RESTful bindings for Parlay X Web Services - Common

Candidate Version 1.0 – 23 Nov 2010

Open Mobile Alliance

OMA-TS-ParlayREST_Common-V1_0-20101123-C

Contents

1. SCOPE 5
2. REFERENCES 6
2.1 NORMATIVE REFERENCES 6
2.2 INFORMATIVE REFERENCES 6
3. TERMINOLOGY AND CONVENTIONS 7
3.1 CONVENTIONS 7
3.2 DEFINITIONS 7
3.3 ABBREVIATIONS 7
4. INTRODUCTION 8
4.1 VERSION 1.0 8
5. COMMON CONSIDERATIONS FOR PARLAYREST 9
5.1 USE OF REST GUIDELINES 9
5.2 NAMESPACES 9
5.3 UNSUPPORTED FORMATS 9
5.4 AUTHORING STYLE 9
5.4.1 Names 9
5.4.2 Case usage for names 9
5.5 CONTENT TYPE NEGOTIATION 9
5.6 RESOURCE CREATION 10
5.6.1 Error recovery during resource creation 10
5.7 JSON ENCODING IN HTTP REQUESTS/RESPONSES 11
5.7.1 Serialization rules: general conversion 11
5.7.2 Serialization rules: structure-aware conversion 13
5.7.3 Rules for JSON-creating and JSON-consuming applications 15
6. SHARED DATA TYPE DEFINITIONS 16
6.1 ADDRESS DATA ITEMS 16
6.1.1 Charging 16
6.1.2 Charging data type 16
6.2 COMMON DATA TYPES 17
6.2.1 Enumeration: NotificationFormat 17
6.2.2 Enumeration: TimeMetrics 17
6.2.3 Type: TimeMetric 18
6.2.4 Type: ChargingInformation 18
6.2.5 Type: CallbackReference 18
6.2.6 Type: ResourceReference 19
6.2.7 Type: Link 19
6.2.8 Type: RequestError 19
6.2.9 Type: ServiceException 19
6.2.10 Type: PolicyException 19
6.2.11 Type: ServiceError 20
6.2.12 Enumeration: RetrievalStatus 20
6.3 HTTP RESPONSE CODES 20
6.3.1 Handling of not allowed HTTP methods. 21
6.3.2 Service and Policy exceptions 21
6.3.3 HTTP Response Codes in response to Notifications 22
APPENDIX A. CHANGE HISTORY (INFORMATIVE) 23
A.1 APPROVED VERSION HISTORY 23
A.2 DRAFT/CANDIDATE VERSION 1.0 HISTORY 23
APPENDIX B. DEPLOYMENT CONSIDERATIONS (INFORMATIVE) 25
B.1 PARLAYREST CLIENT APPLICATION EXECUTING IN A SERVER EXECUTION ENVIRONMENT 25
B.2 PARLAYREST CLIENT APPLICATION EXECUTING IN A MOBILE DEVICE EXECUTION ENVIRONMENT 26

B.3 PARLAYREST CLIENT APPLICATION EXECUTING IN A FIXED DEVICE EXECUTION ENVIRONMENT 27

Figures

Figure 1 ParlayREST API accessed from a server execution environment (e.g. 3rd party Service Provider application) 25
Figure 2 ParlayREST API accessed from a mobile device execution environment 26
Figure 3 ParlayREST API accessed from by a fixed device execution environment 27

Tables

Table 1: NotificationFormat Values 17
Table 2: Time Metrics Values 17
Table 3: TimeMetric Structure 18
Table 4: ChargingInformation Structure 18
Table 5: CallbackReference Structure 18
Table 6: ResourceReference Structure 19
Table 7: Link Structure 19
Table 8: RequestError 19
Table 9 ServiceException 19
Table 10: PolicyExceptionResponse Codes 19
Table 11: ServiceError 20
Table 12: RetrievalStatus 20

1. Scope

The scope of this specification is to specify an HTTP protocol binding for the set of Parlay X Web Services specifications in OMA, using REST architectural style.

The specification defines an HTTP protocol binding for an abstract API, based on existing OMA enablers.

2. References

2.1 Normative References

[ISO4217] "ISO 4217 currency names and code elements", URL: http://www.iso.org/

[JSON] "The application/json Media Type for JavaScript Object Notation (JSON)", D. Crockford, July 2006, URL: http://www.ietf.org/rfc/rfc4627.txt

[ParlayX_Common] "Open Service Access (OSA); Parlay X web services; Part 1: Common", Release 8, Third Generation Partnership Project, URL: http://www.3gpp.org/ftp/Specs/html-info/29-series.htm

[PSA] "Reference Release Package for Parlay Service Access", Open Mobile Alliance™, OMA-ERP-PSA-V1_0, URL: http://www.openmobilealliance.org/

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC2616] "Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding et. al, June 1999, URL: http://www.ietf.org/rfc/rfc2616.txt

[RFC3261] "SIP: Session Initiation Protocol", J. Rosenberg, et. Al, June 2002, URL: http://www.ietf.org/rfc/rfc3261.txt

[RFC3986] "Uniform Resource Identifier (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, January 2005, URL: http://www.ietf.org/rfc/rfc3986.txt

[RFC3966] "The tel URI for Telephone Numbers", H. Schulzrinne, December 2004, URL: http://www.ietf.org/rfc/rfc3966.txt

[RFC4122] "A Universally Unique IDentifier (UUID) URN Namespace", P. Leach, M. Mealling, R. Salz, July 2005, URL: http://www.ietf.org/rfc/rfc4122.txt

[SCRRULES] "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL: http://www.openmobilealliance.org/

[XMLSchema1] W3C Recommendation, XML Schema Part 1: Structures Second Edition, URL: http://www.w3.org/TR/xmlschema-1/

[XMLSchema2] W3C Recommendation, XML Schema Part 2: Datatypes Second Edition, URL: http://www.w3.org/TR/xmlschema-2/

2.2 Informative References

[OMADICT] "Dictionary for OMA Specifications", Version 2.8, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_8, URL: http://www.openmobilealliance.org/

[REST_WP] "White Paper on Guidelines for ParlayREST API specifications", Open Mobile Alliance™, OMA-WP-Guidelines-for-ParlayREST-API-specifications, URL:http://www.openmobilealliance.org/

[XML2JSON] Open source XML to JSON conversion tool URL: http://forge.morfeo-project.org

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

For the purpose of this TS, all definitions from the OMA Dictionary apply [OMA-DICT].

[N/A] [N/A]

3.3 Abbreviations

API	Application Programming Interface
DNS	Domain Name Server
HTTP	Hypertext Transfer Protocol
ID	Identifier
IP	Internet Protocol
JSON	JavaScript Object Notation
OMA	Open Mobile Alliance
PSA	Parlay Service Access
REST	Representational State Transfer
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

4. Introduction

To ensure consistency for developers using the ParlayREST enabler, this "Common" technical specification aims to contain all items that are common across all HTTP protocol bindings using REST architectural style for the various individual interface definitions, such as namespaces, naming conventions and fault definitions. In addition, data types that are shared between two or more protocol bindings are included in this specification as well.

4.1 Version 1.0

This version of the ParlayREST Common Technical Specification contains common namespaces, naming conventions and fault definitions, as well as shared data types for ParlayREST V1_0.

The ParlayREST TSs for version 1.0 addresses the following enablers:

- Short Messaging, as defined in PSA V1_0 [PSA]
- Multimedia Messaging, as defined in PSA V1_0 [PSA]
- Terminal Location, as defined in PSA V1_0 [PSA]
- Payment, as defined in PSA V1_0 [PSA]

5. Common Considerations for ParlayREST

5.1 Use of REST Guidelines

Representational State Transfer (REST) is an architectural style for defining distributed systems. Entities in these systems communicate using the interfaces they expose. For the purpose of REST API specification development for the ParlayREST Enabler, guidelines for defining REST bindings for Parlay X have been collected in [REST_WP]. These guidelines include general key principles that are used in mapping the Parlay X SOAP bindings to REST bindings.

5.2 Namespaces

The namespace for the common data types is:

urn:oma:xml:rest:common:1

The 'xsd' namespace is used in the present document to refer to the XML Schema data types defined in XML Schema [XMLSchema1, XMLSchema2]. The use of the name 'xsd' is not semantically significant.

5.3 Unsupported Formats

Servers must return a 406 Not Acceptable error if a message body format (e.g. XML or JSON) requested by the application is not supported [RFC2616].

5.4 Authoring Style

5.4.1 Names

Names will be meaningful, and not abbreviated in a way that makes the name hard to understand for users of the REST interfaces that are not literate in computer programming. This does not preclude the use of commonly understood acronyms within names (e.g. ID) or commonly used abbreviations (e.g. max). However, the resulting name must still be meaningful.

5.4.2 Case usage for names

Two general cases are provided for, both using mixed case names; one with a leading capital letter, the other with a leading lowercase letter.

Names will start with a letter and be mixed case, with the leading letter of each word capitalized. Words will not be separated by white space, underscore, hyphen or other non-letter character.

The following data types will have a leading uppercase letter –Type names and element names in an enumeration.

The following data types will have a leading lowercase letter – all other names.

For names consisting of concatenated words, all subsequent words start with a capital. For example, “concatenatedWord” or “BothCapitals”. If a lowercase name starts with an abbreviation, all characters of the abbreviation are de-capitalized, e.g. “smsService”.

Resource names are all lowercase.

5.5 Content type negotiation

The Content type of a response used SHALL be established using the following methodology:

As a general rule, content type used in response message body must match content type used in request body. At least XML and JSON content types MUST be supported.

Support for other content types will be specified on a case-by-case basis (e.g. simple name-value pair parameters may be accepted in the URL when using GET and www-form-urlencoding may be supported for the **request** message body when using POST or PUT).

Content type of the request message body SHALL always be determined by Content-Type header of the HTTP message.

Content type of the response body SHALL be determined using the following methodology. When invoking the API, the requesting application SHOULD include the 'Accept' request header, and provide the primary content type choice, and optionally OPTIONALLY any supported substitute content types, in this request Accept header.

a. If the server does not support the content type choice listed as priority in the Accept header, it SHALL attempt to return the next preferred choice if one was provided.
b. If the requesting application does not provide an Accept header or any other indication of desired content type of the response (see further below), and the request message body content type is XML or JSON, then the server SHALL provide a response message body with the content type matching that of the request message body. For example, a request with an XML body and no Accept header will trigger an XML response.
c. If the requesting application requires the response message body to be of a different content type than the request message body or the format resulting from Accept header negotiations (for which requesting application may not have sufficient control), it MUST request that by inserting in the URL path "?resFormat={content type}" (where content type SHALL be either XML or JSON). This option overrides the Accept header provided by the application, if both are present.
d. If the server cannot return any of the content types based on the negotiation steps described, it SHALL return a 406 response code as per [RFC2616].
e. The default format for notification payloads SHALL be XML, unless the client has specified *notificationFormat="JSON"* in the subscription.
f. Content type SHALL accompany HTTP response codes 200, 201, 400, 409 in the conditions dictated by the above specified methodology, and MAY be omitted in other cases.

5.6 Resource creation

Typically, a resource is created either following a POST request (to create a child of an existing resource that is addressed by the request), or following a PUT request (to create a new resource as addressed by the request).

If a resource has been created on the server, the server SHALL return an HTTP response with a "201 Created" header and the Location header containing the location of the created resource, and SHALL include in the response body either a resourceReference element, or a representation of the created resource. Note that this allows the server to control the traffic. Further note that ParlayREST resource representations are designed in such a way that they include a self reference.

5.6.1 Error recovery during resource creation

The following mechanism allows recovery from communication failures that can occur during resource creation using POST.

The client MAY (and in some cases SHOULD) include in the parameter set of the resource creation request the "clientCorrelator" field which uniquely maps to the resource to be created.

Note that this allows the client to retry a resource-creating request for which it did not receive an answer due to communication failure, and prevents the duplicate creation of resources on the server side in case of such retry. Note further that depending on the deployment (e.g. Network Address Translation, Proxies), the server might or might not be able to distinguish between different clients.

It is therefore RECOMMENDED that the client generates the value of the "clientCorrelator" in such a way that collisions (i.e. two unrelated requests use the same "clientCorrelator" value) are impossible or at least highly improbable. The way this is achieved is out of scope of this specification, however, it is pointed out that for example UUID [RFC4122] provides a way to implement such a scheme.

In case the server receives a "clientCorrelator" value in a resource-creating POST request, it SHALL do the following:

- in case the request contains a "clientCorrelator" value that has not been used yet to create a resource, the server SHALL create the resource and respond with "201 Created", as above.
- in case the request contains a "clientCorrelator" value that has already been used to create a resource, the server responds as follows:
- in case this is a valid repeated attempt by the same client to create the same resource, the server SHALL respond with "200 OK", and SHALL return a representation of the resource.
- otherwise, it SHALL respond with "409 Conflict", in this case indicating a clientCorrelator conflict. In such case, the client can retry the request using a new "clientCorrelator" value.

5.7 JSON encoding in HTTP Requests/Responses

5.7.1 Serialization rules: general conversion

Specifications of ParlayREST APIs include XML schema files defining the data structures used by the API, for its direct usage in XML format. The following are general rules for mapping between the Parlay REST XML and JSON data format:

a. XML elements that appear at the same XML hierarchical level (i.e. either root elements or within the same XML parent element), are mapped to a set of *name:value* pairs within a JSON object, as follows:

   (i) Each XML element appearing only once at the same hierarchical level (*"single element"*) is mapped to an individual *name:value* pair. The *name* is formed according to bullet b, while the *value* is formed according to bullet c.

   (ii) XML elements appearing more than once at the same hierarchical level (*"element list"*) are mapped to only one, individual *name:value* pair. The *name* is formed according to bullet b, while the *value* is a JSON array containing one *value* per each occurrence of the XML element. The name is formed according to bullet b whilst values are formed according to bullet c.

   (iii) *Name* and *Value* of JSON objects will go between "". Additionally, any JSON representation of an element of complex type will go between {}, according to [JSON].

b. The *name* of the *name:value* pair is the name of the XML elements (i.e. XML_element_name:value)

c. The *value* is formed as follows:

   (i) when the XML element has neither attributes nor child XML elements, the *value* is equal to the value of the XML element. In case the element is nill (i.e it has no value), it will be indicated as having a "null" value within JSON.

   (ii) when the XML element has child elements and/or attributes, the *value* is a JSON object containing the following *name:value* pairs:

   - one *name:value* pair per each attribute, where *name* is the name of the attribute and *value* is the value of the attribute.

   - one *name:value* pair associated to the text value (simple type content) of the XML element, where *name* is the string "$t" and *value* is the value of the XML element.

   - *name:value* pairs associated to XML child elements. These *name:value* pairs are formed in accordance with bullet a.

Within JSON, there is no need to reflect:

- the first <?xml version="1.0" encoding="UTF-8" ?> tag
- declaration of namespaces or schemaLocations

In order to generate unambiguous JSON from XML instances, based on the rules defined above, the following limitations need to be imposed on the XML data structures:

- it is not allowed that two different elements from different namespaces have the same name, in case they appear at the same level
- within an XML parent element, no attribute is allowed to have the same name as a child element of this parent element.

Note: These general rules have been used to generate the JSON examples from the XML examples in the Technical Specifications of the ParlayREST Enabler.

5.7.1.1 Utility which implements the general conversion rules (Informative)

The general conversion rules are implemented with UNICA XML2JSON utility, an open source tool, distributed, under a AGPL license, within the open source community MORFEO [XML2JSON].

5.7.1.2 Example (Informative)

The following is an example illustrating the guidelines:

Input XML content:

```
<Animals>
   <dog>
      <name attr="1234">Rufus</name>
      <Breed>labrador</Breed>
   </dog>
   <dog>
      <name>Marty</name>
      <Breed>whippet</Breed>
      <a/>
   </dog>
   <dog/>
   <cat name="Matilda"/>
   <a/>
</Animals>
```

Transformed JSON:

```
{"Animals": {
   "a": null,
   "cat": {"name": "Matilda"},
   ""dog": [
      {
         "Breed": "labrador",
         "name": {
            "$t": "Rufus",
            "attr": "1234"
         }
      },
      {
         "Breed": "whippet",
         "a": null,
         "name": "Marty"
      },
      null
   ]
```

```
}}
```

5.7.2 Serialization rules: structure-aware conversion

The general approach as defined above relies only on the information in the XML data instance.

The structure-aware approach defined in this section considers information in a data instance (e.g. XML) plus further information about the data structure definition (such as the allowed number of element occurrences), as documented in the API specifications and XML Schemas.

This structure-aware approach allows having always the same JSON structure to convey lists of elements.

> In this conversion approach, the rules above apply, except for the following modification to the conditions in a (i) and a (ii):If an element is allowed to appear more than once at the same hierarchical level, it SHALL be treated according to a (ii) as element list, otherwise it SHALL be treated according to a (i) as single element.

5.7.2.1 Example (Informative)

The following example illustrates the structure-aware serialization.

In the example, the data instance is represented as XML document:

```
<Animals>
   <dog>
      <name attr="1234">Rufus</name>
      <Breed>labrador</Breed>
   </dog>
   <dog>
      <name>Marty</name>
      <Breed>whippet</Breed>
                        <a/>
   </dog>
   <dog/>
   <cat name="Matilda"/>
   <a/>
</Animals>
```

The information about the data structure is represented as XML schema in this example. Note that the maximum cardinality of the elements is the only piece of information that is used here.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Animals">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="dog" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="name" minOccurs="0">
                                <xsd:complexType>
                                    <xsd:simpleContent>
                                        <xsd:extension base="xsd:string">
                                            <xsd:attribute name="attr" type="xsd:string"/>
                                        </xsd:extension>
```

```
                                    </xsd:simpleContent>
                                </xsd:complexType>
                            </xsd:element>
                            <xsd:element name="Breed" type="xsd:string" minOccurs="0"/>
                            <xsd:element name="a" minOccurs="0"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="cat" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:simpleContent>
                            <xsd:extension base="xsd:string">
                                <xsd:attribute name="name" type="xsd:string" use="required"/>
                            </xsd:extension>
                        </xsd:simpleContent>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="a"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Transformed JSON:

```
{"Animals": {
   "dog": [
      {
         "name": {
            "$t": "Rufus",
            "attr": "1234"
         }
         "Breed": "labrador",
      },
      {
         "name": "Marty"
         "Breed": "whippet",
         "a": null,
      },
      null
   ]
   "cat": [{"name": "Matilda"}],
   "a": null,
}}
```

5.7.3 Rules for JSON-creating and JSON-consuming applications

A JSON-creating application SHALL use either the *structure-aware* or the *general* approach, but not both.

Applications that consume a JSON representation SHALL accept the following two different JSON representations for an array that contains one element:

1. a pair of name and value (e.g. “name”: “one”)
2. a pair of name and array of one value (e.g. “name”: [“one”])

Note: In JSON, according to [RFC4627], the order of objects is not significant, whilst the order of values within an array is.

6. Shared Data Type Definitions

This section contains data type definitions which are shared among two or more REST protocol bindings.

6.1 Address data items

Addresses, unless the specification provides specific additional instruction, MUST conform to the address portion of the URI definition provided in RFC 3966 [RFC3966] for 'tel:' addresses, [RFC3261] for 'sip:' addresses or the definition given below for shortcodes or aliased addresses. Optional additions to the address portion of these URI definitions MUST NOT be considered part of the address accepted by the ParlayREST interfaces, and an implementation MAY choose to reject an address as invalid if it contains any content other than the address portion.

When processing a 'tel:' URI, as specified in [RFC3966], ParlayREST interface MUST accept national addresses (those not starting with '+' and a country code) and MUST accept international addresses (those starting with '+' and a country code).

When specified in the definition of a service operation, the URI may contain wildcard characters in accordance with the appropriate specification (i.e. [RFC3966] or [RFC3261]).

Shortcodes are short telephone numbers, usually 4 to 6 digits in length reserved for telecom service providers' own functionality. They shall be differentiated from national addresses by the use of a 'short:' rather than 'tel:' URI scheme. The short code defined in the URI consists of a string of digits with no non-digit characters.

Support for aliases in addresses is provided by use of the URI defined in [RFC3986]. This allows for arbitrary data to be submitted to the ParlayREST interface. The following is an example of how this could be applied:

<uri scheme>:<generic syntax>

An alias is generally a relatively short character string that holds a scrambled address such that only the application identified in the URI can expand it.

6.1.1 Charging

This section deals with in-band charging, i.e. passing charging data as part of the API request. To enable this capability to be provided across a variety of services in a consistent manner, the information to be provided in the message for charging information is defined as a common charging data type.

6.1.2 Charging data type

The charging information is provided in an XML data type, using the following schema.

```
<xsd:complexType name="ChargingInformation">
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
    <xsd:element name="currency" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="amount" type="xsd:decimal" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="code" type="xsd:string" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

The application accessing the Service provides this information:

- *Description* is an array of text. The first entry of a list will often be used to provide billing text. This text does not have specific required content, but would likely include information on the business, the content or service provided, and a transaction identifier. Credit card statements are a good example of description text provided by different companies.
- When more than one entry is provided, the rest should be references to individual operations relevant to the charging. Reference should be set to a value provided in a response message to the operation as a unique identifier to correlate individual operation.
- *Currency* in which the charge is to be applied. Values for the currency field are defined by ISO 4217 [ISO4217].
- Defines the amount to be charged.

- *Code* specifies a charging code which references a contract under which this charge is applied. The code identifier is provided by the Service Provider.

The charging information provided may not be acceptable to the Service Provider. For example, the Service Provider may limit the amount that may be specified for a particular Service or for a particular Service Requester. If the information provided is not acceptable, an appropriate fault message may be returned to the requester (SVC0007 and POL0012 are defined as a generic charging fault, The 'SVC' and 'POL' service exceptions are defined in [ParlayX_Common]).

Especially in case of charging operation such as creating a charge or refund, it is strongly recommended to convey a list of relevant operations related to charging over a description part as described above.

This is useful especially when a charging operation is performed after a certain set of operations.

Some of the services may be meaningful to the user only when a certain set of operations is completed. In that case, service provider may want to charge a user only upon a completion of the entire process, instead of charging per operation. Also, service provider may want to control the actual amount of charging depending on a certain condition, e.g., service usage volume, independent of the volume control provided by the network operators. This is also the case where it is preferable to perform charging operation after a completion of certain set of operations. In these cases where a service provider charges a user for the consumption of a certain service, the service provider is recommended to provide the references to the individual operations performed as evidences. This information can be referenced by the relevant entities to ensure the validity of charging when necessary.

It should be noted that this is for a service provider to provide a list of evidences of their direct use of operations. Any mapping of underlying operations performed internally in the operator must be performed by the operator if necessary. How to maintain the consistency between the information kept at service provider and the operators is out of scope. Also, charging aspects which do not relate to any operations are not covered.

6.2 Common data types

6.2.1 Enumeration: NotificationFormat

List of notification format values.

Enumeration	Description
XML	Notification about new inbound message would use XML format in the POST request
JSON	Notification about new inbound message would use JSON format in the POST request

Table 1: NotificationFormat Values

6.2.2 Enumeration: TimeMetrics

List of time metric values.

Enumeration	Description
Millisecond	Millisecond
Second	Second
Minute	Minute
Hour	Hour
Day	Day
Week	Week
Month	Month
Year	Year

Table 2: Time Metrics Values

6.2.3 Type: TimeMetric

For services that provide service based on a time interval or duration or similar metric, this type is used to specify the time metric.

Element	Type	Optional	Description
metric	TimeMetrics	No	Metric to use for time measurement
units	xsd:int	No	Number of units of TimeMetrics

Table 3: TimeMetric Structure

6.2.4 Type: ChargingInformation

For services that include charging as an inline message part, the charging information is provided in this data structure.

Element	Type	Optional	Description
description	xsd:string [1..unbounded]	No	An array of description text to be use for information and billing text
currency	xsd:string	Yes	Currency identifier as defined in ISO 4217 [ISO4217]
amount	xsd:decimal	Yes	Amount to be charged/refunded/reserved. The amount to be charged/refunded/reserved appears either directly in the amount-field or as code in the code-field. If both these two fields are missing or empty a service exception (SVC0007) will be thrown.
code	xsd:string	Yes	Charging code, referencing a contract under which the charge is applied

Table 4: ChargingInformation Structure

6.2.5 Type: CallbackReference

An application can use the CallbackReference data structure to subscribe to notifications.

If a parameter *callbackData* has been passed in a particular subscription, the server MUST copy it into each notification which is related to that particular subscription.

Element	Type	Optional	Description
notifyURL	xsd:anyURI	No	Notify Callback URL
callbackData	xsd:string	Yes	Data the application can register with the server when subscribing to notifications, and that are passed back unchanged in each of the related notifications. These data can be used by the application in the processing of the notification, e.g. for correlation purposes.
notificationFormat	NotificationFormat	Yes	Default: XML Application can specify format of the resource representation in notifications that are related to this subscription. The choice is between {XML, JSON}

Table 5: CallbackReference Structure

Note: In case the application requires correlating notifications to the related subscription, it can either submit a different *notifyURL* in each subscription, or use the optional *callbackData* parameter as a correlator.

6.2.6 Type: ResourceReference

Element	Type	Optional	Description
resourceURL	xsd:anyURI	No	The URL that addresses the resource

Table 6: ResourceReference Structure

The *resourceReference* element of type *ResourceReference* is defined as a root element in the XSD.

6.2.7 Type: Link

Attribute	Type	Optional	Description
rel	xsd:string	No	Describes the relationship between the URI and the resource
href	xsd:anyURI	No	URI

Table 7: Link Structure

An element of type *Link* can be provided by the server to point to other resources that are in relationship with the resource The *rel* attribute is a string. The possible values for the string are defined in each REST enabler.

Rel and *href* are realized as attributes in the XSD.

6.2.8 Type: RequestError

Element	Type	Optional	Description
link	Link[0..unbounded]	Yes	Link to elements external to the resource
serviceException	ServiceException	Choice	Exception Details
policyException	PolicyException	Choice	Exception Details

Table 8: RequestError

A requestError element of type *RequestError* is defined as a root element in the XSD.

6.2.9 Type: ServiceException

Element	Type	Optional	Description
messageId	xsd:string	No	Message identifier, with prefix SVC
text	xsd:string	No	Message text, with replacement variables marked with %*n*, where *n* is an index into the list of <variables> elements, starting at 1
variables	xsd:string [0..unbounded]	Yes	Variables to substitute into Text string

Table 9 ServiceException

6.2.10 Type: PolicyException

Element name	Element type	Optional	Description
messageId	xsd:string	No	Message identifier, with prefix POL
text	xsd:string	No	Message text, with replacement variables marked with %*n*, where *n* is an index into the list of <variables> elements, starting at 1
variables	xsd:string [0..unbounded]	Yes	Variables to substitute into Text string

Table 10: PolicyExceptionResponse Codes

6.2.11 Type: ServiceError

In a response to a request, ServiceError is used when an operation involving multiple items fails for only some of the items, whereas ServiceException is used where the entire operation fails.

In notifications, ServiceError is always used to indicate a notification termination or cancellation.

Element	Type	Optional	Description
messageId	xsd:string	No	Message identifier, either with prefix SVC or with prefix POL
Text	xsd:string	No	Message text, with replacement variables marked with %n, where n is an index into the list of <variables> elements, starting at 1
Variables	xsd:string [0..unbounded]	Yes	Variables to substitute into text string

Table 11: ServiceError

6.2.12 Enumeration: RetrievalStatus

Enumeration	Description
Retrieved	Data retrieved. Current data is provided
NotRetrieved	Data not retrieved, current data is not provided (does not indicate an error, no attempt may have been made). Note that this field is useful in case a list of addresses are requested, some items could be marked as "NotRetrieved" in case retrieval could not be attempted for some reason, e.g. to avoid time outs
Error	Error retrieving data

Table 12: RetrievalStatus

6.3 HTTP Response Codes

Following is the list of commonly used HTTP response codes for ParlayREST.

200 – Success

201 – Created

204 – No Content

304 - ConditionNotMet - Not Modified: The condition specified in the conditional header(s) was not met for a read operation.

400 - Invalid parameters in the request

401 - Authentication failure

403 - Application don't have permissions to access resource due to the policy constraints (request rate limit, etc)

404 - Not Found - The specified resource does not exist.

405 - Method not allowed by the resource

409 - Conflict

411 - Length Required: The Content-Length header was not specified.

412 - Precondition Failed: The condition specified in the conditional header(s) was not met for a write operation.

413 - RequestBodyTooLarge - Request Entity Too Large: The size of the request body exceeds the maximum size permitted.

416 - InvalidRange - Requested Range Not Satisfiable: The range specified is invalid for the current size of the resource.

500 - Internal server error

503 - ServerBusy - Service Unavailable: The server is currently unable to receive requests. Please retry your request.

6.3.1 Handling of not allowed HTTP methods.

If a method is not allowed by the resource, then server SHOULD also include the 'Allow: {GET|PUT|POST|DELETE} field in the response as per section 14.7 [RFC2616].

6.3.2 Service and Policy exceptions

In case of errors, additional information in the form of Exceptions MAY be included in the HTTP response.

Exceptions are defined with three data elements.

The first data element is a unique identifier for the message. This allows the receiver of the message to recognize the message easily in a language-neutral manner. Thus applications and people seeing the message do not have to understand the message text to be able to identify the message. This is very useful for customer support as well, since it does not depend on the reader to be able to read the language of the message.

The second data element is the message text, including placeholders (marked with %) for additional information. This form is consistent with the form for internationalization of messages used by many technologies (operating systems, programming environments, etc.). Use of this form enables translation of messages to different languages independent of program changes.

The third data element is a list of zero or more strings that represent the content to put in each placeholder defined in the message in the second data element with the first entry mapping to the placeholder %1.

6.3.2.1 Service exception

The *Service exception* is provided in an XML data type, using the following schema.

```
    <xsd:complexType name="ServiceException">
  <xsd:sequence>
    <xsd:element name="messageId" type="xsd:string"/>
    <xsd:element name="text" type="xsd:string"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="variables" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

When a service is not able to process a request, and retrying the request with the same information will also result in a failure, and the issue is not related to a service policy issue, then the service will issue a fault using the ServiceException fault message. A Service Exception uses the letters 'SVC' at the beginning of the message identifier. The 'SVC' service exceptions are defined in [ParlayX_Common]

Examples of *Service exceptions* include invalid input, lack of availability of a required resource or a processing error.

6.3.2.2 Policy exception

The policy exception is provided in an XML data type, using the following schema.

```
    <xsd:complexType name="PolicyException">
  <xsd:sequence>
    <xsd:element name="messageId" type="xsd:string"/>
    <xsd:element name="text" type="xsd:string"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="variables" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

When a service is not able to complete because the request fails to meet a policy criteria, then the service will issue a fault using the *Policy Exception* fault message. To clarify how a *Policy Exception* differs from a *Service Exception*, consider that all the input to an operation may be valid as meeting the required input for the operation (thus no *Service Exception*), but using that input in the execution of the service may result in conditions that require the service not to complete. A *Policy Exception* uses the letters 'POL' at the beginning of the message identifier. The 'POL' service exceptions are defined in [ParlayX_Common]

Examples of *Policy exceptions* include privacy violations, requests not permitted under a governing service agreement or input content not acceptable to the service provider.

6.3.3 HTTP Response Codes in response to Notifications

Handling of HTTP response codes sent by the client application, in response to a notification from the server:

1. in case of HTTP 2xx response codes, server assumes the notification has been sent successfully.
2. in case of HTTP response codes other than 2xx, the handling is left to the server implementation. The server MAY support different actions as dictated by a service provider policy (out-of-scope for this specification).

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
n/a	n/a	No prior version –or- No previous version within OMA

A.2 Draft/Candidate Version 1.0 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-TS-ParlayREST-Common-V1_0	25 May 2009	All	Baseline TS as per agreed: OMA-ARC-REST-2009-0002-INP_ParlayREST_TS_document Name changed to ParlayREST-Common Versionning fixed
	11 Nov 2009	2, 3, 4, 5, 6	Introduce initial structure (OMA-ARC-REST-2009-0064R01)
	2 Dec 2009	5, 6	Added OMA-ARC-REST-2009-0085R01-CR_Update_to_COMMON_TS.doc and OMA-ARC-REST-2009-0080R03-INP_ParlayREST_Link_issue.doc and OMA-ARC-REST-2009-0111-INP_Common_TS_Naming_Conventions and OMA-ARC-REST-2009-0100R01-CR_REST_Common_TS.doc
	3 Dec 2009	1	Update from OMA-ARC-REST-2009-0112-INP_Cleanup_Scope_of_Common_TS.doc
	3 Dec 2009	2	Updated from OMA-ARC-REST-2009-0114-INP_SMS_TS_Reference_Section
	3 Dec 2009	3	Updated from OMA-ARC-REST-2009-0115R01-INP_SMS_TS_Section_3.doc
	3 Dec 2009	App. A	Added from OMA-ARC-REST-2009-0122R01-CR_Deployment_Considerations
	7 Dec 2009	All	Document clean-up, added lost Exception element tables.
	11 Dec 2009	all	Update after final CC, see OMA-ARC-REST-2009-0170-MINUTES_11Dec2009_CC for details OMA-ARC-REST-2009-0175R01-CR_Authorization_and_protection_of_sensistive_data
	16 Dec 2009	All	Editorial Fixes: History table
	1 Feb 2010	All	Applide the rule from OMA-ARC-REST-2010-0006R02 and OMA-ARC-REST-2010-0007R03
	2 Feb 2010	All	Applied G002, A002, A001, A003, A004, A005, A006, A007, A008, A009, A010, A011, A012, A013, A014, A015, A016
	4 Feb 2010	5	Added OMA-ARC-REST-2010-0048-CR_content_negotiation_case_convention_common, solving F011
	7 Feb 2010	All	Changed font colors, and spelling C0076
	15 Feb 2010	Added 5.6 6.2.5 6.2.6	OMA-ARC-REST-2010-0063R01-INP_Closing_proposal_for_echoing_issue.doc OMA-ARC-REST-2010-0059-CR_Some_fixes_TS_Common.doc
	16 Feb 2010	Added 5.7	OMA-ARC-REST-2010-0060R01-CR_Xml2json_conversion_common.doc
	18 Feb 2010	5.4	Reapplied OMA-ARC-REST-2010-0048-CR_content_negotiation_case_convention_common.doc
	25 Feb 2010	Added	Added OMA-ARC-REST-2010-0074-INP_JSON_type_in_Notifications.doc and OMA-ARC-REST-2010-0081R01-CR_Correlator_resolution_TS_Common
	26 Feb 2010	Added	Added OMA-ARC-REST-2010-0073R01-CR_Echoing_response_decision_not_compliant_with__RFC_2616, the SHALL solution
	27 Feb 2010	Corrected	NotificationFormat, reapplied #74
	07 Mar 2010	5.7.2	Added OMA-ARC-REST-2010-0062R02-CR_XML2JSONTool.doc
	19 Mar 2010	All	Updates from walk-through
	26 Mar 2010	All	Editorial updates/formatting
	30 Mar 2010	6.3	Editorial updates: Caption for Table 10 fixed Numbering of 6.3 fixed

Document Identifier	Date	Sections	Description
Candidate Version: OMA-TS-ParlayREST-Common-V1_0	27 Apr 2010	All	Status changed to Candidate by TP: OMA-TP-2010-0186- INP_ParlayREST_V1_0_ERP_for_Candidate_Approval
Draft Versions: OMA-TS-ParlayREST-Common-V1_0	09 Jun 2010	5.5 5.6 6.2.9 6.2.11 6.3, 6.3.3	Implemented Agreed Changes: OMA-ARC-REST-2010-0180- CR_Fixing_implementation_of_CR83R01_TS_Common OMA-ARC-REST-2010-0248-CR_Wrong_capitalization_in_ServiceError. OMA-ARC-REST-2010-0251-CR_common_1.0_RetrievalStatus OMA-ARC-REST-2010-0263R01- CR_Common_handling_HTTP_responses
	11 Jun 2010	6.2.11	Implemented Agreed Change: OMA-ARC-REST-2010-0231R01- INP_CR_Common_TS_add_ServiceError_structure
Candidate Version: OMA-TS-ParlayREST-Common-V1_0	24 Aug 2010	All	Status changed to Candidate by TP: OMA-TP-2010-0359- INP_ParlyREST_V1_0_ERP_for_Candidate_reapproval
Draft Version: OMA-TS-ParlayREST-Common-V1_0	04 Oct 2010	6.2.4	Implemented Agreed Change: OMA-ARC-REST-2010-0557-CR_amount_in_common_TS
Candidate Version: OMA-TS-ParlayREST-Common-V1_0	23 Nov 2010	All	Status changed to Candidate by TP: OMA-TP-2010-0463R01- INP_ParlayREST_V1_0_ERP_for_Candidate_reapproval

Appendix B. Deployment Considerations (Informative)

Applications using the ParlayREST API can be categorized by their execution environment:

- Application is a ParlayREST client application executing in a server execution environment (e.g. a 3rd party application).
- Application is a ParlayREST client application executing in a mobile device execution environment.
- Application is a ParlayREST client application executing in a fixed device execution environment.

ParlayREST API client can execute in any of the above execution environments.

Issues that are dependent on the ParlayREST execution environment and can impact strategic deployment decisions, interoperability, and scalability include (non-exhaustive list):

- Notifications sent from ParlayREST server to ParlayREST client application, for example:
  i. There must be an active "listener" on the application host (in this case the client device), ready to receive the incoming notification via the HTTP protocol.
  ii. This does not have to be the application itself, but at least some host service/client which can invoke the specific application when needed.
  iii. In a client-server HTTP binding, this requires that the client has the support of an HTTP listener service.
- Security aspects (e.g. ParlayREST client application authentication)

While solutions to particular issues related to the ParlayREST client application execution environment are out-of-scope for the ParlayREST enabler, other OMA enablers should be re-used (where applicable) to address such particular issues.

B.1 ParlayREST client application executing in a server execution environment

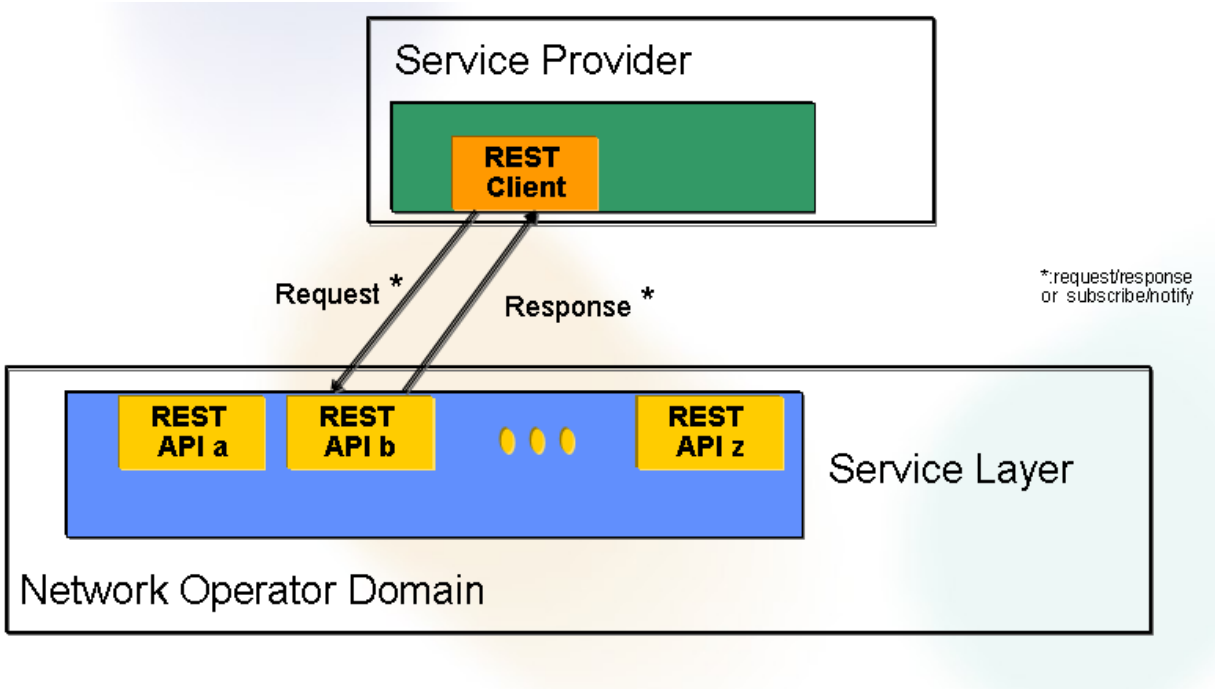

Figure 1 ParlayREST API accessed from a server execution environment (e.g. 3rd party Service Provider application)

The RESTful API exposed by ParlayREST server deployed in the Network Operator service layer domain, may be accessed by a ParlayREST client application executing on a server resident in the Service Provider domain.This deployment can support all resources and operations specified in ParlayREST. There are no particular issues with support of notifications from ParlayREST server to ParlayREST client application. For security aspects, see the Common TS security considerations section.

B.2 ParlayREST client application executing in a mobile device execution environment

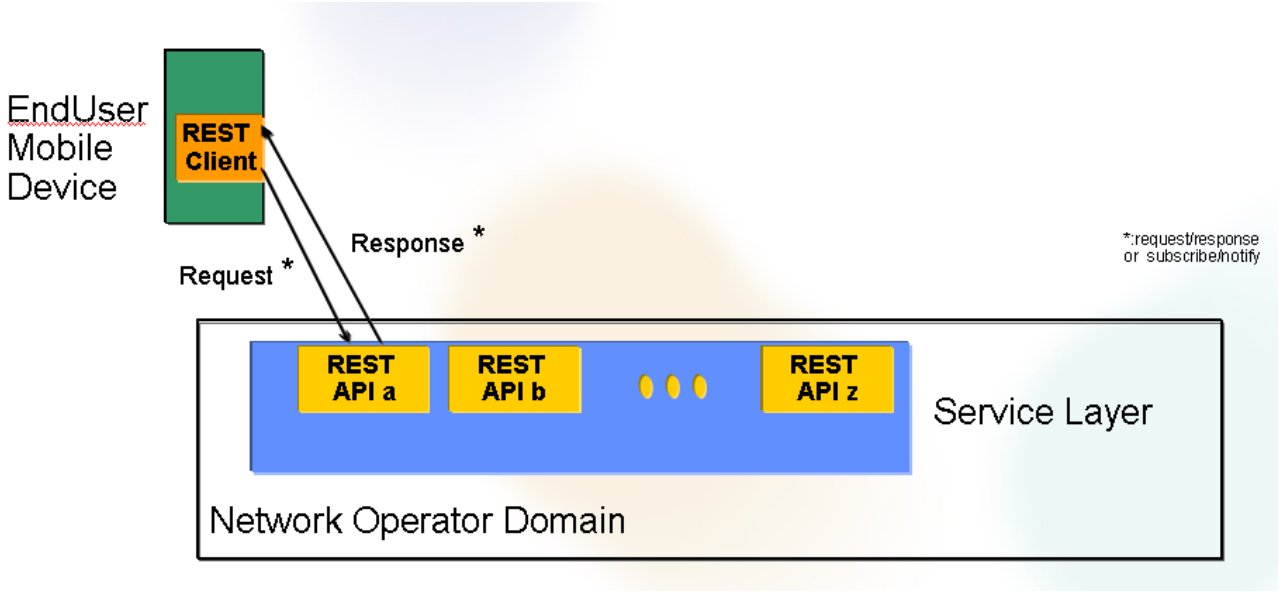

Figure 2 ParlayREST API accessed from a mobile device execution environment

The RESTful API exposed by ParlayREST server deployed in the Network Operator service layer domain, may be accessed by a ParlayREST client application executing on an end user mobile device. This deployment can support most resources and operations specified in ParlayREST. There are however particular issues with support of notifications from ParlayREST server to ParlayREST client application:

- Typically in mobile devices, the client does not have the support for an HTTP listener service. The specified client notification APIs may have to be delivered by alternative means. OMA Push [Push] should be considered to be used to deliver the notifications to the ParlayREST client application.
- It must be possible to actually deliver the notification to the ParlayREST client application, i.e. there must be no boundary across which the protocol is typically blocked. In a client-server HTTP binding, this will typically be an issue as
    - The client is typically within some private network behind a firewall (e.g. PLMN Operator mobile network or home network)
    - The client does not have a fixed IP address or an IP address that is resolvable via DNS.
    - In such cases, a notification service such as OMA Push should be considered to be used to bridge the firewall border and resolve the target address of the notification to an actual client address.

For security aspects, see the Common TS security considerations section.

B.3 ParlayREST client application executing in a fixed device execution environment

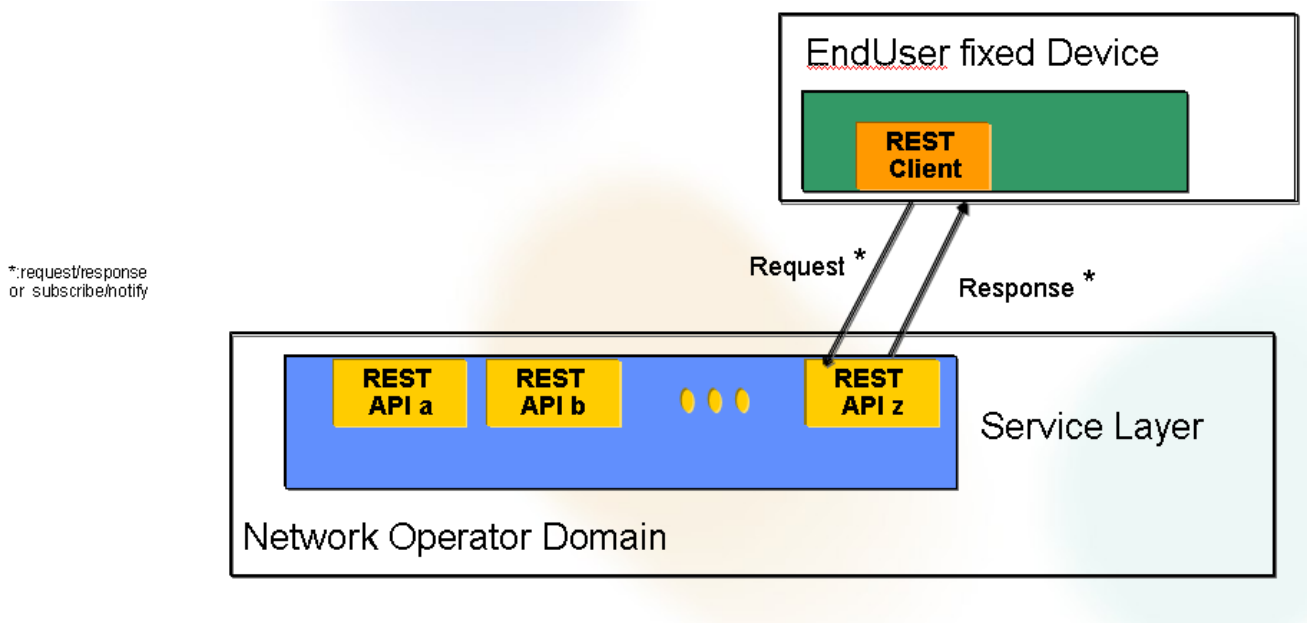

Figure 3 ParlayREST API accessed from by a fixed device execution environment

The RESTful API exposed by a ParlayREST server deployed on the Network Operator service layer domain, may be accessed by a ParlayREST client application executing on a fixed device connected to the Network Operator.

This deployment can support most resources and operations specified in ParlayREST. Some issues with support of notifications from ParlayREST server to ParlayREST client applications may be similar to those mentioned in Appendix B.2. Solutions to those issues may however rely on other mechanisms (e.g. use of COMET).

For security aspects, see the Common TS security considerations section.