RESTful Design for Internet of Things Systems

draft-keranen-t2trg-rest-iot-00

Ari Keränen <ari.keranen@ericsson.com>
with Matthias Kovatsch & Klaus Hartke

W3C Web of Things IG
October 30th 2015, Sapporo, Japan

RESTful Design

- Something similar to building a house
  - We have all seen and used them
  - We have a pretty good idea what we expect the internals to look like
  - Architecture has patterns one should follow
  - But there are also non-obvious reasons why some things are done in a certain way; and it can be easy to miss those

RESTful Design

Photo: Axel Hindemith, License: Creative Commons BY-SA-3.0 de

Goal of the Document

- “Guidance for designing IoT systems that follow the principles of the REST architectural style”
- Collect terminology
- Key information + pointers to details
- With IoT focus in examples etc.
- … while keeping it quick and easy to read

Background

- IRTF Thing-to-Thing Research Group (T2TRG) "REST as we use it" discussion at the IETF #93
- Lots of "REST information" available in RFCs, drafts, PhD theses, blogs, etc., but no single practical source
- Subtle differences in "REST as we use it"

Status

- +20 terms explained
- Some text about key topics
  - architecture (clients, servers, proxies)
  - application/session/resource state
  - resource modeling (media types, etc.)
  - URIs (from RFC3986)
  - methods (RFC7231 & RFC7252)
  - response codes (RFC7231)

Status

- Planning to write more about
  - application state in REST
  - RESTful design patterns
  - how to design resources
  - directories
  - HATEOAS
- Starting point for discussion
- Looking for contributions!

Discussion points

- Do you agree with the term definitions? Key terms missing? Too many already?
- What are the challenges you have faced and would benefit from guidance?
- (Application state)

DETAILS

Defined Terms

- Application State
- Cache
- Client
- Content Negotiation
- Form
- Forward Proxy
- Gateway
- Hypermedia Control
- Idempotent Method
- Link
- Media Type
- Method
- Origin Server
- Proactive Content Negotiation
- Reactive Content Negotiation
- Representation Format
- Representation
- Representational State Transfer (REST)
- Resource State
- Resource
- Reverse Proxy
- Safe Method
- Server
- Uniform Resource Identifier (URI)

Clients, Servers & Proxies

- Component in system is Client or Server
  - Can change role between interactions
- Forward proxy: make requests on behalf of client
- Reverse proxy: server towards clients, forwards requests to actual server
  - legacy encapsulation, etc.

Application state

- Session state with **client only**
  - Every request needs to contain all information needed to process it
  - But servers need no state per client
- Resource state for persistence
  - independent of application control flow

URIs

- Scheme defines namespace; different scheme, different resource (e.g., coap vs. coaps)
- Hierarchical (path) and non-hierarchical (query parameters) parts
- GET has established semantics for query parameters, e.g., filter or paginate
  - PUT/POST/DELETE not that clear

Safe/Idempotent Methods

- Safe method: does not result in any state change on the origin server when applied to a resource
- Idempotent method: multiple identical requests with the method lead to the same visible resource state as a single such request

GET method

- Requests a **current representation** for the target resource
- No semantics for payload (don't use)
- Safe & idempotent

POST method

- Requests that the target resource process the **representation enclosed** in the request according to the resource's **own** specific semantics
- If new resources are created, 201 (Created) is returned with pointer to created resource
- Not safe nor idempotent

PUT method

- Requests that the state of the target resource be **created or replaced** with the state defined by the **representation enclosed in the request message payload**
  - Implication: GET returns the representation
- POST vs. PUT: different intent for the enclosed representation
- (Intent of) PUT is idempotent

DELETE method

- Requests that the origin server removes the association between the target resource and its current functionality
- Representations might or **might not** be destroyed by the origin server depending entirely on the nature of the resource
- DELETE method is not safe, but is idempotent

Modifying properties of resources (drafty ideas)

- Problems with retrieving/modifying parts (properties/attributes) of a resource
- GET can use query parameters
  - And use forms to know how to construct query
- PUT replaces the **whole** representation
- POST with only changed attributes?
- Read-only attributes and GET/PUT symmetry is tricky