An RDAP capability for server specification provisioning

Mario Loffredo, Maurizio Martinelli
IIT-CNR/Registro.it
mario.loffredo, maurizio.martinelli@iit.cnr.it
REST API Specification languages

RDAP
- Servers
- Clients
- Client-Server interaction

Proposal
- Goals
- Implementation
  - Server
  - Client
- Advantages
- Registro.it implementation

Q & A
There is a growing consensus that modern REST APIs should be self-descriptive.

A REST service should provide clients with a machine-processable specification to describe:

- the requests in terms of available paths, parameters and bodies
- the responses in terms of returned properties and values
- the authentication methods
Some of them are available on the web

Each one has its own:

- format

- media type for its delivery as a REST response

- set of tools covering every phase of the API life cycle (design, build, test, documentation and sharing)

- community of developers
A brief list of the most popular REST API specification languages includes:

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenAPI</td>
<td>swagger.io</td>
<td>JSON, YAML</td>
</tr>
<tr>
<td>RAML</td>
<td>raml.org</td>
<td>YAML</td>
</tr>
<tr>
<td>APIBlueprint</td>
<td>apiblueprint.org</td>
<td>proprietary</td>
</tr>
<tr>
<td>JSON API</td>
<td>jsonapi.org</td>
<td>JSON</td>
</tr>
<tr>
<td>JSON Schema</td>
<td>json-schema.org</td>
<td>JSON</td>
</tr>
<tr>
<td>Slate</td>
<td><a href="https://github.com/lord/slate">https://github.com/lord/slate</a></td>
<td>Markdown</td>
</tr>
<tr>
<td>WADL</td>
<td><a href="https://www.w3.org/Submission/wadl/">https://www.w3.org/Submission/wadl/</a></td>
<td>XML</td>
</tr>
</tbody>
</table>

Neither of them is a standard!!

The set of features they can document is very similar
- A specification can be converted into another by automatic tools
REST API Specification languages (4)

- What do they describe?

<table>
<thead>
<tr>
<th>Object</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>title, version (specification, API), description</td>
</tr>
<tr>
<td>server</td>
<td>url, description</td>
</tr>
<tr>
<td>path</td>
<td>endpoints, HTTP methods</td>
</tr>
<tr>
<td>parameter</td>
<td>path, query, header, cookie</td>
</tr>
<tr>
<td>request body</td>
<td>body content, media type</td>
</tr>
<tr>
<td>response</td>
<td>status codes, response schema</td>
</tr>
<tr>
<td>input/output model</td>
<td>common data structures used</td>
</tr>
<tr>
<td>authentication method</td>
<td>HTTP authentications, API key, OAuth2, OpenID</td>
</tr>
</tbody>
</table>
Can be pretty different in both requests and responses:

- search queries cannot be available
- bootstrapping cannot be implemented
- queries can be extended with additional parameters
- authentication can be required if servers want to provide different query capabilities and response information according to the user profile
- standard responses can be extended with new properties or values
- links in responses can be used in different ways
Can’t provide a formal description of their own features:

- whenever the `help` endpoint exists, it returns a human readable but not machine-processable response (https://rdap.pubtest.nic.it/help)

- information in the IANA Registries about extensions and values (https://www.iana.org/protocols):
  - can help to document the response but not the optional query parameters
  - cannot be used for formal validation and on-line specification
As a consequence:

- users might waste time submitting requests that can’t be accepted because they are not implemented by the server or because they are not allowed, according to the user access level

- users/clients must know the features of all the servers they interact with

- if a server changes its features, such a change is not immediately recognized by clients and, normally, it requires an additional effort by client implementers

- if the standard response is extended with some additional properties or values, the client can’t provide users with their on-line description

- responses cannot be formally validated according to a specification (as it happens in EPP by using XML schemas)
RDAP Clients

- Are based on RFC7482
- Provide users with fixed capabilities
The availability of online servers’ specifications seems to be fundamental to speed up client-server interaction.

If a server could provide its own specification, clients would be able to configure themselves in order to issue allowed requests and accept valid responses.
Proposal - Goals

- Describing servers specifications more formally
- Improving client-server interoperability
- Supporting the building of efficient clients
Proposal - Server implementation (1)

- **Request:**
  - RDAP servers provide clients with a new endpoint called "specification" (i.e. https://example.com/rdap/specification)
  - Bootstrapping is implemented through the method as described in RFC8521 (i.e. https://example.com/rdap/specification/{RDAP-provider-tag})

- **Response:**

```json
{
    "rdapConformance": [ "rdap_level_0", "rdap_specification_0" ],
    "notices": {
        "title": "Server specification",
        "description": [ "The list of specifications available for this RDAP server according to different formats" ],
        "links": [
            {
                "value": "http://example.com/rdap/specification",
                "rel": "describedby",
                "title": "OpenAPI-JSON",
                "type": "application/vnd.oai.openapi+json",
                "href": "http://example.com/rdap/specification(openapi.json"
            },
            ..... }
    }
}
```
A server could return its own specification according to different formats

Converters:

- [https://www.apimatic.io/transformer](https://www.apimatic.io/transformer)
  - OpenAPI, RAML, APIBlueprint, etc.

- [https://github.com/LucyBot-Inc/api-spec-converter](https://github.com/LucyBot-Inc/api-spec-converter)
  - OpenAPI, RAML, APIBlueprint, etc.

- [https://mulesoft.github.io/oas-raml-converter/](https://mulesoft.github.io/oas-raml-converter/)
  - OpenAPI, RAML

- [https://github.com/apiaryio/swagger2blueprint](https://github.com/apiaryio/swagger2blueprint)
  - OpenAPI, APIBlueprint
Proposal - Client implementation

- Implementing an RDAP client able to configure itself according to one or more specification formats
  
  - RFC7482 as default specification
  
  - Web UI generation tools
    - https://swagger.io/tools/swagger-ui/
    - https://swagger.io/tools/swagger-codegen/
    - https://github.com/eclipsesource/jsonforms-swagger/
    - https://openapi.tools/
    - https://raml.org/developers/build-your-api
    - https://apiblueprint.org/tools.html#renderers
    - https://jsonapi.org/implementations/
Proposal - Advantages

- **Server side:**
  - providing a machine-processable specification of:
    - the URI templates of non-standard path segments
    - the description and the formal constraints for each property or value extending the response
    - the supported authentication methods
  - announcing any change related to its capabilities to the world and making it suddenly available to clients

- **Client side:**
  - configuring itself, according to any server specification and user access levels
  - enabling the user to submit only valid requests
  - displaying and validating the responses more efficiently
  - adopting open source software available on the web dedicated to validation, data parsing, requests handling and user interface generation
Proposal - Registro.it implementation

- **Server:**
  - Specification is provided in OpenAPI, RAML and APIBlueprint formats ([https://rdap.pubtest.nic.it/specification](https://rdap.pubtest.nic.it/specification))
  - Bootstrapping is simulated:
    - [https://rdap.pubtest.nic.it/specification/STD](https://rdap.pubtest.nic.it/specification/STD)
    - [https://rdap.pubtest.nic.it/specification/GOOGLE](https://rdap.pubtest.nic.it/specification/GOOGLE)
    - [https://rdap.pubtest.nic.it/specification/BRNIC](https://rdap.pubtest.nic.it/specification/BRNIC)
    - [https://rdap.pubtest.nic.it/specification/VRSN](https://rdap.pubtest.nic.it/specification/VRSN)

- **Client:**
  - the target server specification is searched at first
  - if no specification is available, STD is loaded
  - OpenAPI is taken as the internal format
  - the other formats are converted in OpenAPI
  - the web UI is generated by Swagger-UI library
  - the development is still in progress
Thanks for your attention!

Q & A