Roundtrip engineering of NoSQL databases

Jacky Akoka[*,a], Isabelle Comyn-Wattiau[b]

[a] CEDRIC-CNAM & TEM-Institut Mines Telecom, Paris, France
[b] ESSEC Business School, Cergy-Pontoise, France

Abstract. *In this article we present a framework describing a roundtrip engineering process for NoSQL database systems. This framework, based on the Model Driven Engineering approach, is composed of a knowledge base guiding the roundtrip process. Starting from a roundtrip generic scenario, we propose several roundtrip scenarios combining forward and reverse engineering processes. We illustrate our approach with an example related to a property graph database. The illustrative scenario consists of successive steps of model enrichment combined with forward and reverse engineering processes. Future research will consist in designing and implementing the main components of the knowledge base.*

Keywords. Roundtrip Engineering • Forward Engineering • Reverse Engineering • Roundtrip Process • Knowledge Base • NoSQL Database

1 Introduction

As stated by (Mayr et al. 2017), "models are the fundamental human instruments for managing complexity and understanding" . Model driven engineering (MDE) is considered as a methodology providing several benefits such as an improved code quality and a better traceability. It consists of the application of models to increase the level of abstraction required to develop and evolve software products. Its aim is to offer software development approaches in which abstract models of software systems are created and transformed facilitating their implementations. MDE is based on model transformation which takes one or more source models and transform them into one or more target models.

Roundtrip engineering (RTE) represents one facet of MDE. Since code and model are interrelated, changing code will change the model and vice versa. RTE can be considered as a way to improve the software engineering process. It consists mainly of forward engineering and reverse engineering. Forward engineering transforms conceptual models into source code. With reverse engineering, the source code is transformed back into conceptual models. The combination of the two paths leads to roundtrip engineering, keeping the two views consistent (Booch et al. 1998). Demeyer et al. (1999) defines RTE as the seamless integration between design diagrams and source code, between modeling and implementation. Therefore the aim of RTE is to enable a homogeneous integration between the design and the implementation phases. Code generation, described as a ***push method***, is obtained using forward engineering. The transformation of the source code into a conceptual model is obtained by a reverse engineering process based on a ***pull method***. Round-trip engineering corresponds to a ***push-pull method***.

RTE has been first used with UML. It has been extended to other technologies such as graphical user interface design, database design, and to other software modeling artifacts. RTE is different from the addition of forward and reverse engineering. Optimizing forward and reverse engineering leads to incremental transformation. Only the changed modules are transformed, rather than all artifacts.

* Corresponding author.
E-mail. jacky.akoka@lecnam.net

The RTE process preserves the information in the target artifact on the return trip. RTE tends not only to transform models but also to reconcile them. It automatically maintains the consistency of changing software artifacts. In other words, changes made to one model are propagated and reflected in another model using model transformation technologies (Sendall and Kozaczynski 2003; Czarnecki and Helsen 2003). RTE is a solution enabling the synchronization of models by keeping them consistent, thus maintaining conceptual-implementation mappings under evolution. RTE focuses mainly on synchronization. According to (Sendall and Kozaczynski 2003), RTE can be divided into three steps:

- Deciding whether a model under consideration has been changed,
- Deciding whether the changes cause any inconsistencies with the other models, and
- Once inconsistencies have been detected, updating the other models so that they become consistent again.

RTE is supported by methodologies and tools. However most development tools only offer very limited support. This is most probably a consequence of the difficulty in keeping multiple changing artifacts consistent.

The main advantage of RTE is that the design and implementation artifacts are automatically synchronized all the time (Kellokoski 2000). RTE promotes design-led development and improves design traceability since it enables an automatic generation of source code from conceptual models and automatic generation of the latter from source code. One major advantage is a short time to market and an improved quality of the software product. RTE improves the software process as well as its automation. Several qualities are expected from RTE approaches such as the ability to manage trace information and to facilitate the detection of conflicts between RTE activities.

So far, there has been very little research on roundtrip engineering of NoSQL database design. The aim of this paper is to start filling this gap. In particular, we propose a framework facilitating the roundtrip process and we derive requirements that this framework implies. Thus, it can be seen as a roadmap for a roundtrip engineering process of NoSQL databases. This article is organized as follows: The following section presents the state of the art related to RTE. Our framework is described in Section **??**. We then show the results of its application using an illustrative scenario in Section **??**. We derive our framework in Section **??**. We finally indicate in Section **??** some conclusions and future work.

2 Related work

As we pointed out in the introduction, the basis of RTE is a clear definition of required consistency between the models. Therefore issues in RTE are closely related to those of consistency management. The latter is a technique for ensuring that models are consistent (Sendall and Kozaczynski 2003). The methodology presented in (Engels et al. 2001; Küster 2004) can be applied to define consistency for a given set of UML models. (Aßmann 2003) introduces mathematical definitions for RTE. The Fujaba System (Nickel et al. 2000) supports RTE for class diagrams. The CODEX system (Larrson and Burbeck 2003) aims at keeping a model consistent with a set of views on the model. The authors propose a RTE which essentially synchronizes models by keeping them consistent for maintaining conceptual-relational mappings. The added value of (Ciccozzi et al. 2011) is to ensure that extra-functional concerns modeled at design level are preserved at code execution level. They introduce a back annotation model containing information related both to traceability and monitoring results. (Bork et al. 2008) describe an approach towards model and source code RTE. The approach is based on reverse engineering of model-to-transformation (M2T) templates. They use (customizable) code generation templates as a grammar to parse the generated (and later modified) code. (Greiner et al. 2016) propose an RTE approach requiring the specification of QVT-R rules that relate two elements of

the respective meta-models. (Angyal et al. 2008) present an approach for model and code RTE based on differencing and merging abstract syntax trees (AST). (Antkiewicz and Czarnecki 2006) propose an RTE approach based on framework-specific modeling languages. (Hettel et al. 2009) propose an approach towards model RTE based on abductive logic programming. (Macedo and Cunha 2016) proposed an idea on how to circumvent some problems that are related to a QVT-R script by using the language Alloy with their tool Echo. In (Buchmann and Westfechtel 2013), the authors examine a standard use case: incremental round-trip engineering between design models and source code. More specifically, they address the coupling between a UML class diagram and a Java source code. Both model and code may be edited concurrently, and changes may be propagated back and forth to maintain consistency.

Although there are a number of round trip engineering tools available, only a few of them have been adopted by the developers' community. (Nagowah et al. 2013) present a state of the art of these tools, including (Borland Software Solutions 2009; Rational Software 2006; ArgoUML (Odutola et al. 2001); Gentleware AG Poseidon For UML (Boger et al. 2007); JBoss Seam (Orshalick and Assar 2010); Spring Web MVC framework (Winterfeldt 2012); AndroMDA 2012).

RTE is only one aspect of MDE. A general definition of the latter is given by (Hailpern and Tarr 2006). A characterization of MDE can be found in (Ruiz et al. 2017). In the context of MDE, model transformation plays an essential role. It defines transformations rules between a source and a target metamodel (Czarnecki and Helsen 2006). Model transformation includes code generation (Kleppe et al. 2003), models synchronization, in particular at the same or at different levels of abstraction (Ivkovic and Kontogiannis 2004), model evolution (Zhang et al. 2005) and reverse engineering from physical and/or logical levels to conceptual levels and vice versa (Favre 2004).

As it can be seen from this literature review, to the best of our knowledge, there is no approach related to RTE of NoSQL databases. This is precisely the aim of our approach described below.

3 Toward roundtrip engineering of NoSQL databases

It is generally admitted that NoSQL databases offer a flexible and a scalable solution to store and query structured, semi-structured and unstructured data. These databases offer a high level of query performance. They can be designed to meet the requirements of BI applications and analytics. Since they become more and more mature, they have been adopted by many companies. Maintaining these databases require methodologies offering some consistencies between their design models and their implementation. MDE represents such a methodology. It describes a system under consideration by means of high-level models. The latter are refined into low-level models until their level of detail is conform to the underlying platform. A model transformation is the process of mapping one input model into an output model. This transformation process requires the specification of transformation rules referring to metamodels. Therefore, a model transformation defines a set of rules to be applied between source and target metamodels. Transformations can be either unidirectional or bidirectional. Unidirectional transformations allow to map source metamodels to target metamodels, but not the other way around. Bidirectional transformations define mappings in both directions. This bidirectional transformation between meta-models is the main principle underlying roundtrip engineering. The goal of the latter is to keep a set of related metamodels synchronized. Whenever a change is applied to one metamodel, the other metamodels need to be adjusted to restore a consistent state.

For a specific application, roundtrip engineering is a method which allows the automatic synchronization of the source code after modification of the conceptual and/or the logical model and vice versa. Roundtrip engineering allows a bi-transformation between the model and the source code. Besides, roundtrip engineering is a way to

optimize the corrective, adaptive, evolutionary or perfective maintenance of applications. Indeed, anomalies can be detected, functionalities added and/or removed, new tests performed, and migrations to new platforms required during the life cycle of the application.

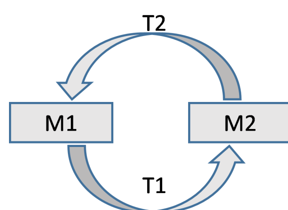

Figure 1: Roundtrip generic scenario

Our roundtrip framework is composed of different transformation rule sets which are all based on a common generic scenario (Figure 1) in which two metamodels (M1 and M2) are connected by two sets of transformation rules (T1 and T2). M1 and M2 can be conceptual, logical or physical metamodels. All possible combinations of roundtrip scenarios are shown in Figure **??**.

M1 \ M2	**Conceptual**	**Logical**	**Physical**
Conceptual	Case 1: Translation	Case 2: Forward Engineering	*Case 3: Forward Engineering*
Logical	Case 4: Reverse Engineering	Case 5: Migration	Case 6: Forward Engineering
Physical	*Case 7: Reverse Engineering*	Case 8: Reverse Engineering	Case 9: Migration

Figure 2: Characterizing roundtrip steps

Forward engineering occurs each time M2 has a lower level of abstraction than M1. Three different cases are to be considered:

- Case 2: M1 and M2 are respectively conceptual and logical metamodels.
- Case 3: M1 and M2 are respectively conceptual and physical metamodels. This is a case of forward engineering. However skipping the logical intermediate level can cause errors and make maintenance of the resulting systems more difficult.
- Case 6: M1 and M2 are respectively logical and physical metamodels.

Reverse engineering is encountered when M2 has a higher level of abstraction than M1. Three different cases occur:

- Case 8: M1 and M2 are respectively physical and logical metamodels.
- Case 7: M1 and M2 are respectively physical and conceptual metamodels. Although it is undesirable to do so, it is nonetheless a case of a possible reverse engineering. Once again, skipping the logical step can lead to inconsistent results.
- Case 4: M1 and M2 are respectively logical and conceptual metamodels.

Mapping a conceptual metamodel M1 into a conceptual metamodel M2 is required when two different conceptual formalisms coexist (Case 1). This the case when, for example, M1 is an Extended Entity Relationship metamodel and M2 is a UML metamodel. When both formalisms are equally expressive, T1 and T2 are reversible transformation rules. This mapping is classical in database engineering and does not present any specific challenge in the context of NoSQL systems.

Finally, two cases deal with *migration* of databases, either at a logical level (Case 5) or at a physical level (Case 9). For example, migrating from Neo4j to OrientDB is an example of physical migration whereas migrating from graph database to relational database illustrates a logical migration.

Roundtrip engineering may encompass several combinations of the cases described above. The generic roundtrip engineering problem may be reduced to the paths described at Figure **??**, without loss of generality. The process also contains enrichment of models due to new requirements and/or reengineering of models.

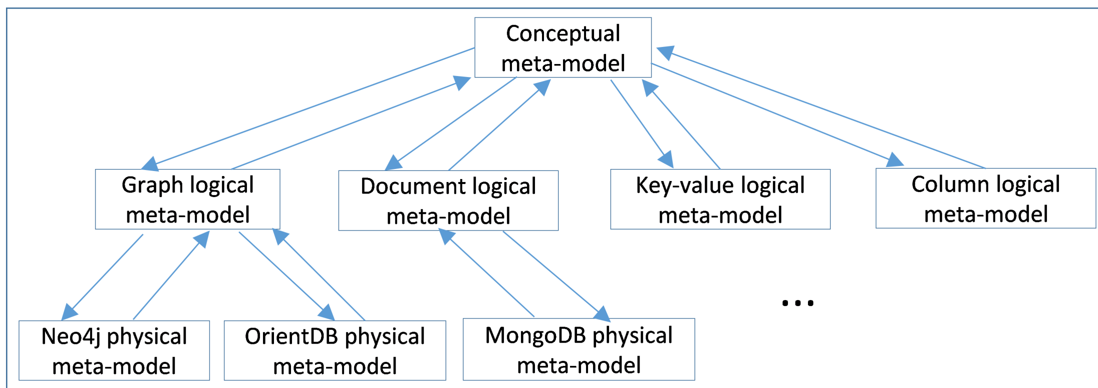

Figure 3: Recommended paths for NoSQL database roundtrip engineering

Each arrow refers to either Case 2, Case 4, Case 6, or Case 8. We don't represent the arrows corresponding to Case 1 since it is not specific to NoSQL systems. We also don't take into account Cases 3 and 7 since they skip the logical level, which is not considered as a good practice. Finally, we don't represent Cases 5 and 9 since we recommend always to maintain the consistency between levels. Migrating from a logical model to another one requires defining a conceptual intermediate step. In the same way, migrating from a physical model to another one requires at least defining a logical intermediate step and, if needed, a subsequent conceptual step.

Even if there is no roundtrip engineering approach for NoSQL databases, nevertheless the literature contains several contributions corresponding either to one arrow of Figure **??** or to two consecutive arrows constituting an acyclic path. Moreover, existing approaches only define transformation rules but don't take into account the additional information required to anticipate the roundtrip paths.

4 Illustrative scenario

In this section, we illustrate our framework with an example. Let's consider the following conceptual model representing information about publications (Fig. 4), taken from (Akoka et al. 2017). Entities contain information about scientific papers, their sources (journals, conferences), their authors, and their affiliations. Let us note the reflexive citation relationship between papers. Terms are keywords characterizing papers. We also added a reviewer entity and a researcher entity. Author and reviewer are subtypes of researcher entity. We suppose that an author may have several affiliations but when he/she publishes a paper, he/she has to declare a unique affiliation. Thus, we represent a ternary relationship between papers, authors, and affiliations.

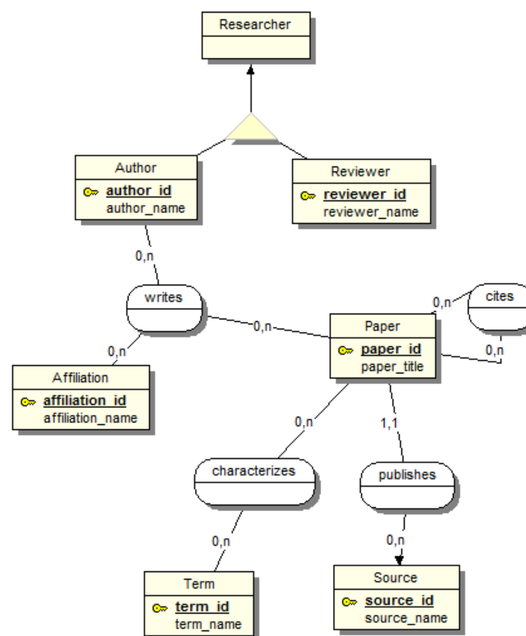

Figure 4: Example of conceptual model (Akoka et al. 2017)

A first step performing a forward engineering process is described as follows. A set of transformation rules (described in (Akoka et al. 2017)) allowed us to generate a logical property graph database (Figure **??**). As an example of a rule, the ternary relationship *writes* in the conceptual model becomes a node of the graph. This node is connected to the three nodes resulting from the transformation of the three entities involved in the relationship. As an additional information, our approach provides also the logical graph with its estimated size (volume attribute). Another set of rules leads to a physical Neo4j graph. In addition, the information regarding the size enables the generation of a Neo4j test database containing as many nodes and edges as estimated by the designer.

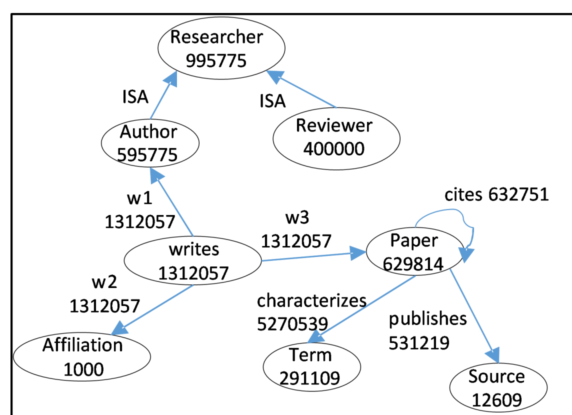

Figure 5: Resulting graph logical model (Akoka et al. 2017)

Let us suppose that the database administrator received additional data describing papers such as year of publication, number of pages, and language. Through a reverse engineering process, this information is propagated into the logical graph and to the subsequent conceptual model. We performed this reverse process by applying the set of rules described in (Comyn-Wattiau and Akoka 2017). The final result is presented at Figure **??**.

Let us consider that the end users then asked to enrich the database with historical information on past affiliations of researchers. This led us to the model of Figure **??** where the relationship *mentions* links researchers, affiliations, and dates. A new forward engineering process containing two steps allowed us to generate the updated Neo4j database.

Finally, let us suppose that the end users would like also to automatically access not only to the information on papers but also to the full text papers. Neo4j is not able to store documents. Thus the database administrator proposes to migrate the database toward an OrientDB environment. OrientDB combines graph and document logical models. The resulting logical schema is presented at Figure **??**. At the physical level, the nodes representing paper information are linked to documents storing full-text papers, but this is not visible at the logical level. Moreover, to maintain the consistency with different modeling levels, we have to generate the updated conceptual model of Figure 9 mentioning the full-text attribute.

We summarize the main steps of our illustrative scenario at Figure **??**. It consists mainly of successive steps of model enrichment, forward engineering, and reverse engineering steps. Let's compare the initial conceptual model of Figure **??** and the final one of Figure **??**. The different enrichments allowed us to complete the description of conceptual objects (entities or relationships) and/or to add new objects. Moreover, the successive execution of forward and reverse engineering processes are not all inverse transformations. In our example, the main difference lies in the ternary *writes* relationship which is transformed into a node and, conversely, becomes an entity with three binary relationships respectively with author, affiliation and paper entities, which is less expressive. A round-trip engineering process must be able to keep trace of the forward transformation in order to be able to reverse it. If this trace mechanism is not available, a quality analysis of the conceptual model generated through the reverse process should be able to detect the semantic poverty of the *writes* entity and its associated relationships *w1*, *w2*, *w3*. This quality analysis may suggest to the designer to provide a better naming of such objects.

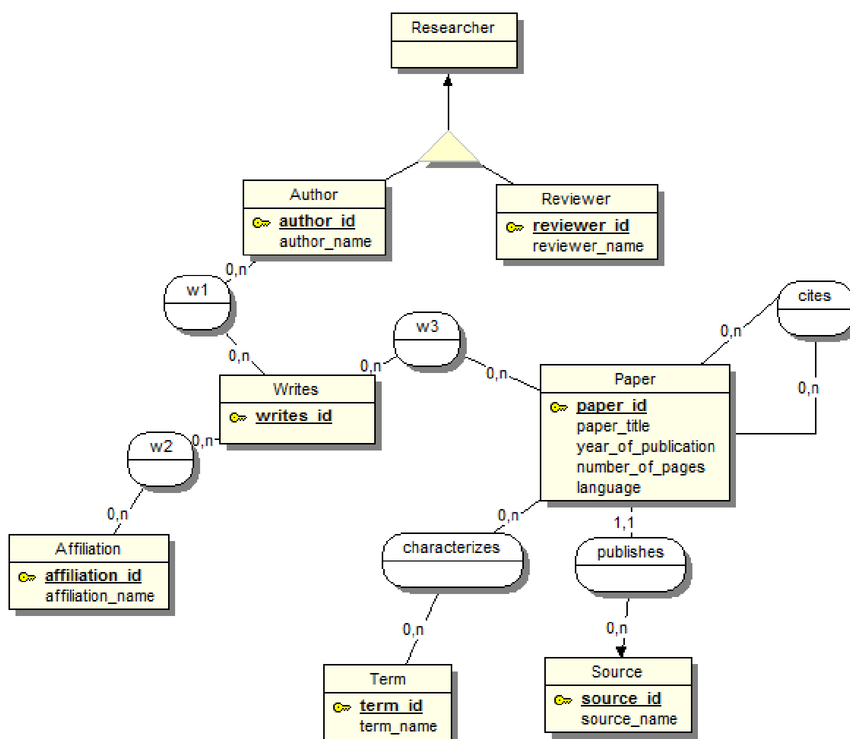

Figure 6: New conceptual model (reversed)

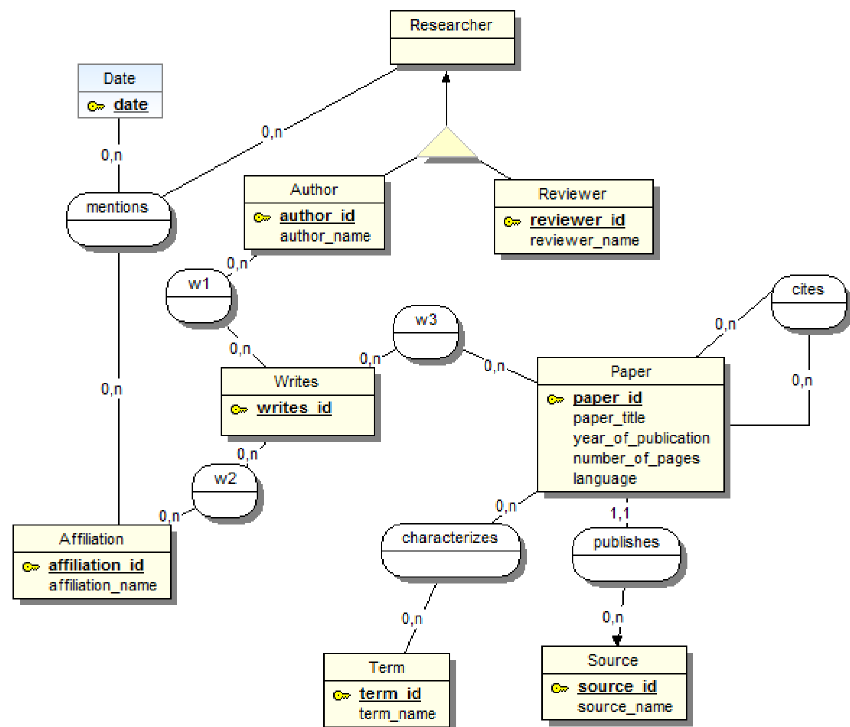

Figure 7: Enriched conceptual model

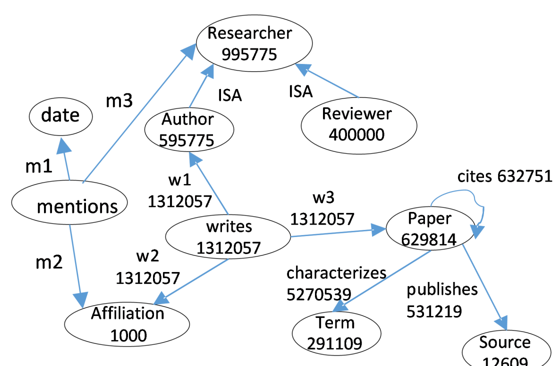

Figure 8: Updated logical graph model

5 The resulting framework

Like any software system, NoSQL databases require maintenance efforts. Software maintenance usually is categorized into four types. Corrective maintenance is dedicated to fix problems. Adaptive maintenance includes modifications applied to keep the software up-to-date and aligned with its environment. Perfective maintenance takes into account new user requirements. Finally preventive maintenance occurs when organizations anticipate future problems.

Corrective maintenance impacts the physical level of the database. Such modification must be propagated at both the logical and the conceptual levels through a reverse process. At each abstraction level, a quality analysis may lead to a roundtrip engineering path composed successively of Case 8, Case 4, Case 2, and Case 6.

Adaptive maintenance may, for example, lead to the migration of a database due to a new release of its platform. In such a case, the editors generally ensure an ascending compatibility, limiting the impact to the physical level.

Perfective maintenance occurs each time new user requirements have to be considered. It should be a classical forward engineering process propagating the new requirements from the conceptual level to the logical and physical levels. However, if the only artifact is the code and/or if previous changes have been integrated at the physical level without performing a backward propagation, a roundtrip engineering path composed successively of Case 8, Case 4, Case 2, and Case 6 must be executed.

Preventive maintenance takes place in many companies where relational databases reach their limits in their capability to meet the volume and/or variety requirements. A migration to a NoSQL system (Case 5) is necessary. In such a situation, the recommended path is composed successively of Case 4, Case 2, and Case 6. Another preventive maintenance happens when non-functional requirements (security, performance, etc.) are no longer met. A migration to another physical environment (Case 9) may be the solution. A path composed successively of Case 8 and Case 6 is recommended.

Roundtrip engineering systematically propagates changes forward or backward. Although this appears to be an additional task, it enables anticipating and facilitating any changes that the maintenance of the system, whatever it may be (corrective, adaptive, perfective, or preventive), requires.

Implementing a roundtrip engineering process therefore requires the design of a knowledge base that combines rule sets, trace templates, and quality assessment models. Some of the latter have been developed for relational databases. Much remains to be done in the field of NoSQL databases. The framework sketched at Figure **??** summarizes our approach.

The first step consists in taking into account maintenance demands and qualifying them. A guiding step then recommends a RTE scenario. The latter is implemented using the knowledge base. It is followed by a quality analysis allowing the software engineer to commit the changes or to reiterate the process.

6 Conclusion and further research

Roundtrip engineering allows us to perform the transformations imposed by different types of software maintenance. In this paper, we studied the case of NoSQL database maintenance. We

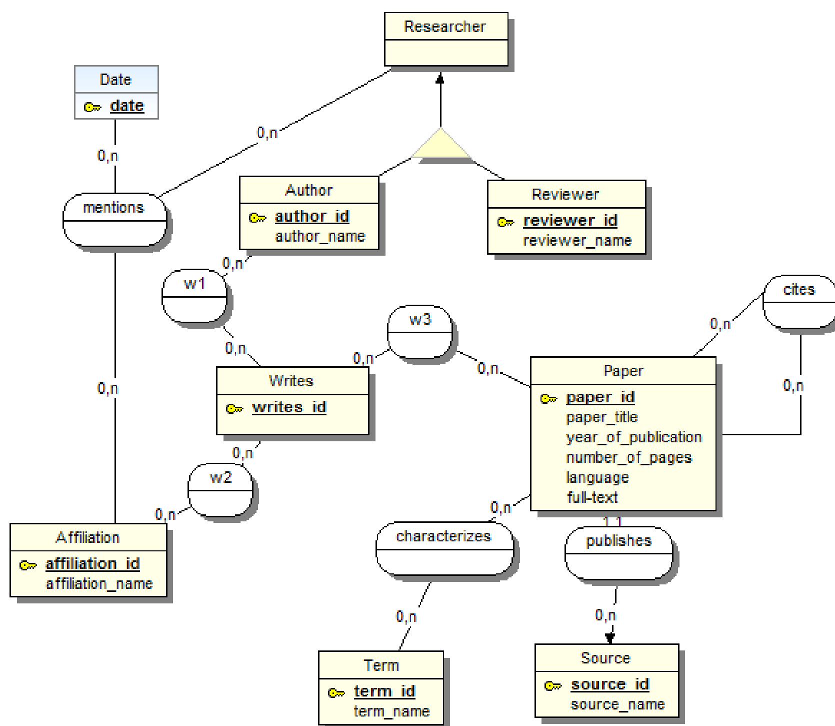

Figure 9: Final conceptual model

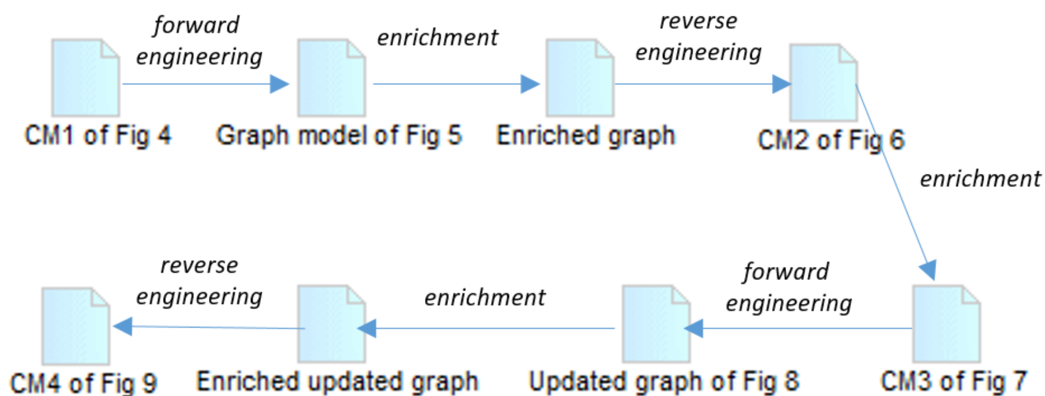

Figure 10: The illustrative process

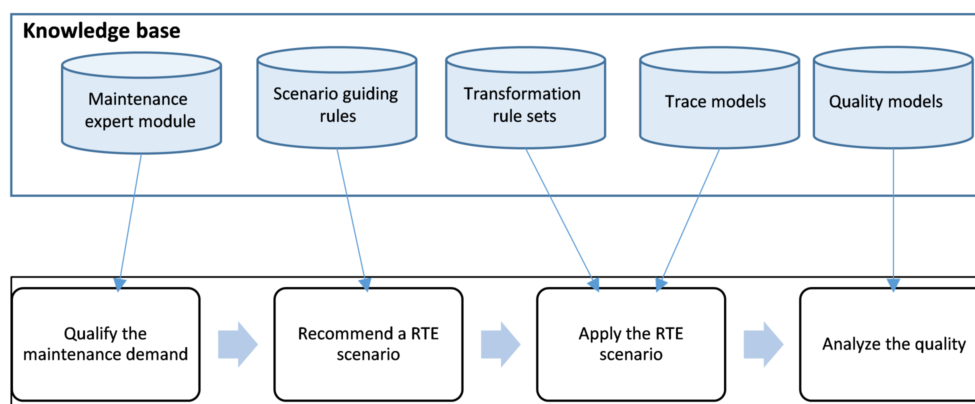

Figure 11: Final conceptual model

proposed a framework dedicated to roundtrip engineering of NoSQL databases. The framework encompasses a knowledge base and a guiding process. The knowledge base is composed of four main modules: a maintenance expert module, a set of guiding rules, several transformation rule sets, trace models, and quality evaluation models. The RTE process starts by qualifying the maintenance demand, then recommends a scenario, applies it and evaluates the resulting quality.

Starting from a roundtrip generic scenario, we propose several roundtrip scenarios combining forward and reverse engineering processes. We demonstrate the feasibility of our approach with an illustrative scenario related to a property graph database. The illustrative scenario consists of successive steps of model enrichment combined with forward and reverse engineering processes.

Future research will consist in designing and implementing the main components of the knowledge base. In order to validate the approach, we will test the resulting prototype with several case studies. Some components of the framework will reuse the results obtained for relational database systems while others will be created ex nihilo.

We dedicate this article to Heinrich Mayr for his fruitful contributions to conceptual modeling.

References

Akoka J., Comyn-Wattiau I., Prat N. (2017) A Four V's Design Approach of NoSQL Graph Databases In: Advances in Conceptual Modeling: ER 2017 Workshops AHA, MoBiD, MREBA, OntoCom, and QMMQ de Cesare S., Frank U. (eds.) Springer International Publishing, pp. 58–68 https://doi.org/10.1007/978-3-319-70625-2_6

AndroMDA. http://www.andromda.org/. Last Access: Accessed: 2017-12-26

Angyal L., Lengyel L., Charaf H. (2008) A Synchronizing Technique for Syntactic Model-Code Round-Trip Engineering. In: 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008), pp. 463–472

Antkiewicz M., Czarnecki K. (2006) Framework-Specific Modeling Languages with Round-trip Engineering. In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems. MoDELS'06. Springer-Verlag, Genova, Italy, pp. 692–706 http://dx.doi.org/10.1007/11880240_48

Aßmann U. (2003) Automatic Roundtrip Engineering. In: Electronic Notes in Theoretical Computer Science 82(5) SC 2003, Workshop on Software Composition (Satellite Event for ETAPS 2003), pp. 33–41

Boger M., Graß E., Köster M. (2007) Poseidon for UML Users Guide

Booch G., Rumbaugh J., Jacobson I. (1998) The Unified Modeling Language User Guide. Addison-Wesley

Bork M., Geiger L., Schneider C., Zündorf A. (2008) Towards Roundtrip Engineering - A Template-Based Reverse Engineering Approach In: Model Driven Architecture – Foundations and Applications: 4th European Conference, ECMDA-FA 2008 Schieferdecker I., Hartman A. (eds.) Springer Berlin Heidelberg, pp. 33–47 https://doi.org/10.1007/978-3-540-69100-6_3

Borland Software Solutions (2009) Borland Together

Buchmann T., Westfechtel B. (2013) Towards Incremental Round-Trip Engineering Using Model Transformations. In: 39th Euromicro Conference on Software Engineering and Advanced Applications, pp. 130–133

Ciccozzi F., Cicchetti A., Sjodin M. (2011) Towards a Round-Trip Support for Model-Driven Engineering of Embedded Systems. In: 37th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 200–208

Comyn-Wattiau I., Akoka J. (2017) Model-Driven Reverse Engineering of NoSQL Property Graph Databases. In: Big Data 2017: IEEE International Conference on Big Data. IEEE, pp. 453–457

Czarnecki K., Helsen S. (2003) Classification of Model Transformation Approaches. In: Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, USA

Czarnecki K., Helsen S. (2006) Feature-based survey of model transformation approaches. In: IBM Systems Journal 45(3), pp. 621–645

Demeyer S., Ducasse S., Tichelaar S. (1999) UML shortcoming for coping with round-trip engineering. In: UML'99 Conference Proceedings. Springer-Verlag

Engels G., Küster J. M., Heckel R., Groenewegen L. (2001) A Methodology for Specifying and Analyzing Consistency of Object-oriented Behavioral Models. In: Proceedings of the 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9). ACM, Vienna, Austria, pp. 186–195 http://doi.acm.org/10.1145/503209.503235

Favre J. M. (2004) CaCOphoNy: metamodel-driven software architecture reconstruction. In: 11th Working Conference on Reverse Engineering, pp. 204–213

Greiner S., Buchmann T., Westfechtel B. (2016) Bidirectional transformations with QVT-R: A case study in round-trip engineering UML class models and java source code. In: 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 15–27

Hailpern B., Tarr P. (2006) Model-driven development: The good, the bad, and the ugly. In: IBM Systems Journal 45(3), pp. 451–461

Hettel T., Lawley M., Raymond K. (2009) Towards Model Round-Trip Engineering: An Abductive Approach In: Theory and Practice of Model Transformations: Second International Conference (ICMT 2009) Paige R. F. (ed.) Springer Berlin Heidelberg, pp. 100–115 https://doi.org/10.1007/978-3-642-02408-5_8

Ivkovic I., Kontogiannis K. (2004) Tracing evolution changes of software artifacts through model synchronization. In: 20th IEEE International Conference on Software Maintenance. IEEE, pp. 252–261

Kellokoski P. (2000) Round-trip Engineering. MA thesis, University of Tampere, Finland

Kleppe A. G., Warmer J., Bast W. (2003) MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc.

Küster J. (2004) Consistency Management of Object-Oriented Behavioral Models. PhD thesis, University of Paderborn, Germany, pp. 1–306

Larrson H., Burbeck K. (2003) CODEX – An Automatic Model View Controller Engineering System.. Proceedings of Workshop Model Driven Architecture, Foundations and Applications, CTIT Technical Report TR-CTIT—03-27, University of Twente

Macedo N., Cunha A. (2016) Least-change bidirectional model transformation with QVT-R and ATL. In: Software & Systems Modeling 15(3), pp. 783–810 https://doi.org/10.1007/s10270-014-0437-x

Mayr H. C., Michael J., Ranasinghe S., Shekhovtsov V. A., Steinberger C. (2017) Model Centered Architecture In: Conceptual Modeling Perspectives Cabot J., Gómez C., Pastor O., Sancho M. R., Teniente E. (eds.) Springer International Publishing, pp. 85–104 https://doi.org/10.1007/978-3-319-67271-7_7

Nagowah L., Goolfee Z., Bergue C. (2013) RTET - A round trip engineering tool. In: International Conference of Information and Communication Technology (ICoICT), pp. 381–387

Nickel U., Niere J., Wadsack J., Zündorf A. (2000) Roundtrip Engineering with FUJABA. In: Ebert J., Kullbach B., Lehner F. (eds.) Proceedings of 2nd Workshop on Software-Reengineering (WSR), Bad Honnef, Germany, Fachberichte Informatik, Universität Koblenz-Landau

Odutola K., Oguntimehin A., Tolke L., Wulp M. (2001) ArgoUML Quick Guide

Orshalick J., Assar N. (2010) JBoss Seam: Agile Ria Development Framework, Red Hat Inc

Rational Software (2006) IBM Rational Rose – Data Sheet

Ruiz F. J. B., Molina J. G., García O. D. (2017) On the application of model-driven engineering in data reengineering. In: Information Systems 72(Supplement C), pp. 136–160

Sendall S., Kozaczynski W. (2003) Model transformation: the heart and soul of model-driven software development. In: IEEE Software 20(5), pp. 42–45

Winterfeldt D. (2012) JSpring by Example, Version 1.2.1

Zhang J., Lin Y., Gray J. (2005) Generic and Domain-Specific Model Refactoring Using a Model Transformation Engine In: Model-Driven Software Development Beydeda S., Book M., Gruhn V. (eds.) Springer Berlin Heidelberg, pp. 199–217 https://doi.org/10.1007/3-540-28554-7_9

This work is licensed under a Creative Commons ‘Attribution-ShareAlike 4.0 International’ licence.