L'implantation de sources de données dans un système NoSQL : formalisation des règles de passage conceptuel/logique.

Fatma Ben Messaoud, Amal Aït Brahim, Gilles Zurfluh

L'implantation de sources de données dans un système NoSQL : formalisation des règles de passage conceptuel/logique

Fatma Abdelhedi[*,**], Amal Ait Brahim[*] et Gilles Zurfluh[*]

[*] IRIT - Université Toulouse Capitole - France -
<prenom.nom>@irit.fr
[**] CBI[2] - Sté TRIMANE Saint Germain en Laye - France -
<prenom.nom>@gmail.com

Résumé. La transformation digitale des entreprises et plus largement celle de la société, entraine une évolution des bases de données vers le Big Data. Nos travaux s'inscrivent dans cette mutation et concernent plus particulièrement les mécanismes d'implantation d'une base de données sur une plateforme NoSQL. Pour automatiser ce processus, nous avons spécifié des algorithmes pour traduire un schéma conceptuel en un schéma logique NoSQL. A partir d'un diagramme de classes d'UML décrivant une base d'objets complexes, nous proposons des procédures de correspondance pour générer un schéma d'implantation destiné à une plateforme NoSQL orientée colonnes. Nous introduisons un schéma intermédiaire de niveau logique afin de limiter les impacts liés aux évolutions techniques des plateformes NoSQL. Une expérimentation du processus de correspondance a été réalisée sur une application médicale.

1 Introduction

Les applications Big Data, Chen et Zhang (2014), développées dans les domaines tels que le spatial, la santé ou la gestion commerciale, répondent à un double objectif : (1) assurer le passage à l'échelle (ou scalabilité), c'est-à-dire répartir les données et distribuer les traitements sur un nombre important de machines afin d'être en mesure de stocker de très grands volumes de données et d'absorber des charges très importantes ; (2) manipuler les données complexes avec des outils qui prennent en compte la répartition logique de ces données.

Notre problématique générale est de proposer des modèles et des outils décisionnels capables de localiser des sources de données pertinentes, d'alimenter des entrepôts et d'exploiter ces derniers à des fins d'analyse. Les présents travaux se situent en amont dans ce processus décisionnel, au niveau des sources de données utilisées pour alimenter les entrepôts. Avant l'avènement du Big Data, ces sources de données étaient principalement constituées de bases de données relationnelles, de fichiers informatiques et de documents formatés en HTML ou XML. Avec la diffusion des plateformes NoSQL, les systèmes de décision doivent intégrer de nouvelles sources de données pour alimenter les entrepôts. Ainsi, des systèmes d'alimentation

d'entrepôts comme Talend [1] offrent des fonctionnalités pour charger, extraire et améliorer (nettoyer et enrichir) des données disparates, tout en tirant parti de la puissance de traitement massivement parallèle des technologies de Big Data, comme Hadoop, Grover et al. (2015) et les bases de données NoSQL, Abhinay et al. (2013). Nos travaux visent donc à intégrer des sources Big Data dans un processus décisionnel. Le présent article consiste à proposer des règles de passage d'un schéma conceptuel décrivant une source Big Data, en un modèle NoSQL orienté colonnes. Une expérimentation est effectuée pour une application d'informatique médicale qui doit être implantée sur la plateforme Hadoop.

2 Motivation

2.1 Étude de cas

Pour illustrer nos travaux, nous utilisons un cas extrait d'une application médicale dont la base de données est représentée avec le formalisme UML. Cet exemple nous permettra de montrer comment passer d'un diagramme de classes (DCL) d'UML (vers un schéma NoSQL, Gajendran (2012). Il s'agit de la mise en place de programmes nationaux ou internationaux pour le suivi de cohortes de patients atteints de pathologies graves. L'objectif majeur d'un tel programme est de collecter des données sur l'évolution temporelle d'une pathologie particulière, d'étudier les interactions de la pathologie avec des maladies opportunes et d'évaluer l'influence des traitements et médications à court et moyen termes. La durée d'un programme est décidée lors de son lancement et peut atteindre trois 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), Doug (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. L'extrait du diagramme UML de la figure 1 montre quelques classes pour un programme médical associé au suivi des patients atteints d'une pathologie particulière.

1. Talend.https ://fr.talend.com/products/big-data

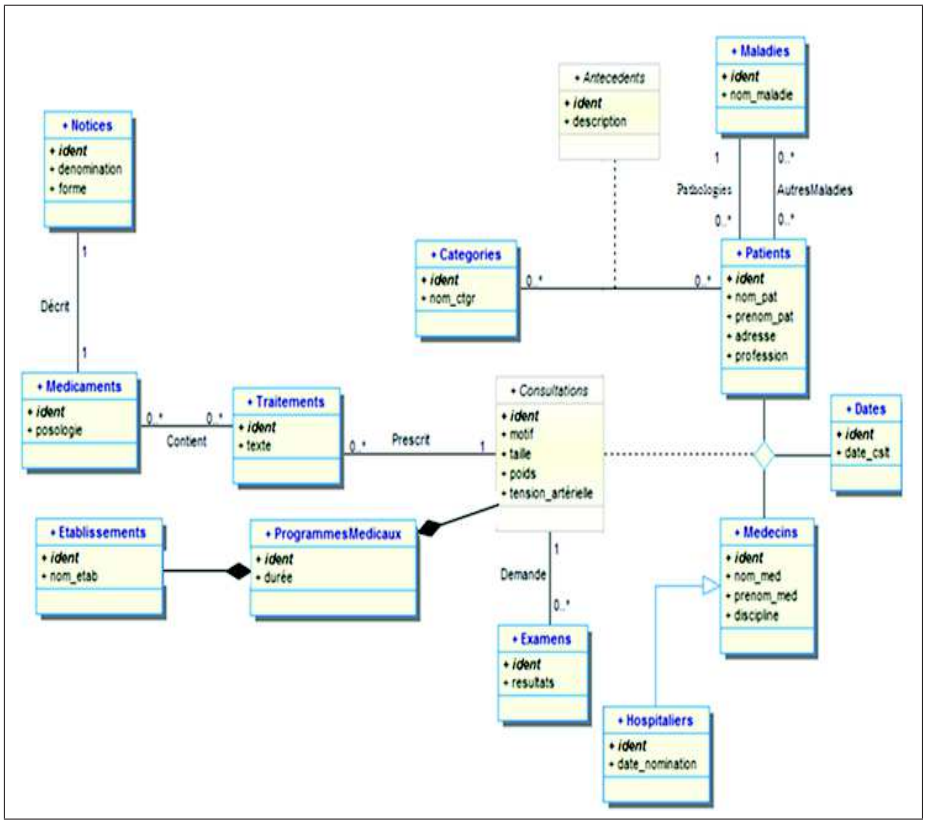

FIG. 1 – *Extrait d'un schéma des données.*

2.2 État de l'art

Une base de données de type Big Data contient des données variées, c'est-à-dire des données de types non standard qualifiés généralement d'objets complexes : textes, graphiques, documents, séquences vidéo, etc. Pour implanter de telles bases de données, des études ont porté sur la modélisation conceptuelle des objets complexes et ont montré que l'on pouvait appliquer ces modèles au Big data. D'autres travaux ont proposé des processus de transformation d'un schéma de bases de données vers un schéma NoSQL.

2.2.1 Modélisation des données complexes

La modélisation des données complexes, Darmont et al. (2005), a fait l'objet de nombreux travaux de recherche ; nous allons nous focaliser sur trois d'entre eux : Pedersen et Jensen (1998), Tanasescu et al. (2005), Midouni et al. (2009) que nous avons considérés comme les travaux les plus marquants dans ce contexte. L'approche de Tanasescu et al. (2005) consiste à concevoir un diagramme UML pour identifier et représenter conceptuellement les données complexes afin de les préparer au processus de modélisation multidimensionnelle. Dans le

domaine médical, Pedersen et Jensen (1998) ont proposé un modèle multidimensionnel pour les données complexes qui modélise les données temporelles et imprécises respectivement par l'ajout du temps de validité et des probabilités au modèle. Nous pouvons citer aussi le travail de Midouni et al. (2009) qui s'intéresse au traitement de la complexité des données médicales ; ils ont étendu un modèle en constellation en introduisant de nouveaux concepts permettant la présentation des données biomédicales. Dans le même article, les auteurs ont proposé une approche de modélisation et d'implantation d'un entrepôt médical en se basant sur le modèle étendu.

Aujourd'hui, le modèle de données UML représente une sorte de référence en matière de représentation de schémas de bases de données complexes. Ce modèle conceptuel, permettant de décrire la sémantique des objets métiers dans une application, peut donc être appliqué à la description des bases de données de type Big Data.

2.2.2 Transformation des modèles

Dans le contexte des entrepôts de données, les travaux de Chevalier et al. (2015) ont défini des règles pour traduire un modèle multidimensionnel en étoile, en deux modèles physiques NoSQL, un modèle orienté colonnes et un modèle orienté documents. Les liens entre faits et dimensions ont été traduits sous la forme d'imbrications.

Dans Li (2010), ont été étudiés les mécanismes d'implantation d'une base de données relationnelle dans le système HBase. La méthode proposée est basée sur des règles permettant la correspondance d'un schéma relationnel en un schéma HBase ; les relations entre les tables (clés étrangères) sont traduites par l'ajout des familles de colonnes contenant des références.

D'autres travaux, Yan et al. (2014), ont étudié la transformation d'un DCL en un schéma de données HBase avec l'approche MDA. L'idée de base est de construire des méta-modèles correspondant au diagramme de classes UML et au modèle de données orienté colonnes HBase puis de proposer des règles de transformation entre les éléments des deux méta-modèles construits. Ces règles permettent de transformer un DCL directement en un schéma d'implantation spécifique au système HBase.

Cet état de l'art montre que peu de travaux antérieurs ont étudié la correspondance d'un modèle conceptuel de données complexes avec un modèle Big Data. Dans le travail le plus proche de notre problématique, Yan et al. (2014), les règles de transformation de schémas proposées ne sont pas compatibles avec d'autres systèmes NoSQL orienté colonnes, tels que Cassandra et BigTable, le modèle de transformation proposé ne considére pas un niveau logique indépendant de toute plateforme technique.

3 Contribution

Nous reprenons l'architecture classique de la modélisation des données qui distingue les niveaux conceptuel et interne, Fankam et al. (2008) ; au niveau interne (ou technique), nous considérons les niveaux logique et physique. A partir de cette architecture, nous proposons un processus de correspondance entre un diagramme de classes conceptuel et un modèle logique NoSQL orienté colonnes. Quant à la correspondance entre les niveaux logique et physique (modèle d'implantation propre à un système propriétaire), il ne fera pas l'objet d'une présentation détaillée dans cet article.

L'étude de ces principes de correspondance sont à la base des travaux que nous menons sur les mécanismes ETL dédiés à l'alimentation d'un entrepôt à partir de sources Big Data. Ces mécanismes, tenant compte de la sémantique des données, nécessitent de connaître les descriptions conceptuelles des sources. La figure 2 illustre notre approche qui consiste à passer d'un DCL à un modèle de type orienté colonnes puis, dans un second temps, à un modèle physique (schéma HBase ou Cassandra).

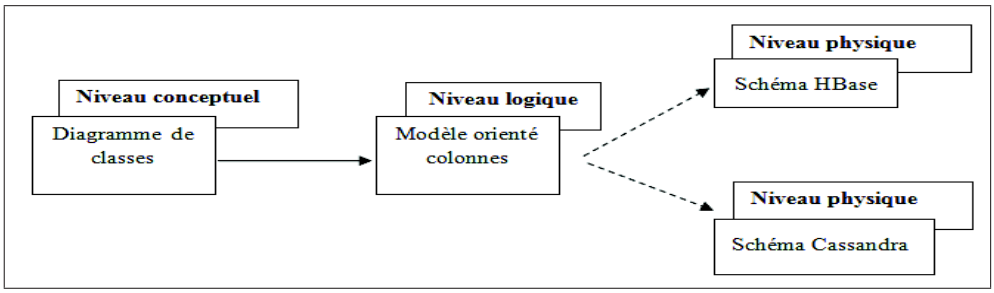

FIG. 2 – *Implantation d'une base de données : les niveaux de description.*

Le passage du niveau conceptuel au niveau logique est assuré par des procédures de correspondance entre les éléments des modèles correspondants.

3.1 Niveau conceptuel

Dans un systéme décisionnel, les données à analyser sont extraites de sources multiples. De nombreux projets reposent sur des sources de type Big Data comme le suggère l'exemple de la section 2.1. UML étant le modèle reconnu par la communauté des bases de données pour représenter des objets complexes [2], nous décrivons le schéma d'une source Big Data sous la forme d'un DCL d'UML.

Un DCL contient un ensemble de classes $\{C_1,...,C_p\}$. Chaque classe regroupe des objets ayant une sémantique et des propriétés communes ; elle est définie par son nom, ses attributs et ses opérations (dans cet article, nous ne prenons pas en compte les opérations).

Le schéma d'une classe C $\in$ DCL est un triplet (N, A, Ident) où :

- C.N est le nom de la classe.
- C.A = $\{a_1,...,a_q\}$ est un ensemble de q attributs. Le schéma d'un attribut a $\in$ A est un couple (N : C) où « a.N » représente le nom de l'attribut et « a.C » la classe qui le définit ; C peut être une classe prédéfinie, c'est-à-dire un type de données standard (String, Integer, Date,...) ou une classe définie explicitement par l'utilisateur appelée classe-utilisateur.
- C.Ident est un identificateur d'objet de type Oid (Object Identifier des systèmes objet) géré automatiquement par le système pour chaque classe du DCL.

Par exemple, la classe Patients de la Figure 1 est définie comme suit :
(Patients , {(nom-pat : String), (prénom-pat : String), (adresse : Adr), (date-naiss : Date), (profession :String)}, (ident : Oid))

2. http ://www.omg.org/spec/UML/2.5/

(Adr , {(cp : Integer, ville : String)}). -- *Classe définie par l'utilisateur et jouant le rôle de type*.

Une association n-aire dans un DCL est exprimée sous la forme d'une classe de n attributs. Chacun d'eux est associé à une classe utilisateur ; ces attributs références permettent d'établir des liens lors de la valorisation des données. Par exemple dans la figure 1, le lien Pathologie entre Patients et Maladies se traduit par une classe définie comme suit :
(Pathologies, {(pat : Patients), (patho : Maladies)},(ident : Oid))

Lors de l'instanciation de Pathologies, les attributs pat et patho prendront respectivement la référence d'un objet de la classe Patients et de la classe Maladies.

Un lien peut être multivalué ; c'est le cas du lien AutresMaladies entre les classes Patients et Médecins dans la figure 1. Le lien s'exprime alors comme suit :
(AutresMaladies , {(pat : Patients), (patho : set(Maladies)}, (ident :OID)). -- *l'attribut patho peut prendre plusieurs valeurs*.

Dans le cas où un lien présente des propriétés dans une classe d'associations, c'est celle-ci qui contiendra les attributs références. La classe d'associations Consultations s'exprime alors comme suit :
(Consultations , {(pat : Patients), (med : Medecins), (dte : Dates), (motif :String), (taille :Float), (Poids : Float)}, (ident :Oid))

Un lien d'agrégation ou de composition dans un DCL s'exprime par l'ajout, dans la classe composite, d'un attribut référençant la classe composante. Cet attribut prendra un ensemble de références d'objets de la classe composante. Par exemple dans la figure 1, le lien de composition entre ProgrammesMédicaux et Etablissements se traduit comme suit : ProgrammesMédicaux : {..., (etab : set(Etablissements)) ,...}

Un lien d'héritage entre deux classes s'exprime par l'ajout, dans la sous-classe, d'un attribut référençant la super-classe. Ainsi le lien d'héritage de la figure 1 qui relie les classes Médecins et Médecins-Hospitalier, se traduit par l'attribut (médec : Médecins) dans la classe Médecins-Hospitalier.

3.2 Niveau logique

Dans le niveau logique de description d'une base de données, les choix d'implantation ne sont pas complètement spécifiés. Les principes d'organisation des données sont précisés mais il est fait abstraction du SGBD utilisé pour implanter la base (ce choix se fait au niveau physique) ; seul le type de SGBD est pris en compte. Nous avons retenu un système NoSQL de type orienté colonnes. Ce choix a été dicté par les besoins de nos applications basés sur des requêtes multicritères faisant intervenir simultanément plusieurs attributs. Or les systèmes orientés colonnes offrent des techniques de stockage qui sérialisent les valeurs des colonnes et permettent ainsi d'accélérer l'accès aux données.

Le problème consiste donc à passer d'un schéma conceptuel de base de données (DCL) vers un schéma physique NoSQL qui fera l'objet d'une implantation. Mais plusieurs systèmes NoSQL orientés colonnes coexistent ; les plus connus sont BigTable, Chang et al. (2008), HBase, Carstoiu et al. (2010), Cassandra [3] et Accumulo [4]. Ils présentent des spécificités techniques propres qui relèvent essentiellement des techniques d'implantation. Pour faire abstrac-

3. http ://cassandra.apache.org/
4. https ://accumulo.apache.org/

tion de ces spécificités, nous intègrerons le niveau logique dans le processus de correspondance entre les schémas. Autrement dit, nous considérerons les passages successifs :
Conceptuel –> Logique puis Logique –> Physique. Au niveau logique, le schéma décrit l'implantation des données en faisant abstractions de considérations techniques propres à tel ou tel système NoSQL.

Selon le modèle orienté colonnes, une base de données (BD) est constituée d'un ensemble de tables. Une table permet de regrouper des objets de taille variable sous forme de lignes ; chacune d'elles est identifiée par un identificateur unique (Id) dont le type est noté « clé-ligne » . Généralement, on regroupe dans une table les objets fortement liés ; par exemple les employés, les services auxquels ils appartiennent et les projets auxquels ils participent. Par défaut, nous stockerons la base de données dans une table unique notée T ; mais ce paramètre peut être modifié par l'administrateur des données. La table T est associée à un ensemble de familles de colonnes $\{f_1,...,f_p\}$. Le schéma d'une famille f est un triplet (N, COL, Id) où :

- f.N est le nom de la famille.
- f.COL = $\{col_1,...,col_q\}$ est un ensemble de q colonnes présentes dans chaque ligne de T décrite par f. Le schéma d'une colonne col est un triplet (N, T, TS) où « col.N » représente le nom de la colonne, « col.T » son type et « col.TS » le TimeStamp (horodatage). Dans cet article, nous ne considérons pas l'horodatage des données.
- f.Id est un identificateur unique de la famille de colonnes de type « clé-ligne ».

D'un point de vue logique, nous pouvons alors illustrer la structure du modèle orienté colonnes comme suit :

```
Table [nom_table]  {
                Id : f[nom_famille]
                        { col[nom_colonne] }
                    }
```

FIG. 3 – *Modèle logique orienté colonnes.*

3.3 Règles de correspondance Conceptuel/Logique

Nous proposons un processus permettant de décrire la correspondance entre les éléments du modèle conceptuel (diagramme de classes) et du modèle logique (orienté colonnes). Cette correspondance est réalisée par trois procédures notés CP (pour Correspondence Procedure) appliquées dans l'ordre suivant : $CP_{class}, CP_{classasso}, CP_{asso}, CP_{composition}$ *et* $CP_{heritage}$.

3.3.1 Transformation de classe

$CP_{class}(C; f)$ est une procédure de transformation de classe en famille de colonnes. Elle comporte les paramètres suivants :

– C : (N, $\{a_1,...,a_q\}$, Ident) est une classe du DCL désignée par un nom N, q attributs et un identificateur Ident.

– f : (N, $\{col_1,...,col_q\}$, Id) est une famille de q colonnes ayant Id pour identificateur de type « clé-ligne » .

Cette procédure correspond à l'algorithme ci-dessous qui élabore la famille de colonnes correspondante à une classe : le regroupement des attributs d'une classe permet de constituer une famille de colonnes distincte.

Algorithm 1 CP_{class}

Input : C , **Output :** f
Begin
  f=∅
  f.N = C.N
  f.Id = C.Ident
  For i=1 **to** q **do**
    col_i.N = a_i.N
    col_i.type = a_i. C
    f = f ∪ col_i
  EndFor
End

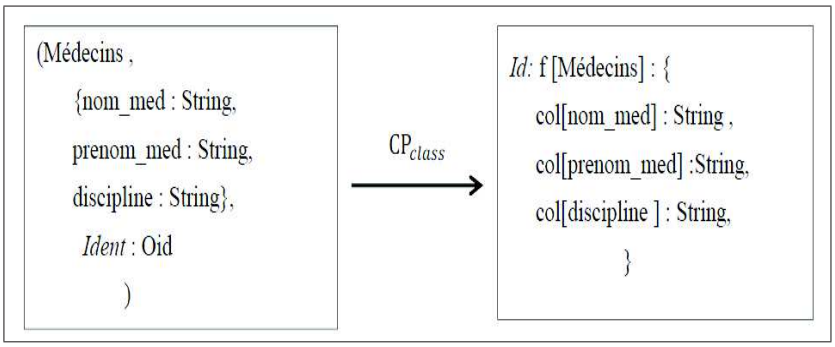

FIG. 4 – *Exemple de résultat de la procédure* CP_{class}.

3.3.2 Transformation de classe d'associations

$\text{CP}_{classasso}(C; f)$ est une procédure de transformation de classe d'associations en famille de colonnes. Elle comporte les paramètres suivants :

– C : (N, {{a_1,...,a_k},{a_1,...,a_h}}, Ident) est une classe d'associations du DCL désignée par un nom N, k attributs, h attributs références et un identificateur Ident.

– f : (N, {col_1,...,col_m}, Id) est une famille de m colonnes, avec m=k+h, ayant Id pour identificateur de type « clé-ligne » .

Cette procédure correspond à l'algorithme ci-dessous qui élabore la famille de colonnes correspondante à une classe d'associations : le regroupement des attributs références et des attributs de la classe d'associations permet de constituer une famille de colonnes distincte.

Algorithm 2 $\text{CP}_{classasso}$

Input : C , **Output :** f
Begin
  f=∅
  f.N = C.N
  f.Id = C.ident
  For i=1 **to** k **do**
    col_i.N = a_i.N
    col_i.type = a_i. C
    f = f ∪ col_i
  EndFor
  For i=1 **to** h **do**
    col_i.N = a_i.N
    col_i.type = clé-ligne
    f = f ∪ col_i
  EndFor
End

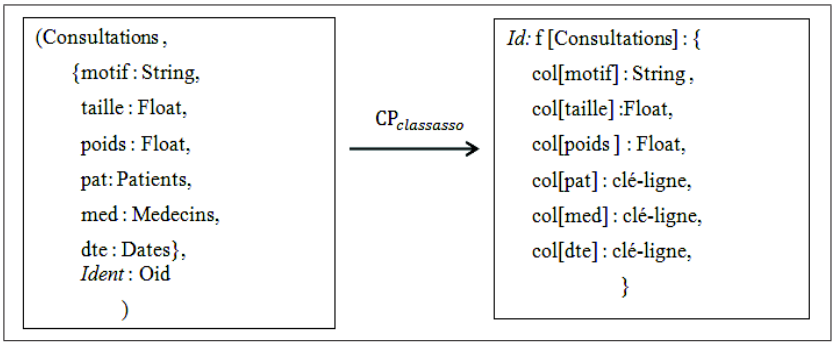

FIG. 5 – *Exemple de Résultat de la procédure* $CP_{classasso}$.

3.3.3 Transformation de lien d'association

$\text{CP}_{asso}(Asso; f)$ est une procédure de transformation d'un lien d'association. Elle comporte les paramètres suivants :

– Asso : (N, {a_1,...,a_n}, Ident) est une association n-aire de n attributs références.

– f : (N, {col_1,...,col_n}, Id) est une famille de n colonnes.

Cette procédure correspond à l'algorithme ci-dessous qui élabore la famille de colonnes correspondante à une association n-aire : les attributs correspondent à des colonnes de type « clé-ligne » qui référencent des familles cibles (classes liées).

Algorithm 3 CP_{asso}

Input : Asso , **Output :** f
Begin
f=∅
f.N = Ass.N
f.Id = C.ident
For i=1 **to** n **do**
col_i.N = a_i.N
col_i.type = clé-ligne
f = f ∪ col_i
EndFor
End

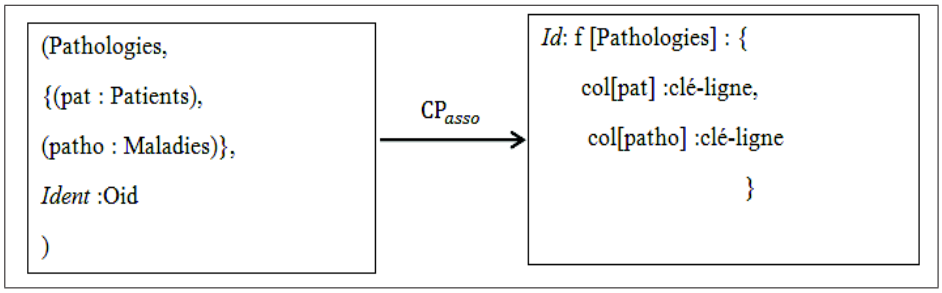

FIG. 6 – *Exemple de résultat de la procédure* CP_{asso}.

3.3.4 Transformation de lien composition/agrégation

$\text{CP}_{composition}(a, f_{composite}; col)$ est une procédure de transformation de lien de composition/agrégation. Elle comporte les paramètres suivants :

– a : est l'attribut référençant la classe composante.
– $\text{f}_{composite}$: est la famille correspondante à la classe composite.
– col : est une colonne de type « ensemble clé-ligne » .

L'algorithme ci-dessous transforme l'attribut référence en une colonne de type « ensemble clé-ligne » puis ajoute cette dernière dans la famille correspondante à la classe composite.

Algorithm 4 $\text{CP}_{composite}$

Input : a, $\text{f}_{composite}$, **Output :** col
Begin
col.N = a.N
col.type = set< clé-ligne >
$\text{f}_{composite}$ = $\text{f}_{composite}$ ∪ col
End

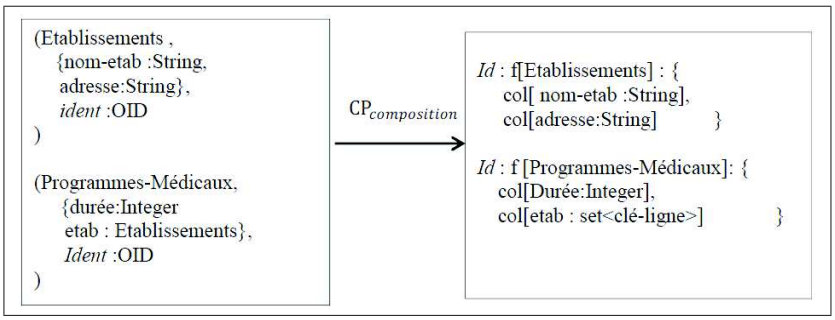

FIG. 7 – *Résultat de la procédure* $CP_{composition}$.

3.3.5 Transformation de lien d'héritage

$\text{CP}_{heritage}(a, f_{sous-classe}; col)$ est une procédure de transformation de lien d'héritage. Elle comporte les paramètres suivants :

– a : est l'attribut référençant la super-classe.

– $f_{sous-classe}$: est la famille correspondante à la sous-classe.

– col : est une colonne de type « clé-ligne » .

L'algorithme suivant transforme l'attribut de référence en une colonne de type « clé-ligne » pour créer le lien entre la super-classe et la sous-classe, puis ajoute cette dernière dans la famille correspondante à la sous-classe.

Algorithm 5 $\text{CP}_{heritage}$

Input : a, $f_{sous-classe}$, **Output :** col

Begin

col.N = a.N

col.type = clé-ligne

$f_{sous-classe} = f_{sous-classe} \cup$ col

End

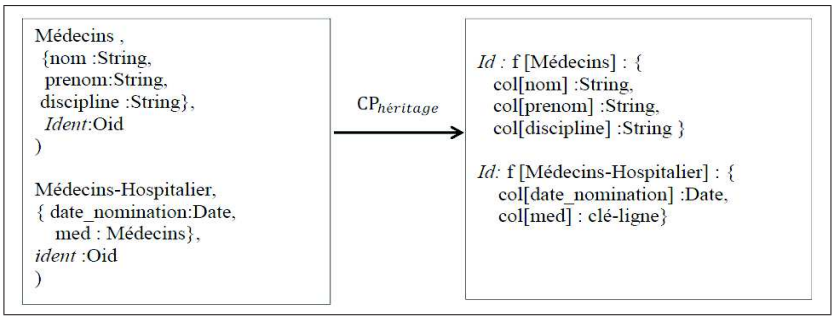

FIG. 8 – *Exemple de résultat de la procédure* $CP_{heritage}$.

4 Niveau physique

Les règles de correspondance que nous venons de définir permettent d'obtenir un schéma logique indépendant de toute plateforme d'implantation. Ce principe assure l'indépendance du niveau logique face aux évolutions techniques des systèmes NoSQL sous-jacents. Nous présentons brièvement deux plateformes d'implantation : Cassandra et HBase. Mais, dans la mesure où cet article est consacré à è la transformation d'un DCL en un modèle logique orienté colonnes, nous ne décrivons pas le passage du modèle logique vers les modèles physiques.

4.1 HBase

HBase est un système NoSQL orienté colonnes qui a été développé au-dessus du système de fichiers HDFS (Hadoop Distributed File System) de la plateforme Hadoop, Vora (2011). Une base de données HBase est par défaut composée d'une seule table notée HTable (l'administrateur peut modifier ce paramètre pour créer plusieurs tables). Lors de la création d'une HTable, on peut lui associer un nombre fixe de familles de colonnes ; seul le nom de la famille est précisé sans mention des noms de colonnes. Une famille est un regroupement logique de colonnes qui seront ajoutées au moment de l'insertion des données. Chaque ligne (ou enregistrement) au sein d'une HTable est identifiée par une clé notée RowKey et choisie par l'utilisateur. Au triplet (RowKey, famille de colonnes,colonne) correspond une cellule unique qui contiendra une valeur.

4.2 Cassandra

Cassandra est un SGBD NoSQL orienté colonnes, initialement basé sur le modèle BigTable de Google, mais qui emprunte également des caractéristiques au système Dynamo d'Amazon [5]. Une base de données Cassandra est par défaut composée d'un seul conteneur de données noté KeySpace. Ce dernier est associé à une ou plusieurs familles de colonnes, chacune d'elles est un regroupement logique de lignes. Une ligne est composée d'un ensemble de colonnes et est identifiée par une clé notée PrimaryKey. Chaque colonne est représentée par un triplet correspondant à un nom, un type et un timestamp.

Notons que les concepts « Table » et « clé-ligne » seront remplacés respectivement par les concepts HTable et RowKey sous HBase et par KeySpace et PrimaryKey sous Cassandra.

4.3 Implantation

Dans cette section, nous évoquons les techniques que nous avons utilisées pour mettre en oeuvre la démarche présentée dans la figure 2.

La version 5 de la distribution Cloudera de Hadoop (CDH 5, Cloudera Distribution Including Apache Hadoop) a été utilisée pour installer Hadoop et HBase. L'utilisation de CDH5 est faite sur VirtualBox (v 4.3.20) avec Cloudera QuickStart Virtual Machine [6]. Il s'agit d'une solution préconfigurée qui facilite l'installation de Hadoop et de plusieurs de ses sous-projets,

5. https ://aws.amazon.com/fr/documentation/dynamodb/
6. http ://www.cloudera.com/content/www/enus/documentation/enterprise/5-3 x/topics/clouderaquickstartvm.html

comme HBase, Carstoiu et al. (2010), Hive, Thusoo et al. (2009) et Impala [7]. Par souci d'efficacité, nous avons choisi cette solution qui contient toutes les composantes nécessaires pour l'implantation de notre base de données.

Sous HBase, plusieurs solutions d'implantation d'une base de données sont possibles. Nous avons opté pour celles qui correspondent le mieux au modèle logique que l'on a proposé ; notamment, nous avons choisi de définir une seule famille de colonnes par ligne de la table. La figure 9 montre un extrait des schémas HBase (a) et Cassandra (b).

FIG. 9 – *Extrait des modéles physiques HBase (a) et Cassandra (b).*

5 Positionnement de nos travaux

Nous positionnons nos travaux au regard de trois articles de recherche dont les problématiques et/ou les solutions proposées sont proches des nôtres.

L'article de Chevalier et al. (2015) s'inscrit dans le contexte de l'entreposage des données puisqu'il étudie les règles de passage d'un schéma multidimensionnel en schémas physiques NoSQL ; deux plateformes ont été retenues : le système orienté colonnes HBase et le système orienté documents MongoDB. Bien que le point de départ du processus (un schéma multidimensionnel) se situe au niveau conceptuel, ce schéma ne présente pas les mêmes caractéristiques qu'un DCL d'UML ; notamment, il comporte exclusivement des classes Faits et Dimensions et un type de lien unique entre ces deux classes.

L'article de Li (2010) traite de la transformation d'un schéma relationnel en un schéma orienté colonnes HBase. Ces travaux répondent bien aux attentes concrètes des entreprises qui, face aux évolutions récentes de l'informatique, souhaitent stocker leurs bases de données actuelles dans des systèmes NoSQL. Mais la source du processus de transformation, ici un schéma relationnel, ne présente pas la richesse sémantique que l'on peut exprimer dans un

7. Impala. http ://www.cloudera.com/documentation/enterprise/latest/topics/impala.html

DCL (notamment grâce aux différents types de liens entre classes : agrégation, composition, héritage,...).

Les travaux présentés dans Yan et al. (2014) ont pour objet de spécifier un processus de transformation MDA d'un schéma conceptuel (DCL) vers un schéma physique HBase. Ce processus ne propose pas un niveau intermédiaire (le niveau logique) qui permettrait de rendre le résultat du processus indépendant d'une plateforme système particulière. D'autre part, la transformation des liens du DCL ne tiennent pas compte des contraintes d'organisation de données qui ont été dictées par les exigences de notre contexte d'application.

6 Conclusion

Nos travaux s'inscrivent dans le cadre de l'évolution des bases de données vers les Big Data, ceci pour prendre en compte le volume, la variété et la vélocité des données présents dans les nouvelles applications liées à la transformation digitale des entreprises. Nos études portent actuellement sur les mécanismes de stockage des données dans des systèmes NoSQL.

Dans cet article, nous avons traité le processus de transformation d'un schéma conceptuel représenté par un DCL d'UML en un schéma physique NoSQL orienté colonnes. Pour automatiser ce processus, nous avons spécifié des algorithmes pour traduire un DCL en un schéma logique NoSQL. Selon notre approche, le schéma logique constitue un niveau intermédiaire qui fait abstraction des considérations techniques propres aux plateformes d'implantation et qui apparaîtront uniquement dans le schéma physique ; ce principe permet de rendre le niveau logique indépendant des évolutions technologiques des plateformes.

Nous avons expérimenté notre démarche et nos modèles sur une application du domaine médical qui porte sur des programmes pluriannuels de suivi de pathologies. Nous avons automatisé le processus de transformation d'un DCL décrivant une base de données en un schéma NoSQL orienté colonnes. Ce schéma a été implanté sur les systèmes HBase et Cassandra.

Références

Abhinay, A., B. Akshata, et C. Karuna (2013). Growth of new databases analysis of nosql datastores. *International Journal of Advanced Research in Computer Science and Software Engineering*.

Carstoiu, D., E. Lepadatu, et M. Gaspar (2010). Hbase - non sql database, performances evaluation. *International Journal of Advancements in Computing Technology*.

Chang, F., J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, Chandra, A. Fikes, et R. Gruber (2008). Bigtable : a distributed storage system for structured data. *ACM Trans. Comput. Syst*.

Chen, C. et C. Zhang (2014). Data-intensive applications, challenges, techniques and technologies : A survey on big data. *Inf. Sci*.

Chevalier, M., M. E. Malki, A. Kopliku, O. Teste, et R. Tournier (2015). Entrepôts de données multidimensionnelles nosql. *EDA*.

Darmont, J., O. Boussaid, J. Ralaivao, et K. Aouiche (2005). An architecture framework for complex data warehouses. *7th International Conference on Enterprise Information Systems (ICEIS)*.

Doug, L. (2001). 3d data management : Controlling data volume, velocity, and variety.

Fankam, C., S. Jean, G. Pierra, et L. Bellatreche (2008). Enrichissement de l'architecture ansi/sparc pour expliciter la sémantique des données : une approche fondée sur les ontologies. *Actes de la 2ème Conférence francophone sur les Architectures Logicielles (CAL'08)*.

Gajendran, J. (2012). A survey on nosql databases.

Grover, M., T. Malaska, J. Seidman, et G. Shapira (2015). Hadoop application architectures. *O'Reilly*.

Li, C. (2010). Transforming relational database into hbase. *In International Conference on Software Engineering and Service Sciences (ICSESS)*.

Midouni, S., J. Darmont, et F. Bentayeb (2009). Approche de modélisation multidimensionnelle des données complexes : Application aux données médicales. *EDA*.

Pedersen, T. B. et C. S. Jensen (1998). Multidimensional data modeling for complex data. *In Proc of 10th Int Conf on Data Engineering (ICDE). IEEE Computer Society*.

Tanasescu, A., O. Boussaid, et F. Bentayeb (2005). Preparing complex data for warehousing. *3rd ACS/IEEE International Conference on Computer Systems and Applications(AICCSA)*.

Thusoo, A., J. S. Sarma, N. Jain, Z. Shao, P. Chakka, H. L. S. Anthony, P. Wyckoff, et R. Murthy (2009). Hive - a warehousing solution over a map-reduce framework. *In Proc. of Very Large Data Bases*.

Vora, M. (2011). Hadoop - hbase for large-scale data. *Computer Science and Network Technology (ICCSNT)*.

Yan, L., P.Gu, et C.Zhang (2014). Transforming uml class diagrams into hbase based on metamodel. *Information Science, Electronics and Electrical Engineering (ISEEE)*.

Summary

In the last decade, Big Data has become a major research area for both academic and industrial side. In this paper, we consider the automatic transformation of Big Data conceptual schema within NoSql System. For this, we have specified algorithms to translate a conceptual schema into a NoSQL model. Starting from UML class diagram that describes a set of complex objects, we propose transformation algorithms to generate, ultimately, a columns-oriented NoSQL model. To ensure efficient automatic transformation, we use a logical model that limits the impacts related to technical developments of NoSQL platforms. We provide experiments of the transformation algorithms in the context of health area..