NOKIA

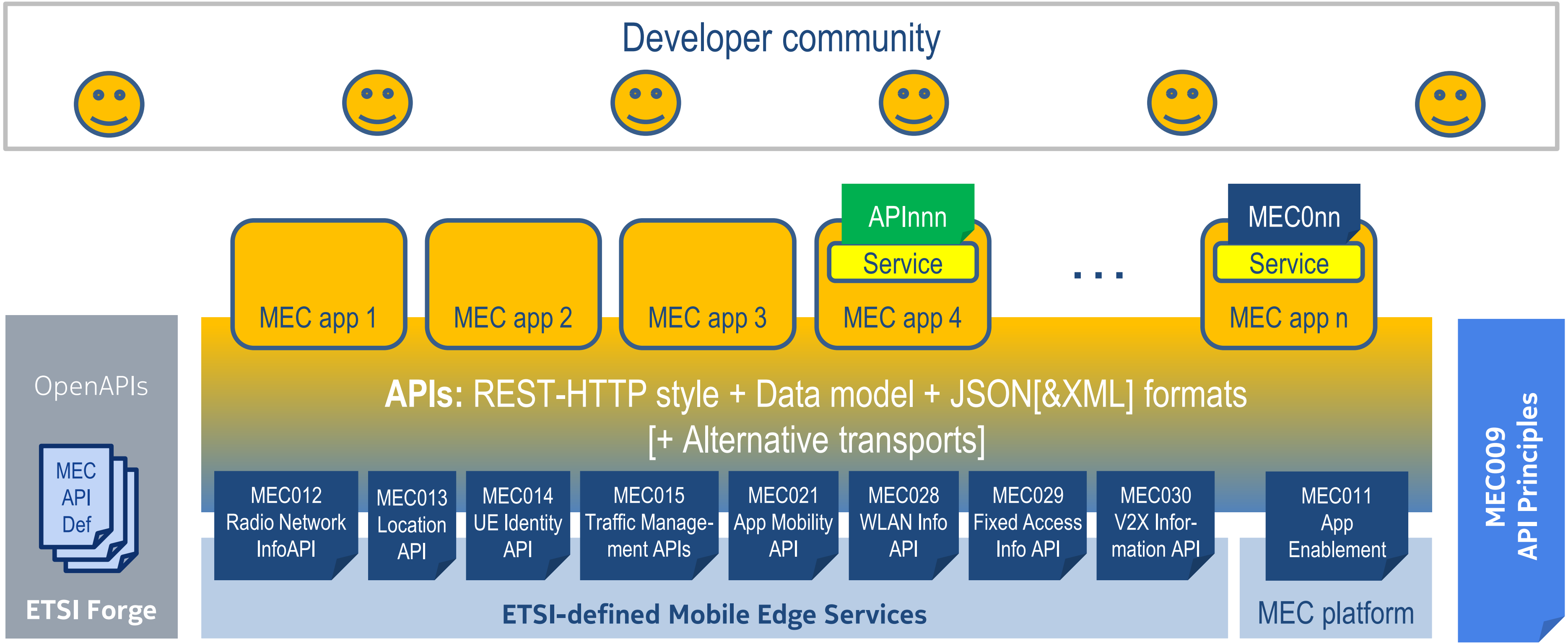
ETSI GS MEC 009

General principles, patterns and common aspects of MEC Service APIs

Uwe Rauschenbach, Nokia, ETSI MEC 009 rapporteur

29 April 2021

MEC APIs ecosystem



Developer community

MEC app 1

MEC app 2

MEC app 3

APInnn

Service

MEC app 4

. . .

MEC0nn

Service

MEC app n

OpenAPIs

MEC API Def

ETSI Forge

APIs: REST-HTTP style + Data model + JSON[&XML] formats [+ Alternative transports]

MEC012 Radio Network InfoAPI

MEC013 Location API

MEC014 UE Identity API

MEC015 Traffic Management APIs

MEC021 App Mobility API

MEC028 WLAN Info API

MEC029 Fixed Access Info API

MEC030 V2X Information API

MEC011 App Enablement

ETSI-defined Mobile Edge Services

MEC platform

MEC009 API Principles

MEC0nn = ETSI MEC API specification

API0nn = Third party MEC service API specification

MEC009 overview

Content

- Defines principles and patterns for RESTful MEC Service APIs
- Defines a REST API specification template
- Defines authorization methods (OAuth 2 and TLS cert based)
- Defines the concept of „alternative transports“

Latest draft

https://docbox.etsi.org/isg/mec/open/gs_mec009v030000_final%20draft%20for%20review_clean.pdf

Patterns in MEC009

- Name syntax in URIs (path segments, queries) and data structures (attributes, types, enums)
- Resource identification
- Resource representations and content format negotiation
- CRUD (Create – Read – Update – Delete) using POST/PUT – GET – PUT/PATCH – DELETE
- Task resources (Escape to RPC)
- Subscribe-Notify (Via HTTP / via Websockets)
- Monitor for async operations
- Links (HATEOAS)
- Error responses
- Authorization with OAuth 2
- Maps and lists in JSON
- Simple queries
- Advanced queries: Attribute selection and attribute-based filtering
- Response size control (paging, pre-defined error message)

Pattern: CRUD (Create – Read – Update – Delete)

Creating by POST:

- POST requestUri addresses the container resource
- Server creates a new child resource in that container; resource ID managed by the server

Creating by PUT:

- PUT requestUri addresses the resource to be created
- Server creates a new resource addressed by the requestURI; resource ID managed by the client

Reading by GET

- Special variant: queries on containers to limit the result set

Updating by PUT

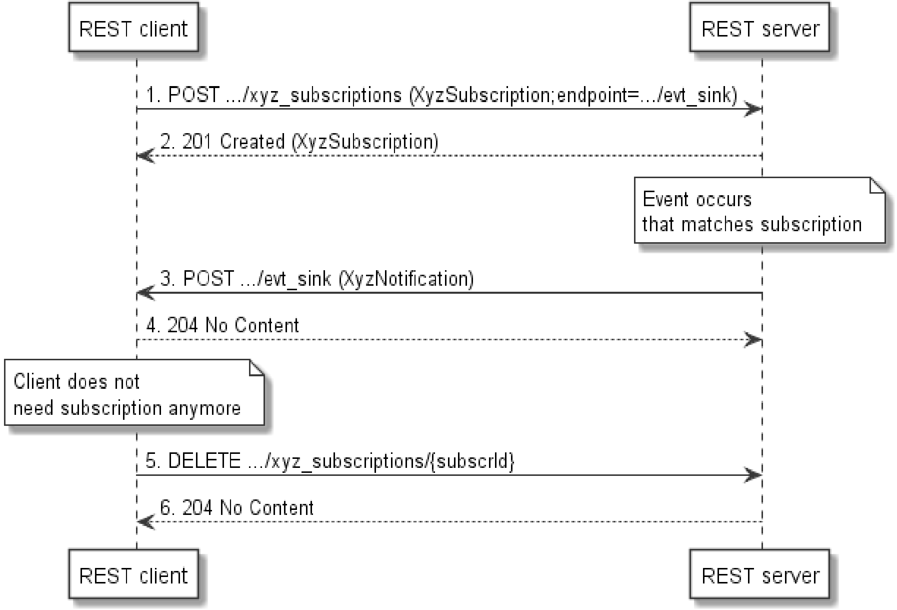
- Overwriting the complete resource

Updating by PATCH

- Providing a „delta document“ that contains information how to change the representation of the resource

Deleting by DELETE

Pattern: Simple subscribe – notify


REST client
REST server
1. POST .../xyz_subscriptions (XyzSubscription;endpoint=.../evt_sink)
2. 201 Created (XyzSubscription)
Event occurs
that matches subscription
3. POST .../evt_sink (XyzNotification)
4. 204 No Content
Client does not
need subscription anymore
5. DELETE .../xyz_subscriptions/{subscrId}
6. 204 No Content
REST client
REST server

Basic concept: „reverse HTTP connection“

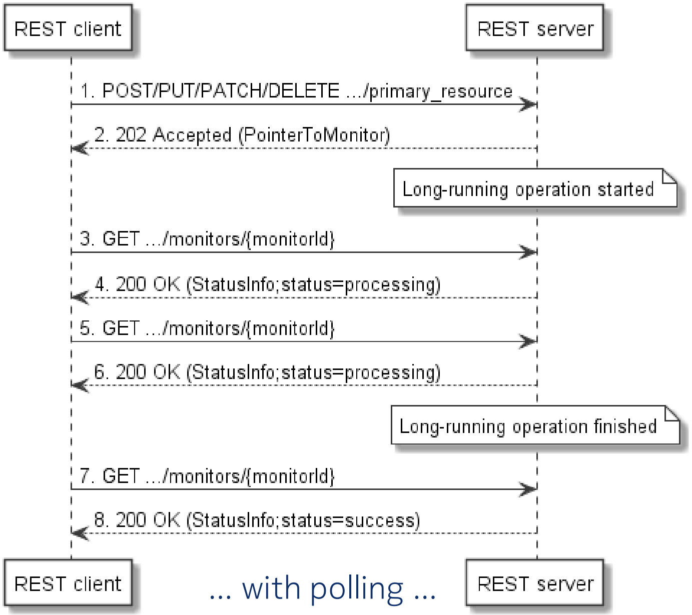
- Client exposes a callback URI
- Client registers callback URI with server on subscription
- Server delivers notifications in POST request towards client if events matches subscription

Variant:
Websocket fallback in case of middleboxes

- Allows penetrating middleboxes that might block reverse HTTP connection
- First try reverse HTTP, then fall back to a Websocket connection
- Aligned with 3GPP T8
- See backup slides…

Pattern: Asynchronous operations with monitors

- Some operations are long-running -> separate the operation from the resource on which it operates



REST client

REST server

1. POST/PUT/PATCH/DELETE .../primary_resource
2. 202 Accepted (PointerToMonitor)

Long-running operation started

3. GET .../monitors/{monitorId}
4. 200 OK (StatusInfo;status=processing)
5. GET .../monitors/{monitorId}
6. 200 OK (StatusInfo;status=processing)

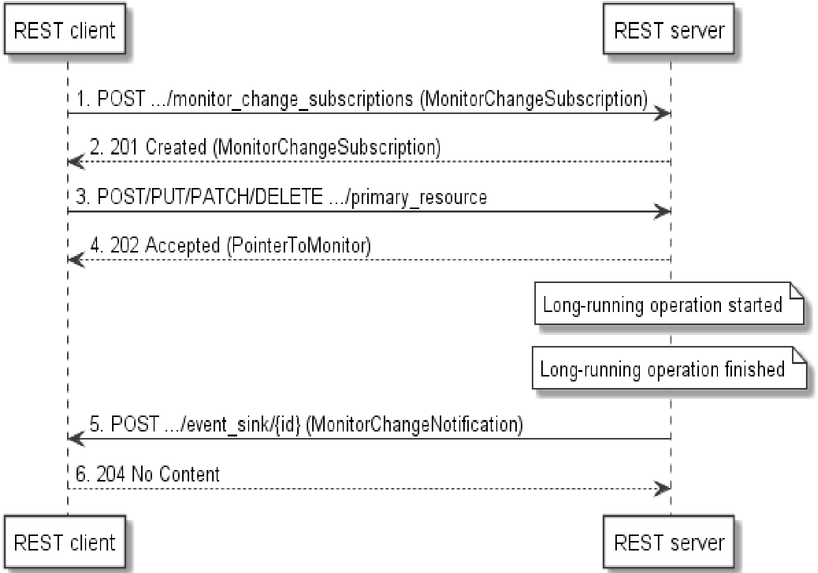
Long-running operation finished

7. GET .../monitors/{monitorId}
8. 200 OK (StatusInfo;status=success)

REST client

REST server

... with polling ...



REST client

REST server

1. POST .../monitor_change_subscriptions (MonitorChangeSubscription)
2. 201 Created (MonitorChangeSubscription)
3. POST/PUT/PATCH/DELETE .../primary_resource
4. 202 Accepted (PointerToMonitor)

Long-running operation started

Long-running operation finished

5. POST .../event_sink/{id} (MonitorChangeNotification)
6. 204 No Content

REST client

REST server

... with notifications ...

The concept of alternative transports

Problem

- REST APIs are not well suited for high volume, low latency information delivery.
- For that, alternative transports such as message buses (MQTT, Kafka) or GRPC are used.
- These are typically non-standardized implementations, which makes it very difficult to select one mechanism in a standard.

Alternative transports – a „semi-standardized“ solution

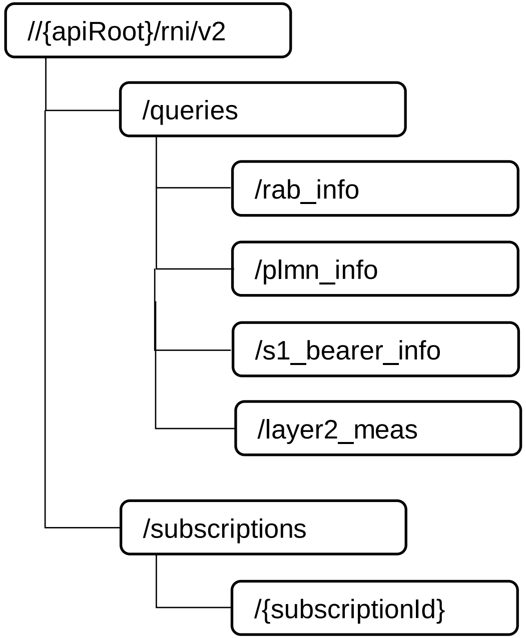
- A transport mechanism (e.g. Kafka) with a serializer (e.g. Protobuf) and associated security (e.g. based on TLS credentials)
- Declared in application descriptor and discovered via the MEC service registry
- Information model of the messages **is** standardized
- Serializer **may be** standardized
- Subscription and delivery **are not** standardized

MEC Service API specification template

<N>	**Sequence diagrams (Informative)**
<N+1>	**Data model (Normative)**
<N+1>.2	Resource data types
<N+1>.3	Subscription data types
<N+1>.4	Notification data types
<N+1>.5	Referenced structured data types
<N+1>.6	Referenced simple data types and enumerations
<N+2>	**RESTful API definition (Normative)**
<N+2>.2	Global definitions and resource structure
	→ Overview of the API‘s resources, their meaning and the applicable HTTP methods
<N+2>.3	Resource: <Resource1> (once per resource)
<N+2>.3.2	Resource definition
	→ URI structure
<N+2>.3.3	Resource methods
	→ separate sub-clauses for GET/PUT/PATCH/POST/DELETE

MEC012: Example of an API definition following the MEC009 template

Resource name	Resource URI	HTTP method	Meaning
RAB information	/queries/rab_info	GET	Retrieve current status of Radio Access Bearer information
PLMN information	/queries/plmn_info	GET	Retrieve current status of PLMN information
S1 Bearer information	/queries/s1_bearer_info	GET	Retrieve current status of S1 bearer information
Layer 2 measurements	/queries/layer2_meas	GET	Retrieve current status of layer 2 measurements information
All subscriptions for a subscriber	/subscriptions	GET	Retrieve a list of active subscriptions for this subscriber
		POST	Create a new subscription
Existing subscription	/subscriptions/{subscriptionId}	GET	Retrieve information on current specific subscription
		PUT	Modify existing subscription by sending a new data structure
		DELETE	Cancel the existing subscription
Notification callback	Client provided call-back reference	POST	Send a notification



Author’s contact

Uwe.Rauschenbach@nokia.com

More information on ETSI MEC

MEC Technology Page: http://www.etsi.org/mec

Published MEC Specifications: https://www.etsi.org/committee/1425-mec

Draft MEC Specifications (work in progress): http://docbox.etsi.org/ISG/MEC/Open