Métriques structurelles pour l'analyse de bases orientées documents

Paola Gómez [1], **Claudia Roncancio** [1], **Rubby Casallas** [2]

1. *Univ. Grenoble Alpes, CNRS, Grenoble INP* [*], *LIG, 38000 Grenoble, France*
   {paola.gomez-barreto,claudia.roncancio}@univ-grenoble-alpes.fr
2. *TICSw, Universidad de los Andes,Bogotá - Colombia*
   rcasalla@uniandes.edu.co

RÉSUMÉ. *La flexibilité dans la structuration des données dans les bases orientées documents est appréciée pour permettre un développement initial rapide. Cependant, les possibilités de structuration des données sont nombreuses et le choix de structuration adopté reste assez crucial par son impact potentiel sur plusieurs aspects de la qualité des applications. En effet, chaque structuration peut présenter des avantages et des inconvénients notamment en matière d'empreinte mémoire, redondance de données engendrée, coût de navigation dans les structures et accès à certaines données, lisibilité des programmes. Dans cet article nous proposons un ensemble de métriques structurelles pour des "schémas" de documents JSON. Ces métriques permettent de refléter la complexité des schémas et des critères de qualité tels que leur lisibilité et maintenabilité. La définition de ces métriques s'appuie, entre autres, sur des expérimentations avec MongoDB, des travaux liés à XML et les métriques utilisées en Génie logiciel pour la qualité du code. La définition des métriques est complétée par un scénario de validation.*

ABSTRACT. *Document oriented bases allow high flexibility in data representation. This facilitates a rapid development of applications and allows many possibilities for data structuration. Nevertheless, the structuration choices remain crucial because they impact several aspects of the document base and application quality, e.g, memory print, data redundancy, querying and navigation facility and performances, readability and maintainability. It is therefore important to be able to analyse and to compare several data structuration alternatives. In this paper, we propose a set of structural metrics of JSON documents. These metrics work on the structure (not the data) considered as a schema. They mesure several aspects of the complexity of the structure in order to be used in criteria helping in the schema design process. This work capitalises on experiences with MongoDB so as proposals for XML and software quality. This paper presents the definition ot the metrics together with a validation scenario.*

MOTS-CLÉS : *NoSQL, métriques structurelles, systèmes orientés document, MongoDB*
KEYWORDS: *NoSQL, structural metrics, document-oriented systems, MongoDB*

*. Institute of Engineering Univ. Grenoble Alpes

1. Introduction

De nos jours, les applications et systèmes d'information doivent gérer des larges quantités de données hétérogènes tout en répondant à des exigences fonctionnelles variées et à des besoins de performance et de passage à l'échelle. Les systèmes de gestion de données NoSQL apportent diverses solutions et offrent, pour la plupart, beaucoup de souplesse dans la structuration des données. Ils permettent une structuration des données avec une grande flexibilité et sans création préalable d'un schéma (contrairement aux SGBD relationnels). Dans ces solutions il n'y a pas de séparation claire des couches logiques et physiques.

Nos travaux portent sur les systèmes orientés documents, et plus précisément ceux utilisant JSON, MongoDB notamment. Dans ces systèmes, les données peuvent être structurées dans des collections de documents avec attributs atomiques ou de types complexes. La flexibilité dans la structuration des données est appréciée pour permettre un développement initial rapide. Cependant, on constate que les possibilités de structuration des données sont nombreuses et que le choix de structuration adopté reste assez crucial par son impact potentiel sur plusieurs aspects de la qualité des applications (Gómez *et al*., 2016). Le problème est comment définir ou analyser si une structure est bien adaptée aux besoins des applications. En effet, chaque structuration peut présenter des avantages et des inconvénients différents en matière d'empreinte mémoire, redondance de données engendrée, coût de navigation dans les structures et d'obtention de certaines données, lisibilité des programmes, parmi d'autres.

Il devient alors intéressant de pouvoir considérer plusieurs structurations candidates pour retenir un choix unique, ou temporel, ou plusieurs alternatives parallèles selon les cas. Effectuer une analyse et comparaison de plusieurs choix de structuration n'est pas évident, d'une part, par le nombre potentiellement grand de structurations possibles et, d'autre part, par l'absence de critères objectifs d'analyse [1]. Notre travail est une contribution dans ce sens. Nous cherchons à faciliter la compréhension, l'évaluation et la comparaison des structures de données orientés documents (JSON/BSON) où les possibilités sont nombreuses. Nous proposons d'abstraire et de travailler avec une notion de "schéma" de données même si MongoDB ne le traite pas en tant que tel. L'objectif est de clarifier les possibilités et caractéristiques de chaque "schéma" et de donner des critères objectifs pour l'évaluer et apprécier ses avantages et ses inconvénients.

La principale contribution de cet article est la proposition d'un ensemble de métriques structurelles pour des "schémas" de documents JSON. Ces métriques permettent de refléter la complexité des schémas et peuvent être utilisées pour établir des critères de qualité tels que leur lisibilité et maintenabilité. La définition de ces métriques s'appuie, entre autres, sur des expérimentations avec MongoDB, des travaux liés à XML et des métriques utilisées en Génie logiciel pour la qualité du code.

1. Pour le modèle de données orienté documents, on ne dispose pas à l'heure actuelle de critères analogues aux anomalies de conception et la normalisation du modèle relationnel.

Ce travail est partie du projet SCORUS, plus vaste, qui vise à assister les utilisateurs dans un processus de modélisation de schéma avec une approche de recommandations. SCORUS permet de (1) générer un ensemble de schémas semi-structurés orientés documents pour un modèle de données UML; (2) analyser ces schémas à l'aide des métriques proposées dans cet article; (3) proposer un top k de schémas semi-structurés selon les préférences identifiées. Cet article se concentre sur l'étape (2), l'analyse des schémas par la proposition d'un ensemble des métriques permettant de les comparer.

Dans la suite, la section 2 rappelle certains éléments de MongoDB et la motivation de nos travaux. Dans la section 3, nous présentons les métriques structurelles proposées pour mesurer les schémas. La section 4 fournit un scénario d'expérimentation qu'utilise les métriques pour comparer des schémas. Les travaux connexes sont décrits en section 5. Nos conclusions et perspectives de recherche sont présentées en section 6.

2. Contexte et Motivation

Comme déjà mentionné nous nous intéressons à des questions de qualité de structuration de données BSON/JSON au sein de systèmes type MongoDB. Rappelons que dans ce système (comme dans la plupart des NoSQL) il n'y a pas de gestion explicite d'un schéma de données. La manière de structurer les données reste néanmoins importante car elle a un impact sur plusieurs aspects de la base de documents et des applications qui les utilisent.

Le format BSON, gère les données comme un ensemble de collections de documents (voir figure 1a, collections *Agencies* et *BusinessLines*). Un document est simplement un ensemble de paires `attribut:valeur`. Le type des valeurs peut être atomique ou complexe. Par complexe nous entendons soit un tableau de valeurs de tout type ou un document qui dans ce cas est dit *imbriqué*. Notons que la valeur d'un attribut peut être l'identifiant d'un document ou la valeur d'un attribut d'un document d'une autre collection. Cela permet de *référencer* un ou plusieurs documents.

Ce système de types est à la fois simple et puissant car il permet beaucoup de flexibilité dans la création de structures complexes. Dans un cas simple comme celui de notre exemple, l'association 1-N entre Agence et "BusinessLines", est susceptible d'être représentée de plusieurs manières. La figure 1a montre un choix pour la collection *Agencies* qui utilise des références vers *BusinessLines* alors que le choix illustré sur 1b imbrique les documents correspondant aux "business". Sur 1b la collection *BusinessLines* n'est pas créée et il n'y a pas de duplication de données.

De manière générale, tout en garantissant la complètude des données, les collections peuvent être structurées et reliées de divers manières, e.g. collections séparées et sans imbrication, collections complètement imbriquées, combinaison d'imbrication et de referencement ou duplication de données. Le choix de la meilleure structuration n'aura probablement pas une réponse unique ni absolue et dépendra des priorités et besoins d'accès du moment.

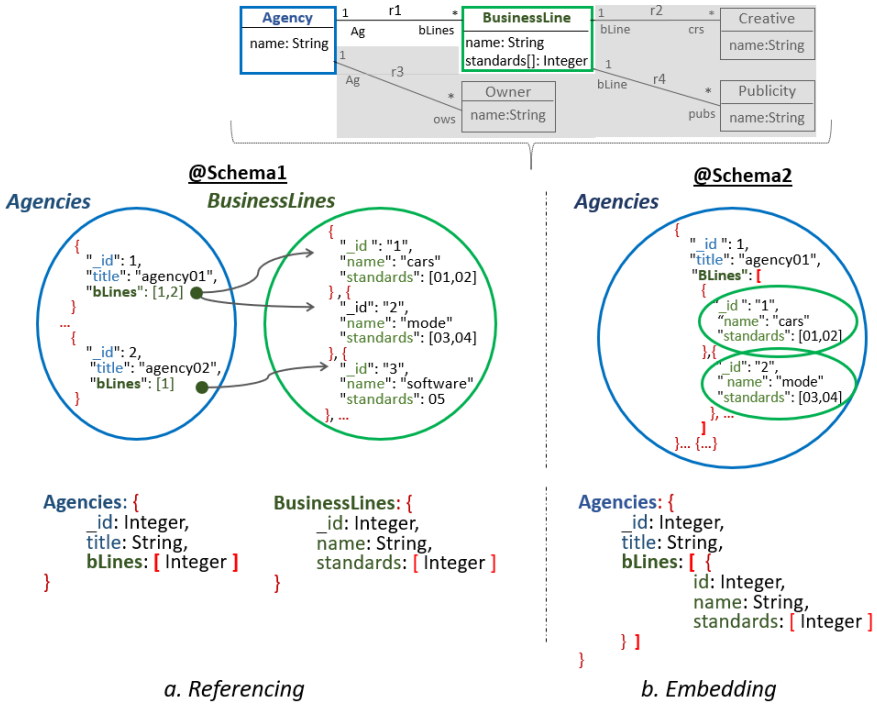

Figure 1. Exemples d'instances MongoDB et abstractions de leurs schémas basés sur un modèle UML

La manière dont sont structurées les données a un fort impact sur la taille de la base, les performances des requêtes et la lisibilité du code des requêtes, ce qu'influence la maintenabilité et l'usabilité de la base ainsi que de ses applications. Cela a été constaté de manière expérimentale (Gómez *et al.*, 2016) où plusieurs patrons de comportement ressortent. Notamment, les collections avec des documents imbriqués sont favorables aux requêtes qui suivent l'ordre de l'imbrication. Cependant, l'accès aux données dans un autre ordre est désavantagé et les performances des requêtes nécessitant l'accès à des données intégrées à différents niveaux dans la même collection sera pénalisé. La raison est que la complexité des manipulations requises dans ce cas, est proche de celle des jointures de plusieurs collections. En outre, les collections avec des documents imbriqués ont une empreinte mémoire plus importante que la représentation équivalente avec des références. Dans le choix de la structuration des données, les priorités peuvent s'avérer divergentes, comme le souhait de dupliquer des documents dans plusieurs collections parce qu'ils sont consultés dans des contextes différents mais aussi le souhait de réduire le coût du stockage.

Nous avons analysé un certain nombre de caractéristiques critiques sur les structures afin d'établir des critères qui aident dans le choix d'un schéma. Dans la suite nous proposons des métriques mesurables sur les schémas de manière à avoir des éléments objectifs d'appréciation et de comparaison au regard des priorités de l'utilisateur.

3. Métriques Structurelles

Dans cette section nous proposons un ensemble de métriques structurelles qui reflètent des aspects clés de la complexité des "schémas" semi-structurés. L'objectif est de faciliter leur analyse et comparaison sans création de la base de données. Des informations statistiques sur les données peuvent bien sur compléter l'analyse lorsqu'elles sont disponibles. La table 1 résume les métriques proposées. Les sections 3.2 à 3.6, définissent et illustrent ces métriques. Dans la suite φ dénote une collection, t un type de document et x un schéma.

Tableau 1. Métriques structurelles proposées

Catégorie	Nom des métriques	Description	Par		
			sch	Col	type
Existence	*colExistence*	Existence d'une collection		x	
	docExistence	Existence d'un type de document dans une collection		x	x
Imbrication	*colDepth*	Profondeur maximale d'une collection		x	
	globalDepth	Profondeur maximale d'un schéma	x		
	DocDepthInCol	Niveau où un type de document se trouve dans une collection		x	x
	maxDocDepth	Niveau le plus profond où apparaît un type de document	x		
	minDocDepth	Niveau le mois profond où apparaît un type de document	x		
Largeur	*docWidth*	"Largeur" d'un type de document		x	x
Référencement	*refLoad*	Nombre de références à une collection		x	
Redondance	*docCopiesInCol*	Copies d'un type de document t dans une collection		x	x
	docTypeCopies	Nombre d'utilisations d'un type de document	x		

Afin de faciliter la manipulation des variantes de schéma (dans le cadre de la génération automatique) et d'évaluer les métriques, nous utilisons la structure de graphe présentée en sous-section 3.1.

3.1. Structure de graphes

Nous considérons un modèle de données UML, avec un ensemble de classes $E = \{e_1, ..., e_n\}$. Les propriétés ou attributs d'une classe e_i sont désignés par le type te_i et ses associations par l'ensemble $R(e_i) = \{r_1, ..., r_n\}$. Pour chaque association on connaît le rôle des entités reliées. Dans la suite nous appelons schéma semi-structuré, l'ensemble de collections et le type des documents qui seront utilisés dans la base pour représenter les données. Différents schémas sont envisageables pour un même modèle entité-association.

Afin de faciliter le calcul des métriques d'un schéma, nous construisons un arbre tel qu'illustré sur la figure 2 pour chaque schéma. Le noeud racine, @ShemaX, a un noeud fils par *collection* présente dans le schéma (e.g . collections *Agencies, Creatives, Owners*). Il s'agit de collections de documents dont le type est représenté par le sous-arbre fils, noeud avec sous-fixe l_0 (e.g. $tAgency@l_0$ pour documents agency au niveau

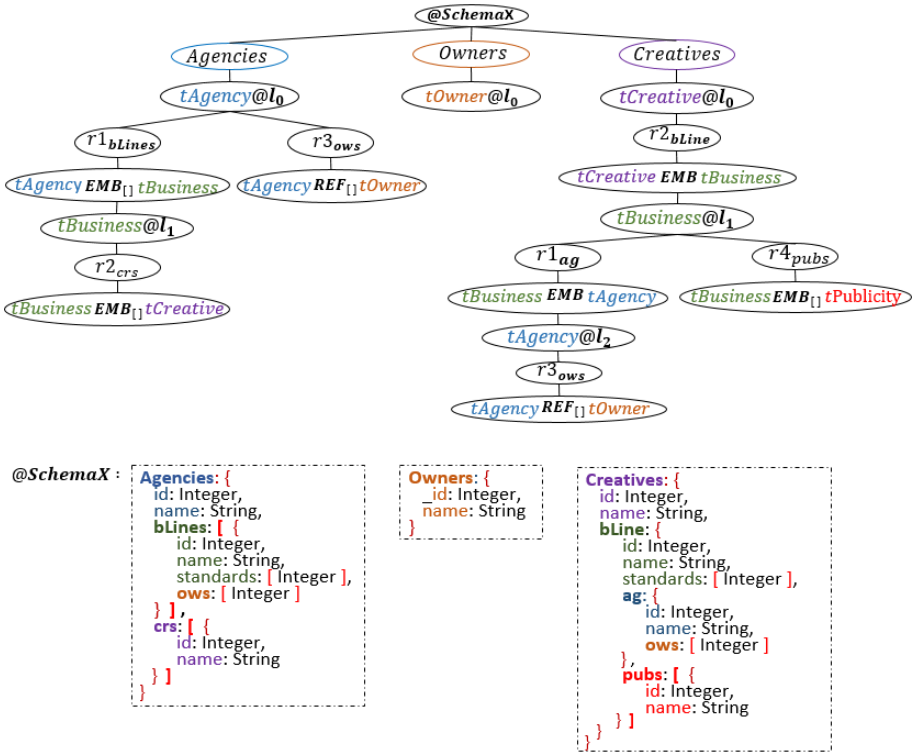

Figure 2. Exemple de représentation des graphes

0). Dans ce sous-arbre, les attributs de type atomique sont dans ce noeud $@l_0$ [2], et les attributs complexes (imbrication ou référence de documents) sont exprimés dans les noeuds fils. Les références sont matérialisées par des *noeuds REF* et les imbrications de documents par des *noeuds EMB* (e.g. $tAgencyEMB[]tBusiness$).

Lors de la création de schémas semi-structurés, les attributs complexes sont utilisés pour stocker les associations $R(e_i)$ des classes e_i. Un noeud avec le nom de l'association est créé (e.g. $r1_{bline}$) avec pour fils un noeud *REF* ou *EMP* selon le choix. Des arrays, notés $[]$, peuvent être utilisés pour les association n-aires. Par exemple, pour les business d'une agence, association $r1_{bline}$, le document agency aura un attribut de type array de documents business (noeud $tAgencyEMB[]tBusiness$).

Les imbrications de documents induisent un niveau de profondeur supplémentaire dans la structure. Ceci est matérialisé dans l'arbre par un *noeud niveau* l_i qui permettra de savoir facilement à quel niveau de profondeur se trouvera un document (e.g. $tBusiness@l_1$). Concernant les références, elles apparaissent à différents niveaux et référencent le type apparaissant au niveau 0 d'une collection.

2. Attributs non listés sur la figure

3.2. Métriques d'existence

Le choix de créer une collection pour un type de document sera motivé principalement par le besoin d'accès rapide ou fréquent à l'extension du type ou à un document du type en question. Au contraire l'imbrication d'un type de document dans un autre peut être motivé par le fait que l'information est fréquemment consultée ensemble. S'assurer qu'un type de document n'est pas imbriqué à certains endroits peut aussi être intéressant, notamment si le document est peu accédé dans ce contexte ou si l'on cherche à réduire la complexité d'une collection. Dans cette catégorie nous définissons des métriques qui reflètent l'existence d'un type de document t dans un schéma. Nous considérons deux cas : (1) la existence d'une collection de documents de type t, et (2) la présence de documents de type t imbriqués dans d'autres documents. Ces cas sont couverts respectivement, par la métrique *colExistence* et la métrique *docExistence* définies ci-après.

Existence de collection :

$$colExistence(t) = \begin{cases} 1 & \text{le noeud } t@l_0 \text{ apparaît dans le schéma} \\ 0 \end{cases} \quad (1)$$

Existence de type imbriqué : l'imbrication de documents de type t est matérialisé dans le graphe par un noeud $*EMB\,t$.

$$docExistence(\varphi, t) = \begin{cases} 1 & t \in \varphi \quad \text{un noeud } *EMB\,t \text{ apparaît dans un chemin partant de } \varphi \\ 0 & t \notin \varphi \end{cases} \quad (2)$$

Notons sur la figure 2 que pour les types *tAgency, towners et tCreative* il y a des collections (noeuds $@l_0$) alors que ce n'est pas le cas pour *tPublicity*. Ceux-ci existent exclusivement imbriqués dans des documents *tCreative*. Notons aussi que des documents de type *tBusiness* sont imbriqués dans deux collections, *Agencies et Creatives*. Nous verrons dans la suite que la prise en compte de ce fait peut s'avérer pertinent dans l'analyse des schémas.

3.3. Métriques d'imbrication

En général, plus une information sera imbriquée profondément, plus il sera coûteux d'y accéder sauf si l'information intermédiaire est aussi recherchée par la requête. Savoir à quel niveau d'imbrication apparaît un type de document permet d'évaluer les coûts de navigation et d'aller-retour entre les niveaux ("intra-joint") pour y accéder, ou des opérations de restructuration nécessaires pour extraire le format le plus approprié. Cette catégorie est consacrée aux métriques qui indiquent le niveau d'imbrication des documents.

Profondeur d'une collection : la métrique *colDepth* (3) indique le niveau de profondeur où se trouve le document le plus imbriqué. L'imbrication des documents est représentée par les noeuds EMB dans le graphe.

$$colDepth(\varphi) = max(depth(p_i)) \quad p_i \text{ est un chemin partant du noeud } \varphi \tag{3}$$

$$depth(p) = n \qquad \text{nombre de noeuds } EMB \text{ dans le chemin } p \tag{4}$$

Profondeur d'un schéma : la métrique *globalDepth* (5) indique le niveau d'imbrication le plus profond des collections d'un schéma.

$$globalDepth(x) = max(colDepth(\varphi_i)) \qquad \forall \text{ collection } \varphi_i \in x \tag{5}$$

Connaître la profondeur d'imbrications des collections aide à mieux cerner leur cas d'utilisation et à estimer la pertinence de la structure. Les imbrications successives contribuent à une certaine forme de complexité mais n'implique pas forcement des requêtes moins performantes. Une collection très imbriquée peut être avantageuse si des requêtes prioritaires nécessitent une majorité des informations imbriquées. Si c'est pas le cas, l'impact des opérations de projections sera à prendre en compte (voir métriques suivantes) ainsi que la restructuration des données pour la réponse si le chemin d'accès de la requête et le sens d'imbrication des données ne coïncident pas.

Sur l'exemple, la profondeur des collections *Owners, Agencies et Creatives* est 0, 2, et 2 respectivement. La profondeur maximale du schéma est de 2. Notons que dans la collection *Creatives*, le type *tAgency* n'ajoute pas de niveau d'imbrication, il ajoute uniquement un tableau avec des références sur *Owners*.

Profondeur d'un type de document : la métrique *docDepthInCol* (6) indique le niveau où apparaît un type de document t dans une collection φ. Si les éléments de la collection sont de type t (noeud $t@l_0$) la profondeur est zéro, sinon on cherche le niveau le plus profond où est imbriqué un document de ce type (noeuds $EMB\,t$) en suivant les chemins racine-feuilles.

$$docDepthInCol(\varphi, t) = \begin{cases} 0 & \text{le noeud fils de } \varphi \text{ est } t@l_0 \\ max(docDepth(p_i, t) & p_i \text{ est un chemin de la racine } \varphi \text{ à une feuille} \end{cases} \tag{6}$$

$$docDepth(p, t) = n \quad \text{nombre de noeuds } EMB \text{ entre la racine et } *\,EMB\,t \tag{7}$$

Par exemple, dans la collection *Creatives*, le niveau d'imbrication de *tPublicity* est 2, celui de *tCreative* est 0. *tCreative* est aussi imbriqué au niveau 2 de la collection *Agencies*. Nous introduisons également les métriques *maxDocDepth* (8) et *minDocDepth* (9) qui indiquent le niveau le plus et le moins profond où le type de document apparaît dans le schéma.

$$maxDocDepth(t) = max(DocDepthInCol(\varphi_i, t)) \quad \varphi_i \in x \ \wedge \ t \in \varphi_i \tag{8}$$

$$minDocDepth(t) = min(DocDepthInCol(\varphi_i), t) \quad \varphi_i \in x \ \wedge \ t \in \varphi_i \tag{9}$$

Connaître les niveaux minimum et maximum permet d'estimer combien de niveaux intermédiaires il faut traiter pour l'accès le plus ou le moins direct. Sur l'exemple, notons qu'il n'y a pas de collection de documents *tBusiness*, *minDocDepth(tBusiness)* = 1.

3.4. Largeur des documents

Ici nous nous intéressons à la complexité d'un type de document en termes du nombre d'attributs et de leur type, atomique ou complexe (documents ou arrays de documents imbriqués). Ces métriques sont motivées par le fait que des documents avec plusieurs attributs complexes peuvent induire des opérations d'accès et de projection plus conséquentes. En effet, pour extraire les attributs nécessaires à l'évaluation d'une requête, il est nécessaire d'enlever les autres attributs ce qui s'avère plus coûteux pour des document "larges". Lors de l'évaluation d'un schéma, on pourra notamment analyser le choix d'une structure à la fois "large" et très imbriquée.

La métrique *docWidth* (10), reflète la "largeur" d'un type de document en se basant sur le nombre d'attributs atomiques (coefficient a=1), le nombre d'attributs qu'imbriquent un document (coefficient b=2), le nombre d'attributs de type array de valeurs atomiques (coefficient c=1) et array de documents (qui ont plus de poids, coefficient d=3).

$$\begin{aligned} docWidth(t,\varphi) = {} & a * nbrAtomicAttributes(t,\varphi) + \\ & b * nbrDocAttributes(t,\varphi) + \\ & c * nbrArrayAtomicAttributes(t,\varphi) + \\ & d * nbrArrayDocAttributes(t,\varphi) \end{aligned} \tag{10}$$

Les métriques indiquant le nombre d'attributs peuvent être utilisées séparément selon les analyses souhaités. La taille des arrays n'est pas prise en compte ici car elle n'est pas forcément disponible. Si c'est le cas, il semble intéressant de différencier les ordres de grandeur des arrays. Des arrays de taille 4 ou 5 sont du même ordre de grandeur, contrairement à des arrays prévus pour de milliers d'éléments.

Les collections textitAgencies et *Creatives* de l'exemple, utilisent des document de type *tBusiness* mais ils n'ont néanmoins pas les mêmes attributs. Dans Creatives le type inclut des arrays d'agences et publicity, *docWidth(Creatives,tBusiness)* = 8 , contrairement à *Agencies* où *docWidth(Agencies,tBusiness)* = 4.

3.5. Taux de référencement

Le maintien de l'intégrité référentielle devient un problème pour les collections dont les documents sont beaucoup référencés par de documents d'autres collections. Pour une collection avec des documents d'un certain type t, la métrique *refLoad* (11) indique le nombre d'attributs (d'autres types) qui sont des références potentielles sur les documents de type t.

$$refLoad(\varphi) = n \qquad \text{soit } t@l_0 \text{ le noeud fils de } \varphi\text{, n est le nombre de noeuds } *REF\ t \text{ du schéma} \quad (11)$$

Pour la collection *Owners* de notre exemple, son type est référencé par 2 collections: *Agencies* la référence au niveau 0 alors que *Creatives* la référence dans un document imbriqué au niveau 2.

3.6. Métriques de redondance

Nous nous intéressons ici à la redondance de données qui peut exister dans la base. La redondance des documents peut accélérer les accès et limiter certaines opérations coûteuses (par exemple des jointures). Cependant, elle impacte l'empreinte mémoire de la base et sa maintenabilité en matière de cohérence (complexité d'écriture des programmes et coût). Pour la métrique de redondance *docCopiesInCol* (12), nous utilisons l'information de cardinalité des associations conjointement avec les choix faits sur le schéma. La redondance apparaît pour certains cas de représentation de l'association par imbrication de documents.

$$docCopiesInCol(t,\varphi) = \begin{cases} 0 & : t \notin \varphi\ docExistence(\varphi,t)=0 \\ 1 & : \text{ le noeud fils de } \varphi \text{ est } t@l_0 \\ \prod card(r_{rol},t) & : \begin{array}{l} r_{rol} \text{ parent de un noeud } EMB \\ \text{dans le chemin entre } \varphi \text{ et } *EMBt \end{array} \end{cases} \quad (12)$$

$$card(r,\varepsilon) = n \qquad \text{cardinalité de } r \text{ du côté } \varepsilon \text{ dans le modèle UML} \quad (13)$$

Dans la collection *Creatives* du schéma de la Figure 2, l'attribut pour business, nommé *bline*, introduit de la redondance pour les agences. Par l'association r1 une agence A peut être associée à n1 business. Il y aura donc autant de copies du document A. Si de plus un business est référence par n2 creatives (association r2) , il y aura n1 x n2 copies du document A.

Par ailleurs, nous proposons la métrique *docTypeCopies(t)* qui indique le nombre de fois qu'un type de document est utilisé dans le schéma. Ceci reflète le nombre de structures qui peuvent potentiellement stocker des documents de type t. Cette métrique utilise la métrique d'existence.

4. Scénario de validation

Comme déjà dit, ce travail s'inscrit dans le cadre du projet SCORUS, qui vise à assister les utilisateurs dans les processus de choix de structuration des documents. Pour un modèle de données UML, SCORUS permettra (1) de générer automatiquement un ensemble de variantes de "schémas" JSON possibles et (2) de donner les métriques pour chacun d'eux. Tenant compte des priorités des applications et des requêtes fréquentes, ces métriques seront utilisées pour établir des critères de choix et comparer les schémas

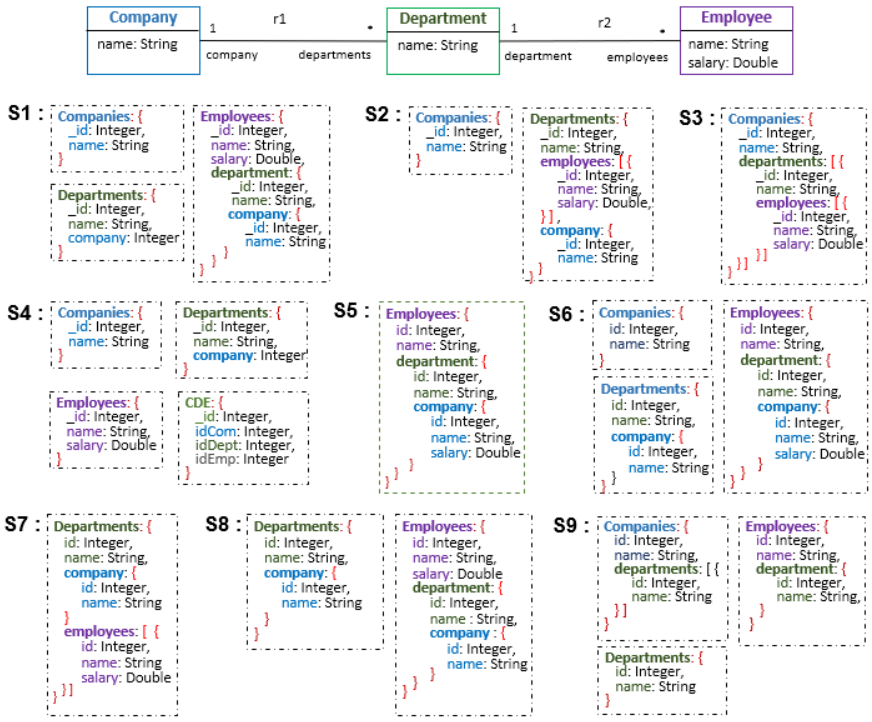

Figure 3. Ensemble de schémas étudiés

entre eux. Il s'agit en priorité de faire émerger le ou les schémas les plus favorables selon certains critères mais aussi d'écarter des choix très défavorables ou encore, d'envisager des schémas alternatifs qui n'étaient pas forcement considérés au départ. Ceci est arrivé lors de notre expérimentation où une des alternatives générées automatiquement s'est avéré être pertinente alors qu'elle ne faisait pas partie des choix "naturels" du développeur.

Dans la suite nous considérons un scénario d'utilisation des métriques proposées. Nous utilisons un cas simple, voir figure 3, pour lequel 9 variantes de structuration JSON sont étudiées. Ce cas a été utilisé lors d'une expérimentation avec des bases MongoDB (Gómez *et al.*, 2016) où l'impact de la structuration des schémas ressort. Nous nous plaçons dans le même contexte applicatif et reprenons des informations sur l'accès aux données afin de les utiliser dans l'analyse des variantes des "schémas" étudiés. Nous avons évalué les métriques pour les 9 schémas. Celles que nous utilisons ici sont indiquées sur la figure 2.

D'un point de vue applicatif, nous disposons des informations suivantes. Les requêtes les plus prioritaires portent sur les entreprises et le nom de leurs départements (priorité forte) mais aussi pour connaître l'employé qui a le salaire le plus élevé dans l'entreprise en donnant l'identifiant de l'entreprise ou le nom de l'entreprise.

Tableau 2. Évaluation des schémas

Métriques \ Schéma	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
colExistence(tCompany)	1	1	1	1	0	1	0	0	1
docCopies(tCompany)	1	1	1	1	1	3	1	1	1
refLoad(Employees)	0			1	0	0		0	0
colExistence(tCompany)	1	0	0	1	1	1	0	0	1
docWidth(Companies,l1)	1	1	3	1		1			3
docExistence(tDepartment,Companies)	0	0	1	0		0			1

Ces informations sur les accès prioritaires ainsi que d'autres informations permettent d'établir des critères d'analyse des schémas. Tenant compte les accès prioritaires, la collection de Companies joue un rôle important (critère1) ainsi que la facilité pour manipuler les instances (critère 5). Les départements sont accèdés via les entreprises (critère 6). De plus on sait que la cohérence des données sur les entreprises est importante. Il est donc préférable de limiter les copies pour ces données (critère 2). Par ailleurs, l'accès à l'ensemble d'employés (critère 4) exclusivement n'est pas prioritaire.

La table 3 illustre la formalisation de certains critères. Chaque ligne montre l'évaluation d'un critère sur les 9 schémas alternatifs étudiés. Les valeurs ont été normalisées (entre 0 et 1) et introduisent un ordre relatif entre les schémas. Par exemple au regard du critère 4, les schémas S1, S4, S5, S6 et S9 sont à privilégier par rapport aux autres.

Tableau 3. Critères

	Critère \ Schéma	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
1	$f_{c_1}(s) = colExistenceCompanies(s)$	1.00	1.00	1.00	1.00	0.00	1.00	0.00	0.00	1.00
2	$f_{c_2}(s) = docCopiestCompany^{min}(s)$	1.00	1.00	1.00	1.00	1.00	0.33	1.00	1.00	1.00
3	$f_{c_3}(s) = refLoadEmployees^{max}(s)$	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
4	$f_{c_4}(s) = colExistenceEmployees(s)$	1.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	1.00
5	$f_{c_5}(s) = levelWidthCompaniesL_1{}^{min}(s)$	1.00	1.00	0.33	1.00	0.00	1.00	0.00	0.00	0.33
6	$f_{c_6}(s) = docDptInCompanies^{min}(s)$	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	1.00

L'analyse des schémas est multi-critères (6 critères dans notre cas). Les critères peuvent avoir le même poids ou bien on peut privilégier certains critères. La fonction d'évaluation d'un schéma, noté *schemaEvaluation* fait la somme pondérée des critères.

$$schemaEvaluation(s) = \sum_{i=1}^{|Criteria|} weight_{criterion_i} * f_{criterion_i}(s) \qquad (14)$$

Nous avons évalué avec trois pondérations différents: même poids pour tous les critères (cas1), priorité aux critères concernant la facilité d'utilisation de companies (cas2), ajout de priorité de la collection employée en supposant qu'il est motivé par un nouveau patron d'accès (cas 3). La figure 4 montre le résultat de l'évaluation des 9 schémas pour les trois cas.

Criteria	Factor		
	Case1	Case2	Case3
Criterion 1	16.7	50	30
Criterion 2	16.7	10	10
Criterion 3	16.7	0	0
Criterion 4	16.7	0	20
Criterion 5	16.7	15	15
Criterion 6	16.7	25	25

a. Ensembles des poids

Schema / Cas	s1	s2	s3	s4	s5	s6	s7	s8	s9
Cas 1	83.33	66.67	72.22	66.67	50.00	72.22	33.33	33.33	88.89
Cas 2	75.00	75.00	90.00	75.00	10.00	68.33	10.00	10.00	90.00
Cas 3	75	55	70	75	30	68.33	10	10	90

b. Evaluation des schémes par cas

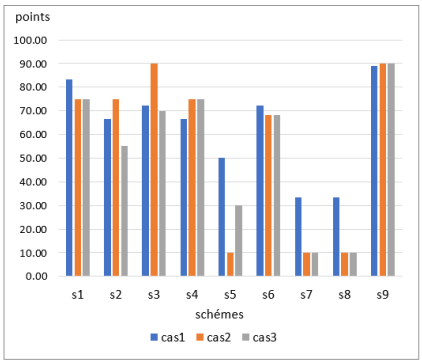

Figure 4. Évaluation des schémas

Les évaluations placent les schémas S5, S7, S8 comme les moins bons dans les trois cas considérés. La structuration dans S5 et S7 se base sur une seule collection qui n'est pas prioritaire dans les critères pris en compte. Alors que S3 se démarque dans le cas 2, pour la forte priorité de la collection *Companies*, seule collection de S3 qu'imbrique des documents en accord avec les critères.

S9 et S6, parmi d'autres schémas, sont stables dans leurs scores pour les trois cas. S9 est le meilleur car il correspond à tous les critères. Les bons résultats aux trois cas dénote une forme de "polyvalence" du schéma qui permet de résister aux évolutions des priorités. S6 paie le coût de ne pas considérer l'imbrication dans la collection *Agencies* et d'introduire de la redondance de documents alors qu'on préfère l'éviter (critère 2).

Les critères à considérer et leur poids [3] dépendent du contexte applicatif mais peuvent aussi correspondre à de bonnes pratiques préconisées pour le développement ou à une priorité générale. Par exemple, adopter des schémas très "compacts" pour limiter l'empreinte mémoire lorsque des données seront peu utilisées. Ou, en donnant priorité à la qualité logicielle, privilégier les schémas les plus "lisibles". Sachant que les critères peuvent diverger et évoluent certainement, l'utilisation des métriques et critères pour un choix de schéma peut aider dans un processus continue de "tuning" de la base qui peut conduire à des évolutions de la structuration ou à la création de copies des données avec des structures différentes. Pendant un certain temps, une base pourrait avoir, pour les mêmes données, une copie avec une schéma Sx et une autre copie avec un schéma Sy.

5. Travaux connexes

Dans l'étude de l'état de l'art, nous nous sommes intéressés aux travaux concernant des système NoSQL ainsi qu'à des propositions antérieures pour des données

3. Le choix des poids est un sujet qui reste à approfondir

complexes, des documents XML et des métriques logicielles. (Klettke *et al*., 2002) s'appuie sur le modèle de qualité de software ISO 9126 et adapte cinq métriques pour évaluer des documents XML en travaillant sur la DTD. Ces travaux nous ont servi de point de départ. Ils travaillent sur une représentation sous forme de graphe et les métriques considèrent le nombre de références, noeuds et font un rapprochement avec la complexité cyclomatique (McCabe, 1976). (Pušnik *et al*., 2014) propose 6 métriques associées chacune à un aspect de qualité telles que la structure, la clarté, l'optimalité, le minimalisme, la réutilisation et la flexibilité. Ces métriques utilisent 25 variables qui mesurent le nombre d'éléments, d'annotations, de références et de types parmi d'autres. (Li, Henry, 1993 ; Chidamber, Kemerer, 1991 ; McCabe, 1976) travaillent sur des métriques de logiciel avec paradigmes procedurale et orienté objet. Plusieurs métriques sont proposées pour refléter notamment les niveaux de couplage entre composants et classes, la taille des objets, des hiérarchies de classes et le nombre de méthodes. Nous avons étendu et adapté ces propositions pour notamment prendre en compte les particularités d'imbrication de documents et types d'attributs JSON en vue d'une utilisation dans une base de documents telle que MongoDB.

(Mior *et al*., 2017) et (Lombardo *et al*., 2012) s'intéressent aux alternatives de "modélisation" des données dans Cassandra (modèle "big table") avec des objectifs d'étude sur le stockage et les performances de requêtes implémentés avec SET et GET. (Lombardo *et al*., 2012) propose la création de plusieurs versions des données avec des structures différentes chacune étant plus adaptée pour l'évaluation d'une requête différente dans l'esprit des requêtes pré-calculées. Dans ce travail, aucune métrique n'est proposée pour évaluer les différentes versions. (Zhao *et al*., 2014) propose un algorithme pour la création systématique d'une base de données orientées documents à partir du modèle entité-relation. Cet algorithme propose une dé-normalisation de ce modèle avec pré-calcul des jointures naturelles par imbrication de documents. Le schéma résultant correspond au modèle du schéma S6 de notre scénario de validation. Ce choix engendre en général de la redondance des données. (Zhao *et al*., 2014) propose une métrique pour cela en utilisant la connaissance du volume des données. Dans cet article, nous proposons un ensemble plus large de métriques structurelles qui inclut deux métriques pour analyser la redondance sans connaissance du volume des données. Si les informations sur les données sont disponibles, elles peuvent être utilisées en complément.

Des travaux récents s'intéressent à l'analyse du schéma d'une base de données déjà implémentée. (Gallinucci *et al*., 2018) propose d'abstraire le schéma en considérant les variantes présentes des documents dans une collection et introduit une métrique d'entropie qui permet de définir la précision avec laquelle les documents ont été classés. Dans les outils existants, notons MongoDBCompass [4] qui permet de monitorer le temps d'exécution des requêtes, de connaître le volume des données d'une collection de documents et d'extraire des informations par rapport à la structure d'une collection. Cela fonctionne donc sur des bases déjà opérationnelles. Nous avons mentionné JSON

4. MongoDBCompass, https://docs.mongodb.com/compass/master/. Accessed: 2018-02-12.

schema [5] , qui est le résultat d'efforts pour faciliter la validation de documents JSON. Certains outils analysent des documents JSON dans le but d'abstraire un "schéma" permettant d'identifier les collections et les types sous-jascents.

D'autres travaux, sans formaliser ou suggérer des métriques sur des schémas semi-structurés, fournissent des lignes directrices, des bonnes pratiques et des aspects à prendre en compte dans le choix des structures. (Abiteboul, 1997) fournit des aspects à considérer pour les données semi-structurées et un aperçu de propositions de modèles et de langages de requête pour ces données. (Sadalage, Fowler, 2012) discutent divers modèles de données et produits NoSQL (MongoDB, Cassandra et Neo4j) et quelques problématiques de migration d'une base relationnelle vers des *BigTables*, documents et graphes. (Copeland, 2013) et MongoDB [6] proposent des lignes directrices pour la création de bases MongoDB en s'appuyant sur des cas appliqués à différents domaines. Ces bonnes pratiques peuvent être formalisées dans les critères que nous proposons afin d'être prises en compte dans l'analyse des schémas.

6. Conclusion et perspectives

Dans ce travail, nous nous sommes intéressés à des questions de qualité des structures de données pour des bases de documents JSON, tel que MongoDB. La flexibilité de structuration de ces bases est appréciée par la souplesse qu'elle permet pour représenter des données semi-structurés. Cependant, cette flexibilité a un coût dans les performances, le stockage, la lisibilité et la mantenabilité des bases et des applications. Ainsi, le choix de la structuration des données est très importante et ne doit pas être négligé. En considérant des travaux en génie logiciel et dans le contexte des bases orienté objet et XML, nous avons défini des métriques structurelles pour des "schémas" JSON. Ces métriques, organisées en catégories, reflètent des éléments de complexité du schéma qui jouent sur des aspects de qualité de la base. Elles peuvent ainsi être utilisées pour analyser et comparer différentes manières de structurer les données.

Nous avons présenté un scénario d'utilisation des métriques avec plusieurs variantes de schéma et certains critères et priorités applicatifs. L'analyse avec les critères, permet d'écarter certains schémas et de mettre en avant d'autres. Ces résultats sur les aspects structurels ont été comparés et, sont bien en phase, avec les résultats d'expériences d'évaluation de performances que nous avons menés avec des bases contenant des données. Il est intéressant de noter que lors du travail sur les structures nous avons pu considérer à "faible coût" plus de variantes de schémas que lors de l'expérimentation avec les bases. Cela a apporté un résultat inattendu qui est l'identification d'un schéma différent avec de très bonnes caractéristiques.

Les métriques proposées n'ont pas l'ambition de représenter un ensemble complet mais sont une base qui va probablement évoluer. La suite des travaux, incluent des

5. Json schema, http://json-schema.org/. Accessed: 2018-02-12

6. Rdbms to mongodb migration guide. (2017, Nov). White Paper. Consulté sur https://www.mongodb.com/collateral/rdbms-mongodb-migration-guide

validations à plus grande échelle. Nous allons notamment poursuivre le développement du système Scorus (mentionné en introduction) afin de compléter l'outil de génération automatique de schémas. Nous allons également travailler dans la formalisation d'un système de recommandation pour faciliter la définition des critères en utilisation les métriques, les requêtes fréquentes et autres préférences fonctionnelles ou non fonctionnelles des utilisateurs potentiels.

Remerciements

Nous remercions G. Vega, J. Chavarriaga et C. Labbé pour les échanges autour de ce travail ainsi qu'aux relecteurs anonymes pour leur retours.

Bibliographie

Abiteboul S. (1997). Querying semi-structured data. In *Proceedings of the 6th international conference on database theory*, p. 1–18. London, UK, UK, Springer-Verlag. Consulté sur http://dl.acm.org/citation.cfm?id=645502.656103

Chidamber S. R., Kemerer C. F. (1991). *Towards a metrics suite for object oriented design* (vol. 26) n° 11. ACM.

Copeland R. (2013). *Mongodb applied design patterns*. Oreilly.

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

Gómez P., Casallas R., Roncancio C. (2016). Data schema does matter, even in nosql systems! In *Research challenges in information science (rcis), 2016 ieee tenth international conference on*, p. 1–6. Grenoble, France, IEEE.

Klettke M., Schneider L., Heuer A. (2002). Metrics for xml document collections. In *International conference on extending database technology*, p. 15–28.

Li W., Henry S. (1993). Object-oriented metrics that predict maintainability. *Journal of systems and software*, vol. 23, n° 2, p. 111–122.

Lombardo S., Nitto E. D., Ardagna D. (2012). Issues in handling complex data structures with nosql databases. In *14th international symposium on symbolic and numeric algorithms for scientific computing, SYNASC 2012, timisoara, romania, september 26-29, 2012*, p. 443–448. Consulté sur http://dx.doi.org/10.1109/SYNASC.2012.59

McCabe T. J. (1976). A complexity measure. *IEEE Transactions on software Engineering*, n° 4, p. 308–320.

Mior M. J., Salem K., Aboulnaga A., Liu R. (2017). Nose: Schema design for nosql applications. *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, n° 10, p. 2275–2289.

Pušnik M., Heričko M., Budimac Z., Šumak B. (2014). Xml schema metrics for quality evaluation. *Computer science and information systems*, vol. 11, n° 4, p. 1271–1289.

Sadalage P. J., Fowler M. (2012). *Nosql distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education.

Zhao G., Lin Q., Li L., Li Z. (2014, Nov). Schema conversion model of sql database to nosql. In *P2p, parallel, grid, cloud and internet computing (3pgcic), 2014 ninth international conference on*, p. 355-362.