Programmable Web Project

JSON/XML based Hypermedia formats

Iván Sánchez Milara. Mika Oja

Outline

- Collection pattern
- Hypermedia content types
- Profiles
- Domain specific designs

COLLECTION PATTERN

Collection pattern

- Stores a set of values.
- PROTOCOL SEMANTICS:
  - GET:
    - Retrieve the collection
    - Retrieve collection item
    - Execute an algorithm (e.g. search)
  - POST:
    - Append element to the collection
  - PUT and PATCH:
    - Modify the state of an item
  - DELETE:
    - Delete an item from the collection

Atom

ATOM FILE TYPE

- Mime-type: application/atom+xml
- Defined in RFC4287
- Link: https://tools.ietf.org/html/rfc4287
- Media type to describe Web feeds.

Atom

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
            <title>Example Feed</title>
            <subtitle>A subtitle.</subtitle>
            <link href="http://example.org/feed/" rel="self" />
            <link href="http://example.org/" />
            <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
            <updated>2003-12-13T18:30:02Z</updated>
            <entry>
                <title>Atom-Powered Robots Run Amok</title>
                <link href="http://example.org/2003/12/13/atom03" />
                <link rel="alternate" type="text/html" href="http://example.org/api/atom03.html"/>
                <link rel="edit" href="http://example.org/api/atom03/"/>
                <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
                <updated>2003-12-13T18:30:02Z</updated>
                <summary>Some text.</summary>
                <content type="xhtml"><!- Type can be text, html, xhtml. Others such as image/jpeg are accepted-->
                        <div xmlns="http://www.w3.org/1999/xhtml">
                                    <p>This is the entry content.</p>
                        </div>
                        </content>
                        <author>
                                    <name>John Doe</name>
                                    <email>johndoe@example.com</email>
                        </author>
            </entry>
</feed>
```

Atom

ATOM PUBLISHING PROTOCOL

- Defined in RFC5023
- Link: http://www.ietf.org/rfc/rfc5023.txt
- Standarized protocol for creating and updating web resources.
- Protocol semantics:
  - Given by HTTP protocol (GET/PUT/POST/DELETE). Search using GET
- Application semantics:
  - Collection pattern (“feed”=> collection and “entry” => item) entrie“collection” and “item”
  - Each entry has the semantics of blog post (author, title, category, etc)

Collection media type

- Mime type: application/vnd.collection+json
- Link: http://amundsen.com/media-types/collection/
- Description:
  - JSON-based read/write hypermedia-type designed to support management and querying of simple collections.
  - It is similar to the The Atom Syndication Format (RFC4287) and the The Atom Publishing Protocol (RFC5023) .
  - Defines both the format and the semantics in a single media type.
  - Supports for Query Templates and expanded write support through the use of a Write Template.
- Protocol semantics:
  - Given by HTTP protocol: GET, PUT, POST, DELETE
- Application semantics:
  - Three elements: “collection” and “item” and “data”

Collection media type

```
{ "collection":
    {
        "version" : "1.0",
        "href" : "http://www.youtypeitwepostit.com/api/",
        "items" : [
          { "href" : "http://www.youtypeitwepostit.com/api/messages/21818525390699506",
           "data" : [
               { "name" : "text", "value" : "Test." },
               { "name" : "date_posted", "value" : "2013-04-22T05:33:58.930Z" }
           ],
           "links" : []
           },

          { "href" : "http://www.youtypeitwepostit.com/api/messages/3689331521745771",
           "data" : [
               { "name" : "text", "value" : "Hello." },
                { "name" : "date_posted", "value" : "2013-04-20T12:55:59.685Z" }
           ],
           "links" : []
           },
        "template" : {
           "data" : [
               {"prompt" : "Text of message", "name" : "text", "value" : ""}
           ]
         }
    }
```

Collection media type

```
// sample collection object
{
  "collection" :
  {
    "version" : "1.0",
    "href" : URI,
    "links" : [ARRAY],
    "items" : [ARRAY],
    "queries" : [ARRAY],
    "template" : {OBJECT},
    "error" : {OBJECT}
  }
}
```

URL representing this collection. Might be used to access to items

Associated links to this collection

Elements in the collection

Used to define algorithms: eg. Search templates

Defines structure of new item

Defines format of the error message

An array is an ordered sequence of zero or more values and it is represented by [] and values are separate by ,

An object contains a set of key-value pairs separated by , . It is represented by a {}

```
// sample items array
  "items": [
     {
       "href" : URI,
       "data" : [ARRAY],
       "links" : [ARRAY]
     },
     ...
     {  }
   ]
```

```
// sample error object
{
  "error" :
  {
    "title" : STRING,
    "code" : STRING,
    "message" : STRING
  }
}
```

```
// sample template
object
{
  "template" :
  {
    "data" : [ARRAY]
  }
}
```

Collection media type

```
// sample data array
"data" : [
      {"prompt" : STRING, "name" : STRING, "value" : VALUE},
      {"prompt" : STRING, "name" : STRING, "value" : VALUE},
      ...
      {"prompt" : STRING, "name" : STRING, "value" : VALUE}
    ]
```

```
// sample links array
"links" :[
          {"href" : URI, "rel" : STRING, "prompt" : STRING, "name" : STRING, "render" : "image"},
          ...
          {"href" : URI, "rel" : STRING, "prompt" : STRING, "name" : STRING, "render" : "link"},
]
```

```
// sample queries array
{
  "queries" :
  [
    {"href" : URI, "rel" : STRING, "prompt" : STRING, "name" : STRING,
      "data" :
      [
        {"name" : STRING, "value" : VALUE}
      ]
    },
    ...
  ]
}
```

Extensions

- Value-types:
  - **URL**: https://github.com/mamund/collection-json/blob/master/extensions/value-types.md
  - **Description**: It allows to substitute the value property by either object (support JSON objects), and array: (support JSON array).

```
"data" : [
   {"name" : "title", "value" : "J. Doe does something strange", "prompt" : "Title"},
   {"name" : "tags", "array" : ["article", "json"], "prompt" : "Tags"},
   {"name" : "contacts", "object" : { "name":"foo"}, "prompt" : "Contacts"}
 ]
```

- Template Data Validation:
  - **URL**: https://github.com/mamund/collection-json/blob/master/extensions/template-validation.md
  - **Description**: Permits using regular expression for data values.

```
"data" : [
   {"name" : "username", "value" : "", "prompt" : "Login Name",
    "regexp"   : "^[a-zA-Z0-9]*$", "required" : "true"}
 ]
```

- Read-only
  - **URL**: https://github.com/mamund/collection-json/blob/master/extensions/read-only.md
  - **Description**: Optional property read-only available for items. Tells if items can be edited (PUT) or removed (DELETE)

Defining structure and protocol semantics

HYPERMEDIA MEDIA TYPES

Hypertext Application language (HAL)

- Mime-type: application/hal+xml or application/hal+json

- Link: http://stateless.co/hal_specification.html
  http://tools.ietf.org/html/draft-kelly-json-hal-06
- Description:
  - HAL is a simple format that gives a consistent and easy way to hyperlink between resources in an API
  - You can adopt HAL's conventions and focus on building and documenting the data and transitions that make up your API.
- Protocol semantics:
  - Arbitrary state transitions through links that may use any HTTP method. Links do not mention HTTP method to use. It must be defined in the profile.
- Application semantics:
  - None

HAL. Example

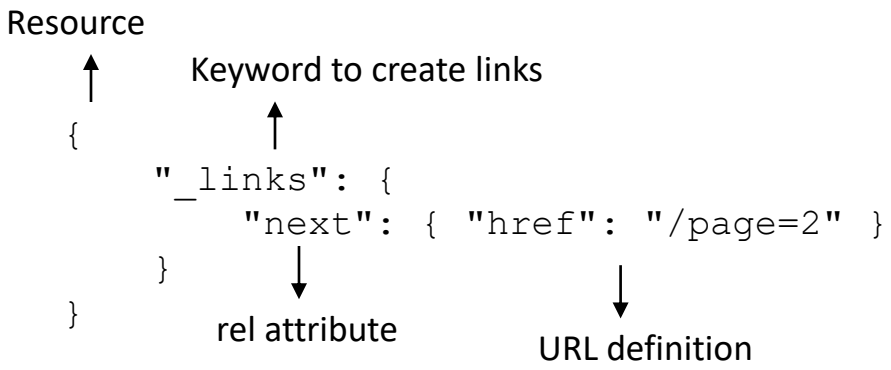
```
{
     "_links": {"self": { "href": "/orders" }, "next": { "href": "/orders?page=2" },
                "find": { "href": "/orders{?id}", "templated": true }},
     "_embedded": {
       "orders": [{
           "_links": {"self": { "href": "/orders/123" },
                      "basket": { "href":"/baskets/98712" },
                      "customer": { "href": "/customers/7809"}
                      },
           "total": 30.00,
           "currency": "USD",
           "status": "shipped"},
                  {
           "_links": { "self": { "href": "/orders/124" },
                       "basket": { "href": "/baskets/97213" },
                       "customer": { "href": "/customers/12369" }},
           "total": 20.00,
           "currency": "USD",
           "status": "processing"
       }]
     },%End of embedded
     "currentlyProcessing": 14,
     "shippedToday": 20
}
```

HAL

- The HAL conventions revolve around representing two simple concepts: *Resources* and *Links*.
- **Resources** have:
  - Links (to URIs)
  - Embedded Resources
  - State
- **Links** have:
  - A target (a URI)
  - A relation aka. 'rel' (the name of the link). It contains the semantic of the link.
  - A few other optional properties to help with deprecation, content negotiation, etc.
  - A link can trigger ANY HTTP request when activated
    - I need to define somewhere what methods can activate the state transition.
- HAL works better for only read APIs. I need to describe the semantics.

HAL

- Resource: { } A JSON object.
- Links:



  - When several links share the same relation

```
{
    "_links": {
      "items": [{
          "href": "/first_item"
      },{
          "href": "/second_item"
      }]
    }
}
```

  - All resources should have a **`self`** relation

HAL

Link properties:

- **`href`**: Its value is either a URI or a URI Template (then it should include a template attribute. It is the only mandatory property
- **`templated`**: Boolean that must be true when the Link Object's "**`href`**" property is a URI Template.
- **`type`**: String used as a hint to indicate the media type expected when dereferencing the target resource.
- **`name`**: Its value may be used as a secondary key for selecting Link Objects which share the same relation type.
- **`profile`**: URI that hints about the profile of the target resource.
- **`title`**: String intended for labelling the link with a human readable identifier.

HAL

- Inserting objects partial information about objects using **`_embedded`**
  - Each embedded resource is an object whose property names are link relation types and values are either a Resource Object or an array of Resource Objects.
  - Each embedded resource MAY be a full, partial, or inconsistent version of the representation served from the target URI
- Curies
  - "CURIE"s help providing links to resource documentation.

```
"_links": {
      "curies": [
                { "name": "doc", "href": "http://haltalk.herokuapp.com/docs/{rel}",
"templated": true }
       ],
       "doc:latest-posts": { "href": "/posts/latest" }
 }
```

  - To retrieve documentation about the latest-posts resource, the client will expand the associated CURIE link with the actual link's 'rel'. This would result in a URL http://haltalk.herokuapp.com/docs/latest-posts which is expected to return documentation about this resource.
  - Similar to namespaces in XML

HAL-FORMS

- Mime-type: application/prs.hal-forms+json
- A HAL extension that defines its own media type for describing forms.
  - A HAL-FORMS document can be linked from a HAL document using rel-links inside the _links element:

```
"http://api.example.org/rels/find": {
  "href": "http://api.example.org/donkeys/find/",
  "title": "Find Donkeys",
  "templated": false
}
```

  - The href property above is the URI to use for sending the actual request. A get to the key URI returns a HAL-FORMS document that explains which query parameters to use.

HAL-FORMS

- Example HAL-FORMS document

```
{
  "_links": {
    "self": {
      "http://api.example.org/rels/find"
    }
  },
  "_templates": {
    "default": {
      "title": "Find Donkeys",
      "method": "get",
      "properties": [
        {"name": "name", "value": "", "prompt": "Name"},
        {"name": "ear_length", "value": "", "prompt": "Ear length", "regex":
          "^[0-9]*$"}
      }
    ]
  }
}
```

HAL-FORMS

- Client procedure
  1. Get the HAL document
  2. Get the linked HAL-FORMS document
  3. Fill in query parameters as instructed in the HAL-FORMS document
  4. Send the query using the url from the original HAL document.
     - E.g.

```
http://api.example.org/donkeys/find?name=eeyore&ear_length=15
```

- More information:
  - http://mamund.site44.com/misc/hal-forms/

Siren

- Mime-type: application/vnd.siren+json
- Link: https://github.com/kevinswiber/siren
- Description:
  - HAL is a simple format that gives a consistent and easy way to hyperlink between resources in an API
  - You can adopt HAL's conventions and focus on building and documenting the data and transitions that make up your API.
- Protocol semantics:
  - Navigation through GET links. Arbitrary state transitions through actions.
- Application semantics:
  - None

Siren

- Represents abstract grouping of data called *entities*
  - Entity concept similar to `<div>` HTML tag.
  - An entity is a URI-addressable resource that has properties and actions associated with it. It may contain sub-entities and navigational links.
    - Sub-entities exist to communicate a relationship between entities, in context.
    - Links are primarily navigational and communicate ways clients can navigate outside the entity graph. The same as `<a>` HTML link.
  - Root entities and sub-entities that are embedded representations should contain a `links` collection with at least one item contain a `rel` value of self and an `href` attribute with a value of the entity's URI.
    - Every sub-entity provide a `rel` describing the relationship between its parent and itself.
- Defines transitions trough siren `actions`:
  - actions concept similar to HTML form method attribute.
  - More sophisticated protocol semantics can be defined.

Siren example

```
{
"class": [ "order" ],
"properties": {
    "orderNumber": 42,
    "itemCount": 3,
    "status": "pending"
},
"entities": [
  {
    "class": [ "items", "collection" ],
    "rel": [ "http://x.io/rels/order-items" ],
    "href": "http://api.x.io/orders/42/items"
  },
  {
    "class": [ "info", "customer" ],
    "rel": [ "http://x.io/rels/customer" ],
    "properties": {
      "customerId": "pj123",
      "name": "Peter Joseph"
    },
    "links": [
      { "rel": [ "self" ], "href":
"http://api.x.io/customers/pj123" }
    ]
  }
],
```

```
"actions": [
    {
      "name": "add-item",
      "title": "Add Item",
      "method": "POST",
      "href": "http://api.x.io/orders/42/items",
      "type": "application/x-www-form-urlencoded",
      "fields": [
        { "name": "orderNumber", "type": "hidden",
          "value": "42" },
        { "name": "productCode", "type": "text" },
        { "name": "quantity", "type": "number" }
      ]
    }
  ],
"links": [
    { "rel": [ "self" ], "href":
"http://api.x.io/orders/42" },
    { "rel": [ "previous" ], "href":
"http://api.x.io/orders/41" },
    { "rel": [ "next" ], "href":
"http://api.x.io/orders/43" }
  ]
}
```

Siren.
Entity properties

- **`class`**: Optional. Describes the nature of an entity's content based on the current representation. Possible values are implementation-dependent and should be documented. MUST be an array of strings.
- **`properties:`** Optional. A set of key-value pairs that describe the state of an entity. E.g. { "name": "Kevin", "age": 30 }.
- **`entities:`** Optional. Array of related sub-entities. A sub-entity might contain an embedded link or the full entity representation itself. A sub-entity MUST contain a `rel` attribute to describe its relationship to the parent entity.
- **`links:`** Array that contains navigation links distinct from entity relationships. Link items should contain a `rel` attribute to describe the relationship and an `href` attribute to point to the target URI.
- **`actions:`** Optional. JSON object representing a collection of actions
- **`title:`** Optional. descriptive text about the entity.

Siren.
Embedded link and link properties

EMBEDDED LINKS

- **`class`**: Same as in entity. Optional.
- **`rel:`** Mandatory. Defines the relationship of the sub-entity to its parent. MUST be an array of strings.
- **`href:`** Mandatory. The URI of the linked sub-entity.
- **`type:`** Optional. Defines media type of the linked sub-entity

LINKS

- **`rel:`** Mandatory. Defines the relationship of the linked resource with the current one. MUST be an array of strings.
- **`href:`** Mandatory. The URI of the linked resource.
- **`type:`** Optional. Defines media type of the linked resource.
- **`title:`** Optional. descriptive text about the nature of the link.

Siren. Actions

- **`name:`** Mandatory string that identifies the action to be performed.
- **`class:`** Optional array of strings. Describes the nature of an action based on the current representation. Possible values are implementation-dependent and should be documented.
- **`method:`** Mandatory. HTTP method. If omitted use GET.
- **`href:`** Mandatory.The URI of the action.
- **`title:`** Optional Descriptive text about the action.
- **`type:`** The encoding type for the request.
- **`fields:`** A collection of fields, expressed as an array of objects in JSON Siren such as { "fields" : [{ ... }] }. Represent controls inside of action.

Siren. Fields.

- **`name:`** Mandatory. A name describing the control.
- **`type:`** The input type of the field. This may include any of the following input types specified in HTML5: `hidden, text, search, tel, url, email, password, datetime, date, month, week, time, datetime-local, number, range, color, checkbox, radio, file, image, button`
- **`value:`** Optional. A value assigned to the field.
- **`title:`** Optional.Textual annotation of a field. Clients may use this as a label.

Mason

- Mime-type: application/vnd.mason+json
- Link: https://github.com/JornWildt/Mason
- Mason adds hypermedia elements, standardized error handling and metadata to JSON representations.

Mason - Metadata

- @meta elements contain data that's useful for client developers but is not used in the client itself.
  - @meta elements can be removed by the server when the request includes `representation=minimal` in the Prefer header
- Properties are: @title, @description, @controls (all optional)
- Example:

```
"@meta": {
  "@title": "Donkey",
  "@description": "This document contains information about a donkey"
}
```

Mason - Namespaces

- Used to expand curies in control element names
  - The primary purpose is to improve human readability of the controls definitions
- Namespaces are defined inside the @namespaces element.
- Example

```
"@namespaces": {
  "dk": {
    "name": "http://donkey-manager-reltypes.org/rels#"
  }
}
```

- See Mason controls slides for use example

Mason – Controls

- Mason's @controls element provides flexible hypermedia controls
- Standard links are named using IANA relationship registry tokens. Non standard links are named using curie (see namespaces) or a complete URI.
  - IANA registry: http://www.iana.org/assignments/link-relations/link-relations.xhtml
- Example using namespaces (from prev. slide)

```
"@controls": {
   "dk:location": {
     "href": "http://api.example.org/donkey/eeyore/location",
     "title": "Location of this donkey"
   }
}
```

Mason – Controls

- @controls can include a number of properties. Some are listed below – see the documentation for full listing
  - href: the only required property. URI or URI template
  - title: the control's title
  - method: HTTP method to use (default is GET)
  - schema: schema definition of request body and template parameters
  - template: request template data, especially useful for providing default values, and to include hidden fields
- Example – simple self link:

```
"@controls": {
  "self": {
    "href": "http://api.example.org/donkey/eeyore"
  }
}
```

Mason - Controls

- Here is another example using schema to instruct how to form the request body:

```
"@controls": {
  "self": {
    "href": "http://api.example.org/donkey/eeyore"
  },
  "dk:mood": {
    "title": "Change mood",
    "href": "http://api.example.org/donkey/eeyore/mood",
    "method": "PUT",
    "encoding": "json",
    "schema": {
      "type": "object",
      "properties": {
        "Mood": {"type": "string"},
        "Reason": {"type": "string"}
    }
  }
}
}
```

Mason - Errors

- Mason also includes a standard way for sending error response documents in the @error element
- An @error must contain a message property. It may also contain (more in the documentation):
  - @id: unique ID for later reference
  - @code: error code (application specific, not HTTP)
  - @details: a more extensive description of the error in human-readable form
  - @httpStatuscode: contains the status code from the response
  - @controls: links for the end user or client developer
  - @time: timestamp (RFC 3339)

Mason - Errors

- Example:

```
"@error": {
  "@message": "Mood reason was not provided",
  "@code": "MISSINGPROPERTY",
  "@time": "2016-02-24T14:57:14.52+02:00"
}
```

UBER Hypermedia

- Mime-type: application/vnd.uber+json
- Link: https://github.com/uber-hypermedia
- Message format that supports both XML and JSON
- Minimal format for simple state transfers and ad-hoc hypermedia-based transitions

UBER Hypermedia

- Documents contain three types of elements: uber, data and error.
  - uber is the root element
  - data is the main element, which may be nested indefinitely
  - error element contains details about a failed request
- Both uber and error are container elements that should contain at least one data element (but may contain more)
- data is the key element used to describe everything.

UBER Hypermedia

- Properties of data elements:
  - id: in-document unique identifier for this element
  - name: name of this element, can be used as a variable
  - rel: link relation value(s), array
  - label: caption for the value property of the same data element
  - url: a resolvable url associated with this element
  - templated: if set to true, the associated url is a URI template
  - action: request verb associated with this element
    - one of: append, partial, read, remove, replace
  - transclude: if set to true, returned content should be embedded
  - model: a template for constructing request bodies
  - sending: media type identifier(s) to use for the request body
  - accepting: media type(s) to expect from the response
  - value: value of the element

UBER Hypermedia - Example

```
{"uber: {
    "version": "1.0",
    "data": [
      {"rel": ["self"], "url": "http://api.example.org/donkey/eeyore"},
      {"rel": ["profile"], "url": "http://api.example.org/profiles/donkey"},
      "data": [{
          "name": "modify",
          "rel": ["http://api.example.org/rels/modify"],
          "url": "http://api.example.org/donkey/eeyore/",
          "action": "replace",
          "model": "n=donkeyName&e=earLength"
        },{
          "name": "delete",
          "rel": ["http://api.example.org/rels/modify"],
          "url": "http://api.example.org/donkey/eeyore/",
          "action": "remove",
        },
        {"name": "donkeyName", "value": "Eeyore", "label": "Donkey name"},
        {"name": "earLength", "value": "30", "label": "Length of ears"}
      ]
    ]
  }
}
```

Problem detail documents

- Mime-type: application/problem+json
- Link: https://tools.ietf.org/html/rfc7807
- Description:
  - Represents error code into a human readable way
- Protocol semantics:
  - Navigation through GET links.
- Application semantics:
  - Error report

Problem detail documents

```
{
”type": "http://example.com/scheduled-maintenance",
”instance": "http://example.com/maintenance/outages/20130533",
”status" : 503
"title": "The API is down for scheduled maintenance.",
"detail": "This outage will last from 02:00 until 04:30 UTC."
}
```

- `"type"` (string): An absolute URI that identifies the problem type. When dereferenced, it SHOULD provide human-readable documentation for the problem . When this member is not present, its value is assumed to be "about:blank".
- `"title"` (string): - A short, human-readable summary of the problem type. It SHOULD NOT change from occurrence to occurrence of the problem, except for purposes of localisation.
- `"status"` (number) : The HTTP status code generated by the origin server for this occurrence of the problem.
- `"detail"` (string): - An human readable explanation specific to this occurrence of the problem.
- `"instance"`(string): - An absolute URI that identifies the specific occurrence of the problem. It may or may not yield further information if dereferenced.

- You can add new properties if needed.

Defining application semantics

PROFILE

Profile

- RFC6906:

  "A profile is defined to not alter the semantics of the resource representation itself, but to allow clients to learn about additional semantics… associated with the resource representation, in addition to those defined by the media type…"

- Defines the vocabulary used in the representations, the application semantics:
  - E.g. the human-readable documentation of the Twitter API is a profile.
- Sometimes it also describe some protocol semantics. What methods can be invoked with each property.
  - Only if
    - the media type does not have hypermedia control.
    - the media type hypermedia control's are not specific enough to define which HTTP request the client should make (e.g. HAL)

Profiles.

- **Link relations**
  - String describing the state transition that will happen if the client triggers a hypermedia control
  - `rel` attribute in all type of hyperlinks
  - All must be documented unless defined by IANA (see next slide)
- **Semantic descriptors:**
  - Short string that indicates what some part of a representation means.
  - Always must be documented
  - Examples:
    - class attribute values: HTML and Siren `<span class="fn">Jenny Gallegos</span>`
    - Keys in JSON or HAL `{"name": "Jenny Gallegos"}`
    - Entity properties in Siren `"properties" : { "name" : "Jenny Gallegos" },`
    - Name of the data field in a Collection+JSON:

```
"data" : [
          { "name" : "family-name", "value" : "Gallegos" }
  ],
```

IANA link relations

- Global register containing about 60 relations.
  - http://www.iana.org/assignments/link-relations/link-relations.xhtml
  - Some useful relations:
    - `collection` and `item` to create collections.
    - `first`, `last`, `next` and `previous` for pagination
    - `replies` to described message thread
    - `latest-version`, `successor-version`, `working-copy` for history of a resource state
    - `edit` and `edit-media` to cover update/delete a resource
- Some document media types defines its own possible relations
- Some profiles include also relations
- If you wanna use your own link relation
  - Use extension relations: *http://mydoma.in/myrelation*

- Microformats Wiki also contains a big set of relations:
  - http://microformats.org/wiki/existing-rel-values
  - DO NOT USE THEM AS SUCH IF YOU HAVE NOT DEFINED THEM IN YOUR PROFILE

Linking to a profile

- Using the **profile** Link relation:
  - RFC 6906 defines a `rel` called **profile**
  - Can be used in any `rel` attribute:
    - `links` (Siren or Collection+Json);
    - `link` defined in HTML, HAL
    - `Link` HTTP header.

```
<html>
  <head>
    <link href="http://microformats.org/wiki/hcard" rel="profile">
```

- Using the **`profile`** Media Type parameter:
  - Added as parameter in the Content-Type header

```
Content-Type = application/collection+json;profile=http://myprofile
```

- Using special purpose hypermedia controls defined in some media types.

Languages to represent a profile

- Plain text
  - Like Facebook or Twitter API
  - We recommend to use this one in your project work.
- XMDP

http://gmpg.org/xmdp/description

- ALPS

http://tools.ietf.org/html/draft-amundsen-richardson-foster-alps-01

- JSON-LD

http://json-ld.org/

  - Can be embedded in the own JSON file.

Profiles stores (I). Microformats Wiki.

- Microformats Wiki contains a wide set of profiles:
  - http://microformats.org/wiki/Main_Page
- You should access the profile of each one the microformat in the previuos page, if they have to extract the semantics
  - You do not need to use the same structure, just the semantics.
- **Examples:**
  - *hCalendar:* Decribes event in time. http://microformats.org/profile/hcalendar
  - *hCard:* Describe people and organizations. http://microformats.org/profile/hcard
  - *XFN*: Describes relationships among people. http://gmpg.org/xfn/11
  - hRecipe: Describe a cooking recipe http://microformats.org/wiki/hrecipe
  - hMedia: Metadata about multimedia http://microformats.org/wiki/hmedia
  - hReview: Describes a rating http://microformats.org/wiki/hreview-profile

Profiles stores (II). Schema.org

- Main source of microdata items.

http://schema.org/docs/schemas.html

- Really consumer focus:
  - E.g. a product is something that a client can buy not a project the client is working on.
- Syntax at http://schema.org/docs/gs.html
- You must specify always which values are mandatory and which ones are optional in your profile description
- Examples:
  - Action: http://schema.org/Action
  - TVSeries: http://schema.org/TVSeries
  - LocalBusiness: http://schema.org/LocalBusiness

JSON Hyper-Schema

- JSON Schema is a format for defining JSON data structure
- JSON Hyper-Schema adds hypertext definitions to JSON Schema definitions
  - Note: It's a machine-readable profile, not a hypermedia format
- JSON Schema:
  - http://json-schema.org/latest/json-schema-core.html
- JSON Hyper-Schema:
  - http://json-schema.org/latest/json-schema-hypermedia.html

JSON Hyper-Schema

- Example schema:

```
{
"title": "Donkey",
"type": "object",
"properties": {
  "id": {
    "title": "Donkey Identifier",
    "type": "number"
  },
  "name": {
    "title": "Donkey Name",
    "type": "string"
  },
  "picture": {
    "title": "Donkey Picture",
    "type": "string",
    "media": {
      "binaryEncoding": "base64",
      "type": "image/png"
    }
  },
  "links": [
    {
      "rel": "full",
      "href": "{id}"
    },
    {
      "rel": "related",
      "href": "{id}/location"
    }
  ]
}
```

JSON Hyper-Schema

- In the previous example links and media are properties added in the Hyper-Schema.
- To use this schema, the response should contain the following header:
  - `Content-Type: application/donkey+json; profile=http://api.example.com/donkey-schema#`
- Hyper-Schema has no support for defining new rel types, the use of existing ones is encouraged instead

JSON Hyper-Schema

- POST example

```
{...
  "links": [
    {
      "title": "Add a new donkey",
      "rel": "create",
      "href": "/donkeys",
      "method": "POST",
      "schema": {
        "type": "object",
        "properties": {
          "name": {"type": "string"},
          "earlength": {"type": "number"}
        }
        "required": ["name", "earlength"]
      }
    }
  ]
}
```

Defining own structure and application and protocol semantics

DOMAIN SPECIFIC DESIGNS

Domain specific designs

- When the problem cannot be solved using any of the existing media types and profiles you MUST define yours.
  - Personalized media types usually do not need to use profiles . Everything is defined in the media type itself.
- A personalized media type MUST define:
  - The representation format.
    - Do not forget to include links and its relations.
  - The protocol semantics
    - Meaning of each transition and which HTTP method must be used
  - The application semantics
    - What does each attribute means in the application
  - A name for the media type:
    - Usually with the format *application/nameofthetype+json*

HOW CAN I DESIGN AN API USING HYPERMEDIA?

How can I design an API using hypermedia?

1. Find a media type that best suits my needs
   - If I cannot find it create your own specific domain design
2. Find a profile that best suits my needs
   - A profile must define:
     - Protocol semantics (if the media type does not provides it).
       - Usually using rel relations
     - Application semantics
       - Define the semantics descriptors
   - If there is no a profile that fits my project:
     - I can create a new profile from a existing ones. I need to clarify the subset of the existing profile I am using.
     - I can create a new profile from several existing ones. I need to clarify the subset of the existing profiles I am using
     - I can create a new profile from scratch.

How can I design an API using hypermedia?

Whatever you do be sure that you provide

1. The format of the representation (in the media type)
2. The protocol semantics (either in the media type or in the profile)
3. The application semantics (either in the media type or in the profile)