

ModelDrivenGuide: An Approach for Implementing NoSQL Schemas

Jihane Mali[1], Faten Atigui[2], Ahmed Azough[1], and Nicolas Travers[2,3](✉)

[1] Université Sidi Mohamed Ben Abdellah, Fès, Morocco
jihane.mali@usmba.ac.ma, ahmed.azough@gmail.com
[2] CEDRIC, Conservatoire National des Arts et Métiers (CNAM), Paris, France
{faten.atigui,nicolas.travers}@cnam.fr
[3] Léonard de Vinci Pôle Universitaire, Research Center, Paris La Défense, France
nicolas.travers@devinci.fr

Abstract. With data evolution in terms of volume, variety and velocity, Information Systems (IS) administrators need to find the best solution to store and manipulate data with respect to their requirements. So far, existing approaches provide rules to transform a source model to a target model, but none of them propose a method to lead the choice of the most suitable solution. ModelDrivenGuide suggests a model transformation approach that focuses on proposing the different relevant solutions to the case of study. It is based on a common meta-model for the 5 families (Relational & NoSQL) and a generation heuristic. Our approach is validated using the TPC-C benchmark.

Keywords: NoSQL · Information Systems · User guide · MDA · Model transformation · Schema denormalization

1 Introduction

For several decades, the storage and the exploitation of data has mainly relied on Relational Databases (RDB). With the advent of Big Data, the volume of data has exploded, the heterogeneity has increased, causing problems of transformation from traditional databases to new storage on the Cloud, whether in terms of storage management, data query, cost or performance. These new data management systems are called *NoSQL* systems since 2009. The *NoSQL* data models correspond to various families of data structures: key-value oriented (KVO), column oriented (CO), document oriented (DO), and graph oriented (GO).

With more than 225 different *NoSQL* solutions, it is difficult for a company's CIO to determine the most suitable solution for its functional needs. Indeed, transferring the database to a NoSQL solution is an extremely heavy and costly process. Inadequate choices can lead to problems of scalability, data consistency or pricing. Our present work aims to provide an answer by driving the choice of digital transformation solutions.

© Springer Nature Switzerland AG 2020
S. Hartmann et al. (Eds.): DEXA 2020, LNCS 12391, pp. 141–151, 2020.
https://doi.org/10.1007/978-3-030-59003-1_9

Existing solutions for the digital transformation of a relational database remain essentially limited in their approach. Indeed, proposed methods are based either on a transformation of the physical model causing problems of scaling, or a purely logical model providing transformations with respect to the business logic. However, they do not take into account the optimization problems inherent to the origin problem of this transformation (scaling up).

Our `ModelDrivenGuide` approach provides logical modeling suitable for models refinement in order to generate all types of optimized physical schemas. Based on rules, it provides a decision making process that integrates the use case to guide the implementation choice of the SQL and/or NoSQL solution(s). The main contributions of this paper are: (i) an approach that subtly combines conception with optimization in DB transformation process; (ii) a common metamodel that supports all families (relational and NoSQL) and schema transformation; (iii) a heuristic to reduce the search space of model generation.

The rest of this paper is organized as follows: the state of the art is presented in Sect. 2. Then our approach is developed in Sect. 3. We finally validate our approach on the TPC-C benchmark in Sect. 4 and conclude in Sect. 5.

2 Related Works

Most of the existing studies have proposed either (i) a comparative study between RDB and NoSQL DB and/or how to transform relational data into NoSQL data or (ii) how to transform a conceptual model into a specific NoSQL DB; (iii) while very few studies have proposed criteria for the choice of physical model and implantation platforms.

Transformation of a Relational DB into NoSQL DB. These approaches define a set of mapping rules that transform a relational schema into a NoSQL schema. We can cite [11] for `CO` families with *HBase*, [18] with several `CO` DB, or [16] for `DO` families with *MongoDB*. Other techniques are concerned with queries to define the target data model [8,10].

Transformation of a Conceptual Model into NoSQL DB. [7] proposes to transform an `Entity Relationship (ER)` model into a `CO` model, based on the definition of a `CO` schema using primary and foreign keys, and on transformation rules. Similarly, [2] suggests a query-driven approach for modeling `Cassandra` starting from an ER model. They define dedicated logical and physical models, as well as transformation rules between models. [5] proposes a conceptual transformation approach which converts an ER model into one of the 4 NoSQL families with an abstract formalization of the mapping rules.

Other studies adopt a model-driven architecture (MDA) to transform a class diagram into NoSQL DB. [1] transforms a class diagram into a NoSQL DB. The authors present a common logical model which describes the four families of NoSQL DB. Then, this logical model is transformed into physical models related to the four families. In [3], the authors propose `UMLtoNoSQL`, a MDA based framework to map conceptual models to several data storage solutions.

Their approach combines model mapping techniques with a set of rules to translate OCL constraints into various query languages. [12] and [4] systematically transform a class diagram into a CO DB with HBase and GO DB respectively.

NoSQL DB Choice Orientation. To guide the choice of the NoSQL DB, we find performance comparisons for dedicated needs [9,15,17], studies of applicability to specific scenarios or software quality choices [13]. A comparative classification model [6] proposes to link functional and non-functional requirements to the techniques of each NoSQL family. The result is presented as a decision tree that guides the choice of the NoSQL system based on sharding, volume of data and CAP properties.

To conclude on the state of the art, existing approaches that compared RDB and NoSQL DB are basically based on technical criteria, and do not cover all the factors that can impact performance such as the data model, the type and the frequency of queries, the optimization structures, and the case of study. Model transformation approaches provide rules that transform a source model into a target model. However, these approaches do not favor the flexibility of the schema offered by NoSQL DB as a whole, and do not allow to move from one model to another, or from one NoSQL family to another *via* transformations; splitting a source concept into several target concepts, or merging several source concepts into one target concept. Thus, the flexibility of models has not been sufficiently exploited to facilitate transformations and improve performances. Finally, the few existing works on NoSQL DB choice orientation are basically based on technical criteria and do not consider data model nor functional requirements.

3 ModelDrivenGuide: Implementing Conceptual Model

We propose an approach for the generation of logical models for each family (NoSQLs and relational). Our approach is based on transformation rules starting from the conceptual model, then going from one logical model to an other by refinement. We adopt a Model-Driven Architecture[1] that offers 3 types of models, namely 1) the *Computation Independent Model* (CIM) that describes the requirements; 2) the *Platform Independent Model* (PIM) which describes the components of the system independently of platforms; 3) and the *Platform Specific Model* (PSM) which describes the components of the system using a precise technical platform. MDA also recommends transforming models by formalizing the transformation rules in a language such as the *Query-View-Transformation* (QVT)[2], which is the standard proposed by the OMG for models transformation.

3.1 Overview of Our Approach

Our approach provides a modeling framework based on those multiple dimensions of choice as illustrated in Fig. 1.

[1] MDA: https://www.omg.org/mda/index.htm.
[2] https://www.omg.org/spec/QVT/About-QVT/.

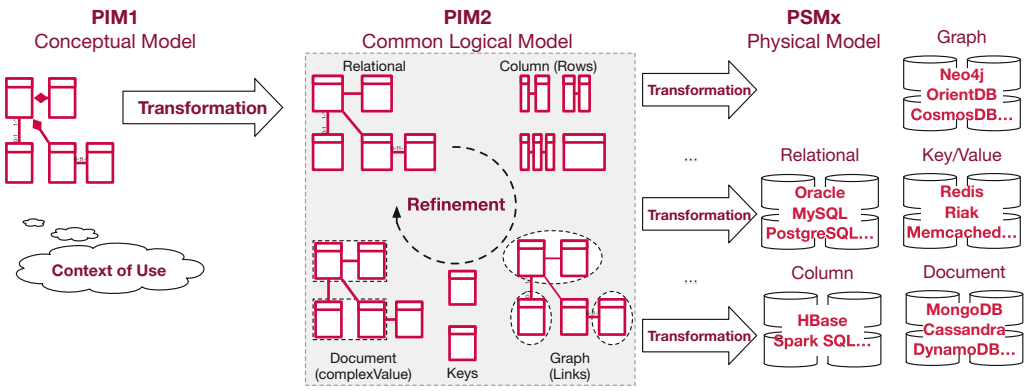


Fig. 1. `ModelDrivenGuide`: From conceptual to physical models

The **PIM1** integrates the functional needs of the IS, both in terms of data and queries. This traditional UML class diagram serves as a basis for modeling the user requirements.

The **PIM2** is the second level independent model, common to the five families of models (NoSQL & relational). It allows to carry out *refinements* by generating all the possible denormalized models using merge and split rules. To reduce the search space, a generation heuristic keeps only the effective solutions by simplification based on the use case.

The **PSMx** are obtained by the transformation of compatible PIM2 into the target data family (e.g. nesting for `DO`, rows for `CO`, edges for `GO`, etc.). Choices of sharding and indexing strategies are obtained using a generic cost model for the 5 families. All solutions can be proposed and sorted in relevance order.

This article focuses especially on the `PIM2 5Families` meta-model used to generate all possible data models, the transformation and refinement rules (between and inside meta-models) and also the heuristic that reduces the search space of PIM2 data-models by removing useless solutions. The choice of proper data-models determined by a global cost model is out of the scope of this paper.

3.2 The `5Families` Meta-Model

The common meta-model seeks to produce different schemas compatible with the PSM constraints while remaining independent. The subtlety of the common PIM2 is to integrate the 5 families of data models. A major advantage of this meta-model is that if a denormalization (or normalization) solution proves to be well suited to a case of study using a family, it may also be the suited one to the four other families. Thanks to the *refinement* rules, it will be possible to merge or split concepts to adapt them to all the data models.

Figure 2 shows the PIM2 `5Families meta-model` integrating all the concepts used in the `5Families` data models: the concepts contain rows (for column-oriented models), key-values with simple or complex values. Concepts can also be linked by edges to facilitate the integration of a graph database.

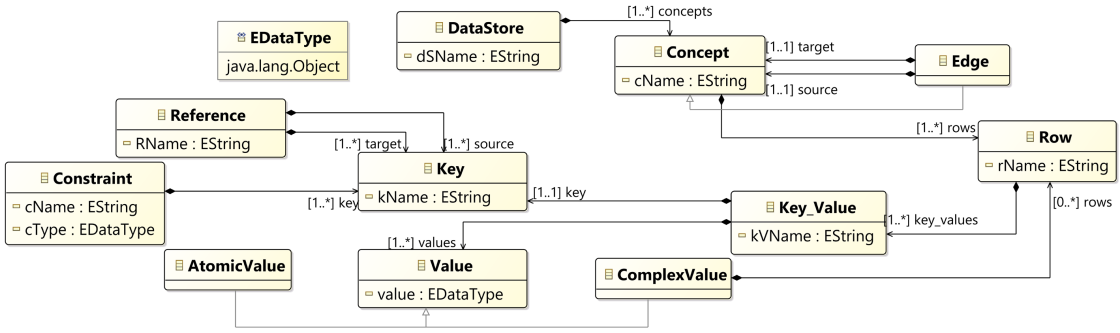


Fig. 2. PIM2 common meta-model for the 5 families of data models

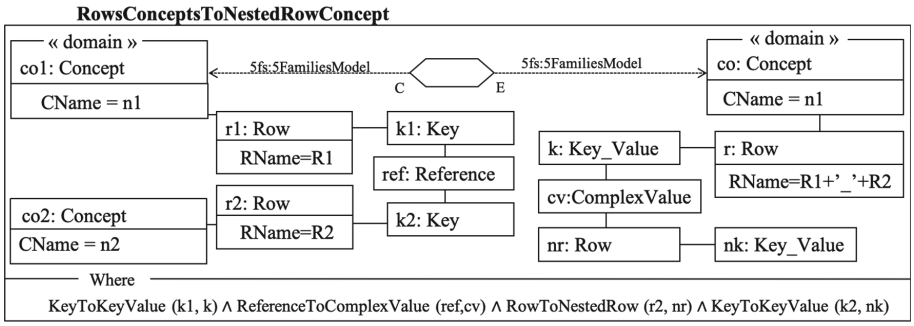


Fig. 3. PIM2 `5Families` Model Refinement - Concepts' Rows Merge QVT Rule

Complex Values are represented by the composition of rows. A loop occurs between rows and complex values, it shows that a value can be multiple, and is structurally identical to a concept. This modeling choice enables the application of refinement rules without iteration limits, allowing the production of new schemas. Meta-constraints are associated with the Complex Value to avoid the mutual composition of instances. Constraints and references are associated with keys, which can belong to distinct concepts. The idea is to be able to transform these constraints into new representations using refinement rules. For example, a *Reference* can be transformed into an *Edge* for graph-oriented models or into a *Foreign Key* in relational databases. UML classes from the PIM1 are transformed to the PIM2 using traditional mappings between concepts, keys and their values. Rules are expressed in `QVT` (for more details, see [1]).

3.3 Common Model Transformation and Refinement

The common meta-model helps to refine models iteratively while respecting the meta-model. The PIM2 refinement relies on three transformation rules:

- **Merge** *rows* between concepts to produce *complex values* for nesting (`CO`, `DO`) or to merge keys for values concatenation (`KVO`),
- **Split** *rows* to produce new columns in a same *concept* (`CO`),
- Transform *references* into the equivalent *edges* (`GO`).

At PIM2 level, models' refinement uses endogenous transformations where both the source model and the target model are conform to the same meta-model, here the `5Families` meta-model. Here, we focus on the `RowConceptsTo-NestedRowConcept` *QVT* rule (Fig. 3) that merges rows belonging to two different concepts. Thus, two rows from two different concepts referring to each other (initially classes associated at the PIM1 level) can be merged into a single row with a complex value. Row *r2* of concept *co2* is linked by reference *ref* to concept *co1* through row *r1*. To merge rows, row *r2* is then nested into a complex value *cv* which corresponds to the transformation of the reference into a new key value *k*. The latter corresponds to the transformation of the reference key *k1*. This rule can be applied in both directions by switching *co1* and *co2* (generating a list of complex values in *k*). As detailed in the following, the generation of data models will rely on those rules by applying them recursively on various combinations of transformations. Each new schema produced by QVT rules can be transformed to target PSMx.

3.4 Transformation Heuristic to Generate Data Models

Models' refinement produces plenty of new data models in which we will find the target solution. The number of generated models and solutions is huge, since splits can be applied on each key and merges can be done in several ways. We need to apply a heuristic to reduce the search space and avoid cycles. It is based on the idea of avoiding to produce the same data models with two different paths. In fact, applying splits and merges in different orders will produce the same effects on the resulting data models. Moreover, one can reverse the effect of a path and produce a cycle in the production of data models. Thus, four main rules have to be adopted and can be summarized in the following:

- A *split* cannot be applied on a *Row* with *Complex Value*s. Schemas generation should not go back after merges to avoid cycles.
- A *merge* cannot be applied between two *Row*s of the same *Concept*, which would cancel a split and generate a cycle.
- A *split* should not separate keys if queries from the use case express a link between them. This will avoid solutions which require instance reconstruction with costly joins.
- The *merge* of *Concept*s is led by the use case with queries combining *row*s of different concepts through *references*. The goal is to reduce the number of joins. It should be noted that the merge can be done in different ways.

4 Experimentations

To illustrate our approach, we used the TPC-C[3] benchmark giving a full use case mixing at the same time transactions, joins and aggregations. For the implementation of our approach, we generated different denormalized schemas that

[3] http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf.

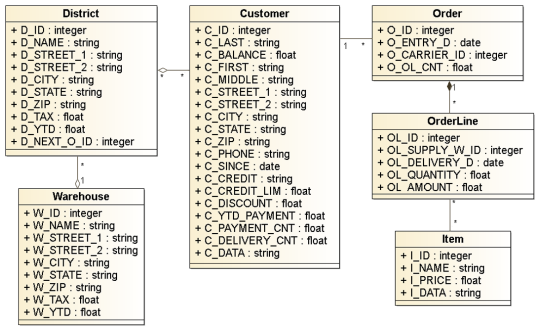


Fig. 4. PIM1 of TPC-C benchmark

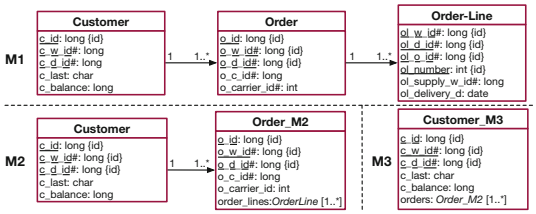


Fig. 5. The Chosen PSM documents after several PIM2 fusions of TPC-C

we have integrated into a `MongoDB` database. The choice was led by the fact that `MongoDB` is one of the rare NoSQL solutions integrating ACID transactions in *sharding* required by the TPC benchmark. All experiments were run on MongoDB 4.2 on a cluster of 30 physical servers with a 3.60 GHz quad-core processor with 16 GB memory. For distribution, each shard corresponds to a distinct server. All showed processing times correspond to the average time given by 10 runs.

The TPC-C benchmark simulates the behavior of a logistic DB on user orders with transaction-oriented stock management (OLTP). To simplify, we will focus on the three main classes (*Customer*, *Order* and *OrderLine*) as illustrated in the PIM1 depicted in Fig. 4. TPC-C covers four transactions describing the activities of a management environment: **T1** - create an order and its delivery, **T2** - update by payments, **T3** - process 10 new orders and **T4** - supply warehouses' stock monitoring. When looking for a client, he is identified by his identifier (*c_id* in T2.1 and T4.1) or by his last name (*c_last* in T2.2 and T4.2).

Our experiments focus on the three most relevant document-oriented PSM models presented in Fig. 5. They correspond to three refinements of PIM2 for which were applied: the normalized model (**M1**), a merge of *Order* & *Order-Line* concepts (**M2** by nesting the list of *Order-Line*), and a second merge of *M2* & *Customer* concepts (**M3** with a list of *Order_M2*). Sharding keys are placed on *c_id* or *o_id*, and indexes on secondary keys *c_last* or nested keys.

4.1 Data Volume Variation

The increase in volume is tested on the M1 schema (equivalence). The data is distributed over two *shards* and the volume of data varies from 5 to 100 warehouses, representing 16 GB of data. Figure 6 shows the execution time of transactions' queries T1 to T4. The T3 query witnesses a different dimension due to its costly grouping operation thus it is shown separately.

We can notice that T1, T2.2 and T4.1 give weakly growing execution times *wrt.* the volume of data as a result of a good use of the sharding key which supports well the load of small updates. However, T2.2 and T4.2 transactions show a strong increase in execution time due to the use of secondary indexes on the *c_last* key. Their low selectivity does not support the increase of volume.

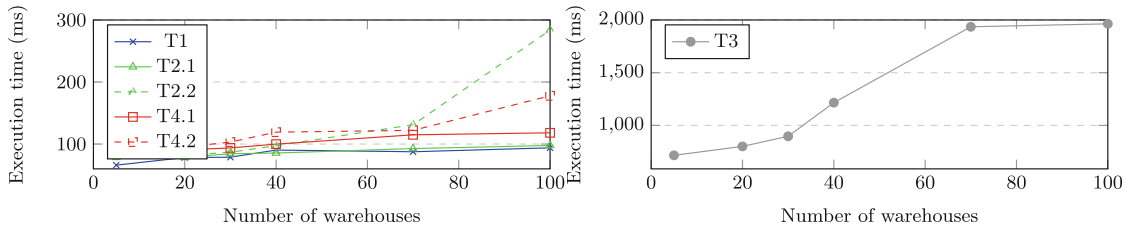


Fig. 6. Variation of data volume with M1

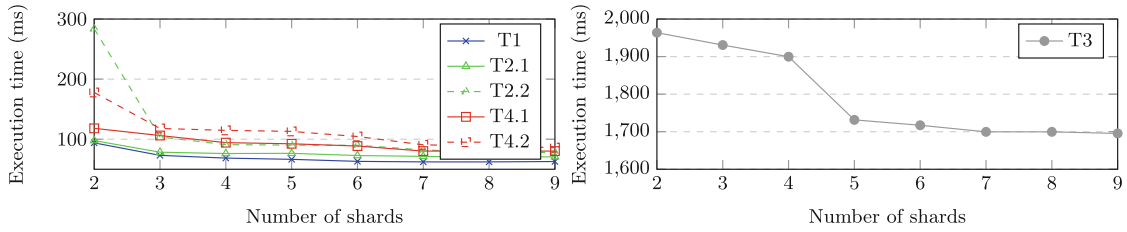


Fig. 7. Variation of the number of servers with M1 for 100 warehouses

The T3 must, on each iteration, process 10 new orders and the associated order lines. In fact, each order must be searched for, modified, and updated according to linked order lines. Thus, the volume of data has a huge impact on this heavy transaction requiring a strong monopolization of *chunks* and reducing the cache effect [14]. The threshold obtained after 70 warehouses corresponds to a null cache effect given the amount of data and the number of available *chunks*.

4.2 Number of Shards Variation

Here, we show the distribution effect by increasing the number of *shards* (nodes/servers). The M1 collection with 100 warehouses is distributed in order to define the number of servers necessary to reach convergence. Figure 7 shows the effect of parallelism until reaching stability starting from 7 *shards*. The gain is low for T1, T2.1 and T4.1 transactions (twice faster) since these are short transactions. For T2.2 and T4.2 based on *c_last*, a significant gain is obtained, since the query is carried out on smaller sets on each server (local indexes). In fact, time was divided by 4. The T3 transaction distributes updates for 10 orders and achieves a constant processing time per order (1,699 ms).

4.3 Data Models Performance

The last test case is to compare the 3 data models in order to see the impact of data modeling on the performance of the system. To do so, we used 100 warehouses on 9 *shards*. Each of the transactions was tested on the 3 data models M1, M2 and M3. Figure 8 shows the execution times for the M1 normalized model, the semi-denormalized M2 and the fully denormalized M3.

The first transaction T1 deals with order's insertion and associated order lines. In the M1 model each element is individually inserted. On the other hand,

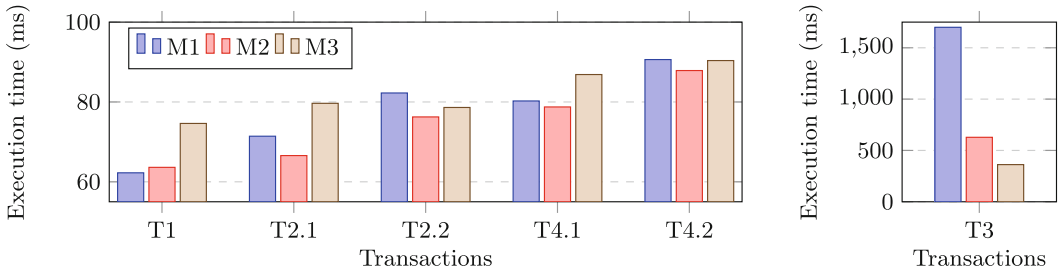


Fig. 8. Data models performance

the M2 model must gather the command lines to nest them as a block in the command, generating a delay. The M3 model generates a substantial additional cost (+20%). In fact, rewriting *chunks* force the placement of the modified document (more space), and thus adds processing time at concerned *shard*'s level.

For transaction T2.1 that consults orders and their order lines for a given customer, the M2 model is more efficient, since it merges them. The M1 model must perform the "join" during the process. However, the M3 model goes through all the orders to retrieve the last one, so it is more expensive. Transaction T2.2 searches for customers by last name. We note that the M3 model becomes more efficient because the additional cost of searching for orders is compensated by the search for the last name. This is not the case for the M1 and M2 models that must search for all of the homonyms before consulting the commands.

For T4.1 and T4.2 which update the payment of customers order, we observe a similar behavior with the additional cost of the updates of orders' "states".

Transaction T3 is very representative of the denormalization gain. Indeed, we can see the significant gain by merging concepts of −63% for M2 and −78% for M3. Indeed this transaction requires to join all concepts in order to process order lines' updates. Consequently, the normalized model M1 requires many "joins", while M2 and M3 merge order lines into orders and thus update a single document to process the transaction. M3 is slightly more efficient, as the delivery status is also notified to the customer (data linked to the document).

We conclude from above experiments, that : a) as expected, normalized standard modeling increases the number of treatments between collections, in particular on linked updates, b) complete denormalization of the schema increases the processing of a certain number of queries due to 1) generating insertions in lists, and 2) browsing lists for each consultation. However, queries requiring to link all the collections are optimal, c) the semi-denormalized data model brings a good compromise between the targeted search with and without "joins".

5 Conclusion and Future Work

In this paper, we have proposed `ModelDrivenGuide`, an IS modeling and implementation MDA approach that guides the user from conceptual model to platform technical choices. The cornerstone of our approach is the common logical meta-model that describes the 5 families of models. We have presented QVT

refinement rules, particularly for merging concepts' rows. Those rules generate optimized schema and produce potential target models. This mix between data modeling and optimization rises to an approach that aims to find the efficient target model among the 5 families of data models. A heuristic reduces the search space and targets relevant models since they rely on the IS's use case.

In future work, we wish to define a generic cost model for PIM2, allowing to compare the solutions produced. This cost model will integrate different dimensions (storage, bandwidth, CPU/energy impact, etc.). We also work on the definition of the eligibility of a PIM2 in the target PSM. Indeed, the generated models must be compatible with a dedicated PSM model and it is necessary to define mapping rules (a key & a value for `KVO`, Concepts & Links for `GO`, etc.).

References

1. Abdelhedi, F., Ait Brahim, A., Atigui, F., Zurfluh, G.: MDA-based approach for NoSQL databases modelling. In: Bellatreche, L., Chakravarthy, S. (eds.) DaWaK 2017. LNCS, vol. 10440, pp. 88–102. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64283-3_7
2. Chebotko, A., Kashlev, A., Lu, S.: A big data modeling methodology for apache cassandra. In: 2015 IEEE ICBD, pp. 238–245. IEEE (2015)
3. Daniel, G., Gomez, A., Cabot, J.: UmltoNoSQL: mapping conceptual schemas to heterogeneous datastores. In: 2019 13th International Conference on Research Challenges in Information Science (RCIS), pp. 1–13 (2019)
4. Daniel, G., Sunyé, G., Cabot, J.: UMLtoGraphDB: mapping conceptual schemas to graph databases. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) ER 2016. LNCS, vol. 9974, pp. 430–444. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_33
5. de Freitas, M.C., Souza, D.Y., Salgado, A.C.: Conceptual mappings to convert relational into NoSQL databases. In: ICEIS (1), pp. 174–181 (2016)
6. Gessert, F., Wingerath, W., Friedrich, S., Ritter, N.: NoSQL database systems: a survey and decision guidance. CSRD **32**(3–4), 353–365 (2017). https://doi.org/10.1007/s00450-016-0334-3
7. Hamouda, S., Zainol, Z.: Document-oriented data schema for relational database migration to NoSQL. In: Innovate-Data 2017, pp. 43–50. IEEE (2017)
8. Hanine, M., Bendarag, A., Boutkhoum, O.: Data migration methodology from relational to NoSQL databases. Int. J. IJECE **9**(12), 2369–2373 (2016)
9. Jatana, N., Puri, S., Ahuja, M., Kathuria, I., Gosain, D.: A survey and comparison of relational and non-relational database. Int. J. IJERT **1**(6), 1–5 (2012)
10. Lee, C.H., Zheng, Y.L.: SQL-to-NoSQL schema denormalization and migration: a study on content management systems. In: SMC 2015, pp. 2022–2026 (2015)
11. Li, C.: Transforming relational database into HBase: a case study. In: ICSESS 2010, pp. 683–687. IEEE (2010)
12. Li, Y., Gu, P., Zhang, C.: Transforming UML class diagrams into HBase based on meta-model. In: ISEEE 2014. vol. 2, pp. 720–724. IEEE (2014)
13. Lourenço, J.R., Cabral, B., Carreiro, P., Vieira, M., Bernardino, J.: Choosing the right NoSQL database for the job: a quality attribute evaluation. J. Big Data **2**(1), 1–26 (2015). https://doi.org/10.1186/s40537-015-0025-0
14. du Mouza, C., Travers, N.: Relevant filtering in a distributed content-based publish/subscribe system. NoSQL Models Trends Challenges **1**, 203–244 (2018)

15. Raut, A.: NoSQL database and its comparison with RDBMS. Int. J. Comput. Intell. Res. **13**(7), 1645–1651 (2017)
16. Rocha, L., Vale, F., Cirilo, E., Barbosa, D., Mourão, F.: A framework for migrating relational datasets to NoSQL. Procedia Comput. Sci. **51**, 2593–2602 (2015)
17. Tauro, C.J., Aravindh, S., Shreeharsha, A.: Comparative study of the new generation, agile, scalable, high performance NOSQL databases. IJCA **48**(20), 1–4 (2012)
18. Vajk, T., Fehér, P., Fekete, K., Charaf, H.: Denormalizing data into schema-free databases. In: CogInfoCom 2013, pp. 747–752. IEEE (2013)