MM-infer: A Tool for Inference of Multi-Model Schemas*

Demo Paper

Pavel Koupil
Department of Software Engineering, Charles University
Prague, Czech Republic
pavel.koupil@matfyz.cuni.cz

Sebastián Hricko
Department of Software Engineering, Charles University
Prague, Czech Republic
sebastian.hricko@gmail.com

Irena Holubová
Department of Software Engineering, Charles University
Prague, Czech Republic
irena.holubova@matfyz.cuni.cz

ABSTRACT

The ***variety*** feature of Big Data, represented by *multi-model data*, has brought a new dimension of complexity to data management. The need to process a set of distinct but interlinked models is a challenging task. In our demonstration, we present our prototype implementation *MM-infer* that ensures inference of a common schema of multi-model data. It supports popular data models and all three types of their mutual combinations, i.e., inter-model references, the embedding of models, and cross-model redundancy. Following the current trends, the implementation can efficiently process large amounts of data. To the best of our knowledge, ours is the first tool addressing schema inference in the world of multi-model databases.

1 INTRODUCTION

The knowledge of a schema, i.e., the structure of the data, is critical for its efficient processing in all types of database management systems (DBMSs) no matter if *schema-full*, *schema-less*, or *schema-mixed*. The problem of *inference of a schema* from a given schema-less data has been studied for several years mainly for XML and JSON. For XML documents the schemas involve regular expressions which describe the structure of particular elements. According to the Gold's theorem [4] regular languages are not identifiable only from positive examples (i.e., sample XML documents), so either heuristics [5, 10] or a restriction to an *identifiable* subclass of regular languages [2] is applied. New approaches for JSON focus mainly on schema inference for Big Data [1, 8]. However, the ***volume*** of Big Data is not its only challenge. The ***variety*** feature represented by the *multi-model data* adds a new dimension of complexity – the need to process a set of distinct but interlinked data models.

Example 1.1. Fig. 1 provides an example of a scenario from multi-model benchmark *UniBench*[1]. It represents an e-shop where customers, members of a social network, order products from various vendors. For our purposes, it was extended with references and redundancy. The arrows represent intra-model (black) and inter-model (red) references. The redundancy can be seen in the case of objects *Product* and *Customer* in the relational and JSON document model. Objects *Customer* also form a subset of objects *Person* in the graph model. □

At the logical level, the transition between two models can be expressed either via (1) *inter-model references* or by (2) *embedding* one model into another (such as, e.g., columns of type *JSONB* in relational tables of *PostgreSQL*[2]). Another possible combination

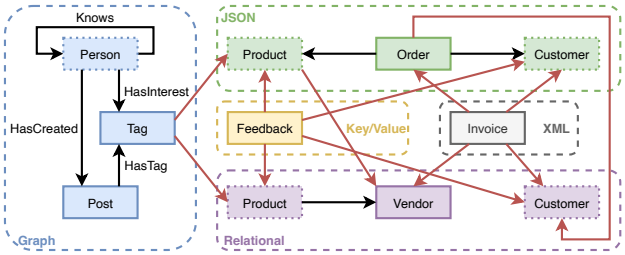

Figure 1: Extended *UniBench* multi-model scenario

of models is via (3) *cross-model redundancy*, i.e., storing the same data fragment in multiple models.

In the case of multi-model data, the inference of a schema is further complicated by contradictory features of the combined models (such as structured vs semi-structured, aggregate-oriented vs aggregate-ignorant, order-preserving vs order-ignoring etc.), cross-model integrity constraints (ICs) involving inter-model references, cross-model redundancy, or the existence of a (partial) schema in some models. Besides, there are verified single-model inference approaches that, however, naturally target only specifics of the particular data model. Furthermore, the question is how to represent the resulting multi-model schema. All these aspects imply the need for a schema inference approach that (1) can be universally applied to multi-model data and cover all specifics of the individual models and (2) can process large amounts of data in distributed multi-model DBMSs efficiently.

To address the key indicated challenges, we extend our previous research results both in the area of inference of an XML schema [5, 10] and unified management of multi-model data [6, 9]. In this demonstration, we present a tool called *MM-infer*[3], a modular and extensible framework that enables one to infer a schema for given multi-model data. It supports all popular data models (relational, graph, key/value, column, and document) and all three types of their combination. If needed, it can process and integrate to the result also an existing partial schema, both user-defined or inferred by a single-model approach. Both types of input, i.e., data eventually with partial schemas, are transformed to a unifying representation called *Record Schema Description* (RSD) using the *local schema inferrer*. At the same time, a set of compressed footprints and statistics describing the data is built. A *global schema inferrer* then merges the local RSDs into the resulting global multi-model schema and uses the footprints and statistics to efficiently detect candidates for local ICs, intra/inter-model references, and cross-model redundancy to be eventually modified by the user. Efficient processing of possibly large input data set is ensured by *Apache Spark*[4] and the idea of footprints that

*Supported by the GAČR project no. 20-22276S.
[1] http://udbms.cs.helsinki.fi/?projects/ubench
[2] https://www.postgresql.org/

[3] https://www.ksi.mff.cuni.cz/~koupil/mm-infer/
[4] https://spark.apache.org/

enable fast filtering of the information. The resulting multi-model schema is provided in a selected textual or graphical format.

In the remainder of this paper, we introduce the ideas implemented in *MM-infer* and outline our demonstration.

2 MULTI-MODEL SCHEMA INFERENCE

The core problem of schema inference in the multi-model world is how to "grasp" all the input models and their specific features. A naive approach would choose one of the models to represent all the data and use a single-model inference strategy. However, since the models naturally do not have the same features, it would lead to imprecise and unnatural structures. First, in Tab. 1 we provide an overview of the popular models we support, their classification, and especially a unification of model-specific terms.

Unifying term	Aggregate-ignorant		Aggregate-oriented		
	Relational	Graph	Key/value	Document	Column
Kind	Table	Label / type	Bucket	Collection	Column family
Record	Tuple	Node / edge	(key, value)	Document	Row
Property	Attribute	Property	–	Field	Column
Domain	Data type	Data type	–	Data type	Data type
Value	Value	Value	Value	Value	Value
Identifier	Key	Identifier	Key	Identifier	Row key
Reference	Foreign key	–	–	Reference	–
Array	–	Array	Array	Array	Array
Structure	–	–	Set / Zset / Hash / ...	Nested doc. / object	Super column

Table 1: Unification of terms in popular models [6]

The logical basic unit for which we infer a schema is a kind. To unify its representation, we introduce the notion of a *Record Schema Description* (RSD). It enables one to describe a schema of one kind regardless of its model(s). Naturally, it covers also the case of a schema for a single record. In the proposed inference process RSD serves for representation of both types of input (data or schema), intermediate schemas, as well as the resulting multi-model schema.

The RSD has a tree structure, and it describes the structure of a property, a record, or a kind (i.e., a set of records). Its root corresponds to the root property of the respective data model (e.g., the root XML element or the anonymous root of a JSON hierarchy) or an artificial root property encapsulating the properties (e.g., in the relational or graph model). An RSD is recursively defined as a tuple $rsd = (name, unique, share, id, types, models, children, regexp, ref)$, where:

- *name* is the name of the property extracted from the data (e.g., `_id`, `person`), or it can be anonymous (e.g., in case of items of JSON arrays or an artificial root property).
- *unique* is the IC specifying uniqueness of values of the property. Its values can be `T` (true), `F` (false), or `U` (unknown) for intermediate steps of the inference process.
- $share = occ/all$ is the number of records in which the property occurs (occ) from all the input records (all). Hence it also bears information about the optionality of a property.
- *id* is the IC specifying that the property is an identifier. Its values can also be `T`, `F`, or `U` with the same meaning.[5]
- *types* is a set of data types that cover the property. For a simple property it involves simple data types (i.e., `String`, `Integer`, ...). For a complex property it involves values `Array` (i.e., ordered (un)named (not)unique child properties – e.g., child XML elements or items of JSON arrays), `Set` (i.e., unordered unnamed unique child properties – e.g., items of `Set` in *Cassandra*[6]), and `Map` (i.e., unordered named unique child properties – e.g., attributes of a relational table). In the final phase of the inference process the set is reduced to a single resulting datatype.
- *models* is a (possibly empty) set of models (`D` = document, `R` = relational, `G` = graph, `C` = column, `K` = key/value) that involve the property. If the set contains more than one item, it represents cross-model redundancy. If the value of *models* within a child property changes, it corresponds to embedding one model to another.
- *children* is a (possibly empty) set of recursively defined child properties.
- (Optional) *regexp* specifies a regular expression over the set *children* (or its subset – e.g., in case of XML elements and attributes, forming together child properties). The respective model does not support a more specific restriction on the order/amount of child properties if not specified.
- (Optional) *ref* specifies that the property references another property. Thanks to the unified representation, we also cover self-references and inter-model references.

Example 2.1. Fig. 2 provides sample RSDs of selected kinds from Fig. 1 in their textual form – one for the relational model (violet) and two for the document model (green and white). In the white case (XML) we can see nonempty *regexp* extracted from a DTD. □

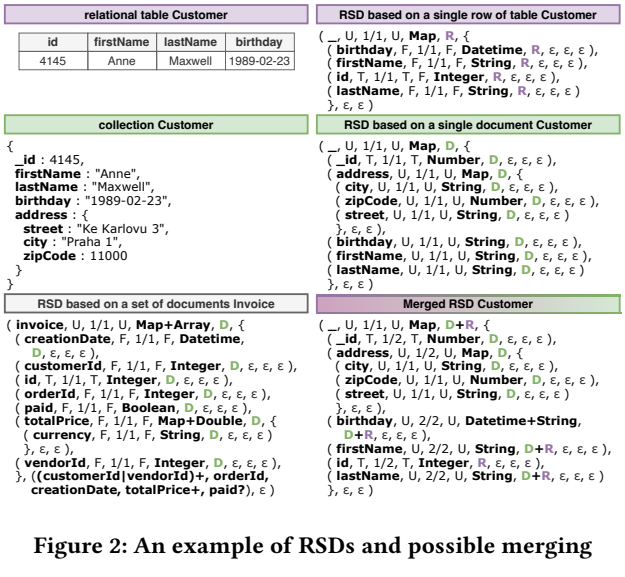

Figure 2: An example of RSDs and possible merging

2.1 Inference Process

With the unifying representation of all possible types of input data, the inference process can treat all of them in the same way, which simplifies the processing and eases the detection of various cross-model aspects, such as redundancy. In general, the approach was designed in order to support:

- various aspects of the combined models and their specifics known for popular multi-model DBMSs [7] (such as sets / maps / arrays / tuples, (un)ordered properties, etc.),
- local ICs, redundancy, and intra/inter-model references,

[5] Note that for the sake of simplicity the current version of *MM-infer* does not support composite identifiers (or references).

[6] https://cassandra.apache.org/

- possible, but not compulsory user interaction influencing the result, such as modification of suggested candidates (for ICs, redundancy etc.) or specification of non-detected cases, and
- processing of Big Data, i.e., efficient filtering and parallel processing of input.

The inference process is based on the fact that two RSDs can be merged to describe a common schema of the respective kinds. The input of the process is formed of the following:

(1) A non-empty set of single/multi-model DBMSs $\mathcal{D}_1, \mathcal{D}_2, ...$ which together contain a set of kinds $\kappa_1, \kappa_2, ..., \kappa_N$. Each kind is associated with its model(s). For each model supported in a particular DBMS $\mathcal{D}_i$, we also know whether it is schema-less/full/mixed and whether the order of sibling properties of a kind must be preserved.
(2) A (possibly empty) set of predefined schemas $\sigma'_1, \sigma'_2, ..., \sigma'_n$ where $n \leq N$, (partially) describing selected kinds.
(3) A (possibly empty) set of user-specified input information which can be of the following types:
   (a) *Redundancy set* of kinds $R = \{\kappa_1, \kappa_2, ..., \kappa_r\}, r \leq N$ which describe the same part of reality, i.e., they will have a common schema σ. (Note that there is no restriction on the models the kinds in R can have.)
   (b) *Simple data type* assigned to a selected property.
   (c) *Local IC* assigned to a selected property. The possible constraints involve identifier, unique, or (not) null.
   (d) *Reference* represented by an ordered pair of properties where the first one is the *referencing property* and the second one is the *referenced property*.

The inference process consists of the following stages:

(1) *Local Schema Inferrer*: For each kind κ we generate its local RSD as follows:
   (a) If κ has a predefined schema σ'_κ we transform it into RSD representation.
   (b) Otherwise, we generate for κ a *basic RSD* using a parallel approach as follows:
      (i) We generate a *trivial RSD* for each record of kind κ.
      (ii) We merge (see Sec 2.1.1) trivial RSDs of κ, and eventually all kinds in its respective redundancy set R_κ, to a basic RSD.
(2) *Gathering of Footprints and Statistics*: Parallel to the local schema inference, for each kind κ we gather its auxiliary statistics (see Sec. 2.1.2) as follows:
   (a) *Phase Map*: We gather statistics for each value of each property p_i^κ of κ.
   (b) *Phase Reduce*: We merge all statistics of values of each property p_i^κ, resulting in aggregated statistics and footprint of property p_i^κ.
   (c) *Candidate Set*: We apply a set of heuristic rules on the merged statistics of each property to produce a set of candidates for redundancy, local ICs, and references.
   (d) The user can confirm/refute the candidates at this stage or add new ones.
(3) *Global Schema Inferrer*: Having a unified RSD representation and confirmed candidates for each input kind κ, we generate the final global RSD as follows:
   (a) We merge (see Sec 2.1.1) all RSDs related to one kind κ.
   (b) (Optionally) we check the validity of the eventual newly detected candidates. Either the user can refute or confirm them, or *MM-infer* can perform a full check.
(4) We transform the resulting set of RSDs and respective ICs to the user-required output (e.g., UML, JSON Schema etc.).

2.1.1 Merging of RSDs. Generation of a basic (local) RSD consists of generating an RSD for each record and their merging into a common schema. When creating the resulting RSD for each kind, this merging is also used at the global (multi-model) level, knowing the whole data set. We check whether merging is possible, merge the information, and modify the regular expression describing the data.

Having two RSDs $rsd_1 = (name_1, unique_1, share_1 = occ_1/all_1, id_1, types_1, models_1, children_1, regexp1, ref_1)$ and $rsd_2 = (name_2, unique_2, share_2 = occ_2/all_2, id_2, types_2, models_2, children_2, regexp_2, ref_2)$, the merging process creates the resulting RSD $rsd = (name, unique, share = occ/all, id, types, models, children, regexp, ref)$ as follows:

- If $name_1 <> name_2$, it has to be resolved by the user/default settings. Otherwise, $name = name_1$.
- $unique = min(unique_1, unique_2)$, where $\mathsf{F} < \mathsf{U} < \mathsf{T}$
- $occ = occ_1 + occ_2$ and $all = all_1 + all_2$
- $id = min(id_1, id_2)$, where $\mathsf{F} < \mathsf{U} < \mathsf{T}$
- $types = types_1 \cup types_2$, whereas if there appear two types $t_1, t_2 \in types$, s.t. $t_1 \subset t_2$, then t_1 is removed from $types$.
- $models = models_1 \cup models_2$
- $children = children_1 \cup children_2$, whereas the child properties with the same name are recursively merged too.
- $regexp$ is a result of merging of regular expressions $regexp_1$ and $regexp_2$ using a verified strategy [5].
- If $ref_1 == ref_2$, then $ref = ref_1$. Otherwise, it has to be resolved by the user/default settings.

Example 2.2. In Fig. 2 bottom right, an example of merging an RSD of a relational table (violet) and an RSD of a document (green) is provided. We can see cross-model redundancy of the two models (denoted by `D+R` for the root property), properties that are a part of only one of the models (`R` for property `id` or `D` for properties `address` and `_id`), and a union type (`Datetime` + `String` for property `birthday`). □

2.1.2 Efficient Data Processing. The second challenge we face is the possible extensive size of data. For this purpose, we exploit parallel processing for inference of the structure of the data (utilizing the *Apache Spark*), i.e., extraction and merging of RSDs. And we apply heuristics that enable us to speed up the detection process of non-structural information. In particular, to detect local ICs (e.g., uniqueness or nullability), references, or redundancy, we would need to check the active domains of all properties and their combinations. For optimization of this process, during the necessary parsing of the input data, we gather (1) basic statistics (e.g., minimum, maximum, average, etc.) and (2) footprints of active domains stored using the Bloom filter [3].

The basic statistics enable to quickly filter, e.g., non-overlapping sets of the value of referencing and referenced properties. After the filtering using basic statistics, we perform filtering using the footprints. In general, the Bloom filter enables to bear smaller information about the active domains of properties to be compared but at the cost of possible false positives. Using this approach, we efficiently generate a small set of candidates for redundancy, references, and local ICs. Then, either the user can confirm or refute them, or a full scan and checking of all values (but just for the particular candidates) can be performed.

2.2 Architecture and Implementation

The frontend of *MM-infer* was implemented in *Dart* using framework *Flutter*[7]; the backend was implemented in *Java* and using *Apache Spark*. The architecture of *MM-infer*, depicted in Fig. 3, reflects the steps of the above-described inference process.

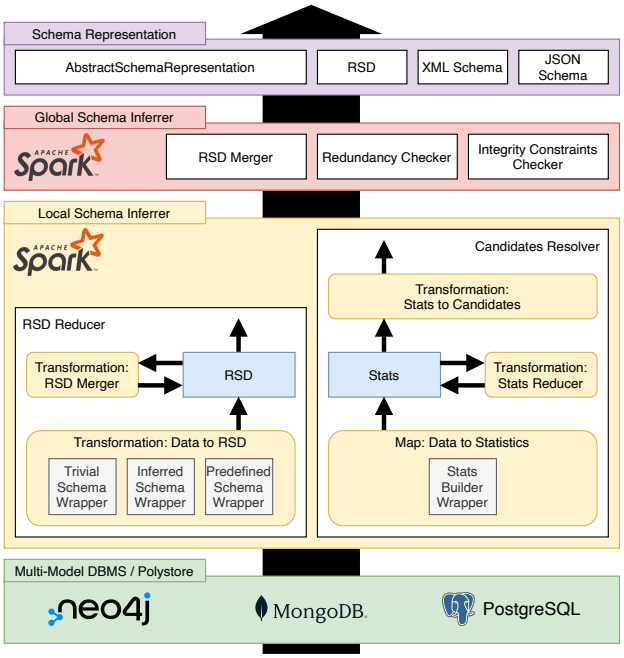

Figure 3: Architecture of *MM-infer*

At the bottom, we can see data sources (green box) – a multi-model DBMS or a set of single/multi-model DBMS (i.e., a polystore-like storage). The local schema inferrer (yellow box) uses three types of wrappers that transform the input data/schema into RSD (predefined, inferred, and trivial). It merges them locally (i.e., within one DBMS) using *Apache Spark*. In parallel, it gathers and merges the data statistics and produces the respective candidates to be eventually modified by the user. The global schema inferrer (red box) merges the RSDs globally (i.e., in the context of all inputs) and checks candidates for ICs and redundancy. The resulting multi-model schema is provided to the user in the chosen representation (violet box).

3 DEMONSTRATION OUTLINE

MM-infer currently supports the following models and DBMSs: *PostgreSQL* (relational and document, i.e., multi-model), *Neo4j*[8] (graph), and *MongoDB*[9] (document) which represents both schema-full and schema-less DBMS.

In our demo of *MM-infer*, we build two use cases around (1) a structurally rich multi-model scenario from *UniBench* depicted in Fig. 1, even extended to demonstrate all interesting cases, and (2) simpler but real-world data from *IMDb*[10] transformed into multiple models (i.e., relational, graph, and document). We will gradually go through the inference process, i.e., (1) choice of

[7]https://flutter.dev/
[8]https://neo4j.com/
[9]https://www.mongodb.com/
[10]https://www.imdb.com/interfaces/

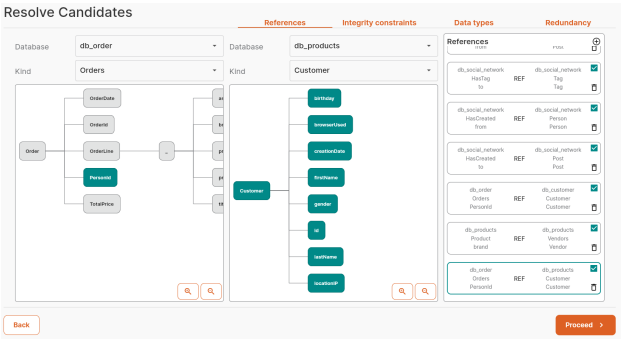

Figure 4: *MM-infer* – Modification of Candidates (left = referencing property, middle = referenced property, right = list of all candidates for references)

particular DBMSs, models, and kinds, (2) modification of candidates for ICs and redundancy (see Fig. 4), and (3) analysis of the inferred result. Using the two distinct data sets, we will show:

- the effectiveness of the approach, i.e. inference of a multi-model schema consisting of three distinct models and involving all three types of cross-model boundaries,
- the advantages of detection of cross-model redundancy,
- the applicability of default candidates and possible user-specified alternatives and their impact on the result.

Since *MM-infer* is a prototype implementation representing the first stage of an ongoing research project, we will also show where and how it can be further extended thanks to its modular design. Our planned (near) future extensions involve support for more complex ICs (e.g., composite identifiers and references or `CHECK` constraints), support of more types of DBMSs, and transformation of the output schema to various representations (e.g., UML or a categorical representation [9]). An essential extension will also be the inference of versions of schemas for the purpose of evolution management.

ACKNOWLEDGEMENTS

This paper was supported by the GAČR grant project no. 20-22276S.

REFERENCES

[1] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. Parametric Schema Inference for Massive JSON Datasets. *VLDB J.* 28, 4 (2019), 497–521.
[2] Geert Jan Bex, Wouter Gelade, Frank Neven, and Stijn Vansummeren. 2010. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. *ACM Trans. Web* 4, 4, Article 14 (sep 2010), 32 pages.
[3] Burton H. Bloom. 1970. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (jul 1970), 422–426.
[4] E. M. Gold. 1967. Language Identification in the Limit. *Information and Control* 10, 5 (1967), 447–474.
[5] Michal Klempa, Michal Kozak, Mário Mikula, Robert Smetana, Jakub Starka, Michal Švirec, Matej Vitásek, Martin Nečaský, and Irena Holubova (Mlýnková). 2013. jInfer: A Framework for XML Schema Inference. *Comput. J.* 58, 1 (12 2013), 134–156.
[6] Pavel Koupil, Martin Svoboda, and Irena Holubová. [n.d.]. MM-cat: A Tool for Modeling and Transformation of Multi-Model Data using Category Theory. In *MODELS '21 Companion, pages = 635–639, publisher = IEEE, year = 2021.*
[7] Jiaheng Lu and Irena Holubová. 2019. Multi-Model Databases: A New Journey to Handle the Variety of Data. *ACM Comput. Surv.* 52, 3, Article 55 (2019).
[8] Diego Sevilla Ruiz, Severino Feliciano Morales, and Jesús García Molina. 2015. Inferring Versioned Schemas from NoSQL Databases and Its Applications. In *ER 2015 (LNCS)*, Vol. 9381. Springer, 467–480.
[9] Martin Svoboda, Pavel Contos, and Irena Holubova. 2021. Categorical Modeling of Multi-Model Data: One Model to Rule Them All. In *MEDI 2021 (LNCS)*, Vol. 12732. Springer, 1–8.
[10] Ondrej Vosta, Irena Mlynkova, and Jaroslav Pokorný. 2008. Even an Ant Can Create an XSD. In *DASFAA 2008 (LNCS)*, Vol. 4947. Springer, 35–50.