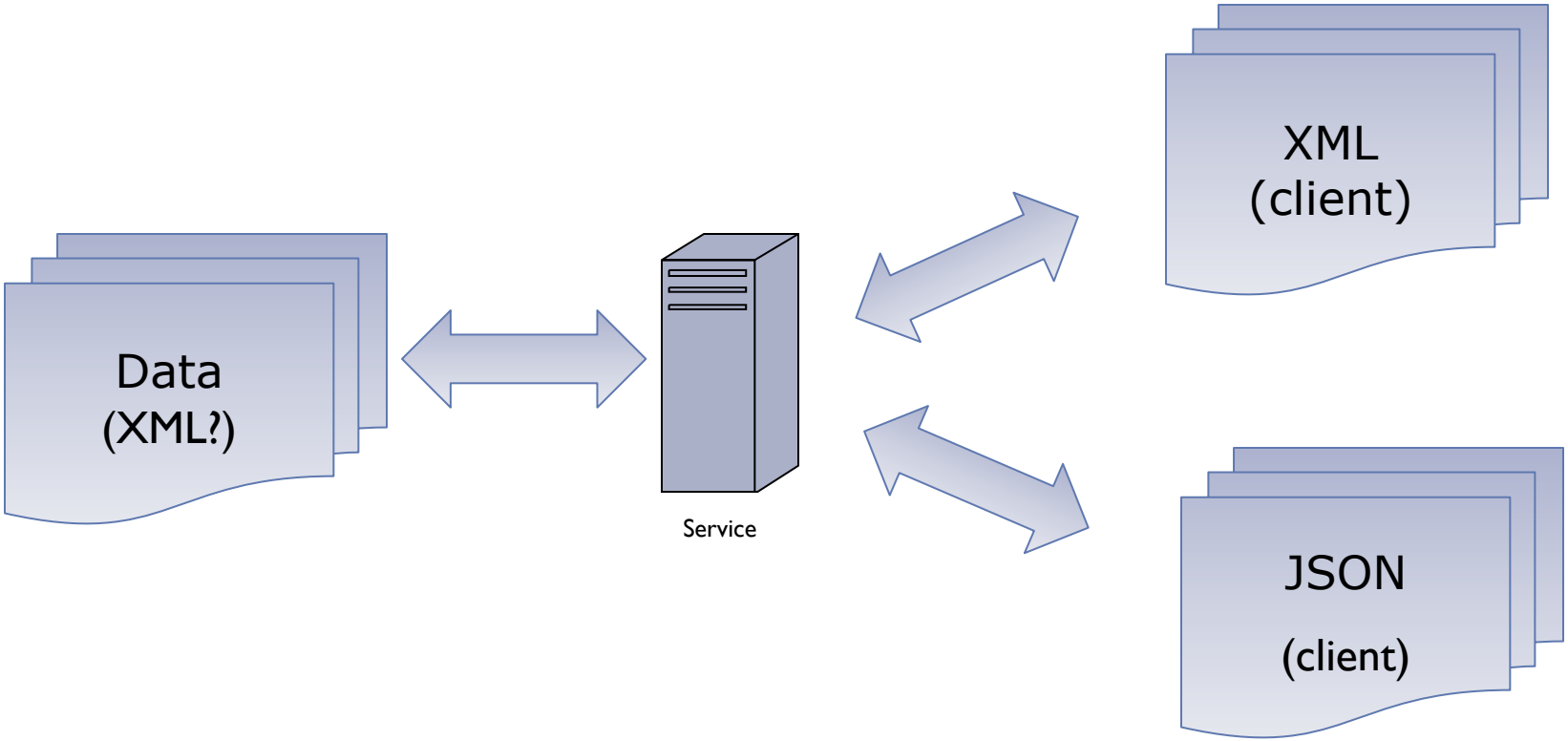
XML ⇔ JSON
Lossless reversible transformation

Epocrates, Inc.

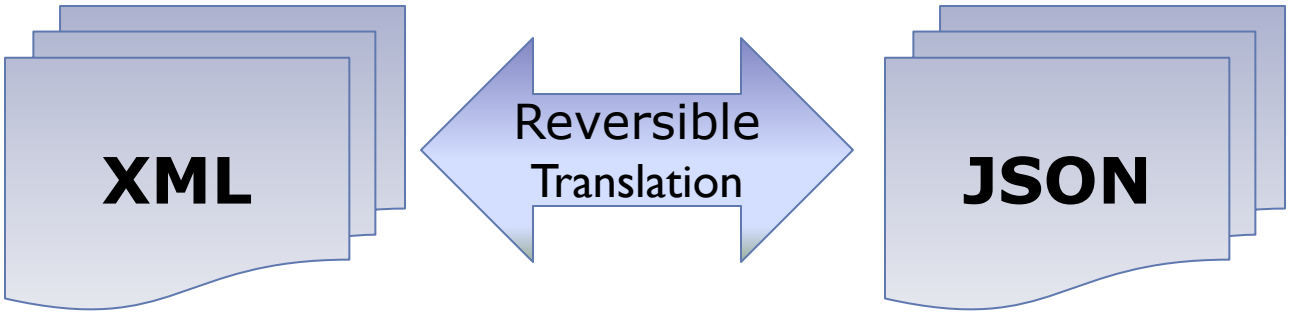
David A. Lee
Senior Principle Software Engineer

Tom Angelopoulos, Staff Engineer

The Problem



The Solution (Easy!)



Not So Easy :(:(:(

- Many mappings of JSON to/from XML
  - Like "Standards" There are so many !
  - Many XML entities not easily translated to JSON
- Few reversible mappings
  - Mapping is usually "One Way"
    or
- "Centric" to one side or another
  - Clean JSON -> Ugly XML
  - Clean XML -> Ugly JSON

- No 'schema' type language that can describe data in both formats
- Mature transformation technology in XML only.

Requirements

- Reversible Translation To/From XML/JSON
  - Round-Trip translation produces equal documents.
- XML and JSON both "Equal Siblings"
  - Markup in "Natural Format" for both XML and JSON
- Use XML tools/technology for JSON Content
  - XPath/XSLT/Xquery
  - XML Databases
  - XML API's
- Use JSON Representation for XML content
  - Load XML source content as JSON into Javascript (browser) client.
  - Portability across many clients (particularly mobile browsers).

Very Nice to Have

- Single Notation to document schema
  - Readable by both JSON and XML developers
- "Real" schema(s) derived from a single source
  - Or a single schema which works for both XML/JSON
- "Natural" representation of native types
  - Atomic types (int/string/double/number …)
  - Array types (JSON arrays)
  - Maps (JSON Objects)

Willing to Sacrifice

- Exact byte-for-byte round-trip
  - Semantic equivalence only
  - Whitespace
- Subset of XML & JSON (hard to map)
  - Namespaces
  - Entities
  - Processing Instructions
  - Identifiers that don't work for both XML & JSON
  - Unnamed JSON objects ?

Atomic Types

JSON Type	JSON Sample	XML Sample
Number	1	1
	-1	-1
	1.5	1.5
	-1.5	-1.5
String	“string”	string
?? Date Time ??		1999-05-21 1999-05-21T13:30:00 ...
Boolean	true	true
	false	false

JSON to XML Structured Types

JSON Type	Sample	XML Type	Sample
Object	{}	Empty Element?	<?/> , <empty/>, ?out of schema?
	{ members }	Element ?	<?>{members}</?> <_unnamed>{members}</_unnamed>
Member – Atomic value	“foo” : value	Element	<foo>value</foo>
	“foo” : value	Attribute	< … foo=”value” >
Member – Object Values	“foo” : { object }	Element	<foo>{object}</foo>
Array – Atomic values	[1 , 2 , 3]	Tokenized Value	1 2 3
		Repeated Elements	<entry>1</entry ??? > <entry>2</entry> <entry>3</entry>
		Wrapped repeated elements	<array> <entry>1</entry> <entry>2</entry> <entry>3</entry> </array>

JSON to XML Array Types (1)

JSON Type	Sample	XML Type	Sample
Array of Objects	[{ “foo” : “bar” }, { “spam” : “bletch”, “hello” : “world” }, 2, “text”]	Wrapped repeated elements with object wrappers	<array> <entry> <foo>bar</foo> </entry> < entry > <spam>bletch</spam> <hello>world</hello> </entry> <entry>2</entry> <entry>text</entry> </array>

JSON to XML Array Types (2)

JSON Type	Sample	XML Type	Sample
Array of Objects	[{ “foo” : “bar” }, { “spam” : “bletch”, “hello” : “world” }, 2, “text”]	“Smarter” entries for specific elements	<array> <entry foo=”bar”/> < entry > <spam>bletch</spam> <hello value=”world”/> </entry> <entry value=”2”/> <entry>text</entry> </array>

XML to JSON - Structured Types (1)

XML Type	Sample	JSON Type	Sample
Empty Element	<foo/>	Member	“foo” : {}
			“foo”: “” “foo”: null
Element with atomic types	<foo>text</bar>	Member	“foo”: “bar”
	<foo>123</bar>	Member	“foo”:123
Element with Attributes	<foo a=”b”/>	Object Member	“foo” : { “a”:”b” }

		Attributes object	“foo” : {
			“attributes” : {
			“a” : “b”
			}
			}

XML to JSON - Structured Types (2)

XML Type	Sample	JSON Type	Sample

Element with Children	<foo>
	<bar>123</bar>
	<spam>hi</spam>
	</foo>
		Object Member	"foo" : {
			"bar" : 123
			"spam" : "hi"
			}

Element with repeated children	<foo>
	<bar>123</bar>
	<bar>hi</bar> </foo>
		Array Member	"foo" : {
			"bar" : [123 , "hi"]
			}

		Child Objects Array	"foo" : [{ "bar" : 123 }, { "bar" : "hi" }]
Element with mixed repeated children	<foo> <bar>123</bar> <bar>hi</bar> <spam>there</spam> <bar>foo</bar> </foo>	Child Objects Array	"foo" : [{ "bar" : 123 }, { "bar" : "hi" }, { "spam" : "there" }, { "bar" : "foo" }]
Element with mixed content	<foo>Hi <B>there</B>World </foo>	Child mixed object array	"foo" : ["hi", { "B" : "there" }, "World"]

XML to JSON - Structured Types (3)

XML Type	Sample	JSON Type	Sample
Element with Attributes and children	<foo a="b"> <bar>hi</bar> </foo>	Object Member	"foo" : { "a" : "b" "bar" : "hi" }

	<foo a="b" b="c">
	<a>Another
	<bar>hi</bar>
	
	<bar>there</bar>
	</foo>
		Object children for elements and attributes
		Attributes can be an object, children must be an array
			"foo" : {
			"_attributes" : {
			"a" : "b" , "b" : "c" },
			"_children" : [
			{ "a" : "Another"} ,
			{ "bar" : "hi" },
			{ "bar" : "there" }]}

		Anonymous arrays for attributes and elements	"foo" : [{ "a" : "b" , "b" : "c" , }, [{ "a" : "Another"} , { "bar" : "hi" }, { "bar" : "there" }]]

Goal #1 – A common type documentation

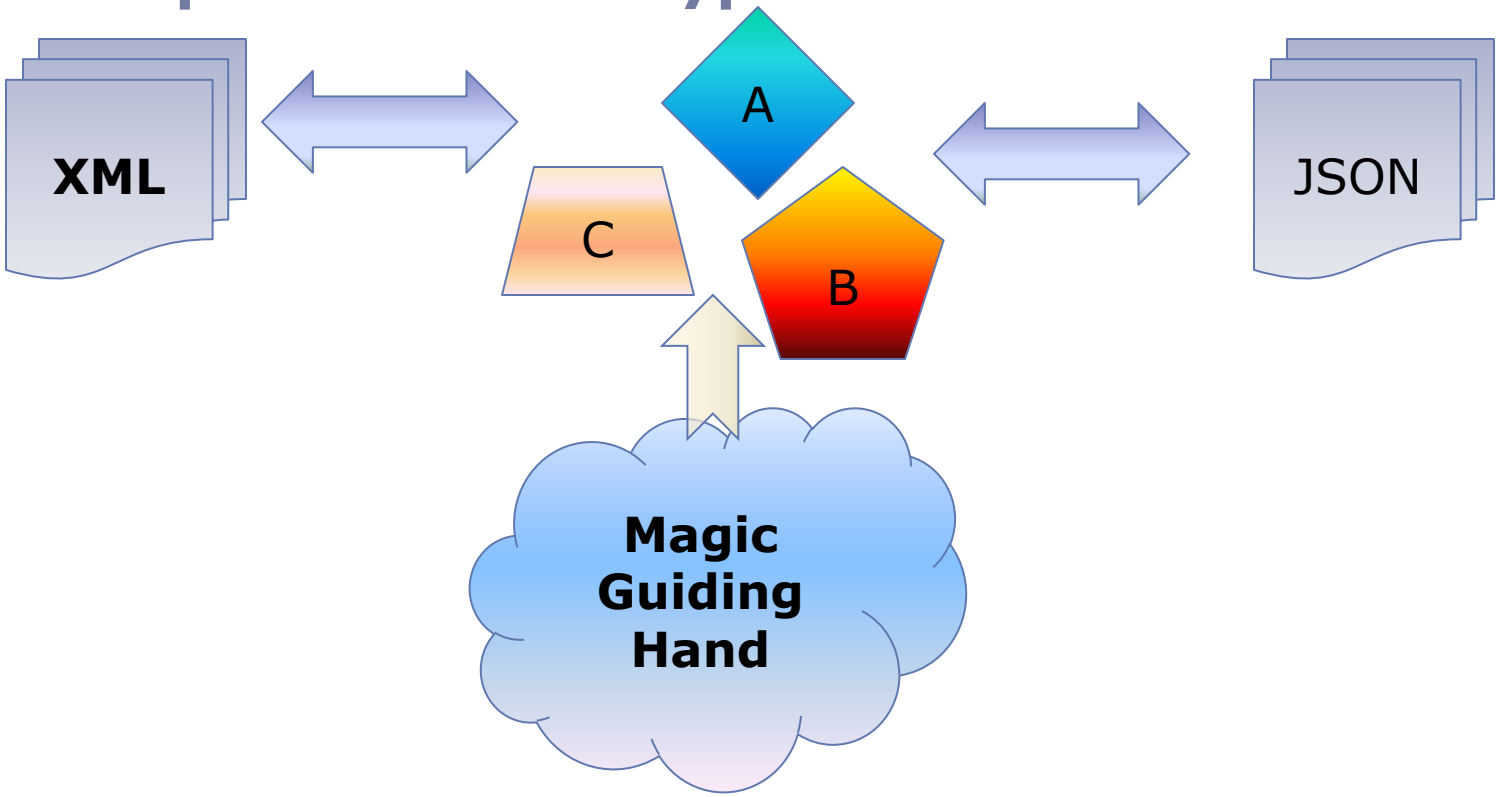
- Use a single Type Description (schema) for describing JSON and XML documents.
  - Human Readable
  - Machine Readable (stretch goal)

- Could XML Schema (some variant) Work ?
  - XSD ? RelaxNG ?
- JSON Schema ?

?? !! Discussion !! ??

Goal #2 – Lossless reversible translation

- If we have a universal schema then we should be able to implement a lossless reversible translation on a per document type basis.



Conclusion

With a lossless reversible translation what do we achieve ?

- **Data transparency between XML & JSON**
- **JSON processing with XML Tools**
- **Serve and consume XML and JSON as equivalent "equal brothers"**
- **Client programmers and server programmers can each use their format and tools of choice**
- **End of the "JSON vs XML" war !**

Discussion !

- Can this be done at all ?
  With what compromises ?
  What subset of JSON and XML supported ?

- Using Existing technologies ?
  - Or write from scratch ?

- Generally useful or niche case ?

??? !! Discussion !! ???