low Energy COnsumption NETworks

REST API for the GAL

Version 1.2

Table of Contents

1. General Concepts

This DRAFT document describes the XML/HTTP REST API binding of the GAL interface described in D4.3.

We consider that document as the conceptual definition of a generic network-, device- and subdevice-level power management interface and also a programming language-style API definition.

While D4.3 can be used in quite a staightforward way to implement a server library, or write a GAL client code (e.g. in „C”), it does not define any binding for network communication between GAL server (e.g. a GAL-aware server) and a GAL network client (e.g. a higher level controlling application).

This document specifies binding for the GAL conceptually equivalent to the D4.3 interface, but using a network protocol (rather then library calls within a runtime) as the „vehicle” of information exchange.

1.1. Place of the GAL interface within the ECONET management framework

Being focused on the energy-related information only (states and measurements), the GAL alone is not expected to be used to monitor and manage a complex network of telecommunication service devices, but rather it is an extension of other interfaces used for the discovery and monitoring of networks from other perspectives (topology, performance, faults, etc.)

This is similarly reflected in the ECONET demonstration architecture, where the GAL interface described here is just one of three interfaces used for communication between the managers and the managed networks and elements (see Fig 1.):

- The Topology interface (Ref. 2) is the principal mechanism for describing the networks, the devices (including their internals if necessary), and the interconnections betweend ports on the devices. In addition, the topology information will contan the definition of the GAL interface access points available for executing GAL commands (and similarly points of performance data access is also described here).
  The Topology inteface is thus the key for all information exchange between the managers and managed systems.
- The other interface is the GAL itself
- The third interface is used to query data from the manaeged systems (networks, devices, etc). that fall outside of the GAL scope, but are necessary for making demonstrating and evaluating the energy-aware behavior of the devices, especially the impact of the energy states, and the consequential trade-offs. Examples of such performance data are momentary

latency, bitrate, temperatures, etc.
The performace data interface is kind of an auxiliary mechanism, as we expect that a production-scale telecom network environment will already have a performance monitoring framework in place.

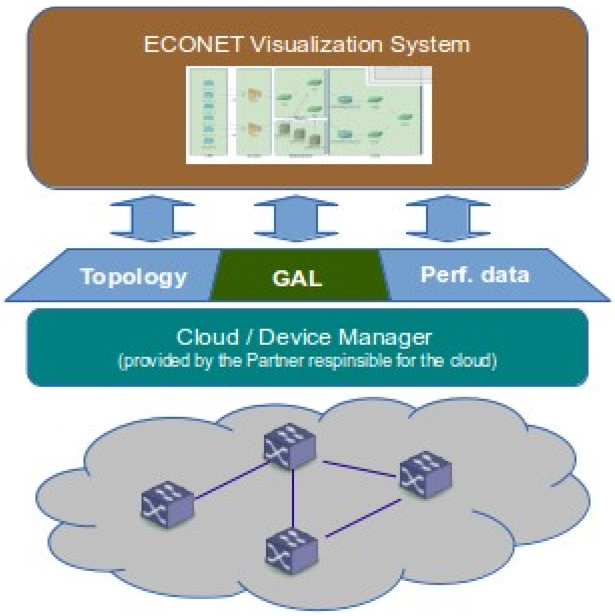

ECONET Visualization System
Topology
GAL
Perf. data
Cloud / Device Manager
(provided by the Partner respinsible for the cloud)

Fig 1. The GAL and other interfaces in the ECONET architecture

This „protocol vehicle” is set of XML fragments used over a REST-style HTTP client-server interface.

1.2. Introduction to the REST.style interfaces

REST (Representational State Transfer), is a popular network API architectural style, which is characterized by the consistent usage of the limited HTTP command vocabulary (GET, PUT, POST, DELETE, etc.), to implement functionality through a series of read and write operations on an object-oriented database. Sections of this database are represented as documents (XML documents in our case) during the client-server communication.

REST is an instance (and competitor) of several similar network API architectural styles, e.g. SOAP, CORBA, DCE, and many other proprietary or language-bound alternatives. It advantages are

- conceptual clarity
- fairly easy implementation on both sides
- it is suitable for proxying, caching and HTTP-level security enforcement

The basis of REST is a conceptual hierarchical „document” which describes the entirety of the information covered and handled by the API. We call it in our case the „GAL REST XML tree” which describes all information covered by the GAL (including writeable state variables). Since the GAL is hierarchical, the tree is also hierarchical: each level represents a GAL resource, with children corresponding to the subordinate resources (e.g. devices in a network, or ports on a linecard, etc.).

Please see the section 2. for the GAL tree.

The GAL REST API works by providing access to parts of this tree through HTTP operations. For example

- discovery is a HTTP GET query where the URL path and the URL query parameters specifying the information to be returned
- provisioning is a HTTP PUT, with the state specified as an XML fragment in the document part of the HTTP request.
- commit and rollback are POST operations.

In all cases the information returned consists of an error code and/or a document containing the data requested, e.g. for successful operations HTTP return code 200/OK is returned, while operations on nonexistent resurces return 404/NOTFOUND, ansd invalid operations return 401/BAD REQUEST status codes.

Please see the section . for a detailed description of the operations.

1.3. Further network API bindings for the GAL

While this document describes the REST API only, it is possible to define alternative network protocols with the same functionality. Indeed, we are considering the definition of further „GAL vehicles” like

- An SNMP API where the resource state is represented as an SNMP MIB, and SNMP protocol (GET/SET/WALK, and maybe TRAP) is used to access this information

- A CWMP TR-069 binding (called 'profile' in the CWMP world), that provides asynchronous access to the GAL exposed by CPE devices. Please note that since these devices are very simple, the GAL to be exposed is also very simplified. On the other hand extended asynchronity of the communication (e.g. delays of up to 24 hours) mandates that the API also exchanges the history of certain states and monitored variables.

- A TMF814-compliant CORBA API to be used with resources that support that protocol.

These alternative bindings are detailed in separate documents.

2. The GAL REST XML Tree

The principal tag of the XML tree is a GAL resource, Each resource is described by the tag <gal:resource>. Each resource is identified by an ID, usually a variable-length dotted format.

Within the gal:resource tag, the following children nodes supported

<gal:provisionedState>: the state provisioned for the resource
<gal:committedState>: the state committed for the resource
<gal:powerStates>: list of the supported powerstates of this device
<gal:monitoring>: a section of monitored parameters along with their momentary values.
<gal:url> this tag indicates that the details of a resource are available from another GAL server (see also Section 4.)

In addition GAL resources can contain further resources described as children:

<gal:physicalChildren>: physcal children
<gal:logicalChildren>: logical children

Please not that this XML tree is not a document and it does not necessarily exist anywhere in its fullness, but it is rather a conceptual information base which can be partially queried and manipulated by the commands described below.

```
<gal:resource  gal_id=”0.0.1” description=”AbcAbc1.0”   type=”linecard” >
        <gal:provisionedState state_id=”s0p0”/>
        <gal:committedState state_id=”s0p1”/>

        <gal:powerStates>
                <gal:state state_id=”s0p0”
                                minimum_power_gain=”1.0”
                                maximum_packet_throughput=”10000”
                                maximum_bit_throughput=”12000000”/>
                <gal:state state_id=”s0p1” ….  />
                <gal:state state_id=”s0p2” ….  />
                <gal:state state_id=”s1” ….  />
        </gal:powerStates>

        <gal:physicalChildren count=”n”>
                <gal:resource gal_id=”0.0.1.1” description=”Port 1”   type=”port” >
                        …
                </gal:resource>
                <gal:resource gal_id=”0.0.1.1” description=”Port 2”   type=”port” >
                        …
                </gal:resource>
        </gal:physicalChildren>
        <gal:logicalChildren  count=”n”>
                <gal:resource gal_id=”0.0.1.A” description=”Xyz1.0.1”   type=”logical” >
                        …
                </gal:resource>
                <gal:resource gal_id=”0.0.1.B” description=”Xyz1.0.1”   type=”logical” >
                        …
                </gal:resource>
                ...
        </gal:logicalChildren>

        <gal:monitoring>
                <gal:momentaryConsumption value=”212”/>
                <gal:accumulatedConsumption value=”199” uptime=”38900”/>
        </gal:monitoring>
</gal:resource>
```

3. GAL operations

3.1. Discovery

HTTP request format

basic format:

```
    GET   http://<address>/GAL/0.0.1/
```

or an equivalent alternative sytax:

```
    GET   http://<address>/GAL?id=0.0.1
```

or extended with parameters:

```
    GET   http://<address>/GAL?id=0.0.1&propertySelector=children
```

Query parameters

id=	an alternative way to specifiy a resource within the GAL
childDepth=	an integer number indicating the depth of the tree returned, 0 the default returns only the entity addressed.
porpertySelector=	a string specifying the fields of the resource to be returned. Default is **'all'** which cover everything in the rest tree, **except the <monitoring> sections**. Values of this parameter can be any of the following: {**provisionedState, committedState, powerStates, physicalChildren, logicalChildren**}; and the following aggregate selectors can also be used {**all, children, states**}. Multiple propertySelectors can be specified, which effectively selects a union of the properties specified.

Please note that the 'monitoring' property section is not addressable by the propertySelector and it is neither included in the 'all' aggregate. Consequently, the Monitoring operation (described at 3.4, below) need to be used to access it.

Children are enumerated by default values of parameters (childDepth=”0”, propertySelector=”all”), but further info on the children is not provided.

HTTP status codes returned

200 OK: resource found and returned

404 NOTFOUND: resource not found

301 MOVED PERMANENTLY: the server instructs the client to use a different URI (which is returned). This may happen if the GAL at one level redirects the client to another GAL , e.g. an network manager (NMS) GAL redirects a request to the GAL of a certain device.

501 NOT IMLEMENTED: this may indicate that certain query parameters are not supported by the GAL serving the request

Document returned

The returned document is a subtree of the full GAL tree shown in the previous section. While the full tree is usually quite extensive, the returned document is normally „clipped" to a manageable size through the **childDepth** and **propertySelector** query parameters. E.g.:

```
GET   http://<address>/GAL?id=A.00.0.1&propertySelector=states

<gal:resource  gal_id="A.00.0.1" description="Card 0.1"
type="linecard" >
     <gal:provisionedState state_id="s0p0"/>
     <gal:committedState state_id="s0p1"/>
</gal:resource>
```

3.2. Provisioning

HTTP request format

```
PUT   http://<address>/GAL/0.0.1/provisionedState
```

Query parameters:

none

Document submitted

A provisionedstate section within the REST XML tree:

```
     <gal:provisionedState state_id="s0p2"/>
```

HTTP status codes returned

to be completed

3.3. Release

HTTP request format

```
DELETE   http://<address>/GAL/0.0.1/provisionedState
```

Query parameters

none

HTTP status codes returned

to be completed

3.4. Monitoring

This returns the monitoring subtree within the REST tree *(to be defined in detail)*

HTTP request format

```
GET   http://<address>/GAL/0.0.1/monitoring
```

Query parameters:

none

HTTP status codes returned

to be completed

Document returned

It is recommended that the REST server returns 3 metrics for monitoring, as shown in the example below:

```
<gal:monitoring>
     <gal:committedState value=”212”/>
     <gal:momentaryConsumption value=”212”/>
     <gal:accumulatedConsumption value=”199” uptime=”38900”/>
</gal:monitoring>
```

3.5. Commit

HTTP request format

```
POST   http://<address>/GAL/0.0.1/committedState
```

Query parameters:

none

Document submitted

```
<gal:commit/>
```

HTTP status codes returned

to be completed

3.6. Rollback

HTTP request format

```
POST   http://<address>/GAL/0.0.1/committedState
```

Query parameters:

none

Document submitted

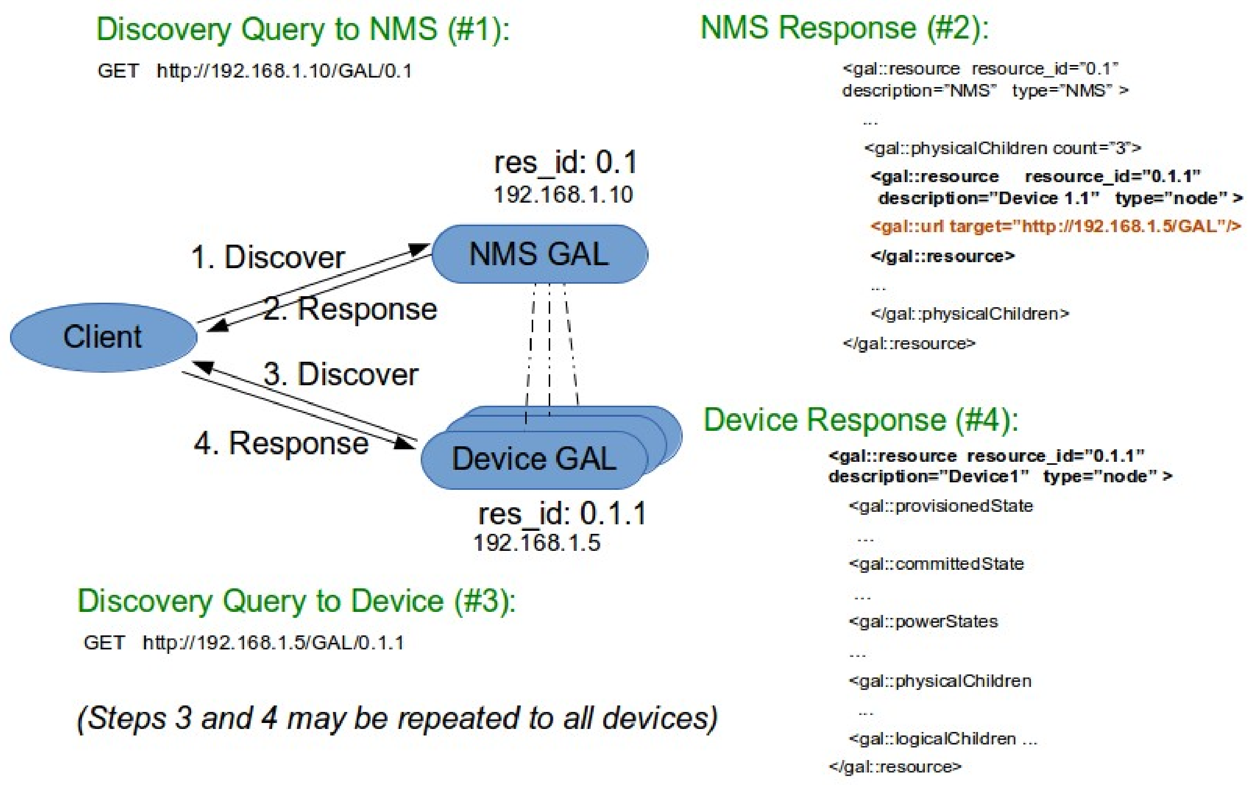
```
<gal:rollback/>
```

HTTP status codes returned

to be completed

4. Distributed GAL access

We expect that GAL is not necessarily a centralized informattion base, but it may also be distributed, in the sense that some information not available (ar writeable) at a higher level, may be available at a different level. The most typical example is a network management system, which lists its managed devices, but it cannot tell details of those devices. This information is accessible by accessing the GAL of the devices themselves only.



```
GET   http://192.168.1.10/GAL/0.1
```

```
GET   http://192.168.1.5/GAL/0.1.1
```

```
<gal::resource  resource_id="0.1"
description="NMS"   type="NMS" >
   ...
   <gal::physicalChildren count="3">
    <gal::resource    resource_id="0.1.1"
     description="Device 1.1"   type="node" >
    <gal::url target="http://192.168.1.5/GAL"/>
    </gal::resource>
    ...
    </gal::physicalChildren>
</gal::resource>
```

```
<gal::resource  resource_id="0.1.1"
description="Device1"   type="node" >
   <gal::provisionedState
    ...
   <gal::committedState
    ...
   <gal::powerStates
   ...
   <gal::physicalChildren
    ...
   <gal::logicalChildren ...
</gal::resource>
```

Fig 2. *NMS GAL only contains a reference to device level GAL-s*

5. References

1. Deliverable D4.3 - Abstraction Layer Final Definition: ECONET website / Documents/ WP4 / Deliverables

2. Topology format description for Econet ***(ECONET_Topology_XML_Interface_1.x)*** *:* ECONET website / Documents/ WP6 / Final demo planning