



HAL
open science

Approche dirigée par les modèles pour l'extraction automatique du modèle NoSQL

Amal Aït Brahim, Rabah Tighilt Ferhat, Gilles Zurfluh

► To cite this version:

Amal Aït Brahim, Rabah Tighilt Ferhat, Gilles Zurfluh. Approche dirigée par les modèles pour l'extraction automatique du modèle NoSQL. Journées Francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2019), Oct 2019, Montpellier, France. pp.45-60. hal-02950715

HAL Id: hal-02950715

<https://hal.science/hal-02950715>

Submitted on 28 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:
<http://oatao.univ-toulouse.fr/26367>

Official URL

<https://editions-rnti.fr/?inprocid=1002545>

To cite this version: Ait Brahim, Amal and Tighilt Ferhat, Rabah and Zurfluh, Gilles *Approche dirigée par les modèles pour l'extraction automatique du modèle NoSQL*. (2019) In: Journées Francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2019), 3 October 2019 - 4 October 2019 (Montpellier, France).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Approche dirigée par les modèles pour l'extraction automatique du modèle NoSQL

Amal Ait Brahim *, Rabah Tighilt Ferhat * et Gilles Zurfluh *

*Institut de Recherche en Informatique de Toulouse (IRIT)
Université Toulouse Capitole - 2 Rue du Doyen-Gabriel-Marty - 31042 Toulouse - France
<prenom.nom>@irit.fr

Résumé. Les systèmes NoSQL permettent de gérer des bases de données (BD) massives vérifiant les 3V : Volume, Variété et Vitesse. La plupart de ces systèmes sont caractérisées par la propriété « schema less » qui signifie absence du modèle de données lors de la création d'une BD. Cette propriété apporte une souplesse indéniable en permettant l'évolution du modèle pendant l'exploitation de la base. Cependant, l'expression des requêtes d'interrogation (type SQL) exige une connaissance précise de ce modèle. Dans cet article, nous proposons un processus pour l'extraction automatique du modèle physique d'une BD NoSQL de type documents. Pour ce faire, nous utilisons l'architecture MDA (Model Driven Architecture) qui fournit un cadre formel pour la transformation automatique de modèles. À partir d'une BD NoSQL, nous proposons des règles de transformation formalisées avec QVT pour générer le modèle physique. Une expérimentation du processus d'extraction a été réalisée sur une application médicale.

1 Introduction

Ces dernières années ont vu l'explosion des données générées et accumulées par les dispositifs informatiques de plus en plus nombreux et diversifiés. Les BD ainsi constituées sont désignées par l'expression « Big Data » et sont caractérisées par la règle dite des « 3V » Chen et Zhang (2014). Celle-ci est due au volume des données qui peut dépasser plusieurs téraoctets et à la variété qui correspond au stockage de données de types et de formats diversifiés tels que des textes, des tableaux, des documents fortement structurés. De plus, ces données sont souvent saisies à très haute fréquence et doivent donc être filtrées et agrégées en temps réel pour éviter une saturation inutile de l'espace de stockage.

Les techniques d'implantation classiques, basées principalement sur le paradigme relationnel, connaissent des limites pour gérer les BD massives Angadi et al. (2013). Ainsi, de nouveaux systèmes de stockage et de manipulation des données ont été développés. Regroupés sous le terme NoSQL Han et al. (2011), ces systèmes sont bien adaptés pour gérer de gros volumes de données dont les modèles sont flexibles. Ils apportent aussi de grandes capacités de passage à l'échelle et de bonnes performances en temps de réponse Angadi et al. (2013).

La plupart des SGBD NoSQL sont caractérisés par la propriété « schema less » qui correspond à l'absence du schéma de données lors de la création d'une BD. Cette propriété apparaît

dans de nombreux systèmes NoSQL tels que MongoDB, CouchDB, HBase et Neo4j. Notons cependant qu'elle est absente dans quelques systèmes tels que Cassandra et Riak TS. La propriété « schema less » offre une souplesse indéniable en permettant de faire évoluer facilement le modèle. Par exemple, l'ajout de nouveaux attributs dans une ligne existante se fait sans modifier les autres lignes de même type préalablement stockées.

Actuellement, les SGBD NoSQL « schema less » ne disposent pas d'une fonctionnalité permettant d'afficher dynamiquement le modèle de la BD. Or, les utilisateurs (développeurs, Data scientists, Data analysts, etc.) ont besoin du modèle de données pour exprimer des requêtes d'interrogation et de mises à jour des données. Dans cet article, notre problématique consiste à extraire dynamiquement et automatiquement le modèle d'une BD NoSQL « schema less ». Pour cela, nous utilisons l'architecture MDA (Model Driven Architecture) qui offre un cadre formel à ce processus.

Le reste du document est structuré comme suit. La section 2 présente l'application médicale qui justifie l'intérêt de nos travaux. La section 3 passe en revue les travaux de l'état de l'art. La section 4 présente notre contribution qui consiste à formaliser avec MDA le processus d'extraction du modèle. La section 5 décrit les caractéristiques du prototype que nous avons développé. La section 6 décrit l'expérimentation et l'évaluation de notre processus.

2 Cas d'étude

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

L'objectif premier d'un tel programme est de collecter des données significatives sur l'évolution temporelle de la pathologie, d'étudier ses interactions avec des maladies opportunes et d'évaluer l'influence de ses traitements à court et moyen termes. La durée d'un programme est décidée lors de son lancement et peut atteindre entre trois et dix ans. Les données collectées par plusieurs établissements dans le cadre d'un programme pluriannuel, présentent les caractéristiques généralement admises pour le Big Data (les 3 V), Douglas (2001). En effet, le volume des données médicales recueillies quotidiennement auprès des patients, peut atteindre, pour l'ensemble des établissements et sur trois années, plusieurs téraoctets. D'autre part, la nature des données saisies (mesures, radiographie, scintigraphies, etc.) est diversifiée et peut varier d'un patient à un autre selon son état de santé. Enfin, certaines données sont produites en flux continu par des capteurs ; elles doivent être traitées quasiment en temps réel car elles peuvent s'intégrer dans des processus sensibles au temps (mesures franchissant un seuil qui impliqueraient l'intervention d'un praticien en urgence par exemple). Le suivi des patients exige le stockage de données variées telles que l'enregistrement des consultations effectuées par les praticiens, des résultats d'examens, des prescriptions de médicaments et de traitements spécifiques. Nous avons donc stocké l'ensemble de ces données dans un système NoSQL de type « schema less ».

Cette étude cas est un exemple typique d'applications où les utilisateurs ont besoin d'un outil permettant d'afficher le modèle de la BD. En effet, les médecins saisissent quotidiennement des mesures liées à l'état de leurs patients et peuvent ajouter, au fil de l'eau, de nouvelles mesures spécifiques si l'état pathologique de tel ou tel patient se modifie (évolution du modèle

par ajout de nouveaux attributs). Ensuite ces médecins qui sont aussi des décideurs, pourront analyser sur plusieurs années les données et suivre l'évolution de pathologies pour plusieurs cohortes de patients. Pour exprimer des requêtes correspondantes à ce type d'analyse, les médecins ont besoin de connaître le modèle de la BD.

3 Etat de l'art

Plusieurs travaux de recherche permettent d'extraire le modèle d'une BD NoSQL « schema less », ceci principalement pour les BD de type documents comme MongoDB . Ainsi, un processus a été proposé dans Klettke et al. (2015) pour extraire le modèle d'une collection de documents JSON stockée sur MongoDB. Le modèle retourné par ce processus est en format JSON ; il est obtenu en capturant les noms des attributs qui figurent dans les documents en entrée et en remplaçant leurs valeurs par leurs types. Les valeurs d'attributs peuvent être de type atomique, des listes ou des documents imbriqués.

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

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

Par ailleurs, l'article de Comyn-Wattiau et Akoka (2017) propose un processus d'extraction d'un modèle à partir des requêtes d'insertion d'objets et de relations dans une BD NoSQL orientées-graphes ; Le processus proposé repose sur une architecture MDA et applique aux requêtes deux types de transformations. Le processus est composé de deux traitements. Le premier consiste à construire un graphe (Nœuds + Arêtes) à partir des requêtes Neo4j. Le deuxième traitement consiste à extraire du graphe un modèle Entité/Association en transformant les nœuds portant la même étiquette en classes d'entités et les arêtes en classes d'associations.

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

D'autre part, dans Baazizi et al. (2017), les auteurs proposent un processus d'extraction du schéma d'une collection volumineuse de documents JSON en utilisant le système MapReduce. La phase Map consiste à extraire le schéma de chaque document de la collection en inférant des

couples (champ, type) à partir des couples (champ, valeur). La phase Reduce consiste à unifier tous les schémas produits dans la phase Map dans le but de fournir un schéma global de tous les documents de la collection. Dans un autre article Baazizi et al. (2019), les mêmes auteurs ont proposé d'étendre ce processus en intégrant le paramétrage de l'extraction au niveau de la phase Reduce. Ainsi, l'utilisateur peut choisir soit d'unifier tous les schémas des documents de la collection, soit d'unifier uniquement les schémas ayant les mêmes champs (noms et types).

Dans le tableau 1, nous synthétisons les travaux précédents en faisant apparaître leurs caractéristiques principales :

	BD comportant		Système NoSQL		Traitement des liens intra ou inter classes	
	Une seule classe	Plusieurs classes	Type documents	Type graphes	Oui	Non
Klettke et al. (2015)	X		X			X
Sevilla et al. (2015)		X	X			X
Gallinucci et al. (2018)	X		X			X
Comyn-Wattiau et al. (2018)		X		X	X	
Maity et al. (2018)	X		X	X		X
Baazizi et al. (2017)	X		X			X
Baazizi et al. (2019)	X		X			X

TAB. 1 – Tableau comparatif des travaux d'extraction du modèle à partir d'une BD NoSQL

Au travers de cet état de l'art, il apparait que ces solutions ne répondent que partiellement à notre problématique. En effet, dans Klettke et al. (2015), Gallinucci et al. (2018), Maity et al. (2018), Baazizi et al. (2017) et Baazizi et al. (2019), les auteurs proposent des processus qui prennent en entrée une unique classe d'objets (documents ou nœuds) et ne considèrent pas les liens entre les objets. De même, les travaux de Ruiz et al. (2015) ne traitent pas les liens bien qu'ils considèrent une BD constituée de plusieurs collections. D'autre part, les travaux de Comyn-Wattiau et Akoka (2017) ne prennent pas en considération les attributs structurés ; ceci est dû au fait que les systèmes orientés-graphes ne permettent pas de déclarer ce type d'attributs.

4 Extraction du modèle NoSQL

L'objectif de cet article est d'automatiser l'extraction du modèle des BD NoSQL « schema less » généralement réparties selon quatre types : clé/valeur, colonnes, documents et graphes. Nous nous limitons au type documents qui est le plus complet en termes d'expression des

liens (utilisation des références et des imbrications). Le processus *ToNoSQLmodel* que nous proposons, extrait automatiquement le modèle d'une BD NoSQL de type documents.

Pour formaliser et automatiser notre processus, nous utilisons l'architecture MDA (Model Driven Architecture) Hutchinson et al. (2011) de l'OMG qui offre un cadre formel pour l'automatisation des transformations de modèles. L'objectif de cette architecture est de décrire séparément les spécifications fonctionnelles et les spécifications d'implantation d'une application sur une plateforme donnée Hutchinson et al. (2011), Pour ceci, elle utilise trois modèles représentant les niveaux d'abstraction de l'application. Il s'agit (1) du modèle d'exigences (CIM pour Computation Independent Model) dans lequel aucune considération informatique n'apparaît, (2) du modèle d'analyse et de conception (PIM pour Platform Independent Model) indépendant des détails techniques des plateformes d'exécution et (3) du modèle de code (PSM pour Platform Specific Model) spécifique à une plateforme particulière. Etant donné que l'entrée de notre processus est une BD NoSQL et que sa sortie correspond à un modèle physique, nous retenons uniquement le niveau PSM.

Le passage d'une BD NoSQL vers son modèle se fait via une séquence de transformations. Nous allons formaliser ces transformations en utilisant le standard QVT (Query View Transformation) défini par l'OMG. La Figure 2 montre un aperçu de notre processus.

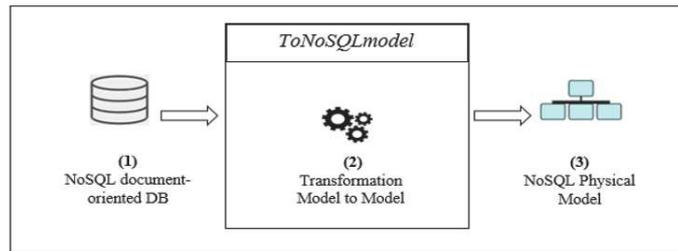


FIG. 1 – Notre processus *ToNoSQLmodel*

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

4.1 Source : une BD NoSQL orientée-documents

Une BD NoSQL orientée-documents est composée de collections. Chaque collection contient un ensemble de documents. Un document est constitué d'un ensemble de champs ; chacun d'eux est un couple (Nom-champ, Valeur). Une valeur peut être atomique, multivalué ou complexe (i.e. composée d'autres champs). Pour exprimer un lien entre les collections, nous avons utilisé un champ intitulé champ de référence selon un principe proposé dans MongoDB (2018). Celui-ci est un cas particulier de champ complexe contenant deux champs atomiques : l'identifiant du document référencé et le nom de la collection où le document référencé se trouve. Notons qu'un lien peut être multivalué ; dans ce cas, le champ de référence est composé d'un ensemble de champs de la forme (identifiant du document, nom de collection). Nous présentons ces différents concepts au travers le méta-modèle de la Figure 2.

Nous précisons que tous les méta-modèles présentés dans cet article sont formalisés avec le langage normalisé Ecore que nous présentons dans la section Expérimentation.

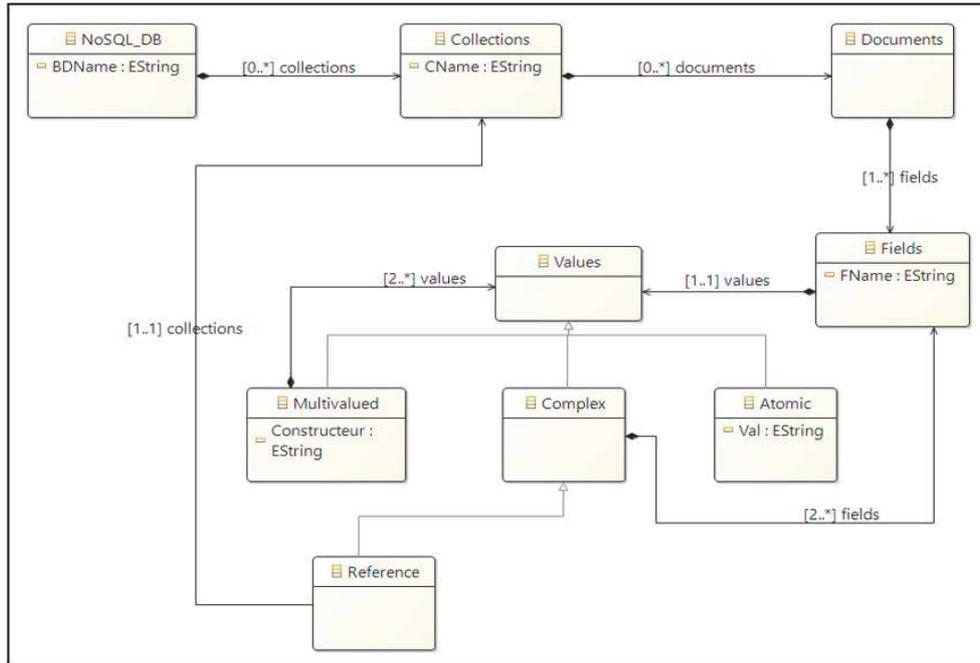


FIG. 2 – Méta-modèle décrivant la source de notre processus

4.2 Cible : Un modèle NoSQL

Le modèle NoSQL généré par notre processus est stocké dans une seule collection de documents. Chaque document contient le modèle d'une collection en entrée, Les documents sont composés d'un ensemble de champs ; Chacun d'eux peut être soit atomique soit complexe. Un champ atomique est présenté sous la forme d'un couple (Nom, Type) où Type correspond à un type de données prédéfini comme Integer, Boolean et String. Un champ complexe est composé d'un ensemble de champs qui, à leur tour, peuvent être soit atomiques soit complexes. Nous formalisons ces concepts à travers le méta-modèle Ecore de la Figure 3.

4.3 Transformations

Après avoir formalisé les concepts présents dans le modèle source (BD NoSQL orientée-documents) et dans le modèle cible (modèle physique NoSQL), nous allons décrire le passage automatique entre les deux modèles sous forme d'une séquence de règles de transformation QVT décrites ci-dessous.

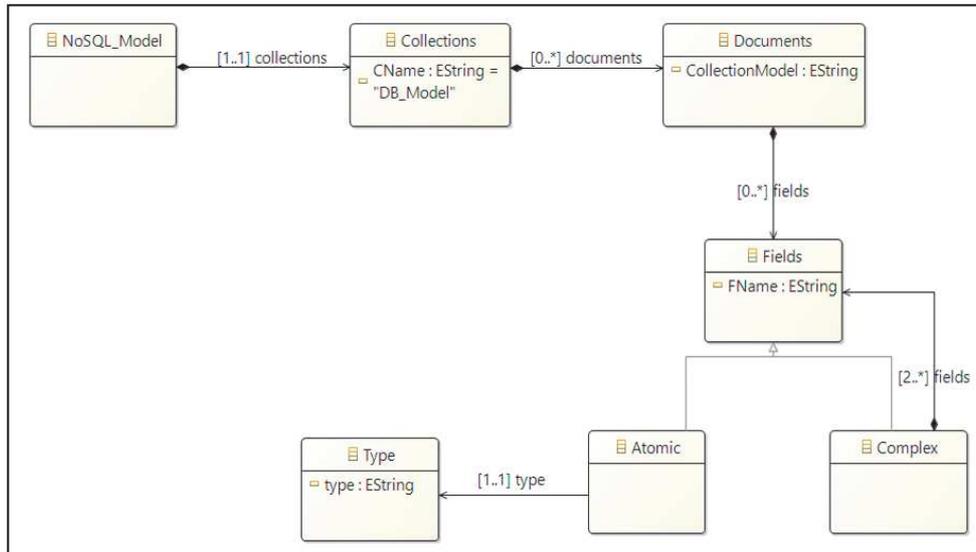


FIG. 3 – Méta-modèle décrivant la cible de notre processus

R1 : Une BD NoSQL orientée-documents est transformée en une collection appelée DB_Model.

R2 : Chaque collection en entrée est transformée en un document. Celui-ci comporte un champ noté CollectionModel qui indique le nom de la collection concernée. Notons que chaque document contient un modèle unifié pour tous les documents qui composent la collection en entrée. Ceci signifie que notre processus génère un modèle de collection unique regroupant l'ensemble des champs des documents. Nous ne considérons donc pas plusieurs versions de modèles pour la même collection en entrée comme dans de Ruiz et al. (2015) et Gallinucci et al. (2018). En effet, nous ne traitons pas le cas des champs synonymes ni le cas où un champ a des types différents au sein d'une même collection.

R3 : Pour chaque champ atomique, le couple (Nom-champ, Valeur) est transformé en un couple (Nom-champ, Type). En effet, le type d'un champ est généré en fonction des caractéristiques de sa valeur. Par exemple, si la valeur est de la forme "xxx", alors son type est String ; si la valeur est de la forme [yyy, zzz, etc], alors son type est [Integer].

R4 : Pour un champ complexe, le processus parcourt l'ensemble des champs qui le composent. Pour chacun d'eux, s'il s'agit d'un champ atomique, alors appliquer R3 ; s'il s'agit d'un champ complexe, alors appliquer R4.

R5 : Pour un champ de référence, le couple (identifiant du document, nom de la collection) est transformé en un couple (ObjectId, nom de la collection) s'il s'agit d'un champ monovalué ou en un couple (ObjectId, [nom de la collection]) s'il s'agit d'un champ multivalué.

Nous présentons dans la figure 4 un exemple d'application de ces règles de transformation. Considérons les deux collections Patients et Doctors décrites dans la source. Après l'exécution de nos règles, nous obtenons le modèle présenté dans la cible. Notons que le résultat de la

transformation du champ de référence monovalué Treating-Doctor est donné en gras :

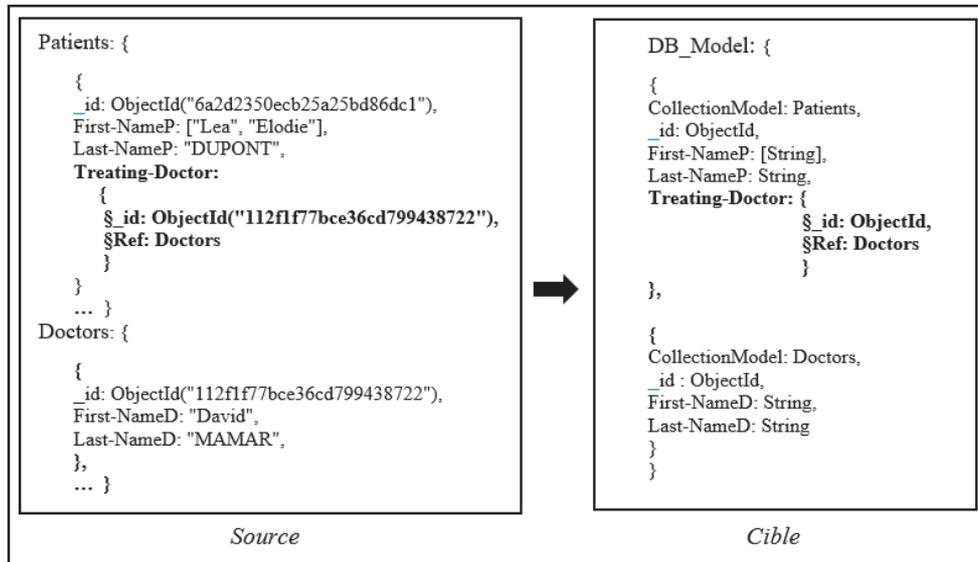


FIG. 4 – Exemple de transformation

5 Développement du prototype

5.1 Environnement technique

Nous décrivons succinctement les techniques que nous avons utilisées pour mettre en œuvre la démarche présentée dans la Figure 1. Etant donné que notre approche est dirigée par les modèles, nous avons utilisé un environnement technique adapté à la modélisation, la méta-modélisation et la transformation des modèles. Nous avons eu recours à la plateforme Eclipse Modeling Framework (EMF) Budinsky et al. (2004). EMF fournit un ensemble d'outils destinés à introduire une approche de développement dirigée par les modèles au sein de l'environnement Eclipse. Ces outils apportent trois fonctionnalités principales. La première est la définition d'un méta-modèle représentant les concepts manipulés par l'utilisateur. La deuxième est la création des modèles instanciant ce méta-modèle et la troisième fonctionnalité correspond à la transformation de modèle vers modèle et de modèle vers texte. Parmi les outils fournis par EMF, nous avons utilisé :

- Ecore : un langage de méta-modélisation utilisé pour la création de nos méta-modèles. Les Figure 2 et 3 illustrent les méta-modèles Ecore source et cible, utilisés par notre processus ToNoSQLmodel.
- XMI (XML Metadata Interchange) : qui est une norme utilisée pour représenter des modèles en XML.

- QVT (Query, View, Transformation) : qui est un langage standardisé pour exprimer les transformations de modèles. Le choix du standard QVT a été fondé sur des critères spécifiques à notre démarche. En effet, l’outil de transformation doit être intégré dans l’environnement EMF pour qu’il soit utilisé aisément avec les outils de modélisation et de méta-modélisation.

Nous avons mis en œuvre notre processus sur des données issues de l’application médicale présentée en section 2. L’aspect essentiel, montrant l’intérêt de notre processus, se situe dans la variété des données utilisées : notamment du texte, des données multivaluées, des documents structurés.

5.2 Implantation du processus ToNoSQLmodel

ToNoSQLmodel est exprimé sous la forme d’une séquence d’étapes élémentaires qui construisent le modèle résultant (modèle NoSQL) pas à pas à partir du modèle source (BD NoSQL de type documents) :

Étape 1 : nous créons des méta-modèles Ecore correspondant à la source (Figure 2) et à la cible (Figure 3).

Étape 2 : nous construisons une instance du méta-modèle source pour produire la BD NoSQL de type documents (cf. Figure 5); cette BD est un extrait des données de l’application médicale décrite dans la section 2 et stockée sous la forme d’un fichier XMI.

Étape 3 : nous implantons les règles de transformation à l’aide du langage QVT fourni par EMF. Un extrait du script QVT est présenté à la Figure 7 ; les commentaires dans le script indiquent les règles utilisées.

Étape 4 : nous testons les transformations en exécutant le script QVT créé à l’étape 3. Ce script prend en entrée le modèle source créé à l’étape 2 et renvoie en sortie le modèle NoSQL. Le résultat est fourni sous forme de fichier XMI, comme illustré dans la Figure 6.

Notre BD source contient des données variées comme le montre la Figure 5 : des champs atomiques multivalués (tel First-name dans la collection Patients), des champs complexes (Adress dans la collection Patients), des champs de référence monovalués (Performed dans la collection Consultations et Competent dans la collection Doctors) ainsi que des champs de référence multivalués (Medical-histories dans la collection Patients).

La Figure 6 montre le modèle de données résultant de notre processus ToNoSQLmodel. Ce modèle est généré sous forme de fichier XMI. Pour une meilleure lisibilité, nous donnons l’équivalent de la figure 6 sous forme d’un DCL comme le montre la figure 8. Il s’agit d’une description graphique des structures de données stockées dans le système MongoDB que nous avons utilisé dans notre expérimentation. Notons que MongoDB étant un système « schema less », il ne fournit pas ce modèle, ni sous forme textuelle, ni sous forme graphique

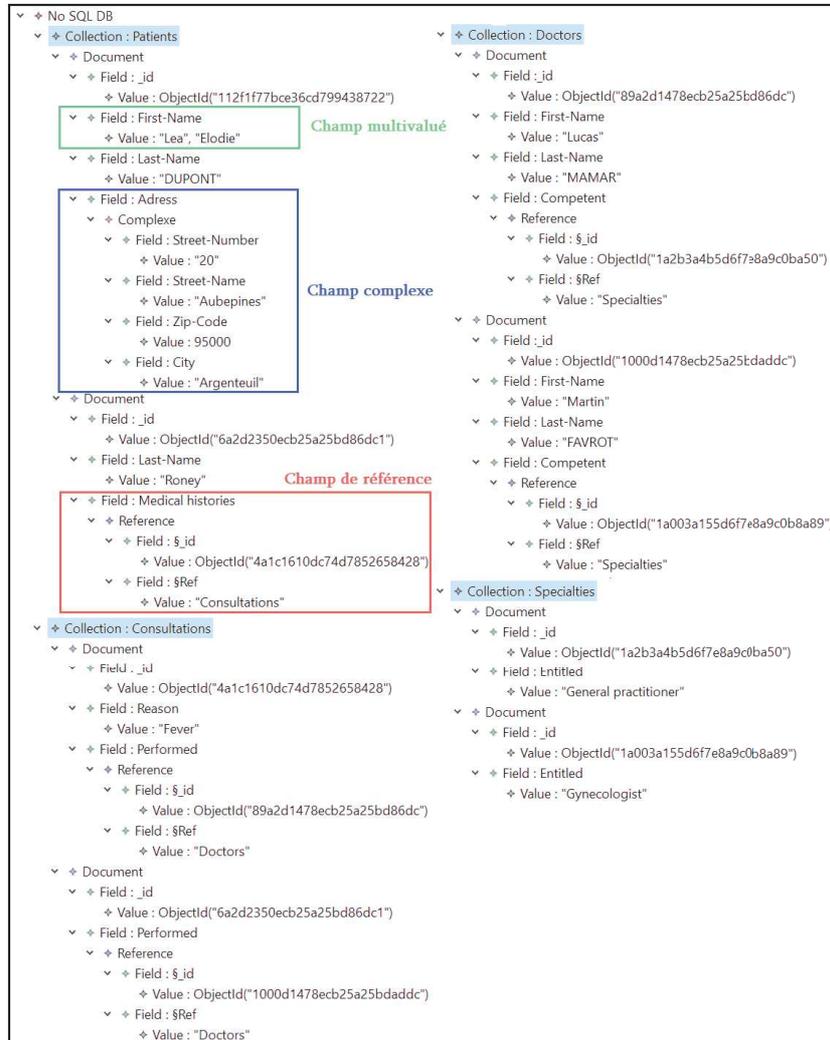


FIG. 5 – Modèle de la source

6 Expérimentation et discussion

Notre problématique consiste à extraire le modèle d'une BD gérée par un système NoSQL « schema less ». Une telle fonctionnalité est destinée à des utilisateurs ne connaissant pas a priori la structure des données tels que des développeurs n'ayant pas créé la BD ou des décideurs ; son intérêt majeur est de permettre la rédaction de requêtes de type SQL comme on peut le faire dans les systèmes relationnels.

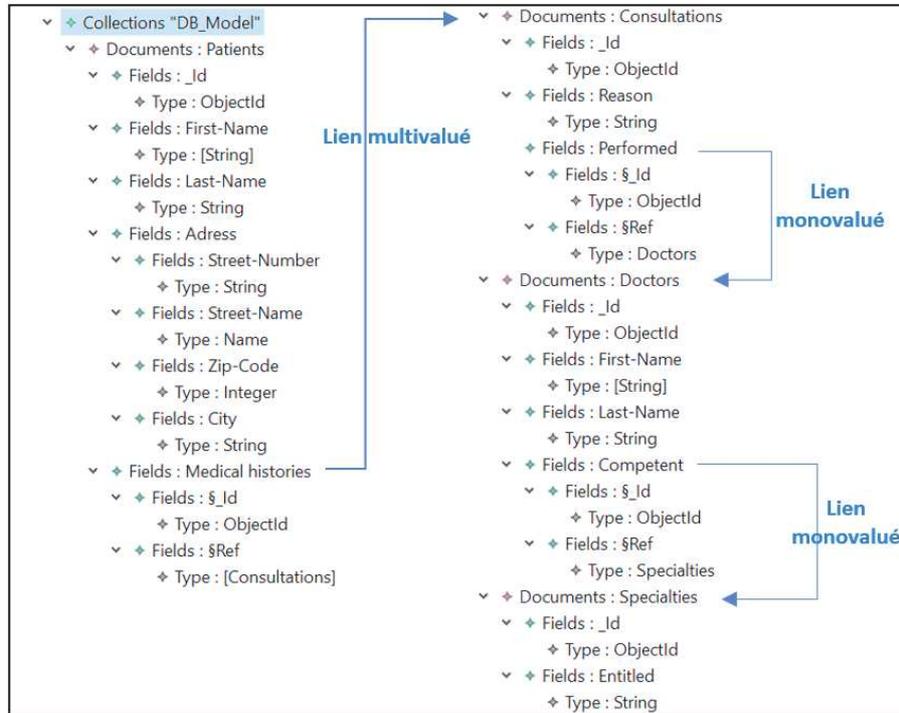


FIG. 6 – Modèle de la cible

6.1 Environnement expérimental

Les expérimentations de notre proposition ont été effectuées sur un cluster composé de 3 machines. Chaque machine présente les spécifications suivantes : Intel Core i5, 8 Go de RAM et 2 To de disque. L'une de ces machines est configurée pour agir en tant que maître ; les deux autres machines ont le statut d'esclave.

Pour mettre en œuvre notre solution, nous avons utilisé des outils de génération de données Data (2018) et Generator (2016). Nous avons produit un ensemble de données de 3 To sous la forme de fichiers JSON. Ces fichiers ont été chargés dans MongoDB à l'aide de commandes shell.

Pour notre expérimentation, nous avons considéré quatre types de requêtes. La complexité de ces requêtes augmente graduellement comme le montre le tableau 2. La requête la plus simple porte sur une collection unique ; la requête la plus complète porte sur deux collections avec un lien multivalué.

```

modeltype NoSQLdb uses "http://nosqldatabaseMM.com";
modeltype NoSQLSchema uses "http://nosqlschemaMM.com";
transformation NoSQLdb2NoSQLschema(in Source: NoSQLdb, out Target: NoSQLSchema);
main() {
  Source.rootObjects()[NoSQL_DB] -> map toNoSQL_Schema();
  mapping NoSQLdb::NoSQL_DB::toNoSQL_Schema():NoSQLSchema::NoSQL_Schema{
    sName:=self.dbName;
    collection:=self.collections -> map toCollection();
    - Transforming Collections
    mapping NoSQL_DB::Collections::toCollection():NoSQL_Schema::Documents{
      dName:=self.cName;
      atomicOUTfield:=self.atomicINfield -> map toAtomicField();
      complexeOUTfield:=self.complexeINfield -> map toComplexeField();
      - Transforming Atomic Fields
      mapping NoSQL_DB::AtomicINField::toAtomicField():NoSQL_Schema::AtomicOUTField{
        fieldOUTname:=self.fieldINname -> map toFieldName();
        fieldOUTvalue:=self.fieldINvalueform -> map toFieldValue1();
        fieldOUTvalue:=self.fieldINvalue -> map toFieldValue2();
      }
      mapping
      NoSQL_DB::FieldINName::toFieldName():NoSQL_Schema::FieldUName{NameU:=self.NameI;};
      mapping NoSQL_DB::FieldINValue::toFieldValue1():NoSQL_Schema::FieldUValue{
        if ((self.FieldIValue = "True") or (self.FieldIValue = "False")) {FieldUValue:= "Boolean";}
        FieldUValue:= "Number"; endif;
      }
      mapping NoSQL_DB::FieldINValueForm::toFieldValue2():NoSQL_Schema::FieldUValue{
        if (self.FieldINValueForm = "") {FieldOUTValue:= "String";} endif;
        if (self.FieldINValueForm = "--/--") {FieldOUTValue:= "Date";} endif;
      }
      - Transforming Complexes Fields
      mapping NoSQL_DB::ComplexeINField::toComplexeField():NoSQL_Schema::ComplexeOUTField{

```

FIG. 7 – Extrait du code QVT

6.2 Discussion

Le tableau 3 compare notre solution à celles des travaux présentés dans la section Etat de l'art. Les requêtes figurant dans le tableau 2 ont été mises en œuvre avec notre système ToNoSQLModel puis évaluées manuellement avec les structures de données proposées dans chaque article. Notons cependant que nous prenons en considération uniquement les travaux portant sur des BD NoSQL orientées-documents à l'exclusion des BD orientée-graphes ; celles-ci n'étant pas adaptées à notre domaine d'application.

L'homogénéité du résultat de la comparaison entre le modèle extrait par notre système et ceux extraits par les systèmes référencés, provient de l'absence de prise en compte des liens entre les collections pour ces derniers. Or, ces liens sont nécessaires pour exprimer des requêtes complexes. Considérons la requête de la figure 9 qui permet d'obtenir le médecin-traitant pour le patient « DUPONT ». Grâce au lien « Treating-Doctors » dans la collection Patients, la requête MongoDB applique une jointure sur les collections Patients et Doctors. Nous voyons bien que nous ne pouvons pas écrire cette requête si nous visualisons pas le lien monovalué entre Patients et Doctors.

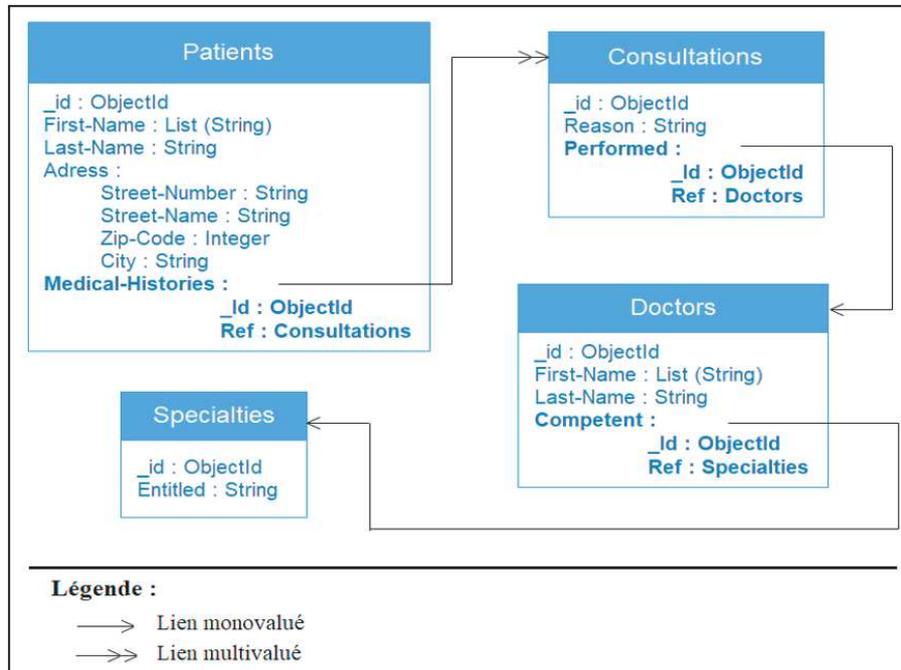


FIG. 8 – Extrait du modèle physique des données

Requête	Caractéristiques
Q1	porte sur une seule collection.
Q2	porte sur deux collections reliées entre elles avec un lien exprimé en utilisant l'imbrication.
Q3	porte sur deux collections reliées entre elles avec un champ de référence monovalué.
Q4	porte sur deux collections reliées entre elles avec un champ de référence multivalué.

TAB. 2 – Tableau descriptif des caractéristiques des requêtes

Catégorie de requêtes	Klettke et al. (2015)	Sevilla et al. (2015)	Gallinucci et al. (2018)	Maity et al. (2018)	Baazizi et al. (2017)	Baazizi et al. (2019)	ToNoSQLModel
Q1	X	X	X	X	X	X	X
Q2		X					X
Q3							X
Q4							X

TAB. 3 – Résultats de comparaison entre notre solution et des travaux de l'état de l'art

```
db.Patients.aggregate ({
  $lookup: {from: "Doctors", localField: "Treating-Doctors_id", foreignField: "_id", as: "Doctors"}},
  {$match: {"Last-Name": "DUPONT"}},
  {$project: {"Doctors.Last-Name": true}}})
```

FIG. 9 – Exemple de requête complexe

7 Conclusion

Nos travaux s'inscrivent dans le cadre des bases de données de type Big Data. Ils portent actuellement sur les mécanismes d'extraction du modèle d'une BD NoSQL « schema less » pour faciliter l'expression de requêtes.

Dans cet article, nous avons proposé un processus automatique pour extraire le modèle physique d'une BD NoSQL de type documents. Ce processus, basé sur l'architecture MDA (Model Driven Architecture), offre un cadre formel pour l'automatisation des transformations de modèles. Il génère le modèle physique d'une BD NoSQL de type Document en appliquant une séquence de transformations formalisées avec le standard QVT. Le modèle retourné décrit la structure des collections qui composent la BD. L'apport majeur de notre solution se situe dans la prise en compte des liens monovalués et multivalués entre les collections. Nous avons expérimenté notre processus sur le cas d'une application médicale qui porte sur des programmes scientifiques de suivi de pathologies ; la BD est stockée sur le système NoSQL MongoDB.

En ce qui concerne les perspectives, nous prévoyons d'étudier la mise à jour du modèle de données au fur et à mesure de l'exploitation de la BD. En effet, le volume des données pouvant atteindre plusieurs téraoctets, la génération du modèle nécessite le balayage de la totalité de la BD. Il n'est pas donc envisageable qu'un utilisateur relance le processus chaque fois qu'il souhaite exprimer une nouvelle requête.

Références

- Angadi, A. B., A. B. Angadi, et K. C. Gull (2013). Growth of new databases & analysis of nosql datastores. *International Journal of Advanced Research in Computer Science and Software Engineering* 3(6).
- Baazizi, M.-A., D. Colazzo, G. Ghelli, et C. Sartiani (2019). Parametric schema inference for massive json datasets. *The VLDB Journal*, 1–25.
- Baazizi, M.-A., H. B. Lahmar, D. Colazzo, G. Ghelli, et C. Sartiani (2017). Schema inference for massive json datasets. In *Extending Database Technology (EDBT)*.
- Budinsky, F., D. Steinberg, R. Ellersick, T. J. Grose, et E. Merks (2004). *Eclipse modeling framework : a developer's guide*. Addison-Wesley Professional.
- Chen, C. P. et C.-Y. Zhang (2014). Data-intensive applications, challenges, techniques and technologies : A survey on big data. *Information sciences* 275, 314–347.

- Comyn-Wattiau, I. et J. Akoka (2017). Model driven reverse engineering of nosql property graph databases : The case of neo4j. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 453–458. IEEE.
- Data, G. T. (2018). Generate test data. <http://www.convertcsv.com/generate-test-data.htm>. Online; 5 July 2019.
- Douglas, L. (2001). 3d data management : Controlling data volume, velocity and variety. *Gartner. Retrieved 6(2001)*, 6.
- Gallinucci, E., M. Golfarelli, et S. Rizzi (2018). Schema profiling of document-oriented databases. *Information Systems 75*, 13–25.
- Generator, J. (2016). Json generator. <http://www.json-generator.com/>. Online; 5 July 2019.
- Han, J., E. Haihong, G. Le, et J. Du (2011). Survey on nosql database. In *2011 6th international conference on pervasive computing and applications*, pp. 363–366. IEEE.
- Hutchinson, J., M. Rouncefield, et J. Whittle (2011). Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 633–642. ACM.
- Klettke, M., U. Störl, et S. Scherzinger (2015). Schema extraction and structural outlier detection for json-based nosql data stores. *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*.
- Maity, B., A. Acharya, T. Goto, et S. Sen (2018). A framework to convert nosql to relational model. In *Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology*, pp. 1–6. ACM.
- MongoDB (2018). MongoDB atlas database as a service. <https://www.mongodb.com/>. Online; 5 July 2018.
- Ruiz, D. S., S. F. Morales, et J. G. Molina (2015). Inferring versioned schemas from nosql databases and its applications. In *International Conference on Conceptual Modeling*, pp. 467–480. Springer.

Summary

NoSQL systems allow to manage databases verifying 3V: Volume, Variety and Velocity. Most of these systems are characterized by the property "schema less" which means absence of the data model when creating a database. This property brings an undeniable flexibility by allowing the evolution of the model during the exploitation of the database. However, query expression requires precise knowledge of this model. In this paper, we propose a process to automatically extract the physical model from a document-type NoSQL database. For this, we use the Model Driven Architecture (MDA), which provides a formal framework for automatic model transformation. From a NoSQL database, we propose formal transformation rules with QVT to generate the physical model. An experimentation of the extraction process was carried out on a medical application.