Web Data Models

Silviu Maniu

JSON Schema

JSON Schema

- JSON tends to be more popular than XML lately (almost all Web APIs provide at least one JSON endpoint)

- not much work on formalisms of JSON schema

- multiple efforts to provide JSON with a schema like in XML, however not standardized

- most used is JSON Schema (https://json-schema.org/, currently at draft 7)

JSON Schema: Principles

- just as in XML Schema, the JSON Schema is a JSON document

- it specifies the types that each has, its restrictions, and the required types

JSON Schema: Simple Example

```
{
  “$schema”: “http://json-schema.org/draft-07/schema#",
  “title”: “Book”,
  “type”: “object”,
  “properties”:  {
    “title”: {
      “description”: “The title of the book”,
      “type”: “string”
    },
    “year”: {
      “description”: “Year published”,
      “type”: “integer”
    }
  },
  “required”: [“title”]
}
```

JSON Schema: General Structure

- for each item one can specify:

  1. its type (`type`): `string`, `number`, `integer`, `object`, `array`

  2. its properties (for `object`), items (for `array`), or pattern (for `string`)

  3. some restrictions (similar to XML schema)

JSON Schema: Grammar

JSDoc := { (**defs** ,)? **JSch** }
defs := "definitions": { **string** : { **JSch** } (, **string** : { **JSch** })* }
JSch := **strSch** | **numSch** | **intSch** | **objSch** | **arrSch** | **refSch** | **not** | **allOf** | **anyOf** | **enum**
not := "not": { **JSch** }
allOf := "allOf": [{ **JSch** } (, { **JSch** })*]
anyOf := "anyOf": [{ **JSch** } (, { **JSch** })*]
enum := "enum": [**Jval** (, **Jval**)*]
refSch := "$ref": "# **JPointer**"

JSON Schema: Strings

strSch	:=	`"type": "string"` (, **strRes**)*
strRes	:=	**minLength** | **maxLength** | **pattern**
minLength	:=	`"minLength":` **n**
maxLength	:=	`"maxLength":` **n**
pattern	:=	`"pattern":` **"regExp"**

```
“phone”: {
  “type”: “string”,
  “minLength”: “8”,
  “maxLength”: “11”,
  “pattern”: “(+[1–9][1–9])?[0–9]*”
 }
```

JSON Schema: Numbers

$$
\begin{array}{lcl}
\textbf{numSch} & := & \texttt{"type": "number"}\ (,\ \textbf{numRes}\ )^* \\
\textbf{intSch} & := & \texttt{"type": "integer"}\ (,\ \textbf{numRes}\ )^* \\
\textbf{numRes} & := & \textbf{min} \mid \textbf{exMin} \mid \textbf{max} \mid \textbf{exMax} \mid \textbf{mult} \\
\textbf{min} & := & \texttt{"minimum":}\ \ \textbf{r} \\
\textbf{exMin} & := & \texttt{"exclusiveMinimum":}\ \ \texttt{true} \\
\textbf{max} & := & \texttt{"maximum":}\ \ \textbf{r} \\
\textbf{exMax} & := & \texttt{"exclusiveMaximum":}\ \ \texttt{true} \\
\textbf{mult} & := & \texttt{"multipleOf":}\ \ \textbf{r} \quad (\textbf{r} \geq \textbf{0})
\end{array}
$$

```
“edition”: {
  “type”: “integer”,
  “minimum”: 1
 }
```

JSON Schema: Objects

objSch := `"type": "object"` (, **objRes**)*
objRes := **prop** | **addPr** | **patPr** | **req**
prop := `"properties":` { **kSch** (, **kSch**)*}
kSch := **string** : { **JSch** }
addPr := `"additionalProperties": false`
req := `"required":` [**string** (,**string**)*]
patPr := `"patternProperties":`
{ **patSch** (, **patSch**)*}
patSch := `"regExp"` : { **JSch** }

```
“author”: {
  “type”: “object”,
  “properties”: {
   “first”: {“type”: “string”},
   “last”: {“type”: “string”},
  },
  “required”: [“lastName”]
 }
```

JSON Schema: Arrays

arrSch	:=	`"type": "array"` (, **arrRes**)*
arrRes	:=	**itemo** | **itema** | **minIt** | **maxIt** | **unique**
itemo	:=	`"items":` { **JSch**}
itema	:=	`"items":` [{ **JSch**} (, {**JSch**})*]
minIt	:=	`"minItems":` **n**
maxIt	:=	`"maxItems":` **n**
unique	:=	`"uniqueItems": true`

```
“address”: {
  “type”: “array”,
  “items”: [
   {“type”: “integer”},
   {“type”: “string”}
  ],
  “additionalItems”: false
 }
```

JSON Schema: Pointers

- JSON schema allows for pointers to a value in the JSON

- the general form is $p=w_1/w_2/\ldots/w_n$, and is evaluated similarly as in XPath

```
[{“name”: “Ullman”}, {“name”: “Knuth”}]
```

$p = 1/name$

$Eval(p) =$ “Knuth”

JSON Schema: Definitions

- JSON Schema can have a `definitions` part which can be referenced using pointers (similar to types in XML Schema)

```
{
 "definitions": {
   "S": {
     "anyOf": [
       {"enum": [null]},
       {"allOf": [
          {"type": "array",
           "minItems": 2,
           "maxItems": 2,
           "items": [
             {"$ref": "#/definitions/S"},
             {"$ref": "#/definitions/S"}]
          },
          {"not": {"type": "array", "uniqueItems": true}}
       ]}
  ]},
   "$ref": "#/definitions/S"
 }
```

JSON Schema: Definitions

- Can lead to schemas which are ill-designed

```
{
    "definitions": {
        "S": {"not": {"$ref": "#/definitions/S"}}
     },
     "$ref": "#/definitions/S"
}
```

- the above allows to define a document that is both itself and not itself
- **way to fix**: a graph of the definitions where a node is connected to another if it is involved in its definition
- schema ok if graph is acyclic (not implemented in the draft specs!)

JSON Schema: Evaluation

- JSON Schema can be evaluated in polynomial time with a complexity of $O(\mathbf{SD})$

general algorithm

1. process document restriction by restriction

2. at the same time, check that the corresponding subschema validates the document

JSON Schema: Conclusion

- popular schema variant for JSON, actively developed and used
- issues with consistency in the schema which have to be addressed
- missing the theoretical underpinnings as in schemas for XML (tree automata and grammars)
- can be evaluated in polynomial time
- however, not all available validators validate the same schemas!

Further Reading

1. Understanding JSON Schema https://json-schema.org/understanding-json-schema/

2. F. Pezoa, J.L. Reutter, F. Suarez, M. Ugarte, D. Vrgoc. Foundations of JSON Schema. WWW 2016 https://martinugarte.com/media/pdfs/p263.pdf