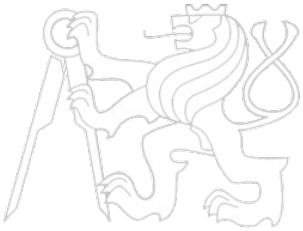
Web 2.0

Lecture 2: REST Architecture 1

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • http://vitvar.com



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/w20

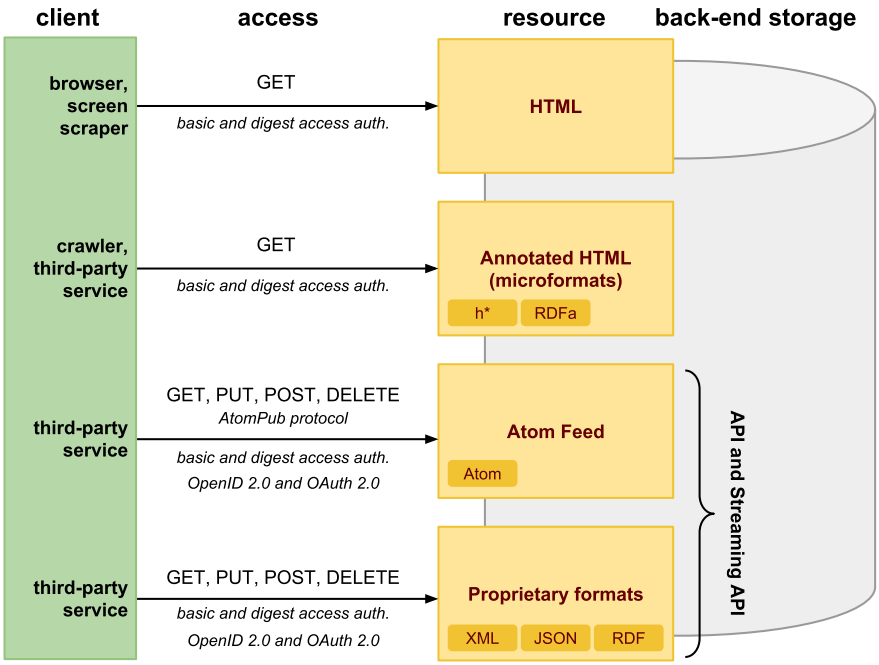
Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Modified: Sun Mar 04 2018, 22:10:55
Humla v0.3

Overview

- Overview of Formats and Protocols
- Introduction to REST
- Uniform Resource Identifier
- Resource Representation
- HATEOAS

Data on the Web



Data Syntax, Structure and Semantics

Semantic Web Layered Cake
syntax and formal semantics

Annotation mechanisms RDFa		
Domain-specific semantics (ontologies)		
Reasoner	OWL Full	Rule Languages
	OWL DL	
	OWL Lite	
	RDF Schema (RDFS)	
SPARQL	RDF	
	XML Schema	
XQuery and XPath	XML	N3 / Turtle
	Namespaces	
	URI	
	UNICODE	
HTTP		

Web Data Formats
syntax and semantics (structure)

Annotation mechanisms microformats (hCard, hResume, ...), microdata		
Domain-specific semantics atom extensions, vCard, vResume, ...		
AtomPub	Atom extensions (e.g., GData)	
	Atom	
	XML Schema	JSON Schema
XQuery and XPath	XML	JSON
	Namespaces	
	URI	
	UNICODE	
HTTP		

Overview

- Overview of Formats and Protocols
- Introduction to REST
- Uniform Resource Identifier
- Resource Representation
- HATEOAS

REST

- REST
  – *Representational State Transfer*
- Architecture Style
  – *Roy Fielding – co-author of HTTP*
  – *He coined REST in his PhD thesis .*
    → *The thesis abstracts from HTTP technical details*
    → *HTTP is one of the REST implementation →* ***RESTful***
    → *REST is a leading programming model for Web APIs*
- REST (RESTful) proper design
  – *people break principles often*
  – *See REST Anti-Patterns for some details.*
- REST and Web Service Architecture
  – *REST is a realization of WSA resource-oriented model*

REST and Web Architecture

- Tim-Berners Lee
  - *"creator", father of the Web*
- Key Principles
  - *Separation of Concerns*
    - *→ enables independent innovation*
  - *Standards-based*
    - *→ common agreement, big spread and adoption*
  - *Royalty-free technology*
    - *→ a lot of open source, no fees*
- Architectural Basis
  - ***Identification:** universal linking of resources using URI*
  - ***Interaction:** protocols to retrieve resources – HTTP*
  - ***Formats:** resource representation (data and metadata)*

HTTP Advantages

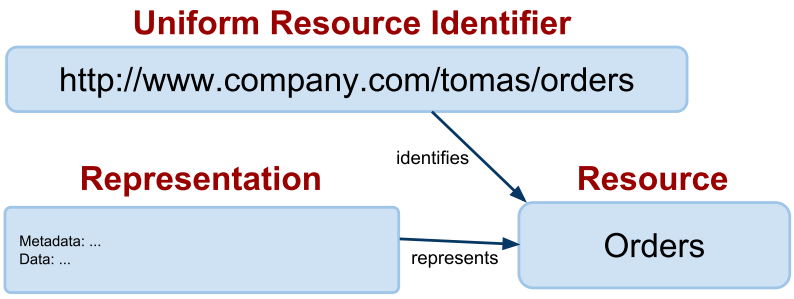
- Familiarity
  - *HTTP protocol is well-known and widely used*
- Interoperability
  - *All environments have HTTP client libraries*
    - *→ technical interoperability is thus no problem*
    - *→ no need to deal with vendor-specific interoperability issues*
  - *You can focus on the core of the integration problem*
    - *→ application (domain, content) interoperability*
- Scalability
  - *you can use highly scalable Web infrastructure*
    - *→ caching servers, proxy servers, etc.*
  - *HTTP features such as HTTP GET idempotence and safe allow you to use caching*

REST Core Principles

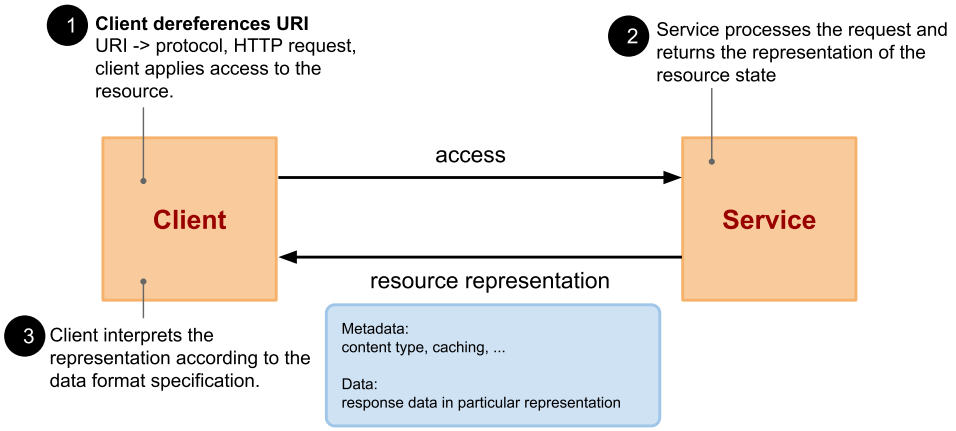
- REST architectural style defines constraints
  - *if you follow them, they help you to achieve a good design, interoperability and scalability.*
- Constraints
  - *Client/Server*
  - *Statelessness*
  - *Cacheability*
  - *Layered system*
  - *Uniform interface*
- Guiding principles
  - *Identification of resources*
  - *Representations of resources and self-descriptive messages*
  - *Hypermedia as the engine of application state (HATEOAS)*

Resource

- A resource can be anything such as
  - *A real object: car, dog, Web page, printed document*
  - *An abstract thing such as address, name, etc. → RDF*
- A resource in REST
  - *A resource corresponds to one or more entities of a data model*
  - *A representation of a resource can be conveyed in a message electronically (information resource)*
  - *A resource has an identifier and a representation and a client can apply an access to it*



Access to a Resource



- Terminology
  - *Client = User Agent*
  - ***Dereferencing URI** – a process of obtaining a protocol from the URI and creating a request.*
  - ***Access** – a process of sending a request and obtaining a response as a result; access usually realized through HTTP.*

Overview

- Overview of Formats and Protocols
- Introduction to REST
- Uniform Resource Identifier
  - *Resources and Application Data*
- Resource Representation
- HATEOAS

URI, URL, URN

- URI – Uniform Resource Identifier
  - *URI only identifies a resource*
    - → *it does not imply the resource physically exists*
  - *URI could be URL (locator) or URN (name)*
- URL – Uniform Resource Locator
  - *in addition allows to locate the resource*
    - → *that is — its network location*
  - *every URL is URI but an URI does not need to be URL*
- URN – Uniform Resource Name
  - *refers to URI under "urn" scheme (RFC 2141)*
  - *require to be globally unique and persistent*
    - → *even if the resource cease to exist/becomes unavailable*

URI

- Definition

```
URI = scheme ":" [ "//" authority ] [ "/" path ] [ "?" query ] [ "#" frag ]
```

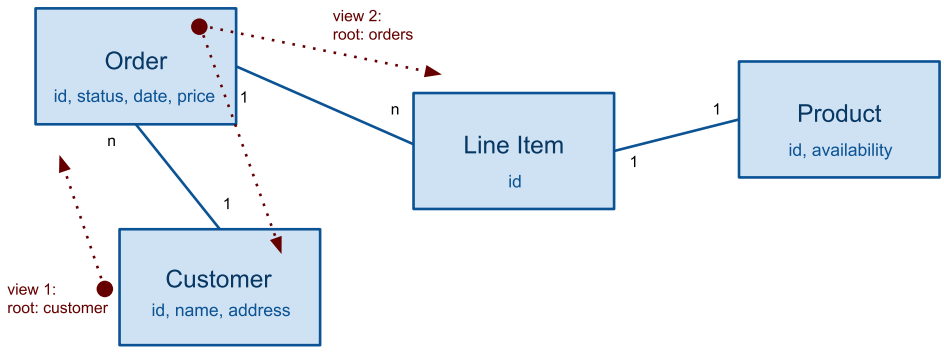
- Hierarchal sequence of components
  - `scheme`
    - → *refers to a spec that assigns IDs within that scheme*
    - → *examples:* `http`, `ftp`, `mailto`, `urn`
    - → `scheme != protocol`
  - `authority`
    - → *registered name (domain name) or server address*
    - → *optional port and user*
  - `path and query`
    - → *identify resource within the scheme and authority scope*
    - → *path – hierarchal form*
    - → *query – non-hierarchal form (parameters key=value)*
  - `fragment`
    - → *reference to a secondary resource within the primary resource*

Overview

- Overview of Formats and Protocols
- Introduction to REST
- Uniform Resource Identifier
  - *Resources and Application Data*
- Resource Representation
- HATEOAS

Resources over Entities

- Application's data model
  - *Entities and properties that the app uses for its data*



- URI identifies a resource within the app's data model
  - `path` – *a "view" on the data model*
    - → *data model is a graph*
    - → *URI identifies a resource using a path in a tree with some root*

Examples of Views

- View 1
  - *all customers:* `/customers`
  - *a particular customer:* `/customers/{customer-id}`
  - *All orders of a customer:* `/customers/{customer-id}/orders`
  - *A particular order:* `/customers/{customer-id}/orders/{order-id}`
- View 2
  - *all orders:* `/orders`
  - *All orders of a customer:* `/orders/{customer-id}`
  - *A particular order:* `/orders/{customer-id}/{order-id}`

⇒ Design issues

- Good design practices
  - *No need for 1:1 relationship between resources and data entities*
    - → *A resource may aggregate data from two or more entities*
    - → *Thus only expose resources if it makes sense for the service*
  - *Try to limit URI aliases, make it simple and clear*

Path vs. Query

- Path
  - *Hierarchical component, a view on the data*
  - *The main identification of the resource*
- Query
  - *Can define selection, projection or other processing instructions*
  - *Selection*
    - → *filters entries of a resource by values of properties*

      `/customers/?status=valid`
  - *Projection*
    - → *filters properties of resource entries*

      `/customers/?properties=id,name`
  - *Processing instructions examples*
    - → *data format of the resource → cf. URI opacity*

      `/customers/?format=JSON`
    - → *Access keys such as API keys*

      `/customers/?key=3ae56-56ef76-34540aeb`

Fragment

- Primary resource
  - *Defined by URI path and query*
  - *could be complex, composed resources*
- Sub-resource/secondary resource
  - *Can be defined by a fragment*
  - *No explicit relationship between primary and sub-resource*
    → *For example, we cannot infer that the two resources are in* `part-of`*, or* `sub-class-of` *relationships.*
  - *Fragment semantics defined by a data format*
- Usage of fragment
  - *identification of elements in HTML*
  - *URI references in RDF*
  - *State of an application in a browser*

Fragment Semantics

- Fragment semantics for HTML
  - *assume that* `orders.html` *are in* `HTML` *format.*

```
1 | http://company.com/tomas/orders.html#3456
```

  ⇒ *there is a HTML element with* `id=3456`
- But:
  - *Consider* `orders` *resource in* `application/xml`

```
1 | <orders>
2 |     <order id="3456">...</order>
3 |     ...
4 | </orders>
```

  - *Can't say that* `http://company.com/tomas/orders.xml#3456` *identifies an order element within the* `orders` *resource.*
  - `application/xml` *content type does not define fragment semantics*

Resource ID vs. Resource URI

- Resource ID
  - *Local ID, part of an entity in a data model*
  - *Unique within an application where the resource belongs*
  - *Usually generated on a server (cf. PUT to update and insert)*
  - *Exposed to the resource URI as a path element*
    `/orders/{order-id}`
- Resource URI
  - *Global identifier, valid on the whole Web*
  - *Corresponds to the view on the data model of the app*
  - *Include multiple higher-level resources' IDs*
  - *Example:*
    `/customers/{customer-id}/orders/{order-id}/`
  - *There can be more URIs identifying the same resource*

Major characteristics

- Capability URL
  - *Short lived URL generated for a specific purpose*
  - *For example, an user e-mail verification*
- URI Alias
  - *Two URIs identifying the same resource*
- URI Collision
  - *Two URIs identifying the same resource (misuse of an URI authority)*
- URI Opacity
  - *Content type encoded as part of an URI*
  - `http://www.example.org/customers.xml`
- Resource versions encoded in an URI
  - *Two URIs identifying the same resource of different versions*
  - `http://www.example.org/v1/customers.xml`
- Persistent URL
  - *URL is valid even when the resource is obsolete*
  - *For example, a redirection should be in place*

Overview

- Overview of Formats and Protocols
- Introduction to REST
- Uniform Resource Identifier
- Resource Representation
- HATEOAS

Representation and Data Format

- Representation
  - *Various languages, one resource can have multiple representations*
    → *XML, HTML, JSON, YAML, RDF, ...*
    → *should conform to Internet Media Types*
- Data format
  - *Format of resource data*
  - *Binary format*
    → *specific data structures*
    → *pointers, numeric values, compressed, etc.*
  - *Textual format*
    → *in a defined encoding as a sequence of characters*
    → *HTML, XML-based formats are textual*

Metadata

- Metadata ~ self-description
  - *Data about the resource*
  - *e.g., data format, representation, date the resource was created, ...*

  1. *Defined by HTTP response headers*
  2. *Can be part of the data format*
     - → *Atom Syndication Format such as* `author`, `updated`, ...
     - → *HTML* `http-equiv` *meta tags*
- Resource anatomy

Content-Type Metadata

- Access
  - *to be retrieved (GET)*
  - *to be inserted or updated (PUT, POST)*
  - *to be deleted (DELETE)*
- Request
  - *HTTP header* `Accept`*, part of content negotiation protocol*
- Response
  - *HTTP header* `Content-Type: type/subtype; parameters`
  - *Specifies an Internet Media Type of the resource representation.*
    - → *IANA (Internet Assigned Numbers Authority) manages a registry of media types and character encodings*
    - → *subtypes of* `text` *type have an optional charset parameter*

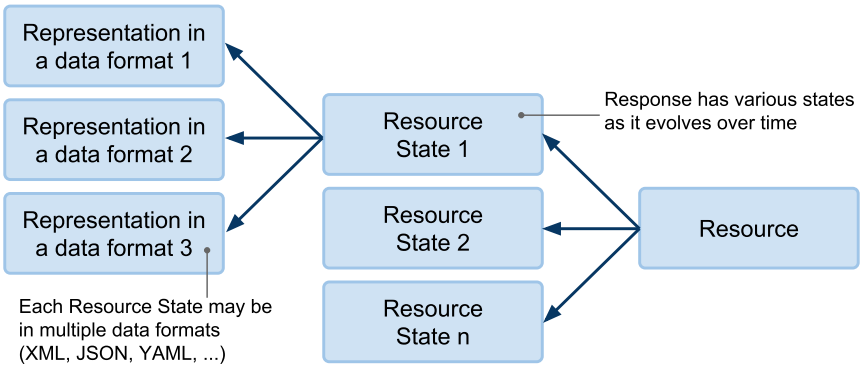
      `text/html; charset=iso-8859-1`
  - *A resource may provide more than one representations*
    - → *promotes services' loose coupling*

Major Media Types

- Common Standard Media Types
  - `text/plain`
    → *natural text in no formal structures*
  - `text/html`
    → *natural text embedded in HTML format*
  - `application/xml`, `application/json`
    → *XML-based/JSON-based, application specific format*
  - `application/wsdl+xml`
    → `+xml` *suffix to indicate a specific format*
- Non-standard media types
  - *Types or subtypes that begin with* `x-` *are not in IANA*
    `application/x-latex`
  - *subtypes that begin with* `vnd.` *are vendor-specific*
    `application/vnd.ms-excel`

Resource State

- State
  - *Resource representation is in fact a* ***representation of a resource state***
  - *Resource may be in different states over time*



- In REST resource states represent application states

Resource State Example

- Time `t1`: client A retrieves a resource `/orders` (GET)

```
<orders>
    <order id="54467"/>
    <order id="65432"/>
</orders>
```

- Time `t2`: client B adds a new order (POST)

```
<order>
    ...
</order>
```

- Time `t3`: client A retrieves a resource `/orders` (GET)

```
<orders>
    <order id="54467"/>
    <order id="65432"/>
    <order id="74567"/>
</orders>
```

- The resource `/orders` has different states in `t1` and `t3`.

Overview

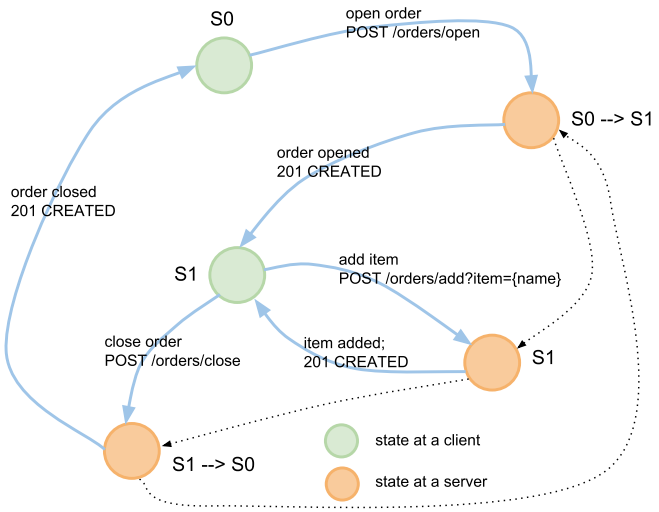
- Overview of Formats and Protocols
- Introduction to REST
- Uniform Resource Identifier
- Resource Representation
- HATEOAS

HATEOAS

- HATEOAS = Hypertext as the Engine for Application State
  - *The REST core principle*
  - ***Hypertext***
    - → *Hypertext is a representation of a resource with **links***
    - → *A link is an URI of a resource*
    - → *Applying an access to a resource via its link = state transition*
- Statelessness
  - *A service does not use a memory to remember a state*
  - *HATEOAS enables stateless implementation of services*

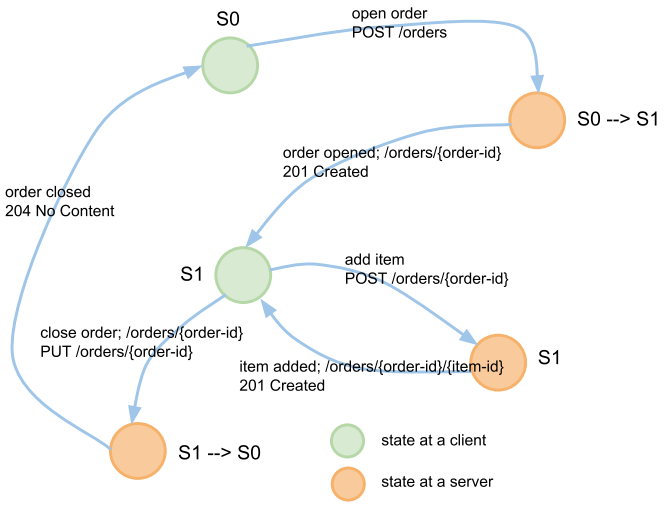
Stateful server

- Sessions to store the application state
  - *Recall HTTP state management in MDW*
  - *The app uses a server memory to remember the state*
  - *When the server restarts, the app state is lost*



Stateless server

- HTTP and hypermedia to transfer the app state
  - *Does not use a server memory to remember the app state*
  - *State transferred between a client and a service via HTTP metadata and resources' representations*

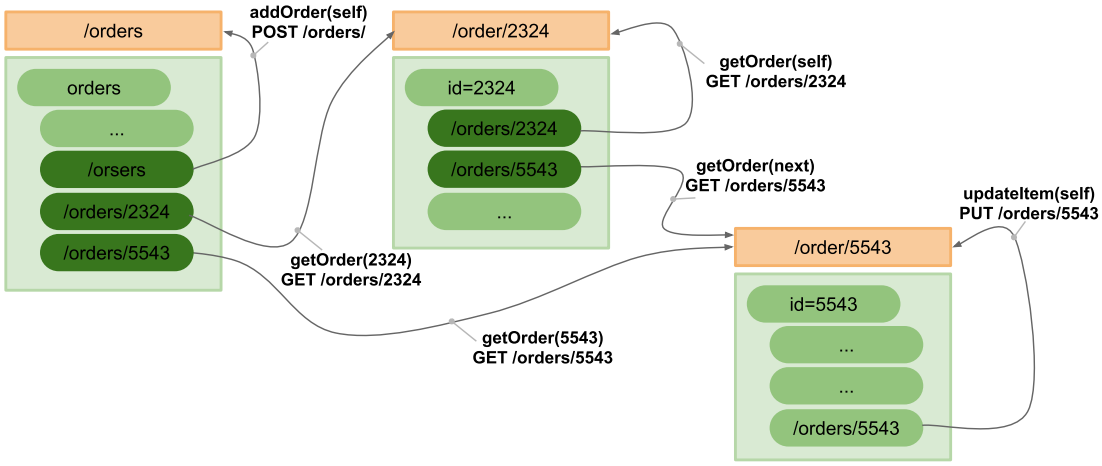


Persistent Storage and Session Memory

- Persistent Storage
  - *Contains the app data*
  - *Data is serialized into resource representation formats*
  - *All sessions may access the data via resource IDs*
- Session Memory
  - *Server memory that contains a state of the app*
  - *A session may only access its session memory*
  - *Access through cookies*
  - *Note*
    → *A session memory may be implemented via a persistent storage (such as in Google AppEngine)*

Link

- Service operation
  - *Applying an access to a link (GET, PUT, POST, DELETE)*
  - *Link: HTTP method + resource URI + optional link semantics*
- Example: `getOrder`, `addOrder`, and `updateItem`



Atom Links

- Atom Syndication Format
  - *XML-based document format; Atom feeds*
  - *Atom links becoming popular for RESTful applications*

```
1  <order a:xmlns="http://www.w3.org/2005/Atom" xmlns="...">
2      <a:link
3          rel="next"
4          href="http://company.com/orders/5543"
5          type="application/xml"/>
6      <customer>Tomas</customer>
7      <items>...</items>
8  </order>
```

  - *Link structure*

    `rel` – *name of the link*

    ~ semantics of an operation behind the link

    `href` – *URI to the resource described by the link*

    `type` – *media type of the resource the link points to*

Link Semantics

- Standard `rel` values
  - *Navigation:* `next`, `previous`, `self`
  - *Does not reflect a HTTP method you can use*
- Extension `rel` values
  - *You can use* `rel` *to indicate a semantics of an operation*
  - *Example: add item, delete order, update order, etc.*
  - *A client associates this semantics with an operation it may apply at a particular state*
  - *The semantics should be defined by using an URI*

```
<order a:xmlns="http://www.w3.org/2005/Atom" xmlns="...">
    <id>2324</id>
    <a:link rel="http://company.com/op/addItem"
        href="http://company.com/orders/2324"/>
    <a:link rel="http://company.com/op/deleteOrder"
        href="http://company.com/orders/2324"/>
</order>
```


Link Headers

- An alternative to Atom links in resource representations
  - *links defined in HTTP Link header, Web Linking IETF spec*
  - *They have the same semantics as Atom Links*
  - *Example:*

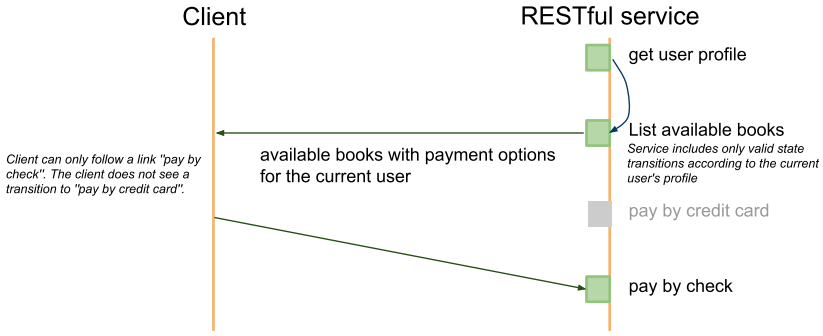
```
> HEAD /orders HTTP/1.1

< Content-Type: application/xml
< Link: <http://company.com/orders/?page=2&size=10>; rel="next"
< Link: <http://company.com/orders/?page=10&size=10>; rel="last"
```

- Advantages
  - *no need to get the entire document*
  - *no need to parse the document to retrieve links*
  - *use HTTP HEAD only*

Preconditions and HATEOAS

- Preconditions in HATEOAS
  - *Service in a current state generates only valid transitions that it includes in the representation of the resource.*
  - *Transition logic is realized at the server-side*



Advantages

- Location transparency
  - *only "entry-level" links published to the World*
  - *other links within documents can change without changing client's logic*
  - *Hypertext represents the current user's view, i.e. rights or other context*
- Loose coupling
  - *no need for a logic to construct the links*
  - *Clients know to which states they can move via links*
- Statelessness and Cloud
  - *Better implementation of scalability*