# SDL Implementation Guideline

Page 1

SDL Implementation Guideline 1

What is SDL Technology 3

Business Motivations 3

Description 3

SDL capabilities 4

Client/Server Architecture 4

Messages formats between applications 5

Applications 5

Tables 5

Messages 6

Server functionnalities 8

Data persistence in server 8

Server scalability 8

Server resilience/redundancy 8

Client API 9

Client Get() function 9

Client Set() function 10

Client CacheStatus() function 10

Client CacheDeleteRow() function 10

Client CacheDeleteTopic() function 10

Client CacheClear() function 10

Test Bench & Performances 12

SDLActor.java test program 12

Client/Server latency performances 12

Cache latency performances 12

Resilience performances 12

Scalability performances 12

## What is SDL Technology

### Business Motivations

SDL API is a software library used to wrap real-time data communication pipeline between different IP connected distant actors. Using this SDL API is a time to market improvement to other pipeline libraries because it provides a unique interface in order to wrap many different pipelines implementations. In other words, with the SDL API you can save time and money because by writing less number of lines of code you can program on more pipelines, limit potential bugs, learn only one API instead of many, facilitate tweaking and support.

User Application (Actors)

SDL API

Kafka

Redis

ZeroMQ

### Description

Shared Data Layer (SDL) software rôle is to real time data pipeline, processing and storing 5G Cloud RAN data. SDL is a point of storage for all the core network data used by Virtualized Network Functions, NFV. Now there is a split between data storage and processing algorithms from other parts of the 4G/5G system. This means that NFV functions can become stateless and don’t longer need to manage their own data.

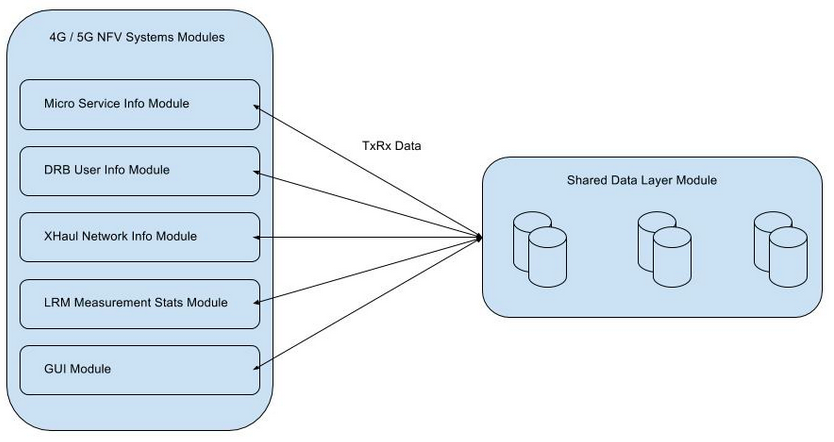


Figure 1 - SDL and 4G/5G VNF interactions

### SDL capabilities

* Data resiliency: Real-time data availability and reduced signaling
* Flexibility: Unified data privacy, unified approach to security, data zoning and data sharing
* SLA-based service: High availability of data, simple automated operation and capacity usage/prediction
* Multivendor interface: Common cloud storage, seamless Integration and reduced time to market
* Real-time, low latency: Ready for session data and subscriber data with proven, real-time geo-redundancy robustness
* VNF efficiency and scalability: Simplified VNF operations, simplified VNF scalability, faster time to market and reduced signaling

Those data will be stored on a NoSQL database that must be highly scalable on demand. There are many types of NoSQL databases. They are categorized with the ‘CAP theorem’ which says that a database is a tradeoff between those three properties:

* ‘C’: Consistency: Every read receives the most recent value or an error.
* ‘A’: Availability: Every request receives a response but not sure it is the last written value.
* ‘P’: Partition Tolerance: System continues to operate in case of network failures.

### Client/Server Architecture

SDL server runs on a server also called brocker that is scalable and performs transmissions of buffers between connected actor applications. A cache is used to store locally the tables received from the topics buffers.

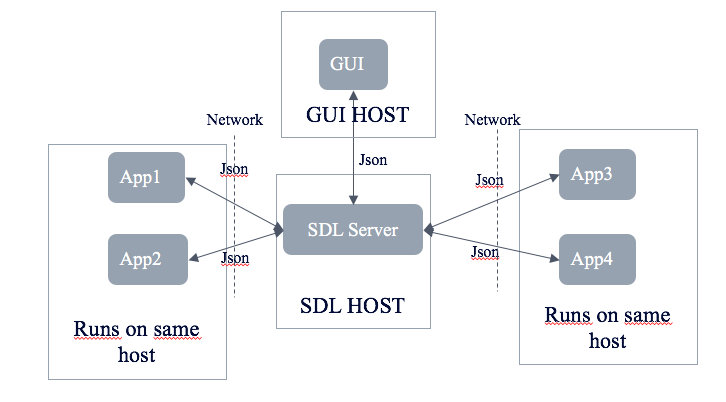


Figure 2 - SDL Client / Server

## Messages formats between applications

### Applications

Tables contain Information Elements (IEs) transmitted from and to separate applications as described in figure 1:

* + GUI
  + SWARM
  + LRM
  + BBSC

### Tables

* + UI\_NS\_VNF\_ORCHESTRATION
  + UI\_NS\_VNC\_SETTING
  + UI\_SS\_TRAFFIC\_PROFILE
  + UI\_CONTROL\_SCREEN\_SETTINGS
  + MICRO\_SERVICES\_INFO
  + LRM\_DB\_MEAS\_IND
  + BBSC\_DB\_REG\_FEU\_EC\_READY
  + BBSC\_DB\_DRB\_ALLOCATION\_AND\_SPLIT
  + XHAUL\_NETWORK\_INFO
  + FZM\_INFO
  + MBB\_TRAFFIC\_PROFILE\_INFO
  + IoT\_TRAFFIC\_PROFILE\_INFO
  + CELL\_VM\_INFO
  + RESOURCE\_INFO

### Messages

Format is JSON (RFC 7159) with a primary Key for each row of the table : The primaryKey is used as parameter in Set() and Get() functions to send and retreive a JSON Row through a SDL server. If a table contains many primaryKeys then the parameter is the concatenation of the keys with a ‘/’ caracter as delimitor. Exemple with the table CELL\_VM\_INFO : the primaryKey of a row is the string **"**vran-cell1/FZM1**".**

* + UI\_NS\_VNF\_ORCHESTRATION
    - **"primaryKey"**:"0"
    - {**"lastUpdateDate"**:1484817084,**"OpenStackEnabled"**:true,**"microServiceDeplEnabled"**:true,**"updatedBy"**:"GUI"}
  + UI\_NS\_VNC\_SETTING
    - **"primaryKey"**:"0"
    - {**"lastUpdateDate"**:1484817084,**"sliceRouting4G"**:"N2",**"sliceRouting5GMBB"**:"N2",**"sliceRouting5GIoT"**:"N2",**"allocatedBandwidth4G"**:10,**"allocatedBandwidth5GMBB"**:10,**"allocatedBandwidth5GIoT"**:10,**"microS1Bandwidth"**:10,**"microS2Bandwidth"**:10,**"updatedBy"**:"GUI"}
  + UI\_SS\_TRAFFIC\_PROFILE
    - **"primaryKey"**:"0"
    - **{"lastUpdateDate"**:1484817084**,"videoTrafficEnabled"**:true**,"gamingTrafficEnabled"**:true**,"MBBTrafficMicroS1Enabled"**:true,"**MBBTrafficMicroS2Enabled**":false,"**MBBTrafficProfileMicroS1**":"high","**MBBTrafficProfileMicroS2**":"high","**IoTTrafficEnabled**":true,"**IoTTrafficSensorCount**":16,"**IoTTrafficProfile**":"high","**MHNetworkDelay**":1000,"**gamingSplitEnabled**":true,"**videoSplitEnabled**":true,"**ECMigrationEnabled**":true,"**IoTSPBCount**":10, "**updatedBy**":"GUI"}
  + UI\_CONTROL\_SCREEN\_SETTINGS
    - **"primaryKey"**:"0"
    - {**"lastUpdateDate"**:1484817084,**"processingThresholdFEU"**:66,**"processingThresholdxUP1"**:66,**"processingThresholdxUP2"**:66,**"microS1BitRatioThresholdGBR"**:66,**"microS2BitRatioThresholdGBR"**:66,**"microS1BitRatioThresholdNGBR"**:66,**"microS2BitRatioThresholdNGBR"**:66,**"updatedBy"**:"GUI"}
  + MICRO\_SERVICES\_INFO
    - **"microServiceId"**:"a13e3c5def37342a"
    - {**"microServiceOrchProvider"**:"Swarm",**"microServiceName"**:"BBSC",**"sourcePortNo"**:51345,**"chainingPortRange"**:1024,**"sourceIpAddress"**:"172.1.0.15",**"status"**:"RUNNING",**"deployedServer"**:"FEU",**"startDate"**:1479829775}
  + LRM\_DB\_MEAS\_IND
    - **"upId"**:"a13e3c5def37342a"
    - {**"lastUpdateDate"**:1484817084,**"bbuId"**:1,**"nwSlice"**:1,**"deployedServer"**:"FEU",**"processingLoad"**:66,**"qoEngbrRatio"**:66,**"qoEgbrRatio"**:66,**"topXdataRadioBearers"**:4,**"mHBandwidthUsed"**:95000,**"migrationRequired"**:true,**"qoEgbrServedRate"**:95000,**"qoEngbrServedRate"**:95000}
  + BBSC\_DB\_REG\_FEU\_EC\_READY
    - **"upId"**:"a13e3c5def37342a"
    - {**"lastUpdateDate"**:1484817084,**"bbuId"**:1,**"ecFeuReady"**:true}
  + BBSC\_DB\_DRB\_ALLOCATION\_AND\_SPLIT
    - **"lcid"**:{**"drb"**:0,**"user"**:1} : **"**0/1**"**
    - {**"lastUpdateDate"**:1484817084,**"bbuIdEC"**:1,**"bbuIdFEU"**:1,**"upIdEC"**:1,**"upIdFEU"**:1,**"reqCPUOcc"**:66,**"qci"**:3,**"split"**:10}
  + XHAUL\_NETWORK\_INFO
    - **"primaryKey"**:"0"
    - {**"actualBandwidth4G"**:1.00,**"actualBandwidth5G"**:2.00,**"videoActualBandwidth5G"**:1.00,**"gamingActualBandwidth5G"**:1.00,**"actualBandwidthIoT"**:1.00,**"physicalTrunk1Bandwidth"**:4.00,**"physicalTrunk2Bandwidth"**:4.00,**"lastUpdateDate"**:1484817084,**"updatedBy"**:"MHNI"}
  + FZM\_INFO
    - **"fzmNo"**:"FZM1"
    - {**"connectionStatus"**:"Connected",**"portNo"**:1234,**"lastUpdateDate"**:1484817084,**"updatedBy"**:"MHNI",**"cellState"**:"OnAir",**"gpsState"**:"Up"}
  + MBB\_TRAFFIC\_PROFILE\_INFO
    - **"profileNameNumber"**:"Profile1"
    - {**"lastUpdateDate"**:1484817084,**"updatedBy"**:"MHNI",**"DRBsInL1GBR"**:20,**"DRBsInL1NGBR"**:20,**"DRBsInL2GBR"**:0,**"DRBsInL2NGBR"**:0}
  + IoT\_TRAFFIC\_PROFILE\_INFO
    - **"profileNameNumber"**:"Uniform"
    - {**"lastUpdateDate"**:1484817084,**"updatedBy"**:"MHNI"}
  + CELL\_VM\_INFO
    - **"primaryKey"**:"vran-cell1/FZM1"
    - {**"connectionStatus"**:"Connected",**"currentState"**:"OnAir",**"numConnectedUE"**:200,**"meanAppLatency"**:2.5,**"meanAggThroughput"**:57.0,**"poolUtilPercentage"**:30,**"lastUpdateDate"**:1484817084,**"updatedBy"**:"MHNI"}
  + RESOURCE\_INFO
    - **"primaryKey"**:"vran-cell1"
    - {**"r1ConsumptionInPercentage"**:40,**"r2ConsumptionInPercentage"**:40,**"r3ConsumptionInPercentage"**:40,**"r4ConsumptionInPercentage"**:40,**"r5ConsumptionInPercentage"**:40,**"r6ConsumptionInPercentage"**:40,**"r7ConsumptionInPercentage"**:40,**"r8ConsumptionInPercentage"**:40,**"r9ConsumptionInPercentage"**:40,**"lastUpdateDate"**:1484817084,**"updatedBy"**:"MHNI"}

## Server functionnalities

### Data persistence in server

SDL Server persistence is the duration of storage of Json Table Rows. Today persistence is set to 30 seconds. It means that the Json Tables Rows are stored in the SDL server for at least 30 seconds and less than 40 seconds. A cleaner process is executed every 10 second to check if it can remove Json Table Rows older than 30 seconds.

Server properties for persistence of data are :

delete.topic.enable=true

log.retention.ms=30000

log.retention.check.interval.ms=10000

### Server scalability

Server scalability is the ability to handle load variations by starting a new process in the same node or in new nodes to handle messages overload.

### Server resilience/redundancy

Server resilience is the ability to fallback to an other server node containing duplicated stored messages in case of failure of the first node.

## Client API

SDL Client Api is a wrapper to Kafka client libraries and SDL API should be used instead of using Kafka client libraries directly. It provides a Set() function to send a Json Table Row and a Get() function to retreive a Json Table Row.

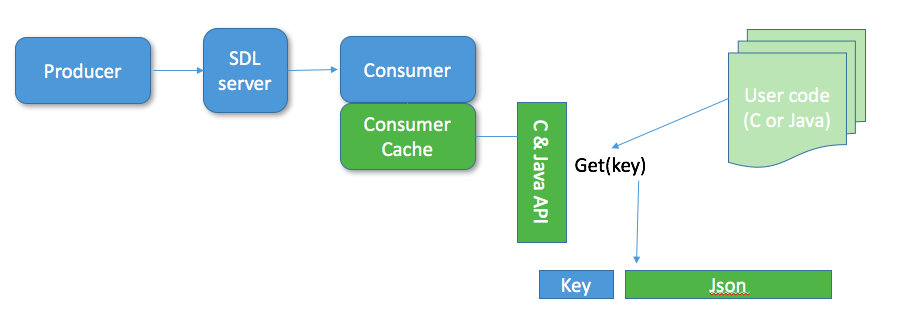


Figure 3 - Client Cache stores locally the received Json Tables Rows.

### Client Get() function

This function retreives a table row as a Json formatted string providing row primaryKey. Data is fetched from application cache.

Java usage is :

1/ import the SDL package :

import com.greenflops.sdl.\*;

2/ Create a APIConsumer java thread with server address and table name :

SDLConsumer SwarmConsumer = new SDLConsumer(address, "MICRO\_SERVICES\_INFO");

3/ Start the polling :

SwarmConsumer.start();

4/ Get the row when needed by its primaryKey (or "0" string if there is no primaryKey):

String microServiceId1 = "a00000000000001a";

String JsonTableRow = SwarmConsumer.get(microServiceId1);

5/ Finally exit the polling thread:

SwarmConsumer.finish();

### Client Set() function

This function sends a Json Row into the server pipelines (same principle of a kafka producer).

Java usage is :

1/ import the SDL package :

import com.greenflops.sdl.\*;

2/ Create a APIProducer java thread with server address and table name :

SDLProducer SwarmProducer = new SDLProducer(address, MICRO\_SERVICES\_INFO);

3/ Start the thread:

SwarmProducer.start();

4/ Send the primaryKey (or "0" string if there is no primaryKey) and the row:

String microServiceId1 = "a00000000000001a";

String JsonTableRow = "{\"status\":\"RUNNING\",\"startDate\":"+ System.currentTimeMillis() + "}";

SwarmProducer.set(microServiceId1, JsonTableRow);

5/ Finally exit the polling thread:

SwarmProducer.finish();

### Client CacheStatus() function

This function returns a status structure of the cache corresponding to number of topics, number of rows, size of used ram.

### Client CacheDeleteRow() function

Delete a cached Json Table Row.

### Client CacheDeleteTopic() function

Delete all rows of a cached Json Table

### Client CacheClear() function

Delete all Json Tables and Rows

## Test Bench & Performances

### SDLActor.java test program

SDLActor is a program written in Java to Set and Get Json tables through a SDL server. It creates a set loop and a get loop. It it used to check the set() -> server -> get() transmission chain.

### Client/Server latency performances

Mesure latency between Producer -> Server -> Consumer

### Cache latency performances

Mesurements of latency for the Get() function in cache

### Resilience performances

Measurement of recover/failover time in case of sever node fallback.

### Scalability performances

Measurements of latency when there is a lot of traffic in the channels.