1. Introduction

Recently WSO2 Telco team released WSO2 Telco Hub 2.0.0 which is developed by moving the mediation layer of the WSO2.Telco Hub to WSO2 ESB. As per the new development architecture, main idea is to ease up the procedure which a developer has to follow to deploy a new API. In this document, it is intended to address those development guidelines.

2. WSO2.Telco Hub overview

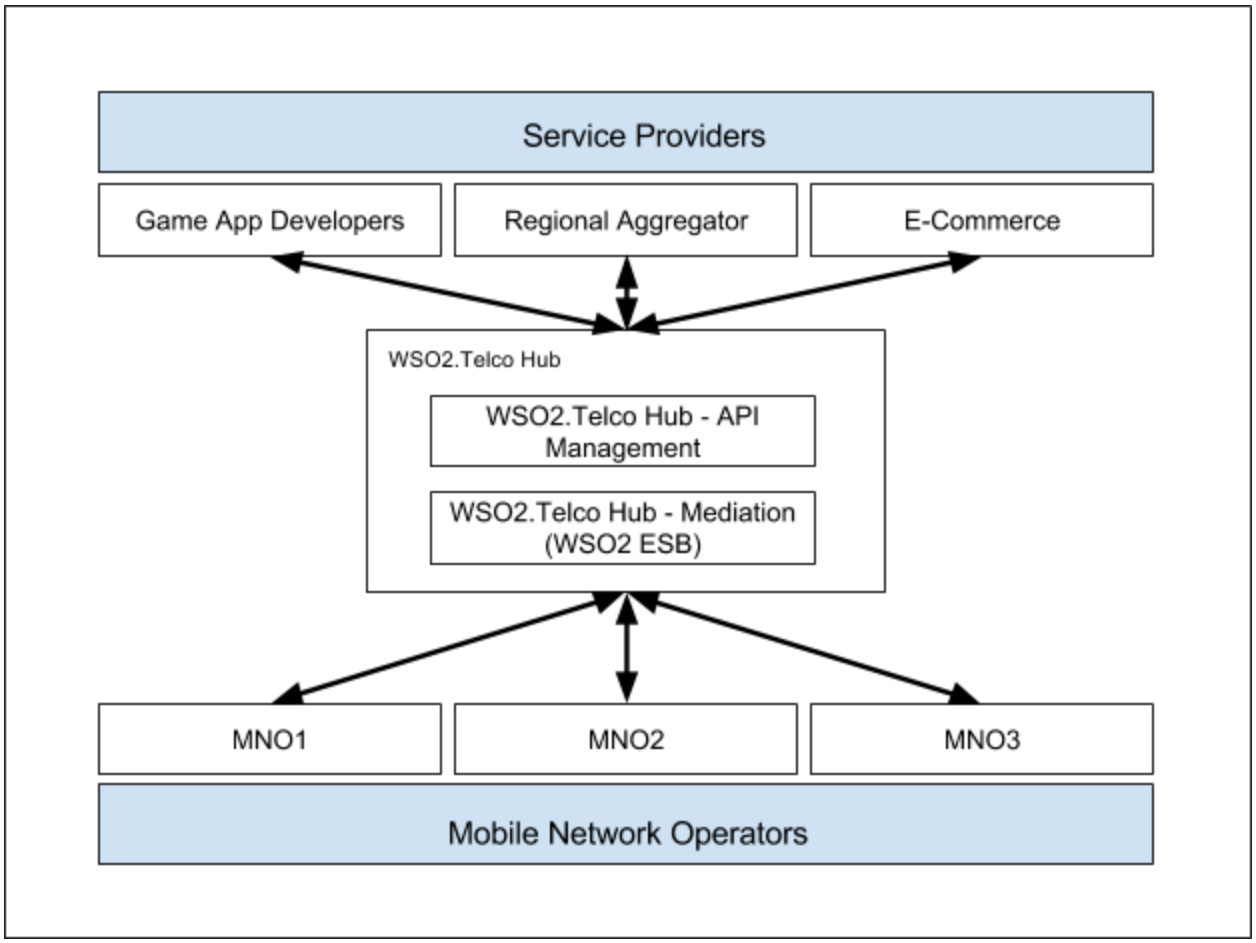

Figure 2.1 Hub overview

As in figure 2.1 illustrates, with the Hub v2.0.0. Mediation layer has been decoupled from the API Management, and moved to WSO2 ESB. Therefore basically to deploy a new API, developer needs to have the swagger definition which is to be deployed to WSO2.Telco Hub and the developed carbon application (capp) with the relevant mediation logic which needs to be deployed to WSO2 ESB.

With this design, API developer can release an API with a full QA cycle without affecting the roadmap or release cycle of another API and can guarantee the hot deployment of that particular API.

3. Adding an API to WSO2 Telco Hub

Once a developer setup the mediation layer (WSO2 ESB) according to the hub release document (README. [2]), developer can use the available common artefacts and mediation logics. Developer only has to implement the API specific logic.
Please refer the 4 and 5 headings to get an understanding on what each repository does and how you should follow the guidelines in developing APIs.

4. Mediation Common Repository

Mediation common repository [3] contains the common mediation logics which an individual API can leverage. It is required to deploy following common artefacts to WSO2 ESB to make use of the common mediation logics.

- Com.wso2telco.dep.common.mediation-2.0.0.jar
  (copy to <ESB>/repository/components/lib/ and restart)
  Dependency jars:
    dbutils.jar (repository: WSO2Telco/core-util)
    mnc-resolver.jar (repository: WSO2Telco/core-util)
    msisdn-validator.jar (repository: WSO2Telco/core-util)
    operator-service.jar (repository: WSO2Telco/component-dep)

- Com.wso2telco.dep.common.capp_2.0.0.car
  (copy to <ESB>/repository/deployment/server/carbonapps/)

```
.
├── com.wso2telco.dep.common.capp
│   ├── pom.xml
├── com.wso2telco.dep.common.mediation
│   ├── pom.xml
│   ├── src
│   │   └── main
│   │       └── java
│   │           └── com
│   │               └── wso2telco
│   │                   └── dep
│   │                       └── common
│   │                           └── mediation
│   │                               ├── dao
│   │                               │   └── APIDAO.java
│   │                               ├── DefaultTokenRetrieverMediator.java
│   │                               ├── DefaultTokenUpdaterMediator.java
│   │                               ├── EndpointRetrieverMediator.java
│   │                               ├── GenerateHashString.java
│   │                               ├── MSISDNBlacklistMediator.java
│   │                               ├── MSISDNWhitelistMediator.java
```

Figure 4.1: Directory structure of the common artefacts repository

Common functionalities available in mediation-dep-common

- Main sequences
  These are the set of mandatory sequences, a developer has to leverage in developing mediation logic of an API.
  - In (request)
  - Out (response)
  - Common fault handler
  - Unexpected error handler
  - Call endpoint
- Util sequences
  - Call notification endpoint
- Validation sequences/ templates
  Common validation templates/ sequences.
  - MSISDN
  - Double
  - Integer
  - Length
  - URL
  - MCC/MNC
  - Purchase category code
  - Mandatory parameter
  - Integer Offset and Limit
  - Channel
- Mediation sequences/ templates
  - Generate Client Correlator
  - Endpoint retriever
  - Select access token
  - Modify notification url
  - Replace hub request identifier
  - Replace service provider identifier
  - Service code retriever
  - Request Identifier generator
- Utilities
  Developer doesn't have to use these templates/ sequences because this has been already applied with the above templates/ sequences. But you may face unique scenarios where you might have to use them directly in your implementation.
  - Logging extension
  - Access token refresh
  - Access token retriever

Common configurations

IMPORTANT
Following configurations are deployed from the common capp and once an ESB instance gets restarted, the changed values will be revert back to default values which are in the capp.

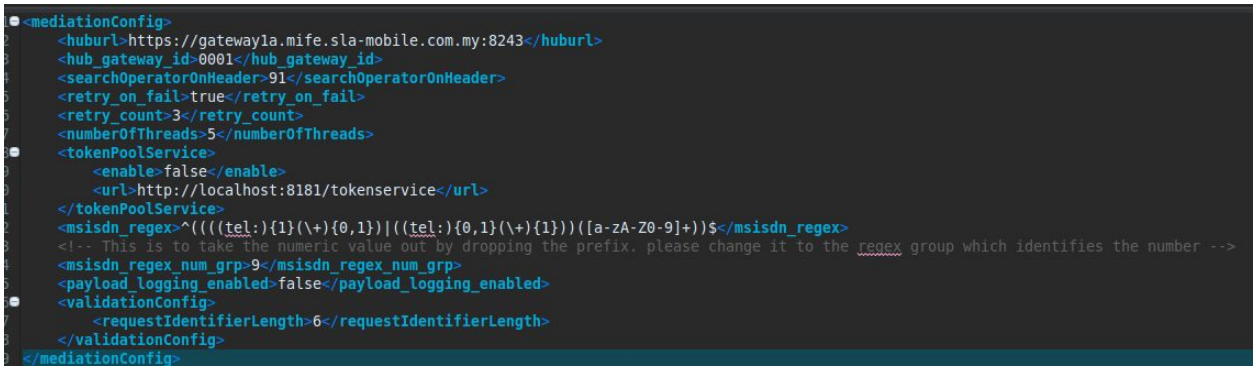

Figure 4.2: Mediation Configuration (mediationConfig.xml)

As in figure 4.2 shows, those are the configuration parameters of the common capp.

- Hub url: hub url after deploying
- Hub gateway id: unique id for the deployment
- Search Operator On Header: comma separated value to check operator value on header
- Token pool service: enable/ disable token pool and to configure token endpoint
- MSISDN regex: validation regex of the msisdn
- MSISDN regex number group: regex group number which contains the numeric value of the msisdn
- Payload logging enabled: logging extension configuration to log the payload
- Validation Configuration: configure request identifier validation length

Adding a common artefact

If a Sequence, Template, Endpoint or etc is something that can be leveraged by other APIs, that is an artefact which should go to the commons capp or mediator jar.

Since the logging extension feature is there to log every incoming and outgoing messages from ESB, you might need to consider if your common artefact is intend to sends/receives requests/ responses.

5. Mediation Hub Repository

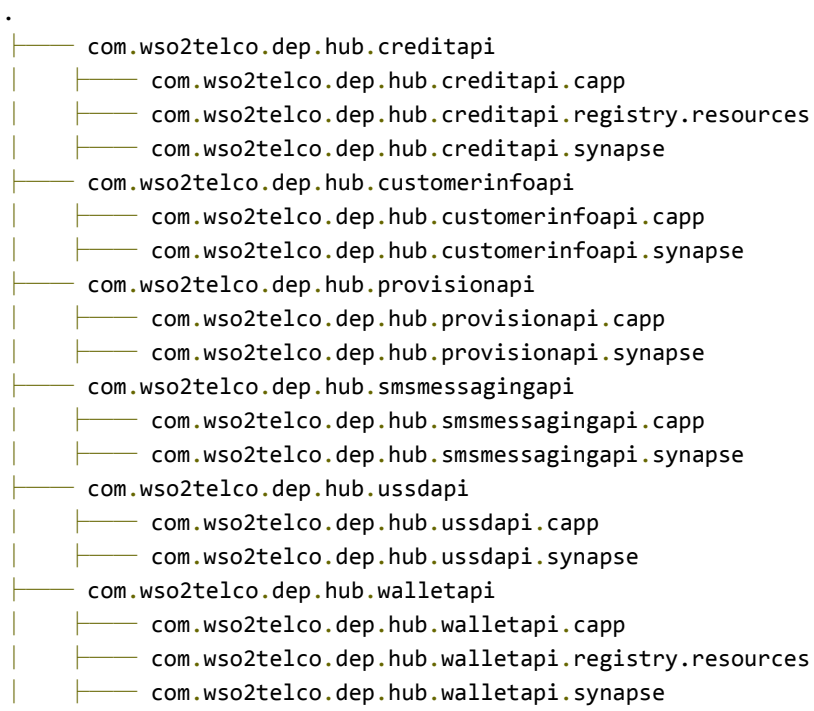

Figure 5.1: Directory structure of the hub API artefacts repository

Figure 5.1 shows the current API implementations' directory structure. Any module (registry or etc) which is API specific should go under this repository. To Deploy any of the APIs available in this repository, developer has to successfully deploy common capp first.
As you can see some of the api contains registry resources. At this moment it hold the API specific configuration file only.

```
<apiConfig>
        <notificationURL>https://localhost:8280/credit/v1/creditnotification</notificationURL>
</apiConfig>
```

Figure: 5.2 creditConfig.xml file.

6. Developing a Carbon Application (capp) in Dev Studio

Mediation layer of a particular API, to be developed as a CApp; developer should have a basic knowledge on developing ESB artifacts using developer studio.
Please find the detailed official documentation from the location [1]. Developer should master developing ESB artefacts using Developer Studio.
Developer is expected to have an understanding on how WSO2 ESB mediation flows [4] work. What are the common error handling best practices. Furthermore Developer should have a

basic knowledge on enterprise integration patterns and how those can be implemented using WSO2 ESB.[6]
Moreover Developer should go through the samples specified in [7] to get an understanding on the mediators and their capabilities. It's also better to have an understanding on templates [9] and endpoint capabilities [10] as well.
Also It's better to have an slight understanding on the threads created in ESB as well. You can find those information in [8]

Naming conventions

- Use fully qualified names for naming directories and artefacts.
    - com.wso2telco.dep.hub.creditapi - Directory structure
    - com.wso2telco.dep.common.main.request.Sequence - sequences
    - com.wso2telco.dep.common.channelValidation.Template - templates

Rule of thumb
Before developing a class mediator to implement the functionality needed please make sure, it cannot be developed using the available mediators. It is quite rare that your requirement cannot be met by the available mediators.

References

[1] https://docs.wso2.com/display/DVS380/Creating+ESB+Artifacts
[2] https://github.com/WSO2Telco/mediation-dep-hub/blob/master/README.md
[3] https://github.com/WSO2Telco/mediation-dep-common
[4] https://docs.wso2.com/display/ESB500/Mediating+Messages
[5] http://wso2.com/library/articles/2012/10/wso2-esb-examples-best-practises-error-handling-wso2-esb/
[6] https://docs.wso2.com/display/IntegrationPatterns/Enterprise+Integration+Patterns+with+WSO2+ESB
[7] https://docs.wso2.com/display/ESB500/Samples
[8] http://soatutorials.blogspot.com/2015/05/understanding-threads-created-in-wso2.html
[9] https://docs.wso2.com/display/ESB500/Working+with+Templates
[10] https://docs.wso2.com/display/ESB500/Working+with+Endpoints