IOP Conf. Series: Journal of Physics: Conf. Series **1069** (2018) 012101 doi:10.1088/1742-6596/1069/1/012101

# Survey on JSON Data Modelling

**Teng Lv[1], Ping Yan[2,] * and Weimin He[3]**

[1]School of Information Engineering, Anhui Xinhua University, Hefei 230088, P. R. China
[2]School of Science, Anhui Agricultural University, Hefei 230036, P. R. China
[3]Department of Computing and New Media Technologies, University of Wisconsin-Stevens Point, 2100 Main Street, Stevens Point, WI 54481, USA
Email: Lt0410@163.com, want2fly2002@163.com(corresponding), whe@uwsp.edu

**Abstract.** JavaScript Object Notation (JSON) is a popular data format based on the data types of JavaScript programming language. As a lightweight semi-structured data format language, it has become the main data exchange format over the World Wide Web in the past several years, and gained popularity in database community research. JSON data model describes the basic data structures and semantics of the underlying JSON data, so it is the fundamental and key aspects for JSON data format. As JSON data model is also the key and foundation for other data management technologies, such as data indexing, data querying, data searching, data mapping, data integrating, and data mining, how to design an efficient and powerful model for JSON data is necessary and important to other related research topics, such as data integration, data search and query, data quality and evaluation, etc. In this paper, we investigate some approaches to modelling JOSN data and corresponding JSON schemas, which are mainly based on tree models. The advantages and disadvantages of each method are analyzed and compared.

## 1. Introduction

JSON (JavaScript Object Notation) [1-3] is a popular lightweight semi-structured data format based on the data types of programming language JavaScript. It has become the main data exchange format over the World Wide Web in the past few years. JSON has also gained popularity in database community research. As a semi-structured data format, JSON not only can be integrated in traditional database systems, but also widely used in NoSQL database systems[4-5]. Some query languages for JOSN such as JSONiq [6,7] and SQL++[8] are already proposed to support JSON data querying.

Data model describes the basic data structures and semantics of the underlying data, so it is the fundamental and key aspects for any data format including JSON data. As data model is also the key and foundation for other data management technologies, such as data indexing, data querying, data searching, data mapping, data integrating, and data mining, how to design an efficient and powerful model for JSON data is necessary and important to other related research topics, such as data integration, data search and query, data quality and evaluation, etc. Now researchers proposed some approaches to modelling JSON documents mainly based on tree models [9-11].

The rest of the paper is organized as follows. A JOSN document is a set of objects, and each JOSN object is a set of key-vlaue pairs. We first discussed JSON object and tree models in Section 2. For arrays in JSON documents, there are two approaches namely ordered approach and unordered approach, which are given in Section 3. As an object can be again its own sub-objects in JOSN data, recursions must be considered in JSON model, which are given in Section 4. JOSN schema definition

and their related schema definitions are also discussed in Section 4. We conclude the paper and point out the future directions of the topic in Section 5.

## 2. Objects and JSON Trees

As each JSON object in a JSON document is a set of key-vlaue pairs, a JOSN document can be naturally represented as a data tree structure called JSON tree. A value can be an atomic value such as a string, an integer, a number, an array, or null. To capture the compositional structure of JSON data, each value can again be a set of JSON objects. We first give the definition of a JOSN document as following:

**Definition 1 (JSON document).** A JOSN document D is defined as

D::={Object[,Object, …]},

where

Object ::= {Key:{Value} [, Key:{Value}, …]},

Key ::= String,

Value ::= String|Interger|Number|Array|Null|Object.

For example, consider the following JSON document D1:

```
{
        "name": {
          "FirstName": "Bill",
                "LastName": "Bush"
                },
          "age": 30
}
```

JSON document D1 contains 2 objects O1 and O2 as following:

```
O1::= {
        "name": {
          "FirstName": "Bill",
                "LastName": "Bush"
                },
      }
```

with key "name" and

```
O2::={
          "age": 30
      }
```

with key "age" and an atomic integer value "30".

The value of key of O1 again contains 2 objects O11 and O12 as following:

```
O11::= {
          "FirstName": "Bill",
        }
```

with key "FirstName" and an atomic string value "Bill" and

```
O12::= {
                "LastName": "Bush"
        }
```

with key "LastName" and an atomic string value "Bush".

Of course, a value may contain nothing, which can be represented by type "Null". For type "Array" and "Number", we will explained in Section 3.

## 3. Arrays and Numbers

Suppose we have the following 2 children O3 and O4 of "Bill Bush" in JSON document D1:

```
O3::={
          "name": {
```

```
                    "FirstName": "John",
                        "LastName": "Bush"
                        },
          "age": 10
        }
```

and

```
O4::={
          "name": {
                    "FirstName": "Mary",
                        "LastName": "Bush"
                        },
          "age": 12
        }
```

To combine O3 and O4 in JSON document D1, we add a new key "children" in D1 and arrange O3 and O4 as an array [O3,O4] and numbered by numbers "1" and "2" sequentially as {1:O3, 2:O4}:

```
{
        "name": {
          "FirstName": "Bill",
                "LastName": "Bush"
                },
          "age": 30,
          "children":{1:O3, 2: O4}
}
```

For the problem of whether an array is ordered or unordered, there are two different treatments. Ref.[10] treated an array to be unordered based on the following two reasons: First, as JSON navigation instructions use random access to access elements in arrays, treat arrays as ordered list will naturally suggest that the navigation instructions support different access methods to access elements of array. Second, no system actually supports the navigation from one element to other element in an array. Ref.[11] also mainly concentrated on unordered arrays, but proposed a total order on atomic keys which can model ordered arrays. We think that arrays to be ordered will add feasibility and flexibility in JSON navigation in the future, which can provide more access functionalities such as accessing the first, accessing the last, accessing the previous, and accessing the next element from a given element or position in an array, etc.

## 4. Recursions

Back to object O3 and O4. From the definition of D1, we can see that the structures of O3 and O4 are identical to the structure of D1. Suppose that we want to represent a family tree such that children O3 and O4 will have children several years later, we can use recursion to achieve this goal. We apply a JSON Schema to specify what a JSON document must look like, the ways to extract information from it, and how to interact with it. We first give the schema definition S0 based on Ref.[9] as following:

```
{
        "definitions": {
          "type": "Object",
                "properties":{
                 "Name": {"type": "Object",
                                "properties":{
                                  "FirstName": {"type": "String"},
                                        "LastName": {"type": "String"}
                                },
                 "type": "Object",
                "properties":{
                 "age": {"type": "Integer"}
                        },
```

```
                "type": "Object",
                "properties":{
                 "children": {"type": "Array",
                                "minItems": 0,
                                "maxItems": unlimited,
                            "items": [
                                        {"$ref": "#/definitions/S0"},
                                        {"$ref": "#/definitions/S0"}
                                    ]
                        }
                }
        }
}
```

Schema S0 defines an object which has sub-objects "Name", "age", and "children", respectively. For sub-object "Name", there are two sub-objects "FirstName" and "LastName", which are both String types. For sub-object "age", it is an Integer type (atomic type). For sub-object "children", it is an Array type (atomic type), which again contains 0-unlimited numbers of object of type S0. Now we give the recursion schema definition of JSON document D1 as the following schema S1:

```
{
        "$ref": "#/definitions/S0"
}
```

which is obviously a recursion definition of S0.

Ref.[12] provides detailed JSON Schema definition and formal grammar specification for the JSON Schema documents. Other schema specification and proposals related to JSON are proposed, which are either formal specification standards, or are accepted only for specific use cases, or applied in a specific application. For example, JSON Schema[13] defines the media type "application/schema+json", a JSON-based format for describing the structure of JSON data. The "application/schema-instance+json" media type provides additional feature-rich integration with "application/schema+json" beyond what can be offered for "application/json" documents. Swagger[14] is a framework of API developer tools for the OpenAPI Specification(OAS), enabling development across the entire API lifecycle, from design to test and deployment based on JSON. RAML(RESTful API Modeling Language) [15] makes it easy to manage the whole API lifecycle from design to sharing based on JSON. You only write what you need to define, which can be reused later. It is a machine readable API design that is also human friendly. JSON-LD[16] is a JSON-based format to serialize Linked Data. It is designed to easily integrate into deployed systems that already use JSON, and provides a smooth upgrade path from JSON to JSON-LD. One of its primary goals is to store Linked Data in JSON-based storage engines.

## 5. Conclusions

This paper presents a survey study of different kinds of models of uncertain data in relational databases, XML data, and graph data. We mainly discuss and review probabilistic uncertain data models as they not only are widely used in many applications and areas nowadays, but also have better tradeoffs between simplicity and expressive power.

The open problems of modelling JSON data include both semantic and computation aspects. For semantic aspect, there is no accepted unified model by all community for JOSN data. In real applications, such semantic problems may dependent on specific applications. It is difficult to give a general guideline to design good JSON document and model. For computational aspect, one major problem of recursions in JSON document and schema definition is that processing JOSN document and schema efficiently is a challenge. Algorithms of non-recursions are difficult to deal with the recursions in JOSN data, which are usually exponential scale. One possible approach is use parallel algorithms to deal with recursions, such as the popular Map Reduce framework. Another open problem is to propose some criteria for modelling JOSN data, such as expressive power,

functionalities, complexity, efficiency, extension, etc. But how to combine all these criteria is also a challenge. A practical approach is a trade-off of all these criteria.

## 6. Acknowledgments

The work is supported by Talent of Discipline and Specialty in Colleges and Universities of Anhui Province (No. gxbjZD54), Academic Leader Foundation (No.2014XXK06), Anhui Province Quality Engineering (No.2015zy073, No. 2017ghjc229), Introduction of Talents Foundation of Anhui Xinhua University (No.2015kyqd002), National Nature Science Foundation of China (No.11201002), and Colleges Nature Science Research Key Project of Anhui Province (No.KJ2015A325).

## 7. References

[1] Introducing JSON 2018. http://www.json.org/
[2] ECMA 2017 the JSON Data Interchange Syntax. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
[3] Internet Engineering Task Force (IETF) 2018 The JavaScript Object Notation (JSON) Data Interchange Format. https://tools.ietf.org/html/rfc7159
[4] MongoDB Inc. 2018 the MongoDB3.0 Manual. https://docs.mongodb.org/manual/
[5] RethinkDB 2018 The open-source database for the realtime web. https://www.rethinkdb.com/
[6] Florescu D and Fourny G 2013 JSONiq: The history of a query language. *IEEE Internet Computing*, 17(5) pp 86-90
[7] Fourny G 2016 JSONiq, the SQL of NoSQL. http://www.28.io/jsoniq-the-sql-of-nosql
[8] Ong K W, Papakonstantinou Y, and Vernoux R 2015 The SQL++ query language: Configurable, unifying and semi-structured. *arXiv:1405.3631*
[9] Felipe P, Reutter J L, Suarez F, and Vrgoč D 2016 Foundations of JSON Schema. *International World Wide Web Conferences Steering Committee*, pp 263-273
[10] Pierre B, and Reutter J L 2017 JSON: Data model, Query languages and Schema specification. *PODS'17*, ACM, pp 123-135
[11] Jan H, Paredaens J, and Jan V D B 2017 J-Logic: Logical Foundations for JSON Querying. *SIGMOD'17*, ACM, pp 137-149
[12] Suárez F, Reutter J, Vrgoc D, Ugarte M, Pezoa F 2018 JSON Schema: syntax and semantics. http://cswr.github.io/JsonSchema/
[13] Wright A 2017 JSON schema: A Media Type for Describing JSON Documents. http://json-schema.org/latest/json-schema-core.html
[14] Swagger 2018 http://swagger.io/
[15] The RAML Workgroup 2018 RAML. http://raml.org/
[16] Sporny M, Kellogg G, and Lanthaler M 2014 JSON-LD 1.0: A JSON-based Serialization for Linked Data. http://www.w3.org/TR/json-ld/