An RDAP capability for server specification provisioning

Mario Loffredo, Maurizio Martinelli

IIT-CNR/Registro.it

mario.loffredo,maurizio.martinelli@iit.cnr.it

Content

- REST API Specification languages
- RDAP
  - Servers
  - Clients
  - Client-Server interaction
- Proposal
  - Goals
  - Implementation
    - Server
    - Client
  - Advantages
  - Registro.it implementation
- Q & A

REST API Specification languages (1)

- 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

REST API Specification languages (2)

- 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

REST API Specification languages (3)

- A brief list of the most popular includes:

Name	URL	Language
OpenAPI	swagger.io	JSON, YAML
RAML	raml.org	YAML
APIBlueprint	apiblueprint.org	proprietary
JSON API	jsonapi.org	JSON
JSON Schema	json-schema.org	JSON
Slate	https://github.com/lord/slate	MarkDown
WADL	https://www.w3.org/Submission/wadl/	XML

- 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?

Object	Features
metadata	title, version (specification, API), description
server	url, description
path	endpoints, HTTP methods
parameter	path, query, header, cookie
request body	body content, media type
response	status codes, response schema
input/output model	common data structures used
authentication method	HTTP authentications, API key, OAuth2, OpenID

RDAP Servers (1)

- 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

RDAP Servers (2)

- 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

RDAP Servers (3)

- 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

RDAP Client-Server Interaction

- 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:

```
{
    "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"
                        },
                        …..
            ]
    }
}
```

Proposal - Server implementation (2)

- A server could return its own specification according to different formats

- Converters:
  - https://www.apimatic.io/transformer
    - OpenAPI, RAML, APIBlueprint, etc.
  - https://github.com/LucyBot-Inc/api-spec-converter
    - OpenAPI, RAML, APIBlueprint, etc.
  - https://mulesoft.github.io/oas-raml-converter/
    - OpenAPI, RAML
  - 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
    - http://json-schema.org/implementations.html#web-ui-generation
    - 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)
  - Bootstrapping is simulated:
    - https://rdap.pubtest.nic.it/specification/STD
    - https://rdap.pubtest.nic.it/specification/GOOGLE
    - https://rdap.pubtest.nic.it/specification/BRNIC
    - 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

Registro.it

Thanks for your attention!

Q & A