Jsonya/dm: A Univocal JSON Interpretation

Miloslav Sredkov

Faculty of Mathematics and Informatics, Sofia University

This work was partially supported by the Bulgarian National Science Research Fund through contract 02-102/2009.
Introduction (1)

• JSON grows more and more popular:
  • Intended to be “the intersection of all modern programming languages”
  • “The thing that everybody can agree on, so it's really easy to pass data back and forth” [1]
• Still defined only as syntax
  • Most developers assume semantics biased towards their tools
  • Potential interpretation clashes
Introduction (2)

• Our idea:
  • Interoperable interpretation should be designed based on a large set of environments

• Our contributions:
  • Overview of the currently used JSON data models
  • Analysis of the ambiguous features of JSON
  • The unambiguous data model Jsonya/dm
  • Analysis of 63 JSON libraries for 10 programming languages
II. EXISTING APPROACHES
JavaScript Interpretation

- JSON is a subset of ECMAScript [2], so why shouldn't its interpretation also be?
- IEEE 754 [3] 64-bit floats:
  - Loss of precision when converting to and from text
  - What about $+\text{Inf}$ or $\text{NaN}$?
  - Some environments may lack 64-bit floats
- Are object members ordered?
  - ECMAScript Standard: No
  - Most browsers: Yes
The XML Metamodel

- Some authors consider JSON as “An alternative physical model for XML metamodels” [4]
  - Tools converting between XML and JSON are present
  - XSLT, XQuery, XForms, etc. can be used
- However,
  - XML has multiple different metamodels
  - JSON and XML are too different — conversion is not trivial
  - Inherited XML problems prevent JSON from being “The Fat-Free Alternative to XML”
The YAML Metamodel

• YAML is stated as a “natural superset of JSON” [5]
  • Many YAML technologies can be applied to JSON
  • Its specification (unlike XML's) explicitly defines an information model
• However,
  • YAML is less popular and less tools are available
  • Its information model is loosely defined, e.g: “The supported range and accuracy depends on the implementation, though 32 bit IEEE floats should be safe.” [5:74]
Other Popular Metamodels

- Work at the syntax level only
  - Pros: developers can pick the most suitable interpretation
  - Cons: less convenient, less interoperable
- Map to the types of the host programming language
  - Pros: better performance, more convenient
  - Cons: less interoperable, e.g. not distinguishing empty arrays from null
- A set of custom data types
  - Pros: flexibility
  - Cons: likely to be influenced by the host language
III. ANALYSIS
Example

{
  "name": "Evgeni V. Plushenko",
  "birth_date": {"year": 1982, "month": 11, "day": 3},
  "best_scores": [261.23, 91.30, 176.52],
  "status": {
    "verified": true,
    "locked": false,
    "external_record": null
  }
}

Could I have written day, month, year instead?

Could I have written 3.0 instead?

Is the trailing zero important?

Could I have omitted it?
Objects

• Some ambiguities:
  • Ordered fields? (RFC 4627: No, Many libraries: Yes)
  • Unique names? (RFC 4627: Probably, Most libraries: Yes)
  • What characters are allowed in field names and how are they compared?

• Common representations:
  • Plain lists / arrays: $O(N)$, ordered
  • Sorted sequences (incl. balanced trees): $O(\log N)$, unordered
  • Hash tables: $O(1)$, unordered
  • Linked hash tables: $O(1)$, ordered
Numbers

- Some ambiguities:
  - \(-0 == 0\)?
  - \(130 == 130.0\)?
  - \(130.0 == 130.00\)? \(130 == 13e1\)?
  - Can we accurately define 0.123456789012345678901?
- Different tools answer these questions differently
- The intersection principle cannot be applied here
- The essential information must be defined explicitly
Strings and Other Ambiguities

- "K" == "\u004b"?
- Can strings contain nil characters?
- Do strings have a maximal length?
- 123 == "123"?
- Are false, null, 0, "", {}, [] distinct?
Design Considerations

• *Explicitness*: We must unambiguously define which JSON details are essential and which are not

• *Determinism*: The same JSON text should denote the exact same information regardless of the environment
  - Any loss of information/precision must be controllable

• *Detail concealment*: The metamodel structure should not expose any inessential information

• *Minimalism*: Only information which is useful to a wide enough set of applications should be included
IV. JSONYA/DM
The Metamodel

• Each *element* has a (distinguishable) kind: *string, decimal, object, array, true, false, or null*

• *Strings*: finite sequences of code points U+0000–U+D7FF and U+E000–U+10FFFF

• *Decimals*: rational numbers with denominators $2^{N5^M}$

• *Objects*: unordered associative arrays whose keys are distinct strings

• *Arrays*: finite sequences of zero or more elements

• *True, false, and null*: no additional information except their kinds
Domain Enumerability

- To formally define the information set, a bijective function \( \text{encode} : \mathbb{N} \rightarrow \text{the set of all elements} \)
- Two JSON texts represent the same element iff they correspond to the same number
- The mapping is based on the Cantor's pair function [6]
- Can also be used to generate test data and for other applications
Implementability (1)

• The information model is designed to follow the core JSON ideas
• For strings and numbers the intersection principle could not be applied
  • The model targets to facilitate determinism instead
• For some environments this model may be too sophisticated
  • Particular limitations can be negotiated explicitly
  • Relayed information must not be inadvertently distorted
The essential defines object model selectors, e.g.:

```java
public interface Element {
    String kind();
    Set<String> keys();
    Element field(String name);
    Element item(int index);
    int size();
    String asString();
    BigDecimal asDecimal();
}
```
Limitations

• The following questions are not answered:
  • How should the unorderness of the `keys()` be achieved?
  • What if a non-existing field or item is requested?
  • How to conceal details available in the used types?
    • E.g.: for Java's BigDecimal 12.0 and 12.00 are different
  • How can the “inessential” information be handled in cases when such is needed?
• Already established technologies may be incompatible with the introduced metamodel
V. EVALUATION
Methodology

- Select the 10 most discussed programming languages according to LangPop.com
- For each of them pick all libraries listed at json.org
- Identify the data model of each library and record its properties, including:
  - Are objects ordered or unordered?
  - What parts of the string or number representation is exposed?
  - What is the supported set of numbers?
  - Are false, null, empty objects and empty arrays distinguishable?
Results

- 63 libraries analysed (C++: 6, C: 9, Java: 18, Python: 4, Haskell: 2, JavaScript: 2, Ruby: 3, C#: 10, PHP: 6, Lisp: 3)
  - More than 11 different integer ranges
  - Almost as much ways to treat non-integers
  - Different handling of strings, empty lists/arrays, nulls
  - Many libraries behaved differently based on platform and runtime version
  - More than half of the libraries treated objects as ordered
- What data-interchange are we talking about then?
Interpretation of Results

- Number handling discrepancy justifies the radical approach of Jsonya/dm.
- Some environments do not fully support Unicode, but there is no suitable substitute
- Unordered objects are more interoperable
- On the negative side:
  - Most libraries could not handle arbitrarily large numbers,
  - Decimal numbers may require additional effort
  - Most libraries used mutable object models, we do not prescribe to efficiently design such
Threats to Validity

- The accuracy of the evaluation may be affected by:
  - All libraries were considered equal, although they vary significantly in features, quality and popularity
  - Some of the libraries may have not been analysed correctly, e.g. used in an incorrect way
  - Some of the libraries may have already changed
Conclusion

- We presented Jsonya/dm — an unambiguous data model for JSON
- Jsonya/dm is aligned with established tendencies and attacks the common causes of discrepancy
- The interfaces of the adhering object models can be simple
- We look forward to integration with some of the already developed JSON tools
- Future work: We need to devise efficient representations for the needs of the various environments
Thank You!

Questions?
References