HATEOAS (Summary)

Ref : https://en.wikipedia.org/wiki/HATEOAS

HATEOAS, an abbreviation for **Hypermedia As The Engine Of Application State**, is a constraint of the REST application architecture that distinguishes it from most other network application architectures. The principle is that a client interacts with a network application entirely through hypermedia provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia. By contrast, in some service-oriented architectures (SOA), clients and servers interact through a fixed interface shared through documentation or an interface description language (IDL).

The HATEOAS constraint decouples client and server in a way that allows the server functionality to evolve independently.

Details

A REST client enters a REST application through a simple fixed URL. The media types used for these are standardized . RESTful interaction is driven by hypermedia. A client does not need to understand every media type and communication mechanism offered by the server. The ability to understand new media types can be acquired at run-time through "code-on-demand" provided to the client by the server.

Origins

The HATEOAS constraint is an essential part of the "uniform interface" feature of REST, as defined in Roy Fielding's doctoral dissertation

What HATEOAS actually means Blog

Ref : https://community.oracle.com/blogs/mkarg/2010/02/14/what-hateoas-actually-means

Flashback

In his dissertation Roy Thomas Fielding explained RESTful architecture (actually it seems that it even *introduced* the word REST), including **hypermedia as the engine of application state (HATEOAS):**

*"**The next control state of an application resides in the representation of the first requested resource**, ... The application state is controlled and stored **by the user agent**... anticipate changes to that state (e.g., **link maps** and prefetching of representations) ... The model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions **in the current set of representations**."*

Okay, so what does that mean and why is most of that wrong what currently is discussed as proposed implementations of HATEOAS?

To understand Fielding's above explanation, we have to remember what his dissertation was about. Fielding was a contributor to theHTTPstandard. In his research he discovered that the method of operation of the world wide web can be abstracted to a general architecture he called REpresentational State Transfer (REST). The thesis behind REST is: Since the WWW is working and scaling so perfectly, while REST is the WWW's architecture, REST itself will be working and scaling well in other domains too, possibly outside of the WWW. In fact he is true, which is why we all are so crazy about REST these days. In detail he identified four key factors that REST is comprising:

*"**REST is defined by four interface constraints:identification of resources; manipulation of resources through representations; self-descriptive messages; and,hypermedia as the engine of application state**."*

Speaking in techniques of the WWW (which is the implementation mostly used to apply the abstract idea of REST to real, physical applications), those four core ideas actually would be the combination of: Using **URI**s to transfer**MIME-typed documents** with **GET / PUT / POST / DELETE** (like their counterparts SELECT / UPDATE / INSERT / DELETE would apply the same idea to SQL) and …? It is the last ellipsis this blog entry (and HATEOAS) is about.

WWW := URI + HTTP + MIME + Hypermedia

Remember

REST is defined by four interface constraints:

1. ***identification of resources;***
2. ***manipulation of resources through representations;***
3. ***self-descriptive messages;***
4. ***and, hypermedia as the engine of application state.***

What are HATEOAS and Hypermedia?

***HATEOAS* is the short form of "*hypermedia* as the engine of state**", as we learn from the disseration. But what does it mean? Let's start with "state". "**State" means the current status of the sum of information found in the system at a particular point in time. For example, if we have an order, it will have (at least) two states: Either it is sent to the customer, or it is not (certainly "state" is neither restricted to a single variable nor a particular type like boolean**; **typically state is a complex construct of several informations**). So what is an "engine of state"? As the example shows, most objects typically have not statically one state for an infinite time, but will change its state from time to time. **An order was not sent to the customer, then got sent, so its new state now is "sent" now. It*transitioned* it's state due to an *action*. The move from one state to another is called "state transition**" and the part of the system that controls the state transitions (i. e. applies a rule set defining what action will result in which state transition, e. g. "if current state is 'new' and action is 'wear' then new state is 'used'") is called a *state engine*.
So now that we know what a state engine is, let's look at hypermedia:

Hypermedia is used as a logical extension of the term hypertext in which graphics, audio, video, plain text and hyperlinks intertwine to create a generally non-linear medium of information. (from WIKIPEDIA)
Or in clear words: **If two or more documents are related *by a link*, this media is called hypermedia**. But this is not what hypermedia in the inner sense of the WWW means, so let's once more cite Mr Fielding's dissertation:
*"The model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions **in** the current set of representations."*
The bold "**in**" is essential to understand what actually *is* HATEOAS and what is *not*:*Only* if the alternative state transitions are found**in** the representations (i. e.**in** the MIME-typed documents that actually renderstate, e. g. XML, HTML or PDF documents, audio files or video streams) -but *not* aside or just near of them- then HATEOAS is true. Why? Because exactly that is what the word HATEOAS itself tells:
HATEOAS := **Hypermedia** as the engine of state

Hypermedia as the state of transtition. It is neither "*the transport protocol* as the state of transition", nor is it "*something beside the representation* as the state of transition". It is clearly *hypermedia* and nothing else and even more clearly it is exactly *the representation*. And the representation is *only* the MIME-typed document but not any headers, URIs, parameters, or whatever people currently are discussing. Unfortunately, Mr Fielding used two divided sentences to explain the concept. It would be much clearer and free of discussion if he would have written what he actually meant to say:
HATEOAS := **hypermedia documents** as the state of state

Why didn't he do so? Let's again check his background: He analyzed the WWW, which comprises mostly of HTML files. **HTML files are hypermedia documents**. They *contain* links (relations) to other documents by <a> elements. I really can understand that he never would have imagined that anybody would ever have the really strange idea to cut away the links out from the hypermedia documents and store them elsewhere and still call that document hypermedia. It would break the whole idea of the WWW if you would remove all <a> elements from all HTML files and store them somewhere else. But that is exactly what people currently are discussing (not for HTML but for different formats)! Moreover, I suspect that he was just used to call even one single HTML file hypermedia due to its theoretical possibility to point to a second one by an <a> element. Or in other words: Fielding's "hypermedia" in fact is *not* any different part of the system but *solely* the document. This is why he wrote in the above cite explicitly

that the state transitions are found **in** the representations. It was just absolutely clear that it makes no sense to have the links outside, as it is not common in the WWW.
Update: BTW, yes, it is RESTful to put links in the "Link" header when using HTTP, as Mr Fielding told me. But don't expect any REST client to take care of that, least of them will, unless there is a good reason (like the entity being an image), just as a browser ignores any <LINK>s in HTML unless it is a relation it *likes* to handle (like RSS feeds). So it is valid, but of potential risk to do so.

What to learn from Mr Fielding?

There is only and exactly one valid implementation of HATEOAS: Having **links inside of the document** (or, *with care*, in the "Link" HTTP header, if HTTP is used).
Just like HTML files link to each other with <a> tags, or just like XML files link to each other with XLink, *all* state transfer has to be done *solely* as a reaction to a link found *inside* of a representation (i. e. inside of a MIME-typed document). Any other technology, like passing the state, possible actions or links outside of the representation, e. g. in HTTP headers etc., is *by definition* **not** HATEOAS.
Moreover, the weird idea of having explicit URIs solely for state transition without passing any document in or getting any document back, is **not** HATEOAS. Looking once more at the concept of the WWW, there typically are no such "pure action links". The typical state transfer in HTML is done by either passing a modified version of the document, or by passing a form containing information to be merged into the new state. But*never* will it be HATEOAS to invoke a link without passing new state. Why?
Once more, the dissertation is providing an absolutely clear an unambiguous definition:
"The application state is controlled and stored by the user agent … freeing the server from the scalability problems of storing state ..."
When you are trying to execute a typical "solution" currently widely discussed

```
POST http://resource/id/the-action-to-execute
```

then this will ask *the server* to do the state transition. **This is *in diametral contrast* to the above cite of the dissertation** which clearly says that it is *not the server but solely the client* that stores and controls state and **thus is explicitly*not* HATEOAS**. It just makes no sense to call a special URI to trigger *a server side* action if *the client* already has switched to the new state. And it shall be *the client* that does the switch but *not the server*. You can just call the "normal" URI of the resource and pass the already state-transitioned version of the resource. By doing so, the server will implicitly learn the new state. No need to tell the server which action was responsible for that (if that would be needed from a business aspect, then it *must* be part of the uploaded document but not of the used transfer protocol)!

So how to do HATEOAS the right way?
In short: Let your client read the possible actions as URIs out of the received document, set up the modified document and then invoke those link.
No clear yet? Let's make an example. We have an order that is not shipped, and now we want to issue shipping. So the client GETs the order (e. g. as an XML document) and inspects its content. Inside it finds XLinks for various actions. One of them is for shipping. The client puts together the needed information from shipping and POSTs that to the XLink's target. In RESTful terms, what we do is creating a new instance of a shipping instruction by uploading an XML containing shipping details (typically containing the order document itself as a child element, or more simple, its URI as a reference). How does our client know what of the contained XLink URIs the one for shipping is? This is a case of definition. XLink for example comes with the role attribute, so we could defined that it must be the one with the role set to "ship".
It's just similar in case your client is a WWW browser and your document format is HTML: You download the order as a HTML file, containing <form> links. You click on the button that has the title "ship" which performs an action of PUT, containing the shipping details you filled in manually. How did you now which button you must press? You don't. You just guessed well or you had clear instructions.
So to come back to XML, it is just a question of clear instructions, which means, definitions: Your client just needs to know that the XLink to search for is called "ship". There is no technical solution. At least this piece of information

must exist. If man does not know that the english word for sending something away is "ship", he wouldn't find the button, too.
And other media types? Well, what will man do when receiving a media file that he has no player for? Nothing! Same for machines. The client needs to be aware of the used media types. It is impossible for the client machine to deal with unknown media types, just as it is impossible for the browser. A good idea would be to use an abstract API that allows to plug in media handlers, just as browsers do.
Tell the truth!
It's not the case that people would not know about all what I wrote above. Most of the people participating in the discussions *have* read and well understood the content of the dissertation. But what they have not understood or what they just *won't believe* (in opposition to Fielding's thesis that an idea that works in the WWW will work everywhere) is that Fielding is just right and that all the problems they had in the past was not caused by REST or HATEOAS but often by not 100% correctly applying it. Also we all are used to apply imperative programming, which means, RPC-style programming. We all are used to call methods from the client and wait for the server to react. This is what we did since the early days of programming and Client/Server, and this is what looks just so easy and simple to do even in RESTful days. But this is neither RESTful not HATEOAS and it is not scalable, so this might be useful and easy to do in many programming projects, but you'll end up with a HTTP based RPC application, not with a RESTful / HATEOAS one.
If you want to gain *all* the benefits of REST, you need to apply *all* four constraints but not just three of them. And this clearly implies *real* HATEOAS in the meaning of the dissertation, not in the interpretation of anybody else besides Mr Fielding himself, and explicitly it doesn't mean HTTP based RPC. If you do not believe that this is the right way, write your own dissertation and fight the thesis "HATEOAS is not feasible". But meanwhile, please stop claiming that it *would* be HATEOAS to have URIs named like verbs or it *would* be HATEOAS to pass state or possible actions as HTTP headers, and whatever strange idea you might find on the web declared a HATEOAS. **This is not HATEOAS**. It is even neither Hypermedia nor RESTful. It is just some use of HTTP that possibly makes your life (currently) easier. Name it as it is, but **don't name it HATEOAS**.
And please, don't ask "How to do server sided workflows and transactions then?". The question is invalid, as it wouldn't be RESTful to have *server sided* workflows. Again, read the dissertation, which says that it is *the client* that modifies and stores state - so it is *the client's* job to run the workflow, either by doing modifications to a document directly or by asking *stateless* servers to send back modified documents (single transaction steps) while tracking internally *in the client* what the next step would be, and how to undo it in case of a failing *transaction*. So there*are* workflows and transactions, but the sole control is*on the client side*. Every attempt to control this*server sided* wouldn't be RESTful *by definition*. If you are unable to turn the workflow control from server to the client (I cannot understand why, actually), then don't do REST: You *will* fail.