Model-Driven NoSQL Data Engineering

Diego Sevilla Ruiz, Severino Feliciano Morales, and Jesús García Molina

Faculty of Computer Science, University of Murcia
Campus Espinardo, Murcia, Spain
{dsevilla,severino.feliciano,jmolina}@um.es

Abstract. While the concept of database schema plays a central role in relational database systems, most NoSQL systems do not require having to formally define an schema. Instead, it is implicit in the stored data. This lack of schema definition offers a greater flexibility. More specifically, schemaless databases ease both the recording of non-uniform data and data evolution. However, this comes at the cost of losing some of the benefits provided by schemas, for instance, static checking that assure that stored data conforms to the database schema. We have started a research work aimed at inferring schemas from NoSQL databases, with the purpose of building database utilities able of automating tasks such as data validation, schema visualization, and data migration. This work has evidenced the benefits of using MDE techniques within the new "NoSQL Data Engineering" field.

Keywords: NoSQL Databases, Schema Inference, Model-Driven Data Reverse Engineering, JSON Schema

1 Introduction

The lack of an explicit data schema is probably the most attractive NoSQL feature for database developers. While relational systems require the definition of the database schema in order to determine the data organization, in NoSQL databases data is stored without the need of having a previously defined schema. Being schemaless, a larger flexibility is provided: databases can store data with different structure for the same entity type (non-uniform data), and data evolution is favoured, due to the lack of restrictions imposed on the data structure. However, removing the need of declaring explicit schemas does not have to be confused with the absence of a schema, since a schema is implicit into data and database applications. Developers must always keep in mind the schema when writing database applications. This is an error-prone task, more so when the existence of several versions of each entity is possible. Therefore, the idea is emerging of combining a schemaless approach with mechanisms (e.g. data validations against schemas) that guarantee a correct access to data [3]. On the other hand, some NoSQL database tools and utilities need to know the schema to offer functionality such as performing SQL-like queries or automatically migrating data. A growing interest in managing explicit NoSQL schemas is therefore arising [1,9,6,4].

A schema of an aggregate-oriented data model is basically formed by a set of entities connected through two types of relationships: aggregation and reference. Each entity will have one or more fields that are specified by its name and its data type. Several versions of an entity can exist given the non-uniformity of the data, and database evolution.

We have defined an MDE approach that implements a reverse engineering strategy to infer the implicit schema in NoSQL databases, and uses the inferred schemas to build database utilities. The main novelty of our work is discovering all the versions of the inferred entities and their relationships (i.e. composition and reference). Moreover, we show how the inferred schemas can be useful to develop data validators and schema viewers. In this article, other possible applications will be indicated such as data migration and data visualization.

Apache Spark SQL [9] and `mongodb-schema` [6], for example, are able to extract schemas from NoSQL databases, but their approach is lacking. The former represents different versions as "sum types" instead of reflecting the exact versions; neither of them extract entities nor relationships. Recently, a schema inference algorithm has been proposed which outputs a JSON-Schema document instead of a model, and entity versions are not extracted [5]. An approach to extract JSON schemas is "JSON Discoverer" devised in the context of REST web services [2]. The extraction of relationships is not considered in any of them.

The approach has been designed to be applied to NoSQL systems whose data model is aggregate-oriented [7], which is the data model of the three most widely used types of NoSQL stores: *document*, *key-value*, and *column family* stores. They organize the storage in form of collections of key-value pairs in which the values can also be collections of key-value pairs, and JSON is the format normally used to store data. Our proposal has been validated for the MongoDB, CouchDB, and HBase stores. and the tools implemented may be downloaded from `http://www.catedrasaes.org/wiki/NoSQLSchemaVersions`.

A first version of our work will be presented at the ER'2015 conference [8].

2 Overview

Figure 1 shows the architecture of the solution, which is organized in three stages. Firstly, a Map-Reduce operation is applied in order to extract a collection of JSON objects that contains one object for each version of an entity, i.e. the minimum number of objects that are needed to perform the inference process. Map-Reduce is germane to most NoSQL databases, and gives an advantage in performance. Secondly, that collection is injected into a model that conforms to a JSON metamodel. Thirdly, the reverse engineering process is implemented as a model-to-model transformation whose input is the JSON model, and generates a model that conforms to the NoSQL-Schema metamodel. The inferred NoSQL-Schema models may be used to build tools. As proof of concept we have developed a data validator generator and a schema viewer.

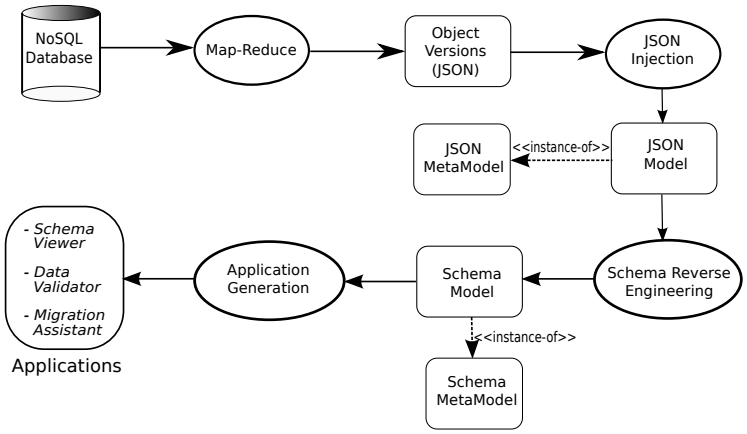

Fig. 1. Overview of the proposed MDE architecture.

We have used MDE techniques to implement the first version of the proposed architecture, in order represent NoSQL schemas as models that conforms to a metamodel, and to take advantage the automation capabilities.

3 Versioned NoSQL-Schema Applications

Several well-known benefits are gained by graphically or textually representing NoSQL schemas. Figure 2(b) shows a textual report that specifies all the entity versions discovered. These reports are automatically generated by a model-to-text transformation that has the schema model as its input. As shown in Figure 2(a) the inferred schemas have been also visualized as UML class diagrams. We have taken advantage of the uniform representation of models and metamodels, and have used the utility integrated into EMF/Ecore to represent metamodels as UML class diagrams. Note that entity versions cannot be explicitly represented in class diagrams, but a new kind of representation is needed.

Object validators (a.k.a. schema predicates) could be created to assure that all the objects a given application retrieves or stores conform to a given type and version. Validators also allow characterizing objects to perform a filter operation on the database. We have generated validator functions by means of model-to-text transformations whose input is a model that specifies relations between versions of an entity type with respect to the JSON object structure.

After this initial effort, future directions include generating data visualizations that take into account the type *and* version of the objects in the database, allowing to visually identify the quantities of objects of each type and version. If a database has overcome several versions migrations, it would be interesting to show how many data belongs to each version.

Generating object version transformers could also be interesting. A developer can describe, by means of a specialized DSL, the necessary steps to convert one

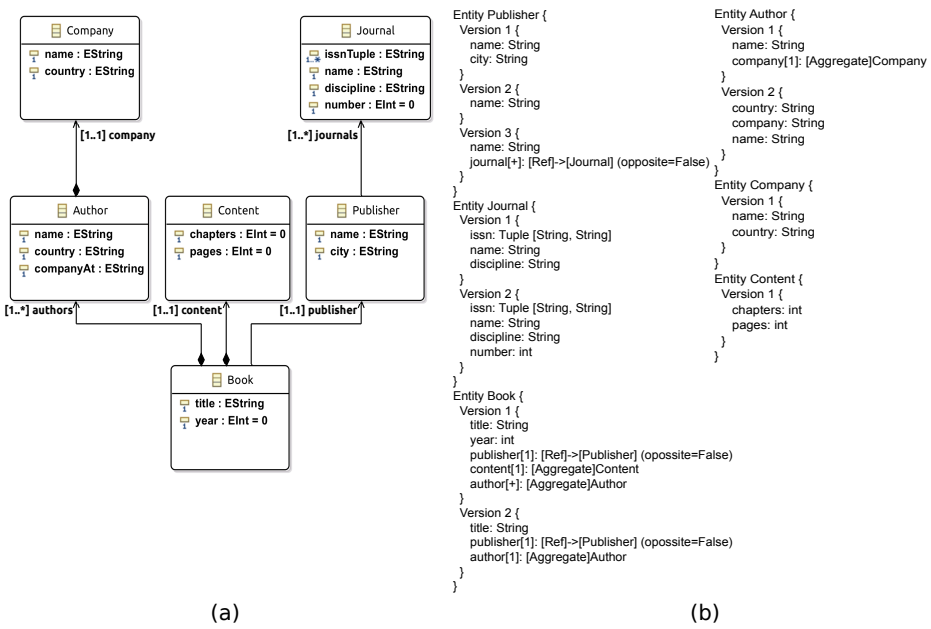

Fig. 2. Graphical representation of entities and textual description of versions.

version of an object to another version. These could be used in at least two ways: (i) A new application may require that all the recovered objects comply with the a new version; (ii) In the case of a batch database migration, Map-Reduce jobs could be generated to transform old version objects into new versions. This is possible given the precise version information stored in the schema.

References

1. Apache Foundation: Apache Drill (Visited Apr 2015), `http://drill.apache.org/`
2. Cánovas, J., Cabot, J.: Discovering Implicit Schemas in JSON Data. In: ICWE. pp. 68–83 (July 2013)
3. Fowler, M.: Schemaless Data Structures (January 2013), `http://martinfowler.com/articles/schemaless/`
4. Karpov, V.: Mongoose NPM package (Visited Apr 2015), `https://www.npmjs.com/package/mongoose`
5. Klettke, M., Störl, U., Scherzinger, S.: Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In: Datenbanksysteme für Business, Technologie und Web (BTW), 16. Hamburg, Germany. pp. 425–444 (2015)
6. Rückstieß, T.: mongodb-schema NPM package (Visited Apr 2015), `https://www.npmjs.com/package/mongodb-schema`
7. Sadalage, P., Fowler, M.: NoSQL Distilled. A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley (2012)
8. Sevilla, D., Morales, S.F., Molina, J.G.: Inferring Versioned Schemas from NoSQL Databases and its Applications. In: ER'15 (October 2015)
9. Zaharia, M., Chowdhury, M., et al.: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In: NSDI (April 2012), `http://spark.apache.org`