VOL-948: TC Layer Resource Manager per PON port

Approach with High Level Design Details

Appoach1 – Resource Manager as a Python Module

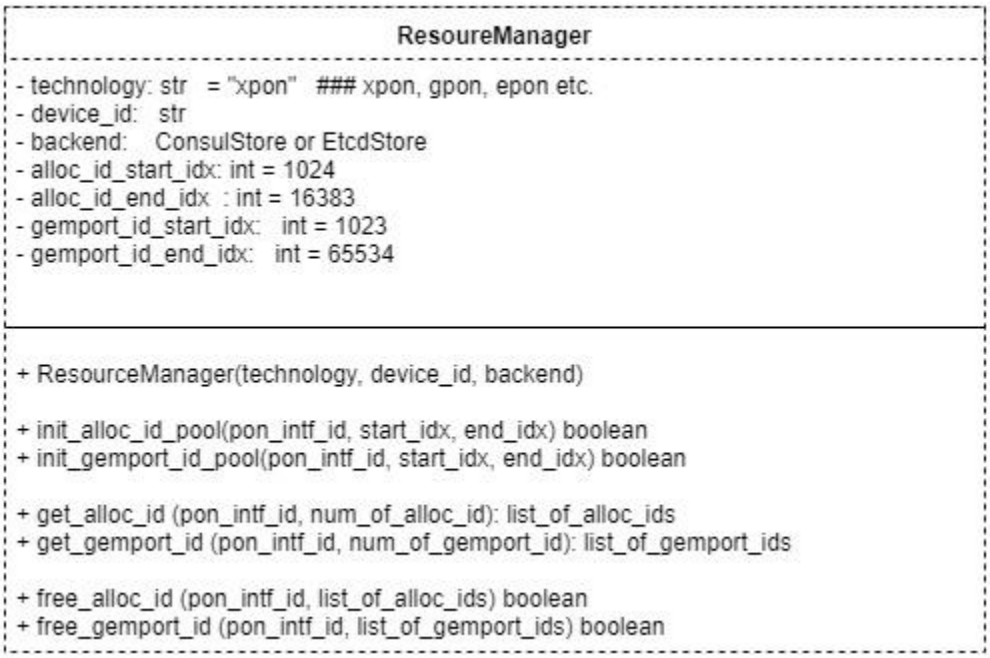
The Resource Manager is a Python module, which implements the `ResourceManager` class.

Adapter should import the module and instance of the `ResourceManager` should be created and initialized when the OLT Device is created.

A single instance of the `ResourceManager` exists per OLT device, and it exposes APIs to create/free alloc_ids/gem_ports.

The `ResourceManager` uses a KV store backend to ensure resiliency of the data.

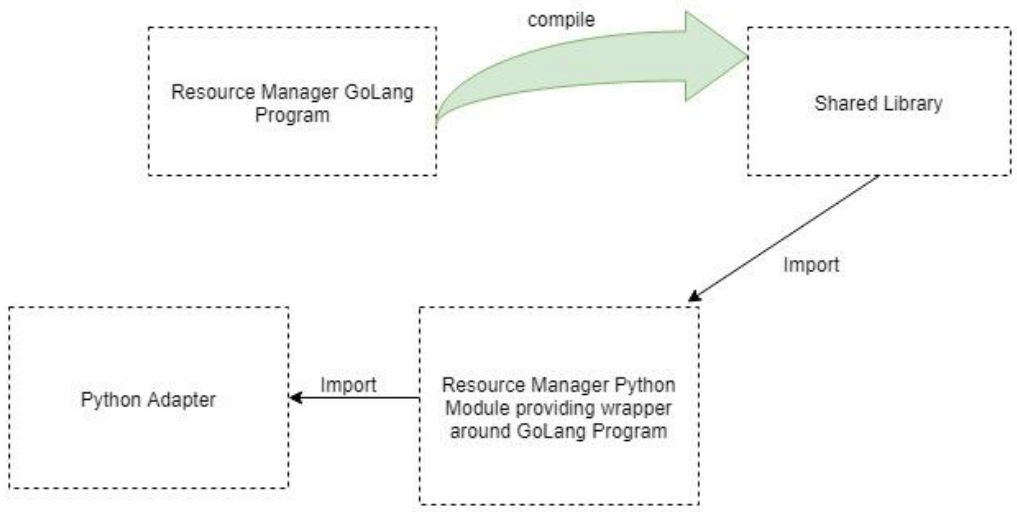
The UML diagram of the `ResourceManager` looks as below.



Approach2 – Resource Manager as GoLang module with Python wrappers

The Resource manager is implemented as a GoLang library with similar interfaces as explained above. This GoLang program is then compiled as a shared library and imported in a Python program. This Python program can be imported by OpenOlt (or other) adapters and work seamlessly.

Below diagram puts things in perspective for Approach2.



Advantages

1. The GoLang program could be re-used if/when we move entirely to GoLang

Disadvantages/Challenges

1. Not all GoLang data types can be easily mapped to their C Struct types in Python.
2. Additional overhead of writing, compiling and importing a GoLang program into Python modules.

Low Level Design Details

The `ResourceManager` will use the `BitString` utility to track the resources.

The K/V store path can look something like below

`/resource_manager/<technology>/<device_id>/alloc_id_pool/<pon_intf_id>/<alloc_id_string_blob>`

`/resource_manager/<technology>/<device_id>/gem_id_pool/<pon_intf_id>/<alloc_id_string_blob>`

Example:

`/resource_manager/xpon/0001a2ee3b54ea5c/alloc_id_pool/0/1000010000…`

Use Case

Initialize the Resource Manager

1) Use `init_alloc_id_pool` and `init_gemport_id_pool` to initialize alloc id and gemport id pools per PON port. Pass optional parameters `start_index, end_index`.

Allocate Resource(s)

1. When new resource(s) (gemport or tcont) is needed, query the `ResourceManager` using the `get_alloc_id` or `get_gemport_id` API.
2. `ResourceManager` responds with list (list size could be 1) of resources. The data is backed up on the K/V store for resiliency purposes.

Free Resource(s)

1. When resource(s) (gem port or tcont) are to be freed, use the `ResourceManager free_alloc_id` or `free_gemport_id` API.
2. `ResourceManager` responds with Boolean on the status of the free action. Data is backed up on K/V store for resiliency purposes.

Corner Cases

There are some corner cases related to adapter container (in Voltha 2.0) going down before resources are committed to device. In such cases, it is possible that the K/V store and device information could be out-of-sync. As of today, there is not much clarity on adapter containerization, resiliency and such topics. These need to be looked into later.