# Documentation

# Application Pattern

**Version: 0.1**


| Author (name) | Date | Address and Contact Details | |
|---|---|---|---|
| Thorsten Heinze | | Telefonica Germany, Munich<br>+49 (0)89 2442 4852 | |
| Approved by (name) | Date | Role/Authority | Signature |
| | | | |

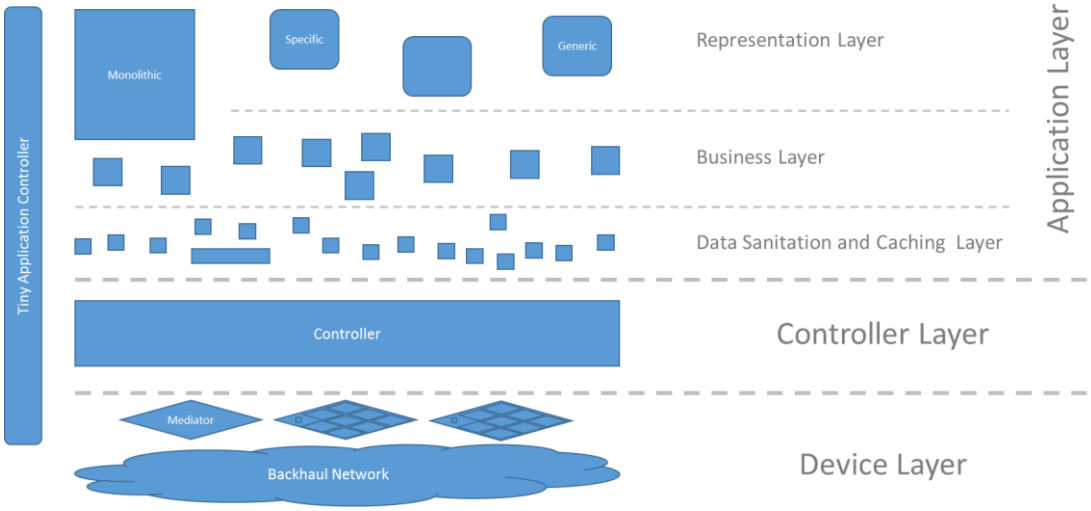
# 1 Introduction

Overall target is to implement an SDN, which facilitates vendor agnostic network automation.

The required application layer shall allow quickly activating innovations and be cost efficient.

The application layer shall be an amalgamation of a few multifunctional monolithic tools for covering a wide range of business tasks and many fine-grained REST Applications for complementing the monolithic tools.

The Application Pattern shall be applied on small to midsize Applications. It will be most relevant on the "Data Sanitation and Caching Layer" and significantly contribute to the "Business Layer".

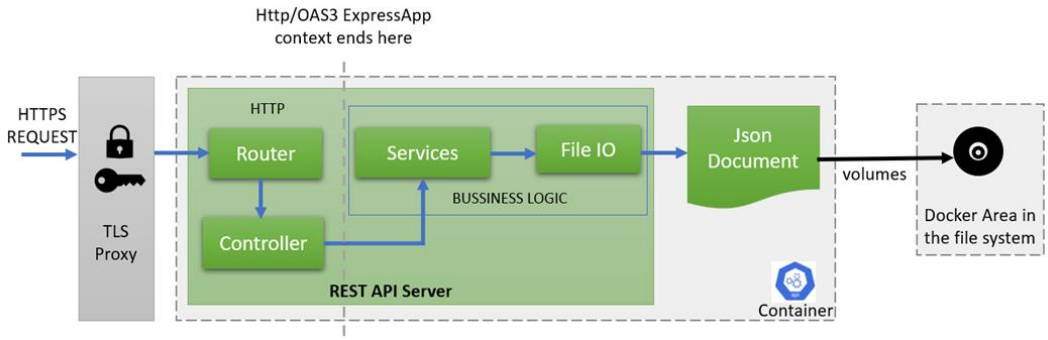



Every Application shall fulfil a single clearly and concisely defined purpose.

Most Applications will not provide an own graphical user interface, but all of them a REST interface for consuming the provided services and managing the Application itself.

The Application Pattern particularly supports implementing Applications in JavaScript code as REST servers, which are run in a node.js runtime environment on the server side.

The internal data of such Applications shall be stored into a JSON document ("Load file"), which is written into a folder that is created on the container engine, but mounted into the individual container of the Application.




The Application Pattern presumes that the encryption layer (if relevant at all) is terminated at some separate web server, which is hosted on the same container engine.

The Application Pattern[1] is composed from the following components, which have to be complemented for specifying an individual Application:

- The REST interface is specified in YAML according to the OAS 3.0.1 (Swagger) specification[2]. This syntactically clean and standardized way of specifying facilitates automated code generation.
- The business logic is implicitly described by providing a collection of test cases in JavaScript code, which are run in Postman[3]. The same test case collection shall be used for test driven development of the Application, acceptance testing and continuous integration.
- The Load file shall also be part of the Application specification. It describes the data structure on the file I/O interface and supports implementation and acceptance testing by putting the Application into a well-defined state.

# 2 REST Interface Structure

**Service and OaM Layers**

It is assumed that there will be several tens of Applications in midterm. They will quickly emerge, but also vanish again. Proper life cycle and connection management will be required to keep control.

A formal process for type approval of Applications shall be established and enforced by API key management.

As a consequence, the REST interfaces at the Applications are required not just for offering Services, but also for managing the Applications themselves. So the interfaces get sub-structured into a Service layer for offering operations to consumers and an OaM Layer for administrational purposes.

While the operations on the Service layer are mostly consumed by other Applications, the OaM layer is mostly used by humans (administrators). Even if many tasks on the OaM Layer will be automated on a later stage, too, they will be executed on behalf of a responsible human.

Consuming the combined Service layers of a conglomeration of existing Applications will be like choosing from a catalog. Short and meaning full path names describing the offered operations will be supportive. On the OaM layer, a harmonized and well-known structure (alike interfaces according to the OSI network layers) for identifying the recurring resources is preferred.

POST method is used for calling an operation on the Service layer, while GET and PUT methods are used for addressing the resources on the OaM layer.

This design makes the REST interface at the Applications very similar to the RESTCONF interface towards physical devices and facilitates applying the same management tools.

**Basic and Individual Components**

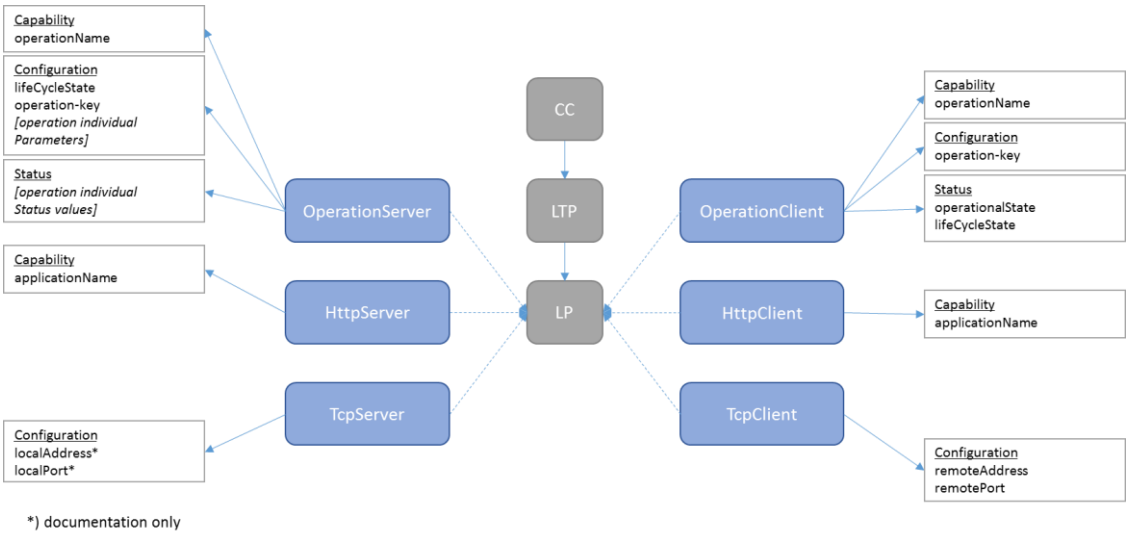
Every Application requires TCP and HTTP Server objects representing their own management interface.

Every Application offering some operation on the Service layer requires at least one Operation Server object, too.

Most Applications will require TCP, HTTP and Operation Client objects for consuming operations, which are provided by other Applications, or addressing the Controller north-bound interface.

As a consequence, the corresponding interface classes have been defined to be parts of the Basic OaM Layer.

[1] The Application Pattern definition relevant for MBH SDN at Telefonica Germany can be found at https://github.com/openBackhaul/ApplicationPattern/tree/tsi .
[2] Documentation can be found at https://swagger.io/docs/specification/basic-structure .
[3] Introduction can be found at https://learning.postman.com/docs/getting-started/introduction .




On the Service Layer, all Applications shall inform about themselves and their release history.

Because the structure of these information are also equal at all Applications, the corresponding operations are part of the Basic Service Layer.

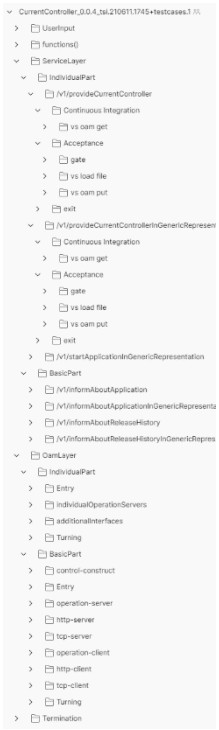
On the other hand, every Application is assumed to offer at least one individual operation, which will be part of the Individual Service Layer.

Likewise, the attributes, which are needed to describe and configure individual operations are augmented to the Basic Operation Server and will be parts of the Individual OaM Layer.

```
openapi: 3.0.0
info: ···

servers: ···

paths:
########################################
#   Service Layer - Individual Part
########################################
  /v1/provideCurrentController: ···
  /v1/provideCurrentControllerInGenericRepresentation: ···

########################################
#   Service Layer - Basic Part
########################################
  /v1/startApplicationInGenericRepresentation: ···

  /v1/informAboutApplication: ···
  /v1/informAboutApplicationInGenericRepresentation: ···
  /v1/informAboutReleaseHistory: ···
  /v1/informAboutReleaseHistoryInGenericRepresentation: ···

########################################
#   OAM Layer - Individual Part
########################################
  /core-model-1-4:control-construct/logical-termination-point=op-s-1000/layer-protocol=0/operation-server-interface-1-0:operation-server-interface-pac/operation-server-interface-configuration/response/current-controller: ···
  /core-model-1-4:control-construct/logical-termination-point=op-s-1001/layer-protocol=0/operation-server-interface-1-0:operation-server-interface-pac/operation-server-interface-configuration/response/response-value-list= ···

########################################
#   OAM Layer - Basic Part
########################################
  /core-model-1-4:control-construct: ···

  /core-model-1-4:control-construct/logical-termination-point={uuid}/layer-protocol=0/operation-server-interface-1-0:operation-server-interface-pac/operation-server-interface-capability/operation-name: ···
  /core-model-1-4:control-construct/logical-termination-point={uuid}/layer-protocol=0/operation-server-interface-1-0:operation-server-interface-pac/operation-server-interface-configuration/life-cycle-state: ···
  /core-model-1-4:control-construct/logical-termination-point={uuid}/layer-protocol=0/operation-server-interface-1-0:operation-server-interface-pac/operation-server-interface-configuration/operation-key: ···

  /core-model-1-4:control-construct/logical-termination-point={uuid}/layer-protocol=0/http-server-interface-1-0:http-server-interface-pac/http-server-interface-capability/application-name: ···

  /core-model-1-4:control-construct/logical-termination-point={uuid}/layer-protocol=0/tcp-server-interface-1-0:tcp-server-interface-pac/tcp-server-interface-configuration/local-address: ···
  /core-model-1-4:control-construct/logical-termination-point={uuid}/layer-protocol=0/tcp-server-interface-1-0:tcp-server-interface-pac/tcp-server-interface-configuration/local-port: ···

########################################
#   Common Components
########################################
components: ···
```

The Application Pattern shall facilitate covering the Basic components of the API specification and the test case collection by constantly re-using the same code for all Applications. Individualization shall exclusively be made by describing the data objects inside the Load file. This shall not be limited to the attribute's values, but also apply on the number of the objects of the respective type.

Applying the Application Pattern shall limit the efforts for specifying a new Application to describing the API paths and test cases of the Individual components on the Service and OaM Layers and all data objects, which are required to define the initial state of the Application.




# 3 Test Case Structure

Alike the API specification, the request collection is structured into Service and OaM Layers and Basic and Individual components.

In addition, test cases for Continuous Integration are separated from the ones to be used for application development and Acceptance testing.

Continuous Integration testing is assumed to take place on an operational network. So, there is no writing included. Values of the attributes of the OaM Layer get compared with the responses of the service calls and responses of the service calls get compared with generic patterns (if included in the API definitions).

During implementation and Acceptance testing, response values get compared against the content of the Load file and dummy values get actively configured for verification of the implementation of the business logic.

It is recommended to deeply analyze one or several existing test case collections and to structure and program as similar as possible to foster future maintainability of the code.

# 4 How to Prepare

- Download Postman (https://www.postman.com/downloads)
- Install Postman and get used to it (https://learning.postman.com/docs/getting-started/introduction)
- Create Workspace for new Application
- Create new API
- Delete example from the API editor
- Download Application Pattern (https://github.com/openBackhaul/ApplicationPattern/tree/tsi)
- Open .yaml of the Application Pattern
- Copy content of the .yaml into the API editor and save
- Import +testcases.json to Collections

# 5 How to Specify

- Define the Individual components of the Service Layer into the API editor (https://swagger.io/docs/specification/basic-structure)
- Complement the operation servers by the Individual components in the API editor
- Define the wished values of the attributes of the Basic components into the Load file
- Complement the Load file by the instances of the individual components
- Copy the Load file content into the body of the UserInput request of the test case collection
- Generate a mock server from the API definition
- Copy the URI of the mock server into the Data file
- Generate an authorization key by sending a request to the mock server and reading from the Code
- Copy the authorization key into the Data file
- Generate a Postman API key
- Copy the Postman API key into the Data file

- Copy the load file content into the Data file
- Complement the test cases collection

**- end of document -**