**3GPP TSG-CT WG3 Meeting #119-bis-e *C3-220026***

**E-Meeting, 17th – 21st January 2022**

**Source: Ericsson**

**Title: Discussion paper on using the RESTful GET method for data retrieval operations**

**Release: Rel-17**

**Agenda item: 17.32 (eSEAL)**

**Work Item / Release: eSEAL / Rel-17**

**Document for: DISCUSSION & AGREEMENT**

# 1. Introduction

The 3GPP OpenAPI design is based on the RESTful architecture. The high number of interconnected OpenAPIs leads to the need for agreement in the OpenAPI design recommendations. This discussion paper proposes to define the RESTful HTTP GET as a primary request type for data retrieval operations.

# 2. Discussion

The different opinions on whether RESTful standard operation GET method or RPC custom operation POST method for TSC\_Stream\_Availability\_Discovery service operations has been discussed during CT3#119e meeting discussion.

This discussion paper formulates a general principle for using the RESTful GET method in the eSEAL WI. However, its results are general and can be applied to the other WIs.

## 2.1 Discussion on the applicability of the RESTful design for the SEAL-S reference point implementation

As the first step, we overview of the recommendations and conclusions for using the RESTful architectural design.

The clause 4.2.2 of 3GPP 29.501 [1] defines the following API principles for query operations.

*When designing a query operation API, i.e. the NF service consumer invokes the API aiming to retrieve certain information from the NF service producer, the following principles should be applied:*

*a) if the query operation does not require any input parameter for the NF service producer, then the REST-style service operation with standard HTTP GET method should be used (see clause 4.6.1.1.2 of 3GPP 29.501 [1]);*

*b) if*

*- the query operation requires input parameter(s) for the NF service producer; and*

*- all the required input parameter(s) are used to identify a particular resource and/or control the content of the result of the query operation;*

*then the REST-style service operation with standard HTTP GET method should be used (see clause 4.6.1.1.2 of 3GPP 29.501 [1]);*

*c) standard HTTP GET method shall not be used for non-safe operations and non-idempotent operations.*

The statement in clause 4.2.2 of 3GPP 29.501 [1] matches with conclusions and recommendations of the CT Working Groups [2-3].

The clause 5.5.1.1.1 of 3GPP TR 29.890 [2] defines recommendations for using the HTTP methods in RESTful architectural design by the CT3 Working Group.

**Recommendations:**

*REST defines a set of architectural principles on how to design services that focus on a system's resources, RSET uses the create, read, update, and delete (CRUD) operations to handle such resources and HTTP methods can be directly mapped to those operations:*

*- To create a resource on the server, use HTTP POST;*

*- To retrieve a resource, use HTTP GET;*

*- To change the state of a resource or to update it, use HTTP PUT;*

*- To remove or delete a resource, use HTTP DELETE.*

*The conclusions on using the RESTful architectural design presented in clause 5.5.2.8 of 3GPP TR 29.890 [2].*

**Conclusions:**

*It is recommended to apply a RESTful framework for the protocol design as follows:*

*- service operations should implement the Level 2 of the Richardson maturity model, with standard API methods, whenever it is a good match for the style of interaction to model, e.g. service operations that can naturally map to one of the standard method (CRUD operations); this should be the preferred modelling attempt;*

*- service operations may use custom API methods (RPC interaction), when it is seen a better fit for the style of interaction to model, e.g. non-CRUD service operations.*

In clause 6.2.2.4.8 of 3GPP TR 29.891 [3], the following conclusions and recommendations by the CT4 Working Group are provided:

1. **Conclusions:**
   1. *RPCs based protocol design might not adequately match the expectation motivating the selection of service based interfaces and could also block a future evolution to a larger compliance with a Service Oriented Architecture;*
   2. *Designing RESTful APIs meeting stage 2 requirements for service based interfaces seems feasible but will require more stage 3 analysis to model resources. RESTful APIs offer the advantage of homogenous, easy to use interfaces, enhanced HTTP visibility (HTTP method accessible e.g. for proxying, logging, monitoring) and a larger decoupling between client and server compared to RPCs.*
   3. *an RPC like design might be necessary, but experience exist how to embed such operations in a RESTful framework (see e.g. custom methods associated with a resource in subclause 6.2.2.4.7 of 3GPP TR 29.891 [1]).*
2. **It is recommended to apply a RESTful framework for the protocol design as follows:**
   1. *service operations should implement the Level 2 of the Richardson maturity model, with standard API methods, whenever it is a good match for the style of interaction to model, e.g. service operations that can naturally map to one of the standard method (CRUD operations); this should be the preferred modelling attempt;*
   2. *service operations may use custom API methods (RPC interaction), when it is seen a better fit for the style of interaction to model, e.g. non-CRUD service operations.*

**Observation 1: The RESTful architectural is preferable for implementing CRUD operations in both CT3 and CT4 Working Groups. However, the RPC architectural design can be used when it is seen a better fit for the style of interaction to model.**

## 2.2 Discussion on the HTTP GET method application in the RESTful architectural design

As the second step, let’s formulate the HTTP GET method application requirements in the RESTful architectural design.

The generic constraint of using the standard HTTP GET method is formulated in [5]: *the HTTP GET method shall not be used for non-safe operations* (methods). 3GPP TS 29.501 [1] extends this constrain with additional "non-idempotent" constraint.

The "safe method" is an operation that does not affect a resource state on the server [5]. IETF RFC 7231 [6]: proposes the guidelines for the "safe method" definition:

*Request methods are considered "safe" if their defined semantics are essentially read-only; i.e., the client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource. Likewise, reasonable use of a safe method is not expected to cause any harm, loss of property, or unusual burden on the origin server.*

*This definition of safe methods does not prevent an implementation from including behavior that is potentially harmful, that is not entirely read-only, or that causes side effects while invoking a safe method. What is important, however, is that the client did not request that additional behavior and cannot be held accountable for it. For example, most servers append request information to access log files at the completion of every response, regardless of the method, and that is considered safe even though the log storage might become full and crash the server. Likewise, a safe request initiated by selecting an advertisement on the Web will often have the side effect of charging an advertising account.*

**Observation 2: The RESTful GET method is applicable for operations that do not imply the server resource state changes.**

The guidelines for the definition of an idempotent operation are presented in IETF RFC 7231 [6]:

*A request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request. Of the request methods defined by this specification, PUT, DELETE, and safe request methods are idempotent.*

*Idempotent methods are distinguished because the request can be repeated automatically if a communication failure occurs before the client is able to read the server's response. For example, if a client sends a PUT request and the underlying connection is closed before any response is received, then the client can establish a new connection and retry the idempotent request. It knows that repeating the request will have the same intended effect, even if the original request succeeded, though the response might differ.*

Let’s consider an example of the current time retrieval operation.

**Given:** the Client wants to retrieve the current time from the Server.

**Assumptions:**

1. the Server is stable;
2. the Server has the requesting resource.

**Observations:**

1. Each "Retrieve the current time" operation returns the different values according to the current time;
2. The "Retrieve the current time" operation reads the same related resource;
3. The multiple identical "Retrieve the current time" operations effects the server resource state in the same way as a single operation.

**Conclusion:** "Retrieve the current time" is an idempotent operation.

This example illustrates the possibility of multiple identical idempotent operations to return different values.

**Observation 3: The multiple identical RESTful GET requests can return different values. However, the RESTful GET method shall have the** **intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request.**

## 2.3 Discussion on the matching of data retrieval operations in the eSEAL WI with the RESTful GET operation requirements

As the third and final step, let’s evaluate the matching of TSC\_Stream\_Availability\_Discovery and Obtaint\_Unicast\_QoS\_Monitoring\_Data (data retrieval) operations in the eSEAL WI with the RESTful GET operation requirements.

Let’s define a typical data retrieval procedure from SEAL server (clause 14.3.7.2 and clause 14.3.3.4.4.2 of 3GPP TS 23.434 [2]) based on TSC\_Stream\_Availability\_Discovery and Obtaint\_Unicast\_QoS\_Monitoring\_Data operations.

First, the VAL server sends "Retrieve data request" to the SEAL. The SEAL Server performs all necessary actions (filtering, validating, searching, handling, aggregating data) to retrieve the requested data. It shall be noted that all internal operations do not imply to change the SEAL server resource state. Then, the SEAL Server sends the "Retrieve data response" to the VAL Server (figure 2.3-1).



**Figure 2.3-1: Sequence diagram for data retrieval from SEAL Server**

**Observation 4: TSC\_Stream\_Availability\_Discovery and Obtaint\_Unicast\_QoS\_Monitoring\_Data operations do not imply the SEAL Server resource state changes.**

**Observation 5: The multiple identical TSC\_Stream\_Availability\_Discovery or Obtaint\_Unicast\_QoS\_Monitoring\_Data operations affect the corresponding SEAL Server resource state in the same way as a single TSC\_Stream\_Availability\_Discovery or Obtaint\_Unicast\_QoS\_Monitoring\_Data operation.**

# 3. Conclusion

It is proposed:

1. note the current recommendations and conclusions for the OpenAPI design principles (**Observation** **1**);
2. note the requirements for the HTTP GET method application in the RESTful architectural design (**Observations** **2** and **3**).

According to **Observations** **4** and **5**, it can be concluded:

1. TSC\_Stream\_Availability\_Discovery and Obtaint\_Unicast\_QoS\_Monitoring\_Data operations are both safe and idempotent methods.
2. The RESTful GET method shall be used for TSC\_Stream\_Availability\_Discovery and Obtaint\_Unicast\_QoS\_Monitoring\_Data operations.

# 4. References

1. 3GPP TS 29.501 5G System; Principles and Guidelines for Services Definition; Stage 3.
2. 3GPP TR 29.890 CT WG3 aspects of 5G System Phase 1 (Release 15).
3. 3GPP TR 29.891 5G System - Phase 1 CT WG4 Aspects (Release 15).
4. 3GPP TS 23.434 Service Enabler Architecture Layer for Verticals (SEAL); Functional architecture and information flows; (Release 17).
5. Dissertation of Roy T. Fielding at the University of California at Irvine, USA (2000): "Architectural Styles and the Design of Network-based Software Architectures", Chapter 5 "Representational State Transfer (REST)", <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.
6. IETF RFC 7231: " Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content".