Towards quality analysis for document oriented bases

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

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

Abstract. Document-oriented bases allow high flexibility in data representation which facilitates a rapid development of applications and enables many possibilities for data structuring. Nevertheless, the structural choices remain crucial because of their impact on several aspects of the document base and application quality, e.g, memory print, data redundancy, querying and navigation facility and performances, readability and maintainability. Our research is motivated by quality issues of document-oriented bases. We aim at facilitating the study of the possibilities of data structuring and providing objective metrics to better reveal the advantages and disadvantages of each solution with respect to user needs. In this paper, we propose a set of structural metrics for a JSON compatible schema abstraction. These metrics reflect the complexity of the structure and are intended to be used in decision criteria for schema analysis and design process. This work capitalizes on experiences with MongoDB, works on XML and software complexity metrics. The paper presents the definition of the metrics together with a validation scenario where we discuss how to use the results in a schema recommendation perspective.

Keywords: NoSQL, structural metrics, document-oriented systems, MongoDB

1 Introduction

Nowadays, applications and information systems need to manage a large amount of heterogeneous data while meeting various requirements such as performance or scalability. NoSQL systems provide efficient data management solutions while offering flexibility in structuring data. Our work focuses on document-oriented systems, specifically those storing JSON documents, including MongoDB[3]. These systems are "schema-free". They support semi-structured data without a previous creation of a schema (unlike relational DBMS) [1]. Data can be stored in collections of document with atomic and complex attributes. This flexibility enables rapid initial development and permits many data structure possibilities for the same information. The choice is quite crucial for its potential impact on several aspects of application quality. [2]. Indeed, each structure may have advantages and disadvantages regarding several aspects, such as the memory footprint of the document base, data redundancy, navigation cost, data access or program readability and maintainability.

*Institute of Engineering Univ. Grenoble Alpes, LIG, 38000 Grenoble, France

[3]MongoDB is largely used nowadays (https://www.mongodb.com). It uses BSON, a binary-encoded serialization of JSON-like documents (http://bsonspec.org)

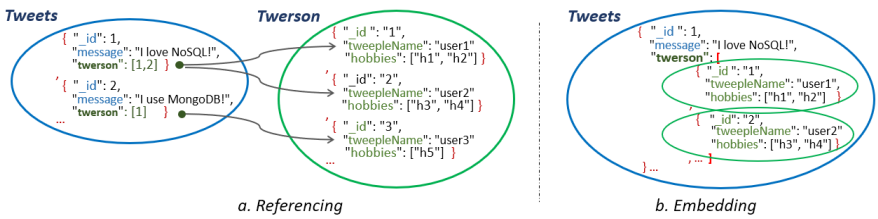

Fig. 1: Example of data in MongoDB using referencing and embedding documents

It becomes interesting to consider several data structure candidates to retain a single choice, a temporal choice or several parallel alternatives. The analysis and comparison of several data structures is not easy because of the absence of common criteria for analysis purposes[4] and because there are potentially too many structuring possibilities.

Our research in the *SCORUS* project is a contribution in this direction. Even if document-oriented systems do not support a database schema, we propose to use a "schema" abstraction. The goal is to assist users in a data modeling process using a recommendation approach. We seek to abstract and to work with a "schema" to facilitate comprehension, assessment and comparison of document oriented data structures. The purpose is to clarify the possibilities and characteristics of each "schema" and to provide objective criteria for evaluating and assessing its advantages and disadvantages. The main contribution of this paper is the proposal of a structural metrics set for JSON compatible scheme abstraction. These metrics reflect the complexity of the data structures and can be used to establish quality criteria such as readability and maintainability. The definition of these metrics is based on experiments with MongoDB, XML-related work and metrics used in Software Engineering for code quality.

In Section 2, we provide a background on MongoDB and the motivation of our proposal. Section 3 provides a brief overview of the *SCORUS* project and introduces the schema abstraction AJSchema to manages it. In Section 4, we propose structural metrics to measure AJSchemes. Section 5 is devoted to the validation. It presents a scenario for schema comparison using our metrics. Related work is discussed in Section 6. Conclusions and research perspectives are presented in Section 7.

2 Background and motivation

We are interested in quality issues of document-oriented databases. We focus on JSON documents managed by systems like MongoDB. Following, we present a brief background of its data type system [3].

Data is managed as collections of documents (see Figure 1) where a document is a set of `attribute:value` pairs. The value type can be atomic or complex. Complex data type means either an array of values of any type or another *nesting* document. An attribute value can be the identifier of a document in another collection. This allows *referencing* one or more documents.

[4]For document oriented data there are no design criteria analogous to normalization theory in the relational model

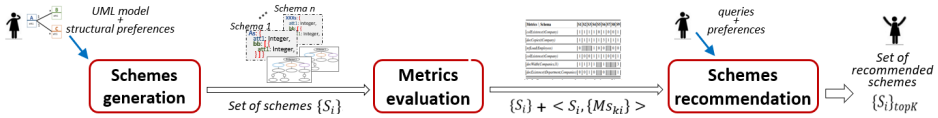

Fig. 2: *SCORUS* overview

This simple type system provides a lot of flexibility in creating complex structures. Collections can be structured and connected in various forms considering or not data replication. e.g. completely nested collections or combination of nesting and referencing. Figure 1 depicts two ways of structuring information about *tweets* and *twerson* in a tweeter context. Figure 1a shows a `Tweets` collection and a `Twerson` collection. Documents in `Tweets` reference documents in `Twerson`. The choice in Figure 1b is different, there is a single collection `Tweets` with nested documents for their "*twerson*". In this example, there is no duplication of data.

There is not a definitive best structure because it depends on the current needs and priorities. However, the characteristics of the data structure have a strong impact on several aspects such as the size of the database, query performance and code readability of queries. The experiments presented in [2] confirm that influence. Our work is motivated by the analysis of how such aspects influence the maintainability and usability of the database as well as applications. In particular, it appears that collections with nested documents are favorable to queries following the nesting order. However, access to data in another order and queries requiring data embedded at different levels in the same collection will be penalized. The reason is that the complexity of manipulations required in such cases is similar to joining several collections. In addition, collections with nested documents have a larger footprint than the equivalent representation with references. When structuring data, priorities may lead to diverging choices, as replicating documents in multiple collections while reducing memory requirement and storage cost.

3 SCORUS Overview and Schema Abstraction

Our research focuses on helping users to understand, evaluate and make evolve semi-structured data in a more conscious way. The main contribution of this paper are the structural metrics presented in Section 4. Hereafter, we provide a brief overview of our larger project, *SCORUS*; we introduce the schema abstraction (Sections 3.2 and 3.3) and tree representation (Section 3.4) we have defined to ease the evaluation of the metrics.

3.1 SCORUS Overview

The *SCORUS* project aims at facilitating the study of data structuring possibilities and providing objective metrics to better reveal the advantages and disadvantages of each solution with respect to the user needs. Figure 2 shows the general strategy of *SCORUS* which involves three steps: (1) from a UML data model, generating a set of schemes alternatives; (2) evaluating such schemes by using the metrics proposed in this paper; and (3) providing a top k of the most suitable schemes according to user preferences and

a. AJSchema abstraction of Figure1a *b. AJSchema abstraction of Figure1b*

Fig. 3: AJSchemes for examples presented in Figure 1

application priorities. The sequence of the three steps can take place in a design process, but each step can be carried out independently for analysis and tuning purposes [5].

3.2 Schema abstraction for structural analysis

To facilitate reasoning about the data structuring choices in document oriented systems, we define a schema abstraction, called here *AJSchema*. It is based on JSON schema approach [4] and on the data types supported by MongoDB.

AJSchema allows us to provide a concise representation of the collections and types of the documents to be stored in the base. Figure 3 shows the abstracted AJSchema for the examples in Figure 1. For each collection the schema describes the type of its documents. A document type is a set of `attribute:type` pairs enclosed by `{}`. Types are those of MongoDB. Arrays are symbolized by `[]` and inside them, the type of its elements (atomic or document). In this paper, we use indistinctly the terms AJSchema or schema to refer to this abstraction. For schema-free systems like MongoDB, such abstractions can be considered as a data structure specification for construction and/or maintainability process.

3.3 An AJSchema from UML Model

Based on a UML model, *SCORUS* provides several AJSchemes that are then compared to each other to improve the selection. Hereafter we illustrate the correspondence between the classes and relationships of the UML model and an AJSchema (Figures 4 and 5) [5].

Considering a UML model, $E = \{e_1, ..., e_n\}$ are the classes. The attributes of a class e_i are designated in the following by the type te_i and its relationships with the set $R(e_i) = \{r_1, ..., r_n\}$. Roles of relationships are known and noted by r_{irol}. Figure 4 shows a UML model with classes `Agency, BusinessLines, Owner, Creative` and `Publicity`. The properties of the class `Agency` are designated by the type `tAgency` [6].

An AJSchema describes the types of the collections that will be used. A collection type includes the attributes of a UML class (te_i) and extra attributes representing the relationships of this class ($R(e_i)$). The latter are named by the target role of r_i from the class e_i. Figure 5 presents a possible schema for the UML data model of Figure 4. It has three collections `Agencies, Owners` and `Creatives`. The type of `Agencies` is formed by the attributes `agencyName` and `id`, corresponding to type `tAgency`, and attributes `bLines` and `ows`, corresponding to relationships `r1` and `r3` respectively.

[5] Schemes generation and recommendation are beyond the scope of this article.

[6] type te_i has by default the attribute `id` corresponding to the MongoDB document identifier.

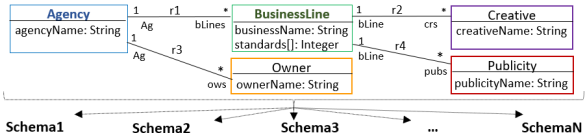

Fig. 4: From UML to several options of schemes

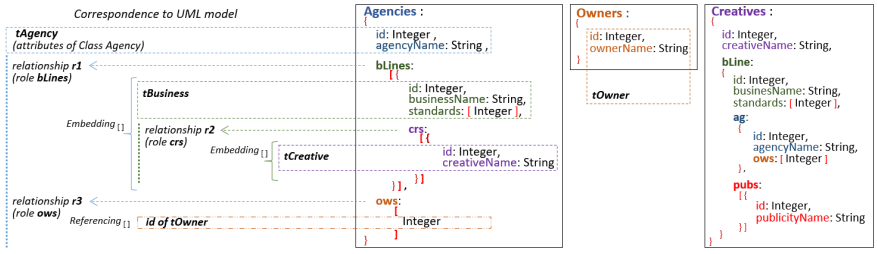

Fig. 5: Example of AJSchema option for the UML model of Figure 4

Relationships can be materialized by referencing or by embedding of documents. This choice, so as the cardinality, determine the type of the attribute. By referencing, the type is Integer, for the id of the referenced document. By embedding, the type is a document of type te_j. A cardinality "many" implies an array of types. Materialization by embedding induces a nested level in the structure. Referencing can occur at any level and forces the existence of a collection with type te_j.

In our example, the attribute `bLines` represents the 1-to-many relationship `r1` by embedding of documents. Its type is therefore an array of documents of type `tBusinessLines`. Attribute `ows` corresponds to the role of relationship `r3`. The 1-to-many cardinality and a representation by referencing lead to an attribute of type array of Integers. Referencing forces the existence of a collection of Owners.

3.4 Tree representation

To facilitate metrics evaluation, we use a tree representation of the AJSchema. The tree contains information about data types, nested levels, and embedded/referenced elements. The tree semantics is illustrated in Figure 6 representing the AJSchema of Figure 5.

The root node has a child per collection in the AJSchema. `Agencies`, `Owners` and `Creatives` in our example. The collection type is represented by the child sub-tree. Nodes of the form *typename@li* indicate that attributes of the *typename* appear in the documents at level *li* (starting with level 0) . Attributes representing the relationships $R(e_i)$ appear as follows. A node with the name (and role used) of each relationship is created (e.g. $r1_{bline}$) with a child node, either *REF*, either *EMB* according to the choice of referencing or embedding documents. Arrays, denoted $[]$, can be used for 1-to-many relationships. For example, the subtree on the left of Figure 6, shows relationship $r1_{bline}$ materialized as an attribute (added to agency) of type array of business, $tAgencyEMB[]tBusiness$. Relation $r2$ of $tBusiness$ causes the embedding of an array of $tCreative$. The nodes indicating a level (e.g. $tBusiness@l_1$) allow to easily identify the depth of a type and its extra attributes associated with the relationships.

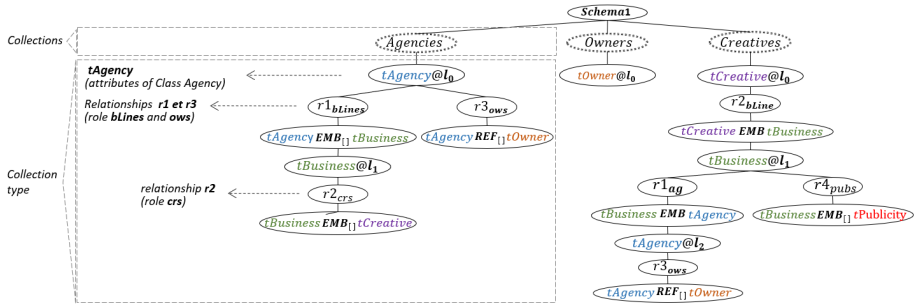

Fig. 6: Tree structure representing AJSchema of Figure 4

4 Structural Metrics

In this section, we propose a set of metrics that reflects key aspects of the semi-structured schema complexity. The purpose is to facilitate schema analysis and comparison. We have defined a set of metrics grouped into 5 categories presented in the Sections 4.1 to 4.5. A summary is presented in Section 4.6. In the following, φ denotes a collection, t a document type and x a schema.

4.1 Existence of types and collections

Having a collection can be mainly motivated by the access improvement to its document type at first level or its nested types. On the other hand, nesting a document into another one can be motivated by the fact that information is often accessed together. It may also be interesting to realize if a document type is nested in many places to help reducing the collection complexity.

In this section, we have defined metrics that allows us to identify the existence of a document type *t* in a schema. We consider two cases: (1) the existence of a collection whose type t is at the first level (l_o), and (2) the presence of such documents nested within other documents. These cases are covered, respectively, by metrics *colExistence* and *docExistence*.

Existence of a collection of documents of type t:

$$colExistence(t) = \begin{cases} 1 & : \text{node } t@l_0 \text{ exists in schema } x \\ 0 & \end{cases} \quad (1)$$

Existence of embedded documents of type t: this is materialized in the graph by a node $*EMB\,t$.

$$docExistence(\varphi, t) = \begin{cases} 1\ t \in \varphi & \text{node } *EMB\,t \text{ exists in the paths child of node } \varphi \text{ in } x \\ 0\ t \notin \varphi & \end{cases} \quad (2)$$

Figure 6 shows collections for the types *tAgency, towners and tCreative* (nodes $@l_0$) but not for *tPublicity*. Documents of type *tPublicity* exist exclusively embedded in *tCreative* documents. Note that documents of type *tBusiness* are embedded in two collections, *Agencies and Creatives*.

4.2 Nesting depth

In general, the deeper the information is embedded, the higher is the cost to access it. This is true unless the intermediary information is also required. Knowing the nesting level of a document type facilitates the estimation of the cost of going down and back through the structure to access the data or to restructure the extracted data with the most suitable format. We propose a set of metrics to evaluate the complexity induced by embedded data. The following two metrics allow us to know the maximum depth levels of collections and schema.

Collection depth: the *colDepth* (3) metric indicates the level of the more deeply embedded document in a collection. Embedded documents are represented by the EMB nodes in the graph.

$$colDepth(\varphi) = max(depth(p_i)) \quad : p_i \text{ is a valid child path of node } \varphi \tag{3}$$

$$depth(p) = n \quad \text{number of nodes } EMB \text{ in path } p \tag{4}$$

Schema Depth: the *globalDepth* (5) metric indicates the deepest nesting level of a schema by considering all collections.

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

Having recurring nested relations increases the schema complexity without necessarily improving query performance. A very nested collection can be advantageous if frequent queries require the joined information. Besides, having such a schema can be ideal if the access pattern matches with the schema. Otherwise, projections and other structuring operations will probably be required, introducing complexity in the data manipulation (see following metrics). This affects code readability and maintainability.

In Figure 6, the depth of collection *Owners* is 0 and the depth of collections *Agencies* and *Creatives* is 2. The maximum depth of the schema is 2. Note that in the *Creatives* collection, the type *tAgency* adds no nesting level as it uses an array with references to *Owners*.

Depth of a type of document: the metric *docDepthInCol* (6) indicates the embedding level of a document of type t in a collection φ. If the items of the collection are of type t (node $t@l_0$), then the depth is zero. Otherwise, the metric is the level of the deepest embedded document of this type ($EMB\,t$ node) according to the root-leaf paths.

$$docDepthInCol(\varphi, t) = \begin{cases} 0 & : t \text{ corresponds to node } t@l_0 \text{ son of node } \varphi \\ max(docDepth(p_i, t) & : p_i \text{ is a valid root-leaf child path of node } \varphi \end{cases} \tag{6}$$

$$docDepth(p, t) = n \quad \text{number of nodes } EMB \text{ between } root \text{ and } *\,EMB\,t \tag{7}$$

In the *Creatives* collection of the example, the nesting level of *tPublicity* is 2, that of *tCreative* is 0. *tCreative* is also nested at level 2 in the *Agencies* collection.

We also introduce the *maxDocDepth* (8) and *minDocDepth* (9) metrics to measure the most and shallowest levels where a document type appears in a schema.

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

$$minDocDepth(t) = min(DocDepthInCol(\varphi_i), t) \quad : \varphi_i \in x \ \wedge \ t \in \varphi_i \qquad (9)$$

Knowing the minimum and maximum levels eases to estimate how many intermediate levels should be treated for the more direct or the less direct access to a document of a certain type. In the example, *minDocDepth(tBusiness)* = 1 as there is no collection of documents of that type.

4.3 Width of the documents

Now we can look at the complexity of a document type in terms of its number of attributes and the complexity of their types. These metrics are motivated by the fact that documents with more complex attributes are more likely to require more complex access operations and projections. The reason is that to extract the attributes required by a query, it is necessary to "remove" the other attributes and data stored together. This operation is more expensive for documents with a larger number of attributes, i.e., with a high width. It may be interesting to choose a scheme by analyzing its "wide" and its nesting level.

The *docWidth*[7] (10) metric of a document type is based on the number of atomic attributes (coefficient a = 1), the number of attributes embedding a document (coefficient b = 2), the number of attributes of type array of atomic values (coefficient c = 1) and array of documents (coefficient d = 3). Arrays of documents have the highest weight as the experiments revealed them as the more complex to manage.

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

The metrics for each type of attributes can also be used separately. The size of the arrays is not considered here because it is not necessarily available in a design phase. If the size is available, it seems interesting to differentiate the orders of magnitude of the arrays, i.e. small ones vs very large ones (less than ten elements, around thousands elements, etc).

In Figure 4, *Agencies* and *Creatives* collections use documents of type *tBusiness* but do not have the same attributes. In *Creatives*, the type includes arrays of agencies and publicity, *docWidth (Creatives, tBusiness)* = 8, unlike *Agencies* where *docWidth(Agencies,tBusiness)* = 4.

4.4 Referencing rate

Referential integrity becomes difficult to maintain for collections which documents are referenced by many other collections. For a collection with documents of a certain type t, the metric *refLoad* (11) indicates the number of attributes (of other types) that are potential references to documents of type t.

$$refLoad(\varphi) = n \qquad n \text{ - number of nodes } * REF\ t \text{ where } t{=}t@l_0 of node\varphi \qquad (11)$$

For the *Owners* collection in Figure 4, $refLoad(tOwner) = 2$: collection *Agencies* references *towner* at level 0 while collection *Creatives* references it in a document embedded at level 2.

[7] this metric is close to the fan-in metric for graphs

Category	Metric	Description	Schema	Collection	Type
Existence	*colExistence*	Existence of a collection		x	
	docExistence	Existence of a document type in a collection		x	x
Depth	*colDepth*	Maximal depth of a collection		x	
	globalDepth	Maximal depth of a schema	x		
	docDepthInCol	Level where a document type is in a collection		x	x
	maxDocDepth	The deepest level where a document type appears	x		
	minDocDepth	The least deep level where a document type appears	x		
Width	*docWidth*	"Width" of a document type		x	x
Referencing	*refLoad*	Number of times that a collection is referenced		x	
Redundancy	*docCopiesInCol*	Copies of a document type t in a collection		x	x
	docTypeCopies	Number of times a type is present in the scheme	x		

Table 1: Structural metrics

4.5 Redundancy

We are interested in estimating potential data redundancy during the schema design because it impacts several aspects. Data redundancy can speed-up access and avoid certain expensive operations (i.e. joins). However, it impacts negatively the memory footprint of the base and makes coherency enforcement more difficult. There is a cost and writing program complexity is increased and impacts the maintainability. As we are working on a structural basis, we do not use data replication information for the metric definition. The metric *docCopiesInCol* (12) is calculated by using the cardinality information of the relationships together with the representation choices in the semi-structured schema. Redundancy occurs for some cases of representation of the relationship by embedding documents.

$$docCopiesInCol(t,\varphi) = \begin{cases} 0 & : t \notin \varphi \; docExistence(\varphi,t) = 0 \\ 1 & : t \text{ corresponds to node } t@l_0 \text{ of } \varphi \\ \prod card(r_{rol},t) & : \begin{matrix} r_{rol} \text{ a valid node } r_{rol} \text{ father of a node} \\ {}_{EMB} \text{ of the path } \varphi \text{ and } *_{EMB}t \end{matrix} \end{cases} \tag{12}$$

$$card(r,\varepsilon) = n \qquad n - \text{ cardinality of } r \text{ on the } \varepsilon \text{ side in the UML model} \tag{13}$$

In the *Creatives* collection of Figure 2, the attribute for business, named *bline* , introduces redundancy for agencies. The relationship r1 may associate an agency A to n1 business instances. This leads to n1 copies of the document A. In the case where a business is referenced by n2 creatives (relationship r2), there will be n1 x n2 copies of the document A.

Moreover, we propose the metric *docTypeCopies(t)* indicating the number of times a document type is used in the schema. It reflects the number of structures that can potentially store documents of type t. This metric uses the metric of existence already introduced.

4.6 Summary

The proposed metrics are summarized in Table 1. These metrics are evaluated in the scope of the collections, types, or the whole schema, to reveal the data structure complexity.

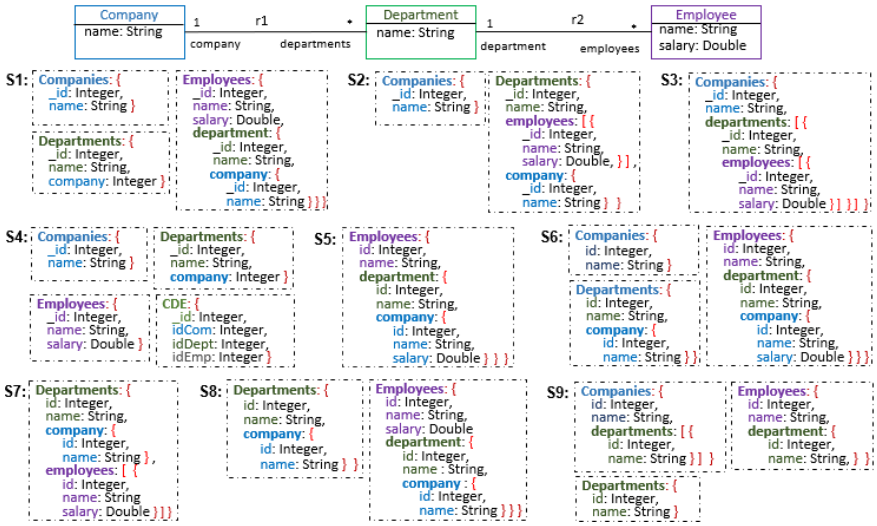

Fig. 7: Case study: UML data and AJSchema alternatives

5 Validation Scenario

As mentioned, our work aims at assisting users in the choice of document-oriented schema. The proposed metrics, together with application priorities, will be used to establish criteria for choosing and comparing schemes. This is primarily to bring out the most suitable schema according to certain criteria but also to exclude unsuitable choices or to consider alternative schemes that were not necessarily considered initially.

In the following we present a usage scenario for the proposed metrics. In Figure 7, we introduce an example with nine structuring alternatives. This case study was already used in the work with MongoDB databases presented in [2], where we discussed the impact of the data-structures on the query evaluation. Here we are considering a similar application context to analyze schema alternatives using the metrics. We evaluate the metrics for the nine schemes. Table 2 reports a subset of them.

Schema analysis will be based on user priorities such as efficient access, data consistency requirements and other user preferences. A criterion corresponds to a preference (maximization or minimization) over one or several metrics to privilege a set of schemes [8].

In our case study, the priorities of the application concern efficient access to companies information, including the names of their departments (high priority), and getting the employee with the highest salary in the company from its identifier or its name. Considering these priorities, the collection *Companies* plays an important role (criterion 1)[9] as well as the manipulation of its instances (criterion 5). The departments are accessed via the companies (criterion 6). Furthermore, it is known that the consistency of business data is important. It is therefore preferable to limit the copies to these data (criterion 2). Moreover, access to all employees (criterion 4) is not a priority.

[8] A criterion is represented by a function in terms of maximization or minimization of metrics

[9] The criterion number facilitates the presentation. It doesn't correspond to a priority.

Metrics \ Schema	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(tEmployee)	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

Table 2: Case study: sub-set of metrics of schema S1 to S9

In Table 3, each line represents one of the six criteria already mentioned. Each criteria has been evaluated for the nine alternative schemes. The values of the criteria evaluation were normalized (between 0 and 1). These values introduce a relative order between the schemes. For example, considering criteria 4, the schema S1, S4, S5, S6 and S9 are preferred over the others.

The analysis of schemes is multi-criteria (6 criteria in our case). Each criterion can have the same weight, or it can be some more important than others. The evaluation function of a schema, noted *schemaEvaluation* is the weighted sum of criteria.

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

We evaluated three different weights: same weight for all criteria (case 1), priority criteria focused on companies (case 2), and priority addition on employees motivated by a new access pattern to its information (case 3). Figure 8 shows the result of the evaluation of 9 schemes for the three cases.

The estimates place schemes S5, S7 and S8 as the worst in the three cases. S5 and S7 are based on a single collection that is not a priority in the current criteria. On the other hand, S3 stands out in case 2 due to the high priority of its unique collection *Companies*. In addition, this collection embeds data in an appropriate order regarding the criterion 6.

Some scheme, as S9 and S6, are stable in their scores for all three cases. S9 is the best because it matches all criteria; its good results in the three cases denotes a form of "versatility" of the schema that can withstand changes in priorities. S6 introduces redundancy which is penalized by criterion 2. Meanwhile, criterion 6 penalizes it by not having embedded documents in collection *Companies*.

The criteria to be considered and their associated weight depends on the applications and the user. They may reflect good practices advocated for development or general priorities. For example, a very "compact" schema limiting memory footprint can be preferred for rarely used data. Knowing that the criteria may evolve and lead to divergent

Criteria \ Schema		S1	S2	S3	S4	S5	S6	S7	S8	S9
Criteria 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
Criteria 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
Criteria 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
Criteria 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
Criteria 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
Criteria 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

Table 3: Criteria evaluation on schema S1 until S9

Criteria	Factor		
	Case 1	Case 2	Case 3
Criterion 1	16.67	50	30
Criterion 2	16.67	10	10
Criterion 3	16.67	0	0
Criterion 4	16.67	0	20
Criterion 5	16.67	15	15
Criterion 6	16.67	25	25

a. Sets of weights

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

b. Schemes evaluation by case

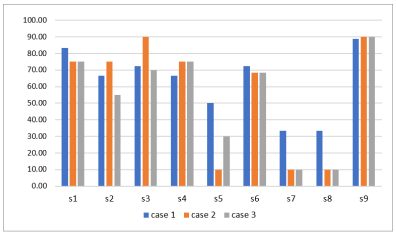

Fig. 8: Schema evaluation

choices, the use of metrics and criteria for a scheme analysis can help in a continuous process of ”tuning” of the base. This can lead to schema evolution and data replication with heterogeneous structures. For a while, a document base may have, a copy (or partial copy) of the data with schema Sx and another copy with scheme Sy.

6 Related work

We studied works concerning NoSQL systems [5–8] [9–12] [3, 13, 14] as well as other proposals for complex data [15, 16] [4, 17], XML documents [18, 19] and software metrics [20–25].

Concerning XML, Klettke et al. [18] propose 5 structural metrics based on the software quality model ISO 9126. They work on a graph representation of the DTD and metrics consider the number of references, nodes and make a link with the cyclomatic complexity [22]. In [19], Pušnik et al. propose six metrics each one associated with a quality issue such as the structure, clarity, optimality, minimalism, reuse and flexibility. These metrics use 25 variables that measure the number of elements, annotations, references and types, among others. We extended and adapted these proposals to take into account particularities of JSON as embedded documents and complex attribute types.

Our metrics are also influenced by software metrics [20–25]. Metrics proposed in [20–23] reflect, for example, the coupling levels between components, the size of the class hierarchies, the size of objects and the number of methods. [25] is an excellent survey of software metrics considering those based on the complexity of code and object oriented concepts.

Concerning NoSQL approaches, some works investigate data modelling alternatives in Cassandra [5,6]. The main concerns of the authors regarding the modelling alternatives with ”big tables” are the storage requirements and query performance. Queries are implemented with SET and GET primitives. Lombardo et al. [6] propose the creation of several versions of the data with different structures. Each version is best suited for a different query in the style of pre-calculated queries. Zhao et al. [7] propose a systematic schema conversion model of a relational database to NoSQL database. It creates a collection per entity type and the documents embed recursively the entities they reference. The structure is in the style of schema S6 in our validation scenario. Using the vocabulary of the relational model, this choice corresponds to a de-normalized structure with pre-calculation of natural joins (represented with embedded documents). The authors propose a metric for the data redundancy generated which uses the data

volume. Among the existing tools working on operational bases, MongoDBCompass [8] allows to monitor the query execution time and data volume of a collection of documents. JSON schema [4] is the result of efforts to facilitate the validation of JSON documents. Tools as json-schema [17] analyze JSON documents in order to abstract a "scheme" with explicit collection and type definitions. Other researchers, as [9–12], work on schemes deduction for existing schema-free document-oriented bases. Their motivation is helping to understand data structuring and explaining its variants.

Guidelines to consider in modelling semi-structured data are discussed in [3, 13–16]. Sadalage et al. [13] analyze various data models and NoSQL systems including MongoDB, Cassandra and Neo4j. Their main concerns are the issues in the migration of a relational database towards BigTables, documents and graphs. [3, 14] propose guidelines for creating Mongo databases based on several use cases. These "best practices" can be formalized in our work as criteria to be taken into account in the schema analysis. To the best of our knowledge, no structural metrics are currently defined in the literature.

7 Conclusion and perspectives

This work is motivated by quality issues in document-oriented bases. We focus on data structuring in JSON documents, supported by MongoDB. The flexibility offered by such systems is appreciated by developers as it is easy to represent semi-structured data. However, this flexibility comes at a cost in the performance, storage, readability and maintainability of the base and it's applications. Data structuring data is a very important design decision and should not be overlooked.

In this work, we briefly described *SCORUS*, our larger project, which aims at helping user to clarify the possibilities of data structuring and to provide metrics allowing to take decisions in a more conscious way. We defined a schema abstraction called AJSchema to reason about semi-structured data and calculate metrics. We proposed a set of 11 structural metrics covering aspects as existential, nesting depth, nesting width, referencing, and redundancy. These metrics reflect the complexity of schema elements that play a role on quality aspects.

We presented a usage scenario of the metrics to analyze several schema variations and certain application criteria and priorities. The criteria analysis can rule out certain schema and highlight others. These findings on structural aspects were compared, and are well in line, with the results of performance evaluation experiments we conducted with databases containing data. It is interesting to note that when working on the structures, it is possible to consider more schema variants than when experimenting with the databases. This brought an unexpected result, that is the identification of a different schema with very good characteristics.

The proposed metrics form a set that is likely to evolve. Further work includes validation on a larger scale. Ongoing and future work include further development of the *SCORUS* system to complete the automatic schema generation.We will also work in formalizing a recommendation system to facilitate the definition of criteria by using the metrics, important queries and other functional or non-functional preferences of potential users.

Acknowledgements: many thanks to G. Vega, J. Chavarriaga, M. Cortés, C. Labbé, E. Perrier and P. Lago for their comments on this work.

References

1. Nayak, A., Poriya, A., Poojary, D.: Article: Type of nosql databases and its comparison with relational databases. International Journal of Applied Information Systems **5**(4) (March 2013) 16–19 Published by Foundation of Computer Science, New York, USA.
2. Gómez, P., Casallas, R., Roncancio, C.: Data schema does matter, even in nosql systems! In: Research Challenges in Information Science (RCIS), 2016 IEEE Tenth International Conference on, Grenoble, France 1–6
3. Copeland, R.: MongoDB Applied Design Patterns. Oreilly (2013)
4. jsonSchema: Json schema. http://json-schema.org/ Accessed: 2018-03-26.
5. Mior, M.J., Salem, K., Aboulnaga, A., Liu, R.: Nose: Schema design for nosql applications. IEEE Transactions on Knowledge and Data Engineering **29**(10) (2017) 2275–2289
6. Lombardo, S., Nitto, E.D., Ardagna, D.: 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
7. Zhao, G., Lin, Q., Li, L., Li, Z.: Schema conversion model of sql database to nosql. In: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on. (Nov 2014) 355–362
8. MongoDBCompass. https://docs.mongodb.com/compass/master/ Accessed: 2018-02-12.
9. Klettke, M., Störl, U., Scherzinger, S.: Schema extraction and structural outlier detection for json-based nosql data stores. Datenbanksysteme für Business, Technologie und Web (BTW 2015)
10. Wang, L., Zhang, S., Shi, J., Jiao, L., Hassanzadeh, O., Zou, J., Wangz, C.: Schema management for document stores. Proceedings of the VLDB Endowment **8**(9) (2015) 922–933
11. Ruiz, D.S., Morales, S.F., Molina, J.G.: Inferring versioned schemas from nosql databases and its applications. In: International Conference on Conceptual Modeling, Springer (2015)
12. Gallinucci, E., Golfarelli, M., Rizzi, S.: Schema profiling of document-oriented databases. Information Systems **75** (2018) 13–25
13. Sadalage, P.J., Fowler, M.: NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education (2012)
14. MongoDB: Rdbms to mongodb migration guide. White Paper (Nov 2017)
15. Abiteboul, S.: Querying semi-structured data. In: Proceedings of the 6th International Conference on Database Theory. ICDT '97, London, UK, UK, Springer-Verlag (1997) 1–18
16. Herden, O.: Measuring quality of database schemas by reviewing–concept, criteria and tool. Oldenburg Research and Development Institute for Computer Science Tools and Systems, Escherweg **2** (2001) 26121
17. jsonschema.net. https://jackwootton.github.io/json-schema/ Accessed: 2018-03-26.
18. Klettke, M., Schneider, L., Heuer, A.: Metrics for xml document collections. In: International Conference on Extending Database Technology, Springer (2002) 15–28
19. Pušnik, M., Heričko, M., Budimac, Z., Šumak, B.: Xml schema metrics for quality evaluation. Computer science and information systems **11**(4) (2014) 1271–1289
20. Li, W., Henry, S.: Object-oriented metrics that predict maintainability. Journal of systems and software **23**(2) (1993) 111–122
21. Chidamber, S.R., Kemerer, C.F.: Towards a metrics suite for object oriented design. Volume 26. ACM (1991)
22. McCabe, T.J.: A complexity measure. IEEE Transactions on software Engineering (4) (1976)
23. Fenton, N.E., Neil, M.: Software metrics: roadmap. In: Proceedings of the Conference on the Future of Software Engineering, ACM (2000) 357–370
24. Fenton, N., Bieman, J.: Software metrics: a rigorous and practical approach. CRC Press (2014)
25. Timóteo, A.L., Álvaro, A., De Almeida, E.S., de Lemos Meira, S.R.: Software metrics: A survey. Citeseer (2008)