HTTP-PATCH

for read-write linked data

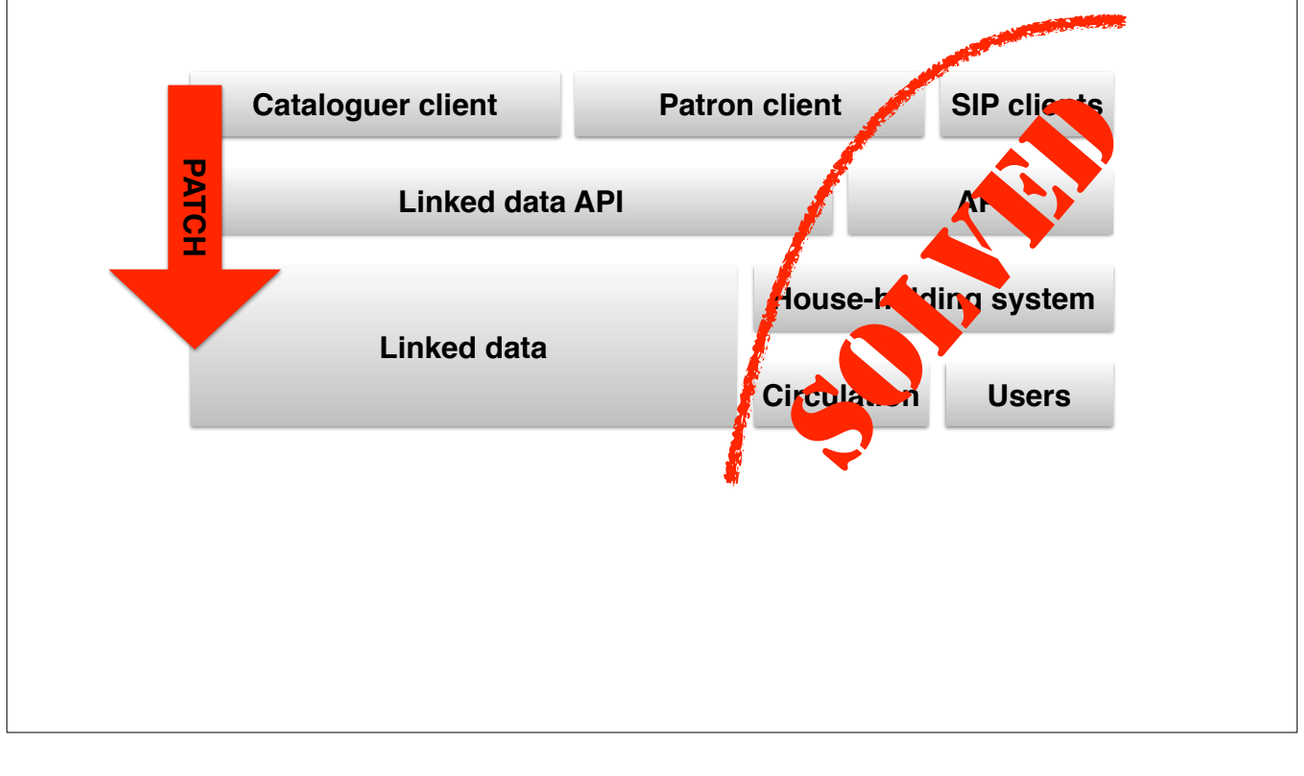
rurik.greenall@computas.com / @brinxmat

DEICHMANSKE BIBLIOTEK

computas

Apart from being a person who likes using all available fonts on one slide…

I work as a consultant for Computas, where I specialise in semantic web and library stuff. I'm not going to give a sales pitch for me or the people I work for — rather, I'm going to talk about the open-source project I work on at Oslo public library.



We're trying to build a new library system for Oslo public library based on linked data and some traditional concepts. The traditional concepts to the right of the image, I'm not even going to touch on; they're largely solved problems like circulation & user other transactions. I'm not going to talk about stuff that is to do with patrons either. However, I am going to talk about HTTP-PATCH, something that goes through the entire stack from the cataloguing point-of-view. Let's start by looking at HTTP methods.

Quick caveat: Since I submitted the abstract for this talk, I became apprehensive about how interesting it would be to a general semantic web in libraries audience — particularly for those who are more interested in metadata. I was given some helpful comments by the programme committee that meant that I have swapped out a good portion of the technical content and tried to push the talk in a direction that shows the practical application of the technology and the implications it has in our development in Oslo.

PATCH is an HTTP method. While HTTP methods like GET and POST are familiar to most people and PUT and DELETE will be familiar to those particularly interested in REST, PATCH seems like a poor relation — little-known and rarely supported.

Show of hands — who has worked with a HTTP-PATCH-enabled service?

Starting with a basic understanding of HTTP methods in order to contextualise why we might need PATCH — from the perspective of bibliographic records, we have POST, which allows us to Create records, GET, which allows us to Read records, PUT, which allows us to Update records and DELETE, which allows us to…well…Delete records.

C

R

U

D

We have the CRUD operations that database people are so keen on talking about

So, with these methods, we seem to have all the possibilities we need. There are however a couple of issues, which I will cover now.

Examples: in JSON, for simplicity's sake. LOL.

Note that my examples in this section are in JSON; I'm currently only trying to show a HTTP-METHOD strategy, so the data is designed to show only differences between client requests and what happens on the server.

Why LOL? Well, we'll see later, maybe.

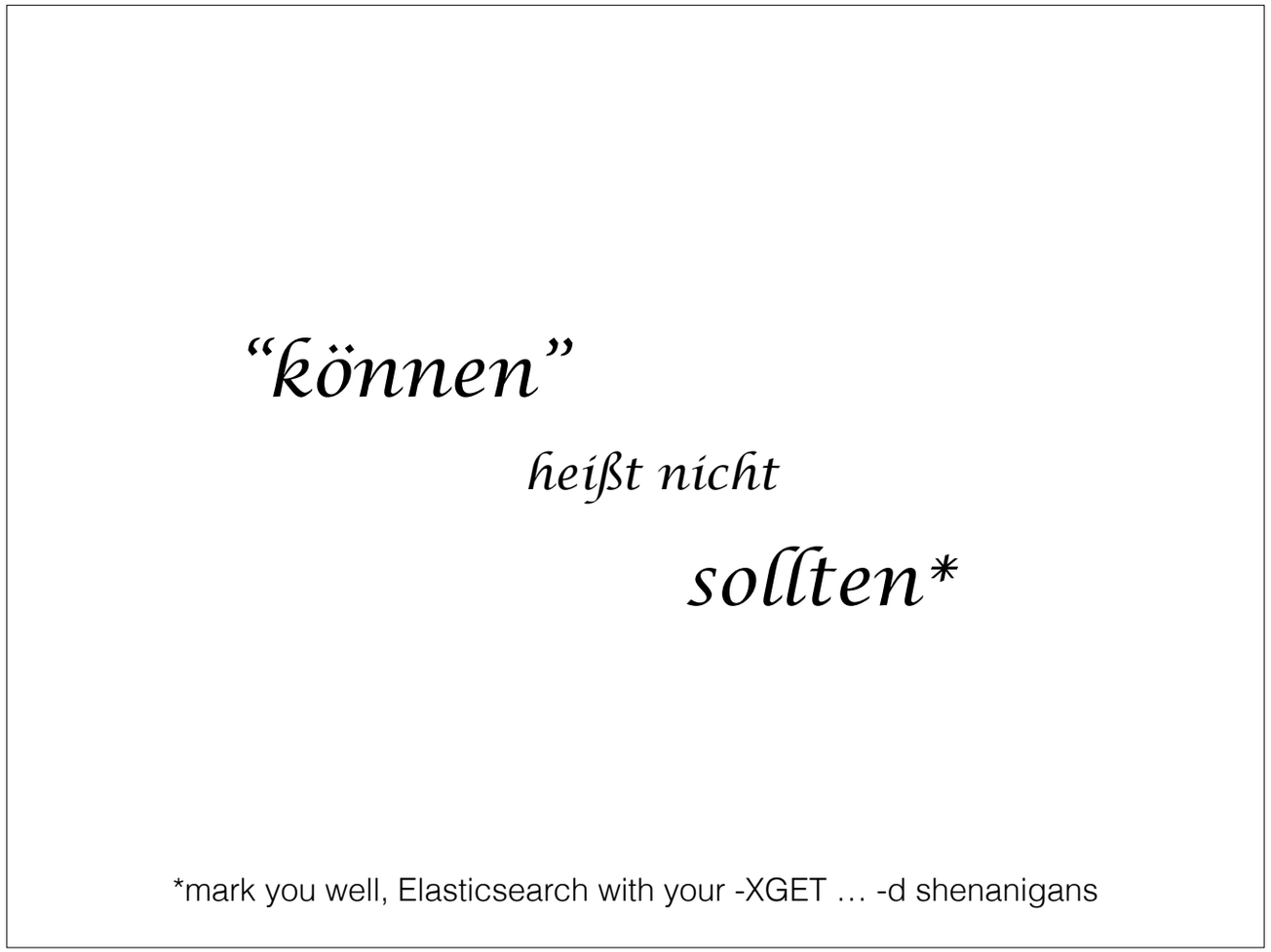
PUT /tier/resource1 →

```
{“animal”:“Cat”}
```

← 201 CREATED

POST and PUT can mean similar things; if you have a resource that you want to create and know the resource’s path, then PUT seems like a usable strategy, while POST is the alternative if you don’t know the path of the resource you’re trying to create.

Note I say create, with PUT…



This, however, seems to be akin to telling the host what it should be doing — and I can’t see any case where I would want a client to tell a server how to organise its resources.

~~PUT /tier/resource1~~ →

```
{“animal”:“Cat”}
```

← ~~201 CREATED~~

POST /tier/ →

```
{
 “animal”:“Cat”
}
```

← 201 CREATED

```
Location: /tier/resource1
```

PUT /tier/resource1 →

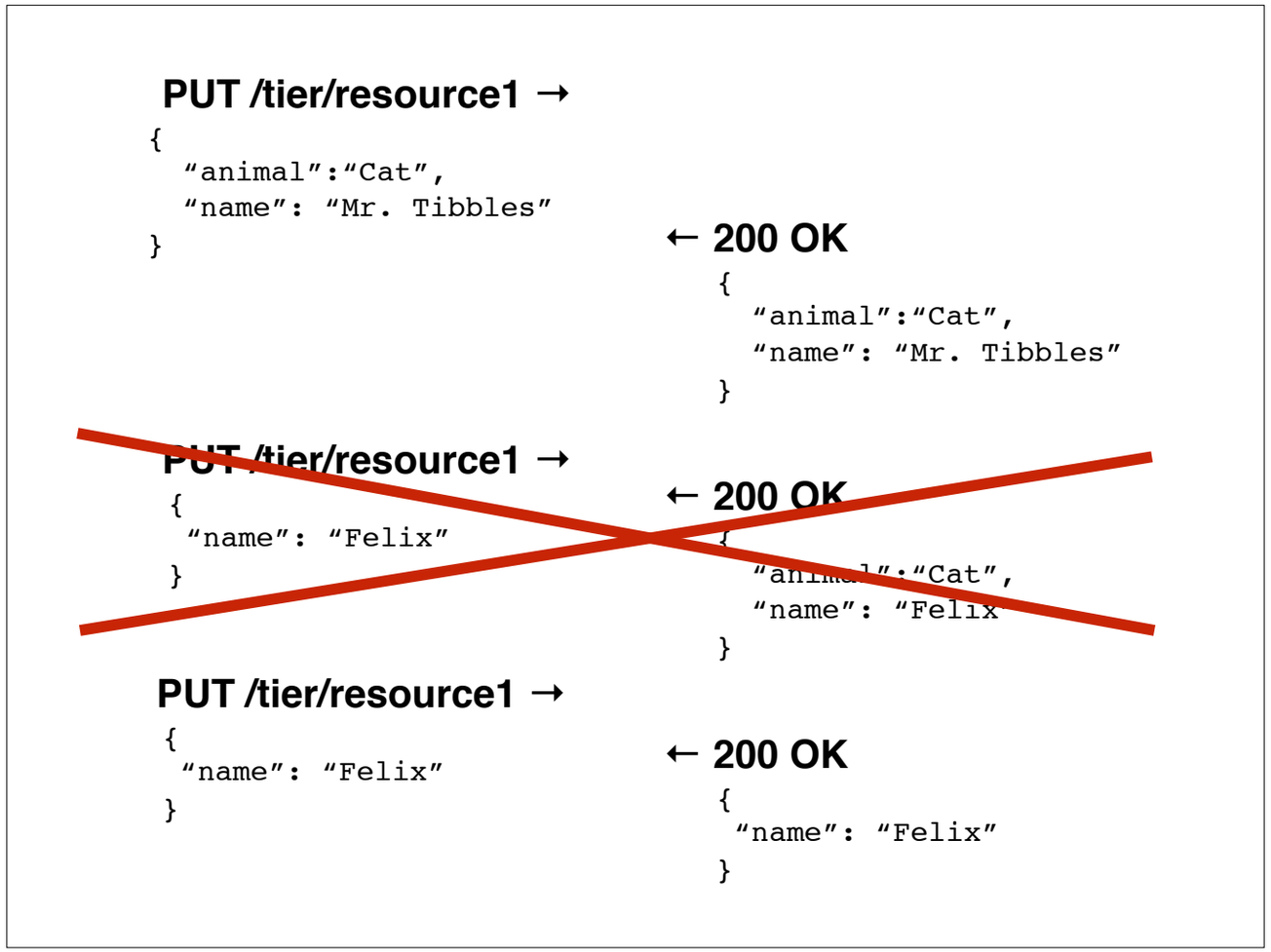
```
{
  “animal”:“Cat”,
  “name”: “Mr. Tibbles”
}
```

← 200 OK

```
{
  “animal”:“Cat”,
  “name”: “Mr. Tibbles”
}
```

So, in a normal situation, I’d like ask for a new resource, to which I can add triples to form a description. Normally, I wouldn’t even post anything in the body, I’d simply request a resource — however, if you’re doing RDF maybe you want to request a typed resource — and requesting a type can then be validated…

I start with a simple request and get a location — our URI — in response — then I PUT a name for the cat onto the now URI-named resource. I get an appropriate response back.



Let’s look at that again, I put the data “animal: cat” and “name: Mr. Tibbles” and got a response with a body that matches (note that the server can manipulate the data, so the response doesn’t actually have to match the input; the terms might be mapped to an ontology or something).

Now I realise that I put the wrong name in, it isn’t “Mr. Tibbles", it’s “Felix”; so I PUT this data…and get the response…actually, no, I don’t.

If you use PUT, it replaces existing resources in their entirety, which wasn’t our necessarily intention, we wanted to add some data, without doing anything with the entire resource.

We intended to update the
resource by adding information,
not by replacing the entire thing

Now, you might point out that this is an unlikely scenario because when we update a resource, we are updating the entire resource — but I think here we need to remember that what we're interested in is updating not a "record", but a resource.

You might see this as splitting hairs, and perhaps not just a little coloured by an interpretation.

But I reckon that the other interpretation is pretty coloured too — just not by RDF, but by a record — or a even a document — concept that doesn't belong to RDF or at least not the linked data model we're using.

When we think of a collection of triples that describes a resource, we think of the data in very simple terms — it is a description of the resource and nothing else. The concept of document or record doesn't really pertain because — while RDF can be represented as documents and probably even records, it isn't what we're doing. We're doing something much simpler — letting RDF just be descriptions that are collections of triples that say something the same resource.

Now that we understand that, we also see why we might not want to update the entire description, but rather add/remove selected triples.

And it is at this point that we can actually talk about a solution to this problem — PATCH.

Why PATCH

- Update a resource, not replace a resource
- Responsive interface without Evil trickery™
- Very simple, predictable sequence for resource creation/deletion and update
- Lightweight, but powerful
- Allows us to explore The Exciting World of Linked Data™ without the baggage of Traditional, Catalogue Record-Based Approaches™

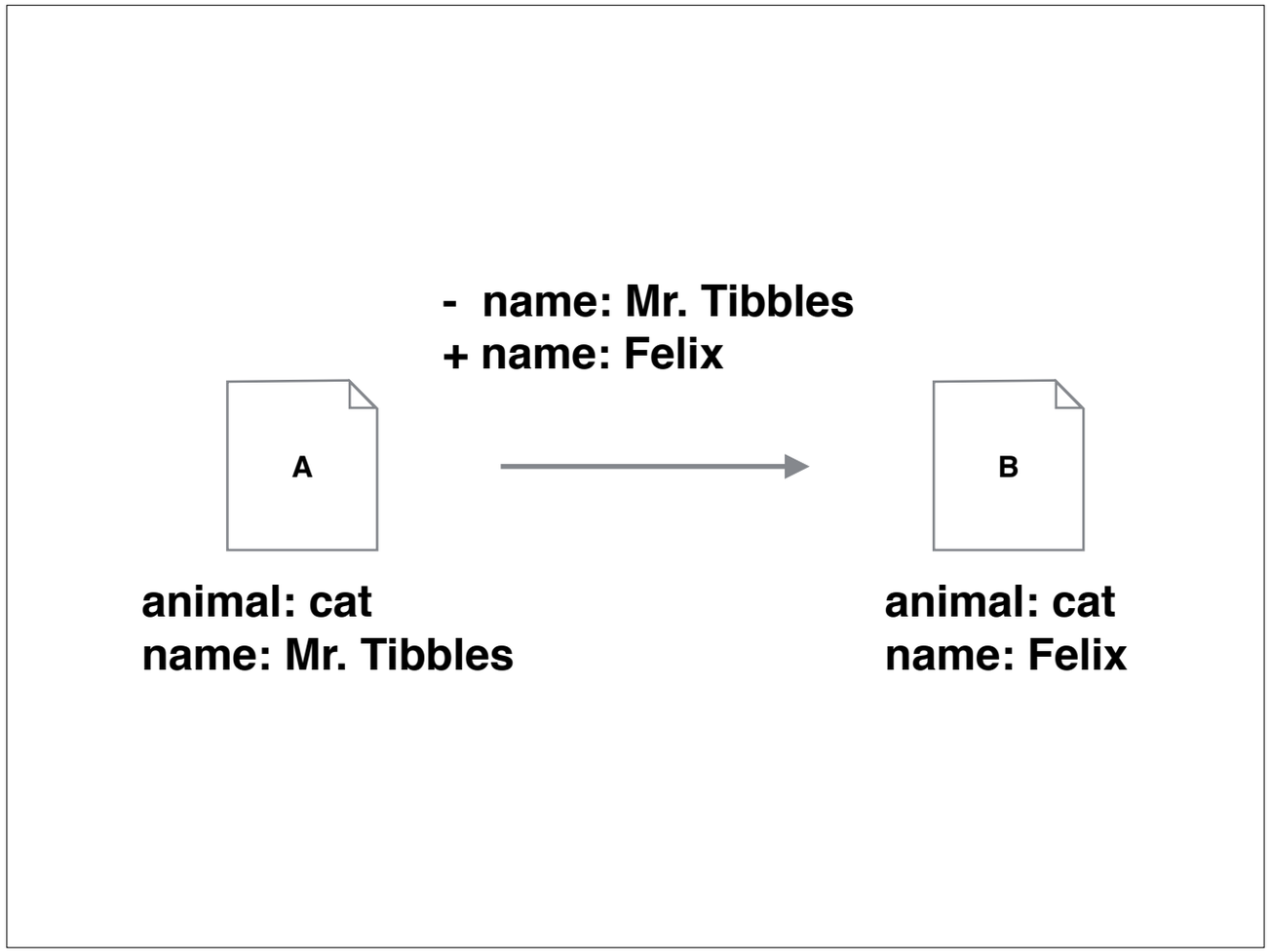
Let's first ask "Why?" I think the answer is that you're doing what you imagine you're doing; updating something, not replacing it. And you're achieving it without having to resort to doing silly things with put/post that make REST-bigots weep.

You're also providing a simple and predictable sequence for creating, updating and deleting resources, which makes for happy developers and better software.

It's also lightweight an yet does what is needed; you're not transferring masses of data for each update.

It also allows us to open up the way we interact with the metadata in the cataloguing workflow.

But, how does it work?



Now, PATCH is a concept derived from the common understanding of "patch"; I say common, but I mean computer science. A patch is basically a description of differences between two states of a resource.

In terms of our previous example, we describe the change we want to make — remove Mr. Tibbles and add Felix. We create a patch that says minus tibbles, and plus felix.

Easy.

Easy.

That’s not a word I’m allowed to use, for obvious reasons.

There is, in fact, no widely implemented way of PATCHING RDF.

Existing approaches: JSON-PATCH

```
{
   "op": "add",
   "path": "/a/b/c",
   "value": "foo"
}
```

https://tools.ietf.org/html/rfc6902

There is, however, an RFC for JSON-PATCH, a patch format for JSON documents.

JSON-PATCH describes various patch operations that can be performed specifically on JSON documents. Let’s look at this.

Existing approaches: JSON-PATCH

```
{
    "op": "add",
    "path": "/a/b/c",
    "value": "foo"
}
```

We have an operation, which is “add”, it could also have been “remove” or replace, move, copy, test.

Existing approaches: JSON-PATCH

```
{
   "op": "add",
   "path": "/a/b/c",
   "value": "foo"
}
```

https://tools.ietf.org/html/rfc6901

We have the path that is to be updated, selecting the node in the document that need adding

Existing approaches: JSON-PATCH

```
{
    "op": "add",
    "path": "/a/b/c",
    "value": "foo"
}
```

And we have the content that is to be added.

Existing approaches: JSON-PATCH

- JSON-oriented
- Resource is JSON-document
- supported operations:
  - add, remove, replace, move, copy, test
- JSON pointer for selectors [https://tools.ietf.org/html/rfc6901]

JSON PATCH is widely implemented (ha-ha) for PATCH, but it is oriented towards JSON resources.

Existing approaches: XML patch

```
<add sel=“doc">
   <foo id="ert4773">This is a new child</foo>
</add>
```

https://tools.ietf.org/html/rfc5261

Similar to JSON-PATCH, there is XML-PATCH…

Existing approaches: XML patch

```
<add sel=“doc">
   <foo id="ert4773">This is a new child</foo>
</add>
```

https://tools.ietf.org/html/rfc5261

Again, we can “add”

Existing approaches: XML patch

```
<add sel="doc">
   <foo id="ert4773">This is a new child</foo>
</add>
```

https://tools.ietf.org/html/rfc5261

We select the node we want to PATCH

Existing approaches: XML patch

```
<add sel=“doc">
   <foo id="ert4773">This is a new child</foo>
</add>
```

https://tools.ietf.org/html/rfc5261

And add a new child.

Existing approaches: XML patch

- Diffing XML using XPath
- XML-oriented
- Supported operations:
  - add, replace, remove
- XPath selector language

https://tools.ietf.org/html/rfc5261

It’s a bit scary because we’re using XML tools and we’ve been there before.

Issues with existing approaches

- Few implementations
- Implement things that we will never use because we're not updating JSON/XML documents
- We're focussed on RDF

From my point of view, we have issues.

There are few implementations and those that there are are great if you like Java abandonware.

Because our workflow is oriented towards triples, we view descriptions as sets of triples; the extent to which documents come into things is when the descriptions are serialised. We really don't think of the data in that way. I'm sure we could have had a document-oriented workflow, but we have wanted to move forward from that — we have used document formats in libraries since the 1960s and it has done us no real favours.

The RDF focus is, I think, the killer here. You can't really do RDF if you're using a document format. Sure, you can do some stuff, but I think it limits you intellectually, always having to think in terms of the serialisation you're using. It seems counter-intuitive when you're dealing with triples as a basic format.

…but RDF

- Resources are not JSON/XML documents
- RDF doesn't require more than add/delete
- RDF triples are much simpler than either JSON or XML
- Less functionality required, easier to implement

Yes, we could have used a JSON or XML interface with mappings to RDF using some manipulations on the server, but I don't want to have to document this or do it in practice. I want something simpler.

RDF is really simple, we don't need a large number of functions, add/delete suffices.

…but JSON-LD

- The complexity of patching JSON documents
- Added weight of JSON-LD on top of JSON
- Would it work out-of-the-box?
- Do we want a pure JSON system?

But we have JSON-PATCH, can’t that be used with JSON-LD?

We bring vast complexity with JSON-LD; JSON-LD isn’t plain-old JSON either. Would it work out of the box without creating invalid JSON-LD?

The answer to the last question, after a year of JSON-LD misery, is no bloody way.

Existing approaches to patching RDF

- SPARQL Update
- SPARQL Patch
- Turtle Patch
- RDF Patch

There are also approaches to patching RDF that are in the pipeline, none of which are standardised for various reasons; I'll provide a quick overview of them here.

SPARQL Update

```
INSERT DATA  {
  <http://example.com/a> <http://example.com/name> “Kim” .
}

DELETE DATA  {
  <http://example.com/a> <http://example.com/name> “Kim” .
}
```

In essence, SPARQL UPDATE provides PATCH-ing via INSERT and DELETE keywords.

This is useful; but as a PATCH format, it isn’t a simple approach and opens for many other features we don’t need in a PATCH.

SPARQL Patch

SparqlPatch is a subset of SPARQL 1.1 Update with only a DELETE, INSERT and WHERE clause. The WHERE clause is restricted to one Basic Graph Pattern with no property paths or variable predicates.

SPARQL Patch addresses these concerns; but is still SPARQL.

Turtle Patch

TurtlePatch is defined as the syntactic and semantic subset of the SPARQL 1.1 UPDATE language

Turtle Patch provides a turtle-like syntax for PATCH-ing, again based on SPARQL Update.

RDF Patch

A <http://example.org/bob> <http://xmlns.com/foaf/0.1/name> "Robert" .
D <http://example.org/bob> <http://xmlns.com/foaf/0.1/name> "Robert" .

Finally, RDF Patch provides a simple, clean way of PATCH-ing.

Notes

- SPARQL Update is complex; there are some simpler sub-sets of SPARQL Update that make patching possible.
- If I were to use SPARQL Update, I would use SPARQL Update rather than sort-of-SPARQL-Update-in-an-attempt-at-creating-a-simple-patch-language. It is always ugly.
- The remaining options are all rather nasty because of how they handle blank nodes

I have some opinions about all of these things, but all of them share the same issues — They're also unimplemented at the time we started working on this in the languages we use.

We have a much lower bar

- We're consciously avoiding using blank nodes
- We're not looking to support everything in the SPARQL/RDF/LDP standards
- We are attempting to do practical updates of named resources where the resource is a set of triples with no blank nodes
- Prefixes are out-of-scope

We're also attempting to do something much simpler — we're not solving the problems of PATCHing arbitrary RDF, we're fixing a particular problem — updating Linked data in our library system.

Simple patch

- JSON because
  - our editing client is Javascript
  - N-Triples need parsing
  - Turtle is hairy
- Add/Delete triples only
- No bnodes

We decided to follow the lead of JSON PATCH for various reasons; we would have preferred to work with N-Triples generally, but JSON (not JSON-LD) is winning the interface fight right there.

We only need to add/delete and don’t have bnodes because no-one on our team ever wants to use bnodes ever.

Simple patch

- Some cues from JSON Patch
- Some cues from RDF Patch
- Supports:
  - Subject URIs
  - Predicate URIs
  - all the possible object types (no bnodes)

So we followed JSON-PATCH's lead, but we also looked at RDF-PATCH because it matches our needs and ideas.

We ended up with rather verbose objects, but I think that this is unavoidable without unpleasantness in the interpretation of JSON. I am a strong believer in being explicit in data in order to avoid complex code.

And it looks like this:

```
{
    "op": “…“,
    "s": “…",
    "p": “…",
    "o": { }
}
```

This is what our PATCH format looks like, it has an operation that can be ADD or DEL, a subject, a predicate and an object. These concepts should be familiar to those who work with the semantic web.

Note that the object is the only thing that takes a value that is a complex object, the rest are strings. Let’s look at a real-world example.

```
{
    "op": "add",
    "s": "http://example.com/a",
    "p": "http://example.com/b",
    "o": {
      "value": "http://example.com/c",
      "type": “http://www.w3.org/2001/XMLSchema#anyURI"
    }
}
```

Here, we add a simple triple consisting of three URIs. Note that the S and P positions can only ever contain URIs and thus JSON’s strings are subject to no interpretation, while the O is always interpreted — and here we specify a type, otherwise it will be interpreted as an RDF 1.1 plainLiteral).

```
{
    "op": "add",
    "s": "http://example.com/a",
    "p": "http://example.com/b",
    "o": {
        "value": "Text"
    }
}
```

And here we see exactly that case, an rdf:plainLiteral.

```
{
    "op": "add",
    "s": "http://example.com/a",
    "p": "http://example.com/b",
    "o": {
        "value": "Text",
        "lang": "en"
    }
}
```

A language literal is expressed by adding a langstring to a plain literal

```
{
    "op": "add",
    "s": "http://example.com/a",
    "p": "http://example.com/b",
    "o": {
        "value": "2",
        "type": "http://www.w3.org/2001/XMLSchema#integer"
    }
}
```


We can also have other datatypes; as we saw before, xsd:anyURI is interpreted to mean that the object is a URI.

```
[
   {
      "op": "del",
      "s": "http://example.com/a",
      "p": "http://example.com/b",
   "o": {
     "value": "http://example.com/c",
     "type": “http://www.w3.org/2001/XMLSchema#anyURI"
   }
   },
   {
      "op": "add",
      "s": "http://example.com/a",
      "p": "http://example.com/b",
   "o": {
     "value": "http://example.com/c",
     "type": “http://www.w3.org/2001/XMLSchema#anyURI"
   }
   }
]
```

We can combine multiple patch objects and submit, for example, an add-delete operation.

Recap

- Any subject, predicate in a base object is interpreted as an IRI
- Objects that are not simple strings are explicitly typed are strings
- A patch can be a single object or an array of objects

Just to recap [not to be read].

Implementation

- Java
- JAX-RS
  - PATCH annotation (JAX-RS 2.0)
  - OPTIONS (CORS)
- Patch parser
- Patch object
- Patch handler (creates SPARQL Update queries)

We've implemented this in Java. It's worth noting that the PATCH actually ends up as a simple SPARQL Update query.

Things we learnt

- YAGNI
  - Implementing “everything” is stupid
  - Standards are great, but only the bits we need
- Time is money
  - Fix the problem in the simplest, fastest way, work outwards from there
  - Spend too much time doing something and someone will fix the problem in a different way
  - Doing this without JAX-RS would have been painful
- It works!
  - Does exactly what we thought it would
  - We don’t use PUT, just GET, POST, PATCH and DELETE

Running code is running code. The standardisation track for this may be far away.

Only do what you need to do.

We started implementing Linked data platform, but we only use GET, POST, PATCH & DELETE.

Don’t waste your time developing things you don’t need.

Our cataloguing interface relies upon PATCH.

Way forward

- Building a workflow unfettered by concepts like documents or records
- Starting point is “creator disambiguation”
  - Then “work”
  - Then “publication”
- Triple-by-triple

Our way forward builds on the knowledge we’ve gained, making small steps, incrementally deconstructing and reconstructing the core concepts of bibliographic data.

Our current sketch is that we start with creator disambiguation, link in a work concept and then finally publications. This approach simplifies our cataloguing & makes space for linking entities in an exciting way.

And it’s all done using PATCH, triple-by-triple

Talk to me

- @brinxmat
- rurik.greenall@computas.com

I was told a joke about 100$a

I can’t repeat it