets. Ces caractéristiques confirment le choix d'une BD NoSQL pour stocker les données. Pour des raisons de confidentialité, nous avons modifié les noms des objets manipulés.

Pour évaluer nos travaux, huit développeurs expérimentés (consultants en informatique) de Trimane avaient pour mission de gérer des opérations de maintenance pour les trois applications. Ces développeurs n'avaient aucune connaissance préalable des données manipulées par ces applications. Chaque

développeur est chargé de formuler dix requêtes de plus en plus complexes pour chacune des trois applications et ceci dans trois contextes différents :

- sans disposer de schéma de données
- en utilisant le schéma logique généré par ProdSchemas
- en utilisant les schémas conceptuel et logique générés par ProdSchemas.

Le rôle de ces contextes est de montrer :

- d'une part l'utilité d'un schéma de données pour réduire le temps d'écriture d'une requête
- d'autre part la complémentarité des deux schémas conceptuel et logique.

Il convient de noter que nous ne fournissons qu'un extrait de schéma contenant une dizaine de classes. Ainsi, pour des raisons de confidentialité, nous montrons dans les figures 6.23 et 6.24 un exemple de schémas conceptuel et logique de la BD de l'application « Juris-Dépôt » avec des noms de classes et de liens modifiés.

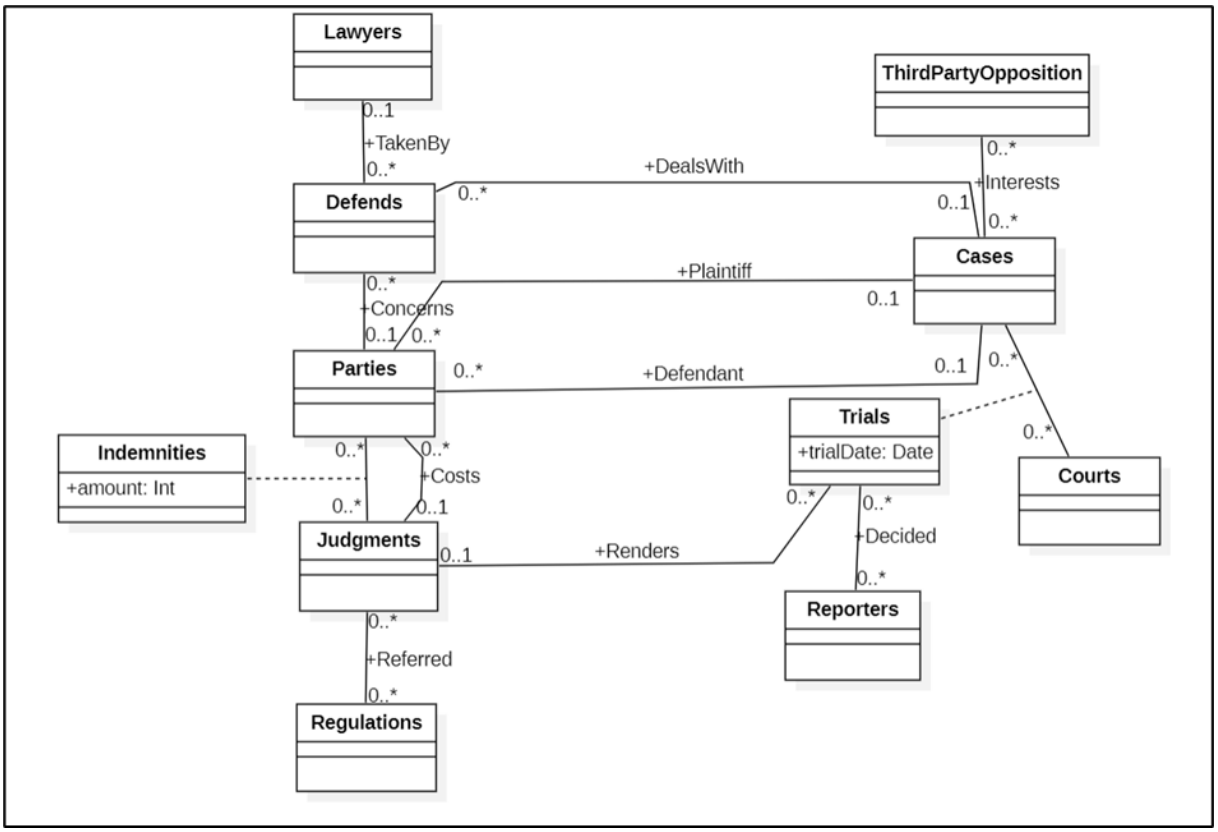


Figure 6. 23 - Exemple de schéma conceptuel de la BD de l'application « Juris-Dépôt »

Chaque BD est associée à un ensemble de dix requêtes dont les énoncés en langage naturel ont été fournis aux huit développeurs. L'affectation des développeurs sur les trois BD est illustrée dans la figure 6.25 : par exemple, le développeur (Dev1) a écrit trois ensembles de 10 requêtes :

- Le premier ensemble de requêtes est appliqué sur la BD1 de l'application « Juris-Dépôt » ; Dev1 ne dispose d'aucun schéma de données.

- Le deuxième ensemble est appliqué sur la BD2 de l'application « Disponibilités Des Agents » (BD2) en utilisant le schéma logique de cette BD généré par notre système.

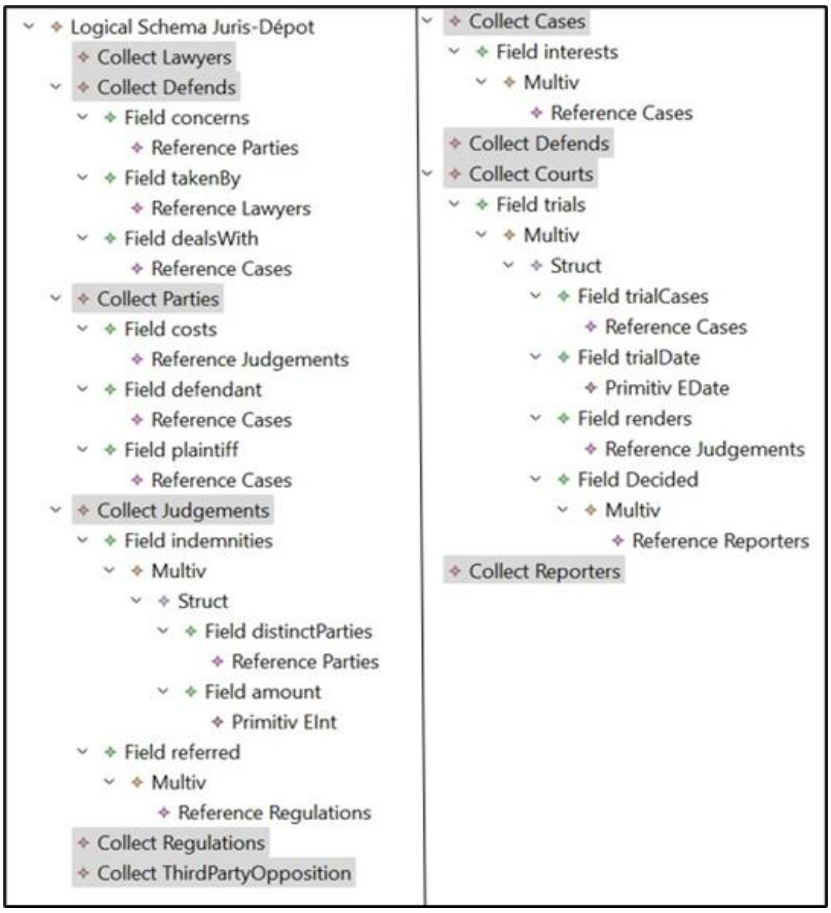


Figure 6. 24 - Exemple de schéma logique de la BD de l'application « Juris-Dépôt »

- Le troisième ensemble est appliqué sur la BD3 de l'application « GMAO » en utilisant les schémas conceptuel et logique produits par notre système.

Comme le montre le tableau 6.1, nous avons enregistré le temps passé par chaque développeur dans l'écriture de chaque ensemble de requêtes. Par exemple, Dev1 a passé 52 minutes pour écrire l'ensemble de requêtes sur la BD1 en ne disposant d'aucun schéma de données, 25 minutes pour l'ensemble de requêtes sur la BD2 en utilisant le schéma logique et 18 minutes pour l'ensemble de requêtes sur la BD3 en utilisant les schémas conceptuel et logique. Nous avons également calculé la moyenne de ces temps obtenus par les huit développeurs dans chaque contexte : sans schéma, avec le schéma logique et avec les deux schémas. Comme le montre le tableau 6.1, le temps moyen le plus court passé par les 8 développeurs est celui dans le contexte d'écriture de requêtes en utilisant les schémas conceptuel et logique. Ainsi, nous avons constaté un faible écart entre les temps moyens consacrés à l'écriture des requêtes dans les contextes d'utiliser seulement le schéma logique et les deux schémas conceptuel et logique. Ceci a permis de confirmer notre hypothèse de départ selon laquelle le schéma de données peut contribuer à réduire le temps d'écriture des requêtes. Les deux schémas conceptuel et logique sont

complémentaires et aident un développeur à rédiger plus rapidement les requêtes sur une BD NoSQL “Schemaless” présentant des structures de données complexes.

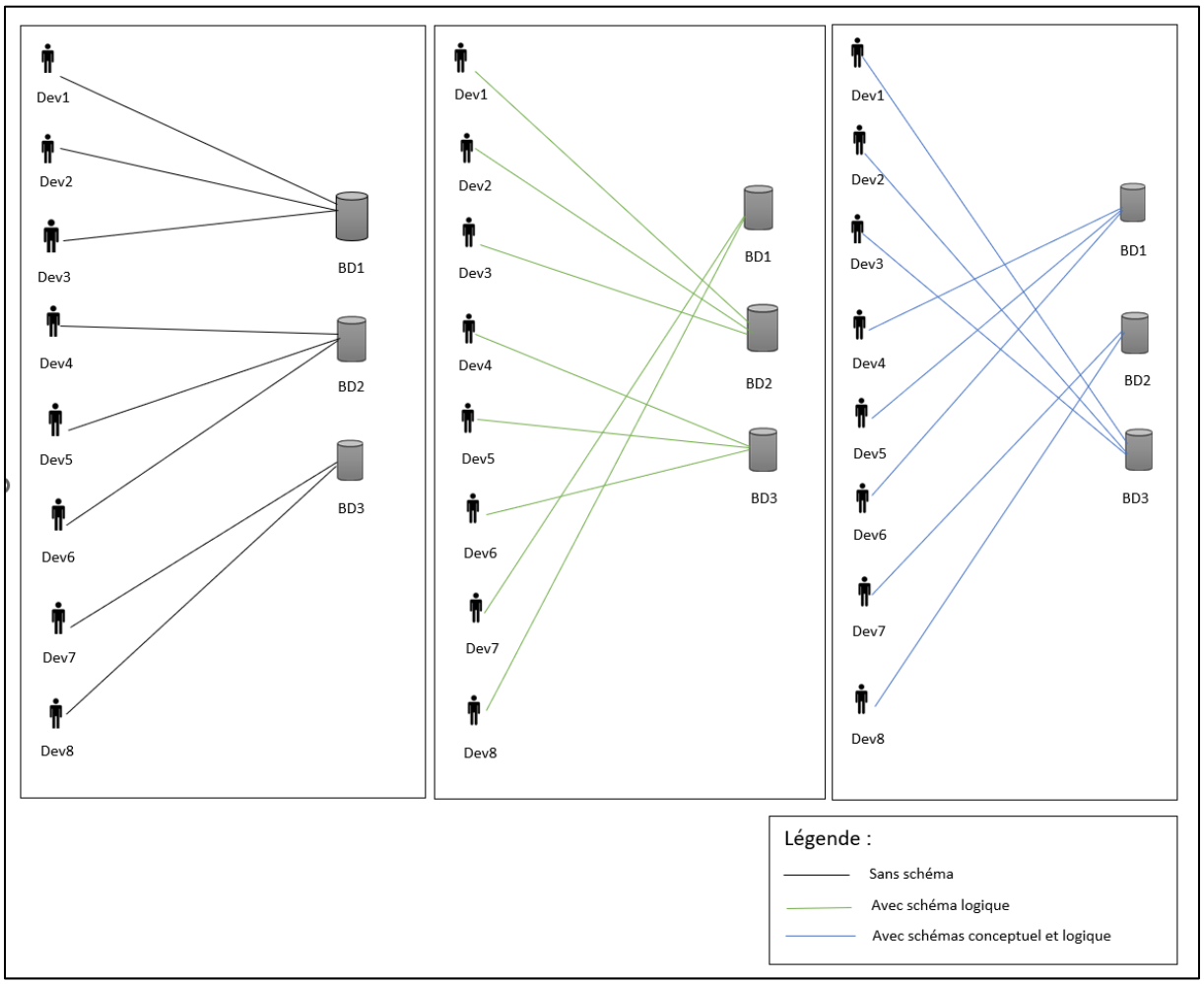


Figure 6. 25 - Affectation des développeurs sur les trois BD

6.5 Conclusion

Dans ce chapitre, nous avons décrit la réalisation de notre système de production de schémas “ProdSchemas”. Ce système est composé de deux modules : ToLogSchemas qui est chargé d’extraire le schéma logique d’une BD NoSQL et ToConceptSchema qui transforme ce schéma logique en un schéma conceptuel. L’implantation de ProdSchemas repose sur MDA. A cet effet, nous avons utilisé la plateforme EMF. Nous avons présenté les outils techniques que regroupe EMF, notamment Ecore pour créer des métamodèles, XMI pour créer des instances de métamodèles et QVT pour réaliser des transformations M2M. Nous avons détaillé les deux modules composant ProdSchemas en présentant, pour chacun, l’entrée, la sortie, les transformations et les étapes de son implantation.

Pour tester notre système, nous avons réalisé deux expérimentations : une première a porté sur deux BD-test implantées sur OrientDB et MongoDB et la seconde sur une BD volumineuse issue de notre application médicale. Nous avons également évalué

l'apport de notre solution pour les décideurs au sein de la société TRIMANE. Actuellement, les décideurs accèdent aux données de manière empirique, sans avoir recours à un schéma ; ce n'est qu'avec l'expérience acquise au sein de l'application qu'ils parviennent à améliorer leurs performances. L'objet est de comparer le temps de rédaction de requêtes sur trois BD pour des applications développées par cette entreprise et ceci dans trois contextes : sans disposer de schéma de données, avec le double schéma. Les résultats ont montré qu'avoir à disposition à la fois un schéma conceptuel et un schéma logique permet d'accélérer la rédaction des requêtes sur une BD NoSQL "Schemaless" complexe.

Dev / **Contexte**	**Sans schéma**	**Avec le schéma logique**	**Avec les schémas conceptuel et logique**
Dev1	52 min (BD1)	25 min (BD2)	18 min (BD3)
Dev2	48 min (BD1)	23 min (BD2)	16 min (BD3)
Dev3	45 min (BD1)	22 min (BD2)	15 min (BD3)
Dev4	50 min (BD2)	24 min (BD3)	18 min (BD1)
Dev5	43 min (BD2)	21 min (BD3)	14 min (BD1)
Dev6	45 min (BD2)	25 min (BD3)	18 min (BD1)
Dev7	49 min (BD2)	22 min (BD3)	16 min (BD1)
Dev8	50 min (BD3)	24 min (BD1)	16 min (BD2)

Tableau 6. 1 - Temps de formulation des requêtes sur les BD des trois applications

Chapitre 7
Conclusion générale

Nous présentons successivement un bilan sur l'extraction d'un double schéma des données ainsi que les différentes activités à venir permettant de compléter et améliorer ces travaux.

7.1 Synthèse de nos travaux

Les travaux de recherche présentés dans ce mémoire s'inscrivent dans le contexte des données massives appelées "Big Data". Ces données sont caractérisées par la règle des 3V : Volume, Variété et Vélocité qui correspond aux propriétés majeures que doit vérifier le Big Data. Le Volume renvoie à la taille considérable que peuvent atteindre ces données, de l'ordre du téraoctets. La variété représente la complexité des structures de données qui peuvent se présenter sous différents types et formats. La Vélocité fait référence à la vitesse à laquelle les données sont générées, collectées, traitées et analysées ; elle implique la capacité à gérer des flux continus de données en temps réel. Face à ces caractéristiques, les SGBD relationnels n'offrent pas des fonctionnalités suffisantes pour gérer les Big Data. En effet, Ils reposent sur des schémas rigides et des modèles tabulaires, ce qui les rend peu adaptés pour gérer la variété et la complexité des Big Data. De plus, les SGBDR sont limités en termes de capacité à gérer des volumes massifs de données. La vitesse élevée à laquelle les données sont générées et traitées dépasse également les capacités de traitement des SGBDR.

Pour gérer efficacement les Big Data, une nouvelle catégorie de SGBD a émergé sous le nom de systèmes NoSQL. Ces systèmes sont mieux adaptés que les SGBDR pour gérer les Big Data. En effet, ils sont plus flexibles en termes de schéma de données c'est-à-dire qu'ils n'imposent pas de schéma fixe défini au préalable. Cette absence de schéma fixe permet aux données d'être intégrées sous divers formats sans nécessité à respecter une structure rigide prédéfinie. En outre, les systèmes NoSQL sont conçus pour faciliter le passage à l'échelle, ce qui signifie qu'ils peuvent gérer de grands volumes de données. Ils sont aussi adaptés à des vitesses élevées et permettent de collecter, de traiter et d'analyser rapidement les données générées à un rythme rapide. L'absence de schéma prédéfini (propriété "Schemaless") est présente dans la plupart des NoSQL. Bien que celle-ci offre une flexibilité indéniable permettant aux utilisateurs d'insérer de nouvelles données sans recourir à un administrateur de BD, elle rend l'expression des requêtes complexe. En effet, pour formuler efficacement des requêtes, il est essentiel de connaître la structure des données (noms des collections, noms et types des champs) et les relations entre les objets. Ces informations se trouvent dans un schéma de données.

Dans ce mémoire, nous nous sommes intéressés à la génération de deux schémas de données pour une BD NoSQL "Schemaless" existante. Il s'agit des schémas logique et conceptuel. Le schéma logique décrit comment les données sont organisées dans la BD et permet d'exprimer des requêtes. Le schéma conceptuel, indépendant de tout aspect technique, permet de comprendre la sémantique des données. Pour ce faire,

nous avons proposé le système ProdSchemas. Ce dernier est composé de deux processus :

- ToLogSchema : extrait un schéma logique en format XMI à partir d'une BD NoSQL "schemaless". Ce schéma affiche des collections, des champs avec leurs types de données ainsi que des relations (notamment des références et des héritages). Ce module, basé sur l'architecture MDA, consiste à définir un métamodèle source qui représente la BD, un métamodèle cible qui représente son schéma logique et un ensemble de règles de transformation formalisées avec le langage standard QVT. Cet ensemble assure le passage entre la source et la cible.

- ToConceptSchema : transforme le schéma logique extrait par le processus ToLogSchema en un schéma conceptuel exprimé en format XMI. Ce schéma contient des classes d'objets et des liens (association, classe d'associations, composition et héritage). Compte tenu de la complexité des données stockées dans la BD, le schéma conceptuel XMI s'avère difficile à comprendre par un non-informaticien. Par conséquent, nous le présentons également sous la forme d'un DCL proposée par la norme UML.

L'originalité de notre proposition réside dans :

- La génération d'un double schéma mis à la disposition des décideurs pour décrire une BD NoSQL "Schemaless : un schéma conceptuel exempt de détails techniques permettant de comprendre la sémantique de données et un schéma logique permettant d'exprimer des requêtes.

- La formalisation du schéma conceptuel en se basant sur le standard UML. Ce schéma contient donc les principaux liens présents dans UML (association, classe d'associations, composition et héritage).

- L'utilisation de l'architecture MDA basée sur la métamodélisation et la définition des règles de transformation. Cette démarche de développement de logiciels génère des artefacts aisément modifiables pour traiter d'autres types de BD NoSQL (par exemple les BD graphe).

Pour vérifier la faisabilité de notre proposition, nous avons implanté le système ProdSchemas en utilisant la plateforme EMF. Cette plateforme présente un cadre adapté à la métamodélisation et la transformation de modèles. Elle fournit un ensemble d'outils nécessaires à la mise en œuvre de notre système : Ecore permet de définir des métamodèles, XMI permet de représenter des instances de métamodèles (les modèles) et QVT permet de formaliser des règles de transformation. Nous avons ensuite effectué deux expérimentations pour nous assurer du bon fonctionnement du système. La première expérimentation a été réalisée sur deux BD-test (une implantée avec MongoDB et l'autre avec OrientDB). L'objectif de cette expérimentation est donc de s'assurer que notre système peut être généralisé en s'appliquant sur plusieurs BD orientées documents gérées par des systèmes différents. Notons cependant que le passage à d'autres modèles (graphe,

orienté colonnes, clés-valeurs) n'a pas été étudié dans le cadre de cette thèse. La deuxième expérimentation a été réalisée sur notre cas d'étude (une BD médicale).

De plus, nous avons validé notre système dans un cadre professionnel au sein de l'entreprise TRIMANE. Pour ceci, des développeurs ont été chargés d'exprimer des requêtes sur trois BD différentes développées par cette entreprise. L'expression des requêtes a été faite dans trois contextes : sans disposer de schéma de données, en utilisant le schéma logique généré par notre système et en utilisant les schémas conceptuel et logique générés par notre système. Les résultats obtenus ont montré que les deux schémas, conceptuel et logique, aident les développeurs à rédiger plus rapidement les requêtes sur une BD NoSQL "Schemaless".

Les travaux de cette thèse ont été présentés dans les publications suivantes : [2], [3], [4] et [5].

7.2 Perspectives

Dans le cadre de la poursuite de nos travaux, nous souhaitons apporter des compléments visant à améliorer notre processus. En effet, la découverte des liens de composition a été systématisée à partir des champs multivalués structurés. Or, certains champs multivalués structurés, imbriqués dans une collection, ne sont pas considérés par les utilisateurs comme des classes composantes ; par exemple, le champ multivalué structuré "Addresses" dans une collection Employees. Mais dans un processus de rétroconception, le système ne peut pas connaître ces aspects sémantiques sans l'aide d'un dispositif complémentaire. L'utilisation d'une ontologie s'avère pertinente dans ce contexte. La première perspective consiste à développer une ontologie qui formalise la signification des champs multivalués structurés et à l'intégrer dans notre processus.

Dans notre cas d'étude (une application médicale), les données stockées dans la BD peuvent évoluer rapidement, ce qui peut rendre rapidement obsolètes les schémas extraits. De plus, la génération de nouveaux schémas exige le parcours de la totalité de la BD. Par conséquent, il n'est pas envisageable de réexécuter le processus d'extraction des schémas à chaque fois qu'une mise à jour est apportée à la BD. La deuxième perspective consiste donc à développer un module pour capturer l'évolution des données et mettre à jour les schémas de la BD en temps réel. Le principe implique qu'au lieu de revoir l'ensemble de la BD pour extraire un nouveau schéma à chaque modification, il est possible de maintenir un ensemble de métadonnées qui enregistre les changements structurels au fil du temps. Des travaux de recherche ont traité certains aspects de ce problème, notamment les travaux de [45] et [46], mais certaines spécificités doivent être mises en œuvre pour notre cas d'étude.

Par ailleurs, notre système a été conçu pour fonctionner avec des BD orientées document gérées par MongoDB et OrientDB. La troisième perspective consiste alors à étendre notre processus pour inclure un autre modèle de BD NoSQL. Ceci est

d’autant plus important que le développement des lacs de données [47] exige la cohabitation de BD multimodèles. Pour cette extension, nous avons choisi d'utiliser OrientDB dont la spécificité est de prendre en charge à la fois le modèle orienté document et le modèle graphe. Ainsi, l’extraction de la structure des nœuds dans le modèle graphe peut être rapprochée de celle des collections dans le modèle orienté document et l’extraction des relations peut être rapprochée des références. Dans un second temps, nous envisageons d'étendre notre système pour le rendre compatible avec tous les modèles de BD NoSQL afin de garantir une flexibilité maximale dans le support des diverses structures de données présentes dans ces BD. L’extraction de schémas pour différents modèles de BD NoSQL (appelée extraction multimodale) a fait l’objet de certains travaux comme ceux présentés dans [48] et [49].

La quatrième perspective consiste à intégrer dans notre système un module qui traite de la transformation logique vers conceptuel et de son inverse pour implanter une BD sur un système supportant un autre modèle. Par exemple, on pourrait transformer une BD MongoDB en une BD Redis (graphe) ou une BD Cassandra (Colonne). L’objectif de cette perspective est d'offrir aux utilisateurs une solution complète de gestion de schémas qui simplifie la transition des schémas d’un niveau à un autre pour différentes plateformes de BD NoSQL. Ceci permet d'accroître la flexibilité, la portabilité et l'efficacité dans la gestion des données dans divers contextes technologiques.

Bibliographie

[1] L. Parkin, B. Chardin, S. Jean, A. Hadjali et M. Baron, « A cooperative treatment of the plethoric answers problem in RDF », *Knowl Inf Syst*, vol. 64, nº 9, p. 2481-2514, sept. 2022. doi: 10.1007/s10115-022-01710-8.

[2] F. Abdelhedi, A. Brahim, H. Rajhi, R. Ferhat et G. Zurfluh, « Automatic Extraction of a Document-oriented NoSQL Schema », in *Proceedings of the 23rd International Conference on Enterprise Information Systems*, Online Streaming : SCITEPRESS - Science and Technology Publications, p. 192-199, 2021. doi: 10.5220/0010433501920199.

[3] F. Abdelhedi, H. Rajhi et G. Zurfluh, « Extraction Process of the Logical Schema of a Document-oriented NoSQL Database », in *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development*, Online Streaming : SCITEPRESS - Science and Technology Publications, p. 61-71, 2022. doi: 10.5220/0010899000003119.

[4] F. Abdelhedi, H. Rajhi et G. Zurfluh, « Extraction of Semantic Links from a Document-Oriented NoSQL Database », *SN COMPUT. SCI.*, vol. 4, nº 2, p. 148, janv. 2023, doi: 10.1007/s42979-022-01578-z.

[5] F. Abdelhedi, H. Rajhi et G. Zurfluh, « Conceptual modeling of a document-oriented NoSQL database », AICCSA, Giza, Egypt, (8 pages), déc. 2023. (A paraître)

[6] A. B. Angadi, A. Angadi, K. Gull, et L. David, « Growth of New Databases & Analysis of NOSQL Datastores », International Journal of Advanced Research in Computer Science and Software Engineering, 2013.

[7] M. Neji, F. Ghorbel, B. Gargouri, N. Mimouni et E. Metais, « An Ontology based Smart Management of Linguistic Knowledge », *Journal of Data Mining & Digital Humanities*, p. 9251, sept. 2022. doi: 10.46298/jdmdh.9251.

[8] Y. Demchenko, C. Ngo et P. Membrey, « Architecture framework and components for the big data ecosystem », Journal of System and Network Engineering, 2013.

[9] J. Han, E. Haihong, G. Le, P. Center et J. Du, « Survey on NoSQL database », in *6th International Conference on Pervasive Computing and Applications*, Port Elizabeth, South Africa: IEEE, p. 363-366, oct. 2011. doi: 10.1109/ICPCA.2011.6106531.

[10] D. W. Embley et S. W. Liddle, « Big Data—Conceptual Modeling to the Rescue », in *Conceptual Modeling*, W. Ng, V. C. Storey, et J. C. Trujillo, Éd., in Lecture Notes in Computer Science, vol. 8217. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 1-8, 2013. doi: 10.1007/978-3-642-41924-9_1.

[11] C. Woo, « The Role of Conceptual Modeling in Managing and Changing the Business », in *Conceptual Modeling – ER*, M. Jeusfeld, L. Delcambre, et T.-W. Ling, Éd., in Lecture Notes in Computer Science, vol. 6998. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 1-12, 2011. doi : 10.1007/978-3-642-24606-7_1.

[12] B. Combemale, « Ingénierie Dirigée par les Modèles (IDM) - Etat de l'art », 2008.

[13] J. Bezivin et O. Gerbe, « Towards a precise definition of the OMG/MDA framework », in *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA: IEEE Comput. Soc, p. 273-280, 2001. doi: 10.1109/ASE.2001.989813.

[14] Object Management Group OMG, « OMG Meta Object Facility (MOF) Core Specification », 2013. Disponible sur : http://www.omg.org/spec/MOF/2.4.1/PDF/

[15] Object Management Group OMG, « MDA Guide Version 1.0 », 2003. Disponible sur : https://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf

[16] A. G. Kleppe, J. B. Warmer et W. Bast, « *MDA explained: the model driven architecture: practice and promise* », 5. print. in Addison-Wesley object technology series. Boston Munich: Addison-Wesley, 2007.

[17] L. Wang *et al.*, « Schema management for document stores », *Proc. VLDB Endow.*, vol. 8, n° 9, p. 922-933, mai 2015. doi : 10.14778/2777598.2777601.

[18] M.-A. Baazizi, H. Ben Lahmar, D. Colazzo, G. Ghelli et C. Sartiani, « Schema Inference for Massive JSON Datasets », Extending Database Technology (EDBT), Venise, Italy, mars 2017. Disponible sur : https://api.semanticscholar.org/CorpusID:4867725.

[19] M.-A. Baazizi, D. Colazzo, G. Ghelli et C. Sartiani, « Parametric schema inference for massive JSON datasets », *The VLDB Journal*, vol. 28, n° 4, p. 497-521, août 2019. doi : 10.1007/s00778-018-0532-7.

[20] A. A. Frozza, R. D. S. Mello et F. D. S. D. Costa, « An Approach for Schema Extraction of JSON and Extended JSON Document Collections », in *IEEE International Conference on Information Reuse and Integration (IRI)*, Salt Lake City, UT: IEEE, p. 356-363, juill. 2018. doi : 10.1109/IRI.2018.00060.

[21] M. Fruth, K. Dauberschmidt et S. Scherzinger, « Josch: Managing Schemas for NoSQL Document Stores », in *IEEE 37th International Conference on Data Engineering (ICDE)*, Chania, Greece : IEEE, p. 2693-2696, avr. 2021. doi: 10.1109/ICDE51399.2021.00306.

[22] Z. Brahmia, F. Grandi, S. Brahmia et R. Bouaziz, « A Graphical Conceptual Model for Conventional and Time-varying JSON Data », *Procedia Computer Science*, vol. 184, p. 823-828, 2021. doi : 10.1016/j.procs.2021.03.102.

[23] E. Gallinucci, M. Golfarelli et S. Rizzi, « Schema profiling of document-oriented databases », *Information Systems*, vol. 75, p. 13-25, juin 2018. doi: 10.1016/j.is.2018.02.007.

[24] M. Klettke et S. Scherzinger, « Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores », Datenbanksysteme für Business, Technologie und Web, 2015.

[25] U. P. D et P. Santhi Thilagam, « ClustVariants : An Approach for Schema Variants Extraction from JSON Document Collections », in *IEEE IAS Global Conference on Emerging Technologies (GlobConET)*, Arad, Romania: IEEE, p. 515-520, mai 2022. doi : 10.1109/GlobConET53749.2022.9872382.

[26] A. N. Istiqamah et K. R. S. Wiharja, « a Schema Extraction of Document-Oriented Database for Data Warehouse », *ijoict*, vol. 7, n° 2, p. 36-47, déc. 2021. doi : 10.21108/ijoict.v7i2.584.

[27] D. Sevilla Ruiz, S. F. Morales et J. García Molina, « Inferring Versioned Schemas from NoSQL Databases and Its Applications », in *Conceptual Modeling*, P. Johannesson, M. L. Lee, S. W. Liddle, A. L. Opdahl, et Ó. Pastor López, Éd., in Lecture Notes in Computer Science, vol. 9381. Cham: Springer International Publishing, p. 467-480, 2015. doi : 10.1007/978-3-319-25264-3_35.

[28] A. H. Chillon, D. S. Ruiz, J. G. Molina et S. F. Morales, « A Model-Driven Approach to Generate Schemas for Object-Document Mappers », *IEEE Access*, vol. 7, p. 59126-59142, 2019. doi : 10.1109/ACCESS.2019.2915201.

[29] A. Hernández Chillón, J. R. Hoyos, J. García-Molina et D. Sevilla Ruiz, « Discovering entity inheritance relationships in document stores », *Knowledge-Based Systems*, vol. 230, p. 107394, oct. 2021. doi : 10.1016/j.knosys.2021.107394.

[30] A. A. Frozza, E. Dias Defreyn et R. Dos Santos Mello, « An Approach for Schema Extraction of NoSQL Columnar Databases: the HBase Case Study », *jidm*, vol. 12, n° 5, nov. 2021. doi : 10.5753/jidm.2021.1966.

[31] A. A. Frozza, S. R. Jacinto et R. D. S. Mello, « An Approach for Schema Extraction of NoSQL Graph Databases », in *IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, Las Vegas, NV, USA: IEEE, p. 271-278, août 2020. doi :

10.1109/IRI49571.2020.00046.

[32] F. Machado, D. Saccol, E. Piveta, R. Padilha et E. Ribeiro, « A Text Similarity-based Process for Extracting JSON Conceptual Schemas », in *Proceedings of the 23rd International Conference on Enterprise Information Systems*, Online Streaming : SCITEPRESS - Science and Technology Publications, p. 264-271, 2021. doi : 10.5220/0010475102640271.

[33] J. L. Cánovas Izquierdo et J. Cabot, « JSONDiscoverer: Visualizing the schema lurking behind JSON documents », *Knowledge-Based Systems*, vol. 103, p. 52-55, juill. 2016. doi : 10.1016/j.knosys.2016.03.020.

[34] I. Comyn-Wattiau et J. Akoka, « Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j », in *IEEE International Conference on Big Data (Big Data)*, Boston, MA: IEEE, p. 453-458, déc. 2017. doi : 10.1109/BigData.2017.8257957.

[35] P. D. Muñoz-Sánchez, C. J. Fernández Candel, J. García Molina, et D. Sevilla Ruiz, « Managing Physical Schemas in MongoDB Stores », in *Advances in Conceptual Modeling*, G. Grossmann et S. Ram, Éd., in Lecture Notes in Computer Science, vol. 12584. Cham: Springer International Publishing, p. 162-172, 2020. doi : 10.1007/978-3-030-65847-2_15.

[36] M. E. Maicha, Y. Ouinten et B. Ziani, « UML4NoSQL : A Novel Approach for Modeling NoSQL Document-Oriented Databases Based on UML », *cai*, vol. 41, n° 3, p. 813-833, 2022. doi: 10.31577/cai_2022_3_813.

[37] N. Roy-Hubara, A. Sturm et P. Shoval, « Designing Document Databases : A Comprehensive Requirements Perspective », in *Advances in Conceptual Modeling*, I. Reinhartz-Berger et S. Sadiq, Éd., in Lecture Notes in Computer Science, vol. 13012. Cham: Springer International Publishing, p. 15-25, 2021. doi : 10.1007/978-3-030-88358-4_2.

[38] S. Hamouda, Z. Zainol et M. Anbar, « A Flexible Schema for Document Oriented Database (SDOD) », in *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Vienna, Austria : SCITEPRESS - Science and Technology Publications, p. 413-419, 2019. doi : 10.5220/0008353504130419.

[39] F. Abdelhedi, A. A. Brahim, F. Atigui et G. Zurfluh, « UMLtoNoSQL: Automatic Transformation of Conceptual Schema to NoSQL Databases », in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, Hammamet: IEEE, p. 272-279, oct. 2017. doi : 10.1109/AICCSA.2017.76.

[40] F. Abdelhedi, A. Ait Brahim, F. Atigui et G. Zurfluh, « MDA-Based Approach for NoSQL Databases Modelling », in *Big Data Analytics and Knowledge Discovery*, L. Bellatreche et S. Chakravarthy, Éd., in Lecture Notes in Computer Science, vol. 10440. Cham: Springer International Publishing, p. 88-102, 2017. doi : 10.1007/978-3-319-64283-3_7.

[41] G. Daniel, G. Sunyé et J. Cabot, « UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases », in *Conceptual Modeling*, I. Comyn-Wattiau, K. Tanaka, I.-Y. Song, S. Yamamoto, et M. Saeki, Éd., in Lecture Notes in Computer Science, vol. 9974. Cham: Springer International Publishing, p. 430-444, 2016. doi : 10.1007/978-3-319-46397-1_33.

[42] M.-A. Baazizi, D. Colazzo, G. Ghelli et C. Sartiani, « A Type System for Interactive JSON Schema Inference (Extended Abstract) », 13 pages, 2019. doi : 10.4230/LIPICS.ICALP.2019.101.

[43] F. Budinsky, Éd., « *Eclipse modeling framework: a developer's guide* », 6. printing. in The eclipse series. Boston, Mass. Munich : Addison-Wesley, 2007.

[44] X. Blanc et O. Salvatori, « *MDA en action : ingénierie logicielle guidée par les modèles* ». in Architecte logiciel. Paris: Eyrolles, 2005.

[45] A. H. Chillón, D. S. Ruiz et J. G. Molina, « Towards a Taxonomy of Schema Changes for NoSQL Databases: The Orion Language », in *Conceptual Modeling*, A. Ghose, J. Horkoff, V. E. Silva Souza, J. Parsons, et J. Evermann, Éd., in Lecture Notes in Computer Science, vol. 13011. Cham: Springer International Publishing, p. 176-185, 2021. doi : 10.1007/978-3-030-89022-3_15.

[46] U. Störl et M. Klettke, « Darwin: A Data Platform for Schema Evolution Management and Data Migration », DataPlat : 1st International Workshop on Data Platform Design, Management and Optimization, 2022.

[47] F. Abdelhedi, R. Jemmali et G. Zurfluh, « DLToDW: Transferring Relational and NoSQL Databases from a Data Lake », *SN COMPUT. SCI.*, vol. 3, n° 5, p. 381, juill. 2022. doi : 10.1007/s42979-022-01287-7.

[48] P. Koupil, S. Hricko et I. Holubová, « A universal approach for multi-model schema inference », *J Big Data*, vol. 9, n° 1, p. 97, août 2022. doi : 10.1186/s40537-022-00645-9.

[49] C. J. F. Candel, D. S. Ruiz et J. J. García-Molina, « A Unified Metamodel for NoSQL and Relational Databases », *Information Systems*, vol. 104, p. 101898, févr. 2022. doi : 10.1016/j.is.2021.101898.