Schemas And Types For JSON Data

Mohamed-Amine Baazizi
Sorbonne Université, LIP6 UMR 7606
France
baazizi@ia.lip6.fr

Dario Colazzo
Université Paris-Dauphine, PSL Research University
France
dario.colazzo@dauphine.fr

Giorgio Ghelli
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
ghelli@di.unipi.it

Carlo Sartiani
DIMIE
Università della Basilicata
Potenza, Italy
sartiani@gmail.com

ABSTRACT

The last few years have seen the fast and ubiquitous diffusion of JSON as one of the most widely used formats for publishing and interchanging data, as it combines the flexibility of semistructured data models with well-known data structures like records and arrays. The user willing to effectively manage JSON data collections can rely on several schema languages, like JSON Schema, JSound, and Joi, or on the type abstractions offered by modern programming languages like Swift or TypeScript.

The main aim of this tutorial is to provide the audience with the basic notions for enjoying all the benefits that schemas and types can offer while processing and manipulating JSON data. This tutorial focuses on four main aspects of the relation between JSON and schemas: (1) we survey existing schema language proposals and discuss their prominent features; (2) we review how modern programming languages support JSON data as first-class citizens; (3) we analyze tools that can infer schemas from data, or that exploit schema information for improving data parsing and management; and (4) we discuss some open research challenges and opportunities related to JSON data.

1 INTRODUCTION

The last two decades have seen a dramatic change in the data processing landscape. While at the end of the last century data were usually very structured and managed inside relational DBMSs, nowadays they have very different characteristics: they are big, usually semistructured or even unstructured, without a rigid and predefined schema, and hosted and produced in data processing platforms that do not embrace the relational model. In this new scenario, where data come without a schema, and multiple data models coexist, JSON is affirming as a useful format for publishing and exchanging data, as it combines the flexibility of XML with well-known data structures like records and arrays. JSON is currently employed for publishing and sharing data in many application fields and for many different purposes: for instance, JSON is used as the result format for many web site APIs (e.g., Twitter, New York Times), as a common format for the remote interaction of modern web applications (e.g., Facebook's GraphQL is entirely based on JSON), as a common format for exchanging scientific data as well as public open data (e.g., the U.S. Government's open data platform: `https://www.data.gov`).

Given the wide diffusion of JSON and its use in scientific as well as mainstream applications, the need to directly manipulate JSON data inside applications rapidly emerged. To this aim, a schema language, specifically designed for JSON, has been introduced, but its adoption is not growing at a fast pace, since its specification is somewhat complex and many modern programming languages, like Swift and TypeScript, directly support JSON data through their own, simple type systems; furthermore, Walmart Labs endowed JavaScript, which is inherently untyped, with a powerful schema language for JSON objects by means of JavaScript function calls.

In this tutorial proposal, we will present and discuss existing schema and type languages for JSON data, and compare their features; we will also discuss several schema-related tools, with a particular focus on approaches for schema inference. The main aim of this tutorial is to provide the audience and developers with the basic notions for enjoying all the benefits that schemas and types can offer while processing, analyzing, and manipulating JSON data.

Outline. This *1.5-hour* tutorial is split into five main parts:

(1) **JSON primer (∼ 10 min.)**. In this very introductory part of the tutorial, we review the basic notions about JSON together with its JavaScript legacy, and present a few examples, coming from publicly available datasets, that we will use throughout the remaining parts of the tutorial.
(2) **Schema languages (∼ 20 min.)**. In this part of the tutorial we focus on existing schema languages for JSON data collections and discuss their most prominent features.
(3) **Types in Programming Languages (∼ 15 min.)**. In this part of the tutorial we review how modern programming languages support JSON data as first class citizens. In particular, we focus on programming and scripting languages for web and/or mobile applications, where JSON data interchange is a crucial task.
(4) **Schema Tools (∼ 30 min.)**. In this part of the tutorial we analyze tools that exploit schema information for improving JSON data processing. We focus on the problem of inferring a meaningful schema for schemaless JSON collections, as well as on the exploitation of schema information for improving data parsing and management.
(5) **Future Opportunities (∼ 10 min.)**. Finally, we outline open research problems as potential directions for new research in this area.

In what follows we describe at a very high level the technical content covered in each of the last four aforementioned parts.

2 SCHEMA LANGUAGES

In this part of the tutorial we will focus our attention on several schema languages for JSON data, with particular emphasis on JSON Schema [4] and Walmart Labs Joi [6].

JSON Schema emerged in the academic community and has been developed without a specific programming or scripting language in mind. JSON Schema allows the programmer to specify a schema for any kind of JSON values, and supports traditional type constructors, like union and concatenation, as well as very powerful constructors like negation types.

JSON Schema has already been studied. Indeed, in [21], motivated by the need of laying the formal foundations for the JSON Schema language [4], Pezoa et al. present the formal semantics of that language, as well as a theoretical study of its expressive power and validation problem. Along the lines of [21], Bourhis et al. [15] have recently laid the foundations for a logical characterization and formal study for JSON schema and query languages.

On the contrary, Joi has been developed by Walmart as a tool for (i) creating schemas for JSON objects and (ii) ensuring the validation of objects inside an untyped scripting language like JavaScript; furthermore, to the best of our knowledge, Joi has not been studied so far. Joi only allows the designer to describe the schema for JSON objects, but it still provides the ability to specify co-occurrence and mutual exclusion constraints on fields, as well as union and value-dependent types.

We will analyze the most prominent features of these languages, and compare their capabilities in a few scenarios. We will also briefly discuss JSound [5], an alternative, but quite restrictive, schema language, as well as a few other schema-related proposals, such that described in [24], where Wang et al. present a framework for efficiently managing a schema repository for JSON document stores. The proposed approach relies on a notion of JSON schema called *skeleton*. In a nutshell, a skeleton is a collection of trees describing structures that frequently appear in the objects of a JSON data collection. In particular, the skeleton may totally miss information about paths that can be traversed in some of the JSON objects.

3 TYPES IN PROGRAMMING LANGUAGES

Unlike XML, which found no space as a first class citizen in programming languages, with the obvious and notable exception of XQuery, JSON has been designed starting from the object language of an existing scripting language. Therefore, given its wide use in web and mainstream application development, JSON support has been introduced in several strongly typed programming and/or scripting languages.

To directly and naturally manage JSON data a programming language should incorporate the ability to express record types, sequence types, and union types. While record and sequence types can be easily found in many programming languages, union types are quite rare and usually confined to functional languages only.

In this part of the tutorial we will discuss the support for JSON objects inside the type systems of TypeScript [9] and Swift [8]. TypeScript is a typed extension of JavaScript, while Swift is an Apple-backed programming language that is rapidly becoming the language of choice for developing applications in the Apple ecosystem (iOS + macOS). These languages show similar features, but also very significant differences in the treatment of JSON objects.

We will also compare the features offered by these languages with those of the schema languages we presented in the second part of the tutorial.

4 SCHEMA TOOLS

In this part of the tutorial we will present several schema-related tools for JSON data. We will first discuss existing approaches for inferring a schema starting from a dataset and then move to parsing tools that are able to exploit dynamic type information to speed-up data parsing.

4.1 Schema Inference

Several schema inference approaches for JSON data collections have been proposed in the past. In [10–12] authors describe a distributed, parametric schema inference approach capable of inferring schemas at different levels of abstraction. In the context of Spark, the Spark Dataframe schema extraction [7] is a very interesting tool for the automated extraction of a schema from JSON datasets; this tool infers schemas in a distributed fashion, but, unlike the technique described in [10–12], its inference approach is quite imprecise, since the type language lacks union types, and the inference algorithm resorts to `Str` on strongly heterogeneous collections of data. Other systems, like Jaql [13], exploit schema information for inferring the output schema of a query, but still require an externally supplied schema for input data, and perform output schema inference only locally on a single machine.

There are also a few inference tools for data stored in NoSQL systems and RDBMSs. Indeed, in the context of NoSQL systems (e.g. MongoDB), recent efforts have been dedicated to the problem of implementing tools for JSON schema inference. A JavaScript library for JSON, called mongodb-schema, is presented in [22]. This tool analyzes JSON objects pulled from MongoDB, and processes them in a streaming fashion; it is able to return quite concise schemas, but it cannot infer information describing field correlation. Studio 3T [19] is a commercial front-end for MongoDB that offers a very simple schema inference and analysis feature, but it is not able to merge similar types, and the resulting schemas can have a huge size, which is comparable to that of the input data. In [23], a python-based tool is described, called Skinfer, which infers JSON Schemas from a collection of JSON objects. Skinfer exploits two different functions for inferring a schema from an object and for merging two schemas; schema merging is limited to record types only, and cannot be recursively applied to objects nested inside arrays. Couchbase, finally, is endowed with a schema discovery module which classifies the objects of a JSON collection based on both structural and semantic information [3]. This module is meant to facilitate query formulation and select relevant indexes for optimizing query workloads.

When moving to RDBMSs, in [16] Abadi and al. deal with the problem of automatically transforming denormalised, nested JSON data into normalised relational data that can be stored in a RDBMS; this is achieved by means of a schema generation algorithm that learns the *normalised, relational* schema from data. This approach ignores the original structure of the JSON input dataset and, instead, depends on patterns in the attribute data values (functional dependencies) to guide its schema generation.

4.2 Parsing

There are a few novel parsing tools for JSON data that take into account dynamic type information for improving the efficiency of the applications relying on them.

In a recent work [20], Li et al. present streaming techniques for efficiently parsing and importing JSON data for analytics tasks; these techniques are then used in a novel C++ JSON parser, called Mison, that exploits AVX instructions to speed up data parsing and discarding unused objects. To this end, it infers structural information of data on the fly in order to detect and prune parts of the data that are not needed by a given analytics task.

In [14], Bonetta and Brantner present Fad.js, a *speculative*, JIT-based JSON encoder and decoder designed for the Oracle Graal.js JavaScript runtime. It exploits data access patterns to optimize both encoding and decoding: indeed, Fad.js relies on the assumption that most applications never use all the fields of input objects, and, for instance, skips unneeded object fields during JSON object parsing.

5 FUTURE OPPORTUNITIES

We finally discuss several open challenges and opportunities related to JSON schemas, including the following ones.

Schema Inference and ML. While all schema inference approaches covered in the previous part of the tutorial are based on traditional techniques, a recent work by Gallinucci et al. [17] shows the potential benefits of ML approaches in schema inference; furthermore, ML-based inference techniques have already been used for non-JSON data, as shown by Halevy et al. in [18]. Hence, a promising research direction is to understand how these methods can be efficiently applied to large collections of data and whether they can overcome some limitations of previous approaches.

Schema-Based Data Translation. While JSON is very frequently used for exchanging and publishing data, it is hardly used as internal data format in Big Data management tools, that, instead, usually rely on formats like Avro [1] and Parquet [2]. When input datasets are heterogeneous, schemas can improve the efficiency and the effectiveness of data format conversion. Therefore, a major opportunity is to design schema-aware data translation algorithms that are driven by schema information and use it to improve the quality of the translation.

6 INTENDED AUDIENCE AND COVERAGE

Our goal is to present a coherent starting point for EDBT attendees who are interested in understanding the foundations and applications of schemas and types for JSON data processing. We will not assume any background in JSON schema languages, but will introduce them starting from the roots, giving broad coverage of many of the key ideas, making it appropriate for graduate students seeking new areas to study and researchers active in the field alike.

7 BIOGRAPHICAL SKETCHES

Mohamed-Amine Baazizi (Ph.D.) is an assistant professor at Sorbonne Université. He received his PhD from Université of Paris-Sud and completed his postdoctoral studies in Télécom Paristech. His research focuses on exploiting schema information for optimizing the processing of semi-structured data.

Dario Colazzo (Ph.D.) is Full Professor in Computer Science at LAMSADE - Université Paris-Dauphine. He received his PhD from Università di Pisa, and he completed his postdoctoral studies at Università di Venezia and Université Paris Sud. His main research activities focus on static analysis techniques for large scale data management.

Giorgio Ghelli (Ph.D.) is Full Professor in Computer Science, at Università di Pisa. He was Visiting Professor at École Normale Supérieure Paris, at Microsoft Research Center, Cambridge (UK), and at Microsoft Co. (Redmond, USA), member of the W3C XML Query Working Group, member of the board of the EAPLS. He worked on database programming languages and type systems for these languages, especially in the fields of object oriented and XML data models.

Carlo Sartiani (Ph.D.) is an assistant professor at Università della Basilicata. He received his PhD from Università di Pisa, and he completed his postdoctoral studies at Università di Pisa. He worked on database programming languages and data integration systems, and his current research activities focus on semistructured and big data.

REFERENCES

[1] Apache Avro. `https://avro.apache.org`.
[2] Apache Parquet. `https://parquet.apache.org`.
[3] Couchbase auto-schema discovery. https://blog.couchbase.com/auto-schema-discovery/.
[4] JSON Schema language. `http://json-schema.org`.
[5] JSound schema definition language. `http://www.jsoniq.org/docs/JSound/html-single/index.html`.
[6] Object schema description language and validator for JavaScript objects. https://github.com/hapijs/joi.
[7] Spark Dataframe. `https://spark.apache.org/docs/latest/sql-programming-guide.html`.
[8] Swift. `https://swift.org`.
[9] TypeScript. `https://www.typescriptlang.org`.
[10] Mohamed Amine Baazizi, Houssem Ben Lahmar, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2017. Schema Inference for Massive JSON Datasets. In *EDBT '17*.
[11] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2017. Counting types for massive JSON datasets. In *Proceedings of The 16th International Symposium on Database Programming Languages, DBPL 2017, Munich, Germany, September 1, 2017*. 9:1–9:12.
[12] Mohamed-Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. Parametric schema inference for massive JSON datasets. *The VLDB Journal* (2019). https://doi.org/10.1007/s00778-018-0532-7
[13] Kevin S. Beyer, Vuk Ercegovac, Rainer Gemulla, Andrey Balmin, Mohamed Y. Eltabakh, Carl-Christian Kanne, Fatma Özcan, and Eugene J. Shekita. 2011. Jaql: A Scripting Language for Large Scale Semistructured Data Analysis. *PVLDB* 4, 12 (2011), 1272–1283.
[14] Daniele Bonetta and Matthias Brantner. 2017. FAD.js: Fast JSON Data Access Using JIT-based Speculative Optimizations. *PVLDB* 10, 12 (2017), 1778–1789. http://www.vldb.org/pvldb/vol10/p1778-bonetta.pdf
[15] Pierre Bourhis, Juan L. Reutter, Fernando Suárez, and Domagoj Vrgoc. 2017. JSON: Data model, Query languages and Schema specification. In *PODS '17*. 123–135.
[16] Michael DiScala and Daniel J. Abadi. 2016. Automatic Generation of Normalized Relational Schemas from Nested Key-Value Data. In *SIGMOD '16*. 295–310.
[17] Enrico Gallinucci, Matteo Golfarelli, and Stefano Rizzi. 2018. Schema profiling of document-oriented databases. *Inf. Syst.* 75 (2018), 13–25.
[18] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing Google's Datasets. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. 795–806.
[19] 3T Software Labs. 2017. Studio 3T. `https://studio3t.com`.
[20] Yinan Li, Nikos R. Katsipoulakis, Badrish Chandramouli, Jonathan Goldstein, and Donald Kossmann. 2017. Mison: A Fast JSON Parser for Data Analytics. *PVLDB* 10, 10 (2017), 1118–1129.
[21] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON Schema. In *WWW '16*. 263–273.
[22] Peter Schmidt. 2017. mongodb-schema. `https://github.com/mongodb-js/mongodb-schema`.
[23] scrapinghub. 2015. Skinfer. `https://github.com/scrapinghub/skinfer`.
[24] Lanjun Wang, Shuo Zhang, Juwei Shi, Limei Jiao, Oktie Hassanzadeh, Jia Zou, and Chen Wangz. 2015. Schema Management for Document Stores. *Proc. VLDB Endow.* 8, 9 (May 2015), 922–933.