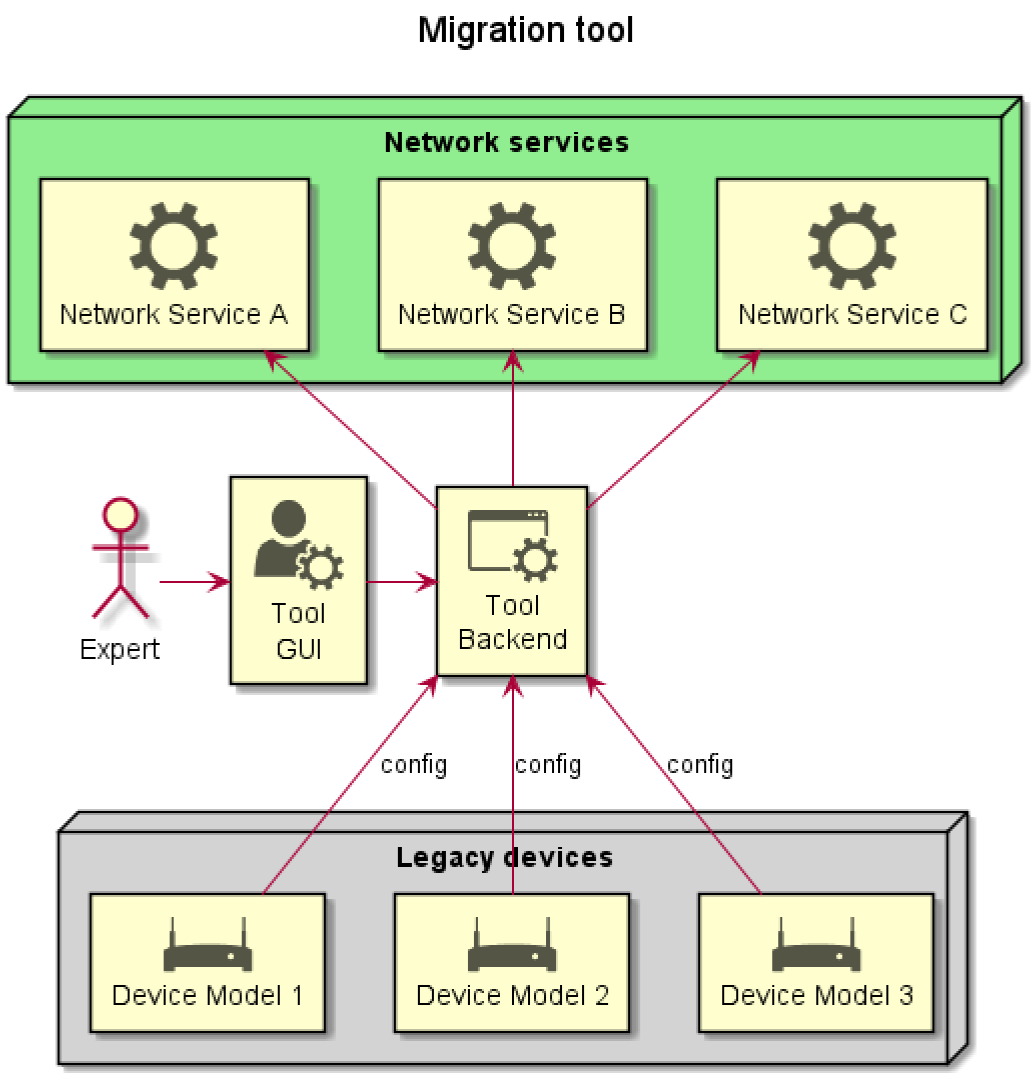
Migration Tool

Overview

The tool is responsible to help our Customers migrating their existing network services to a new product which is based on a new technology. The Customers have extensive netwoks with large complexity and feature variations. The tool should reduce manual procedures as much as possible by automating the legacy network analysis, decision making in the selection of target solution and finally execution of the migration itself.



Layers

1. Legacy network services

- The current Customer network is based on Cisco devices (Cisco IOS)
- There are a few dozends model / firmware combinations in the field
- Up-to few hundred devices per Customer networks
- Majority of the devices follow a standard and comparable configuration (80%)
- Minority of the devices are heavily customized with many active features (20%)

2. Migration Tool

Two main interactions are expected with the tool:

- Expert personal - migration preparation phase
  - The expert will, with the help of the tool, analyze the existing Customer network and decide on the target solution.
    - multiple 'runs' are expected over a configurable amount of devices or selected ones
    - the tool will be used in interactive way (for example, mapping rules are adapted depending on the results obtained so far)
    - this phase is less time critical
- External system - migration execution phase
  - During the live migration, the migration tool will be responsible to provide the input data for the fulfillment systems executing the migration
    - this process is time-critical

3. Network Services

- This layer is describing the new product, it's features and services which are available for the Customer.
- The tool must be able to decide based on the result of the analysis which one if best suitable

Interfaces

---

1. Legacy devices <--> Migration Tool back-end -> the up-to-date configuration of each device can be retrieved from a central storage (assume HTTP)

2. Migration Tool back-end <--> Tool GUI -> the back-end will expose various APIs to trigger actions and provide data back for representation purposes

3. Expert <--> Migration Tool (GUI or back-end) -> Either a person is interacting with the tool or an external system which is fully automated
   - the back-end should expose APIs meant to be consumed by other automated systems

4. Migration Tool <--> Network Services -> API based, the tool is to be consumed

Exercise

Develop a Java application that simulates the role of the Tool Backend from the diagram above. The application must be able to read configuration files from more devices of different models, then, produce the list of network services that each device is running.

Please consider the following non-functional and usability needs when designing the application:

- the number of devices will be of order of hundreds, and the number of models a few dozens
- mapping rules can be changed by the expert user and applied at the next tool run
- the tool is going to run in a cloud environment, so it needs to be designed with the "12 factors" in mind

The application developed for this exercise must be able to read the files from `input` folder and generate the ones in the `output` using files syntax and mapping rules described below.

Device configuration files

The device configuration files depend on the device model, so below are shown 2 examples to be used for this exercise:

- Device model 1

```
{
   "deviceId": "id11",
   "param1": "value11",
   "param2": "value12"
}
```

- Device model 2

```
uuid=uuid21
param1=value21
param2=value22
param3=value23
```

Network services

The description of network services available on each device must be structured in the following way:

- Network service A

```
  network_service_type: "A"
  device_id: "identifier"
  parameters:
     paramA1: "value"
     paramA2: "value"
```

- Network service B

```
  network_service_type: "B"
  device: "identifier"
  configuration:
     configB1: "value"
     configB2: "value"
```

- Network service C

```
network_service_type: "C"
dev: "identifier"
conf:
   configC1: "value"
   configC2: "value"
```

Mapping rules

- Mapping for device model 1
  - if `param1` exists, then create **network service A** with `device_id: id11` and `paramA1: value11`
  - if `param2` exists, then create **network service B** with `device: id11` and `configB1: value11` and `configB2: value12`

- Mapping for device model 2
  - if `param1` exists and `param2` does not exist, then create **network service A** with `device_id: uuid21` and `paramA1: value21` and `paramA2: "0"`
  - if `param2` exists, then create **network service B** with `device: uuid21` and `configB1: value21` and `configB2: value22`
  - if `param1` exists and `param2` does not exist and `param3` exists, then create **network service C** with `dev: uuid21` and `configC1: value21` and `configC2: value23`