Falconieri: Remote Provisioning Service as a Service

A new, modern, open source and cloud native remote provisioning service gateway.

nethesis

Matteo Valentini

Intro: Remote Provisioning Service Theory

What is it a Remote Provisioning Service?

The scope of Remote Provisioning Service is to solve the problem of the first time phone configuration.

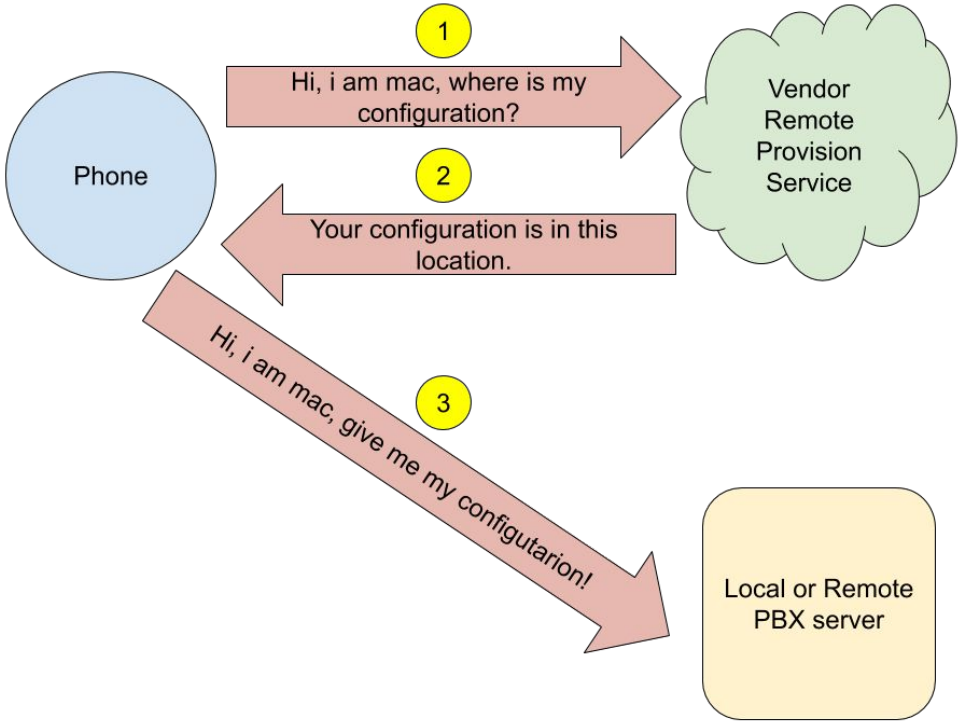
A VoIP phone before his first configuration

What is it a Remote Provisioning Service?

Without a RPS the phone must rely on local mechanism for initial provisioning, like:

- DHCP Option 66: for be effective you must have access to the DHCP server
- UPnP: can be tricky to manage the IP multicast routing

What is it a Remote Provisioning Service?

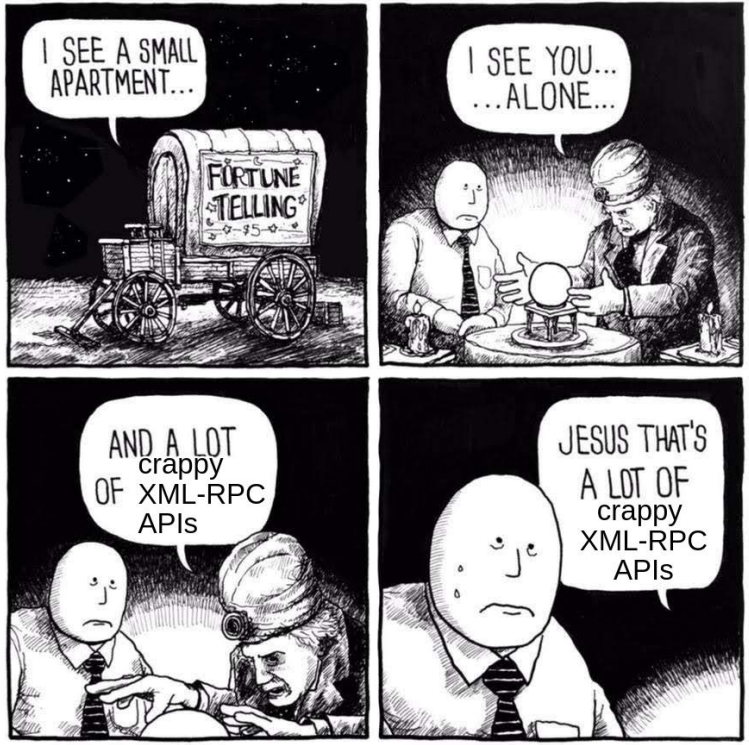


What can do a Remote Provisioning Service?

- Assign a configuration to a device even before is out of the box
- Massive configuration of multiple device via APIs

Why building a RPS gateway?

Vendors implementations



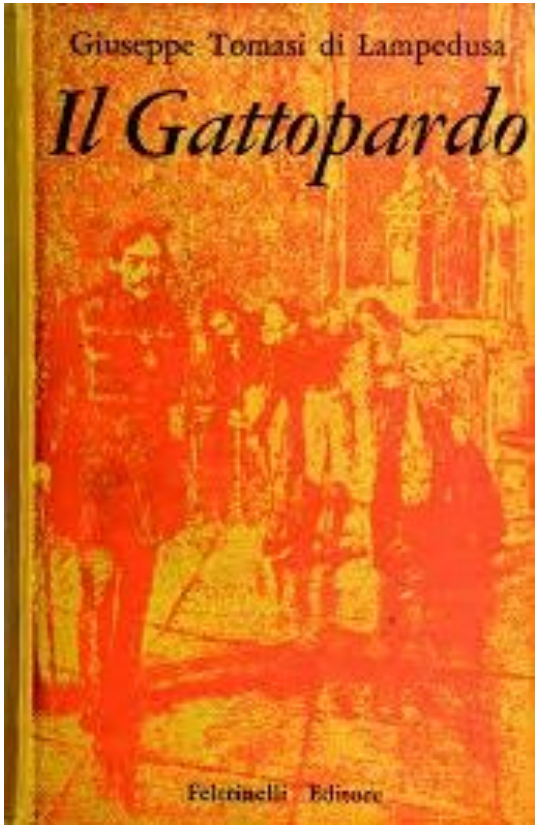
Vendors implementations

- Not standard set of features between vendors
- Different APIs each vendors
- XML-RPC

_Amygos
Matteo Valentini

The Leopard project

The scope of the project is refactoring the phone provisioning component of NethVoice, the Nethesis PBX solution.



_Amygos

Matteo Valentini

“If we want things to stay as they are, things will have to change.”

The Leopard project goals

- Use most modern technologies
- Introduction of new provisioning mechanisms (like RPS)
- Support of a well defined set of selected phone vendors: *SNOM*, *Gigaset*, *Yealink*, *Fanvil*
- Release most of the project's components as Open Source projects

_Amygos
Matteo Valentini

Tancredi Falconieri

Tancredi: phone provisioning engine ideal for internet deployments.

Falconieri: remote provisioning gateway.

The role of Falconieri

The role of Falconieri is to:

- Provide a unified HTTP rest interface to the vendors RPS service
- Store the credentials for access to the vendors RPS services

The vendors APIs

The semantic

For every vendor we want create an API that:

- Given a specific mac address, ***create a new configuration*** for that mac address if the mac address is not already configured
- Given a specific mac address, ***override the previous configuration*** for that mac address if the mac address was already configured

The ideal API endpoint

`PUT /providers/:provider/:mac`

Path variables

- ***provider***: Name of the remote provider.
- ***mac***: Mac address of the device

Body

A JSON object with the url of the configuration server.

Vendors interface objects

Every vendor have implemented this objects in some way:

- **MAC Address**: MAC address of the phone
- **Redirect Url**: the url where the actual phone configuration is stored
- **Group**: A list of phone with the same configuration

The vendors APIs

The Good, the Bad and the Ugly

(Fanvil, Gigaset, SNOM, Yealink)

_Amygos

Matteo Valentini

The Good: SNOM

- Good documentation
  - https://service.snom.com/display/wiki/XML-RPC+API
- Simple APIs
  - 7 APIs
- HTTPS endpoint

The Good: SNOM

Api calls for implementing Falconieri semantic:

1. `redirect.registerPhone(mac, provisioningUrl)`

The Bad: Gigaset

- Public documentation
  - https://teamwork.gigaset.com/gigawiki/display/GPPPO/Gigaset+Redirect+server
  - Better documentation in the service portal (after obtained a user/password from Gigaset)
- Simple APIs
  - 7 APIs
- HTTPS endpoint

The Bad: Gigaset

Why the Bad?

- Require a CRC code within the mac
- The CRC code in printed in the phone label (with no public formula for calcualtion)
- The mandatory CRC code make almost impossible an automated device discovery and configuration.

But maybe you can have the CRC code disable for your account if you ask.

The Bad: Gigaset

Api calls for implementing Falconieri semantic:

1. `autoprov.deregisterDevice(macID)`
   - **`macID`**: ”<MAC address> - <CRC code>”
   - We don’t care about success or not!
2. `autoprov.registerDevice(macID, provisioningUrl, Provider)`
   - **`Provider`**: in this case can be anything

The Ugly

- Yealink
- Fanvil

The Ugly: Yealink

- Public documentation
  - http://support.yealink.com/documentFront/forwardToDocumentDetailPage?documentId=257
- Too many APIs
  - 16 APIs
- HTTPS endpoint

The Ugly: Yealink

Why in the ugly?

- The APIs are overloaded and redundant.
- Very bad API design

The Ugly: Yealink

Api calls for implementing Falconieri semantic:

1. `redirect.registerDeviceWithUniqueUrl(mac, serverName, provisioningUrl, isOverride)`
   - **`serverName`**: in this case can be anything, `provisioningUrl` take the precedence
   - **`isOverride:`** if 1 override the previous configuration

The Ugly: Fanvil

Fanvil:

- No public documentation!
- Too many APIs!
  - 19 APIs!
- HTTP endpoint...

The Ugly: Fanvil

Why the Ugly?

- No HTTPS, require a double hash of the password for the authentication (`md5(md5(password))`)!
- Too many steps to implement the simple Falconieri semantic.

The Ugly: Fanvil

1. `redirect.addServer(serverName, provisioningUrl)`
   - The **`serverName`** and **`provisioningUrl`** actually are the same
   - Don’t care if the Server already exist
2. `redirect.deRegisterDevice(mac)`
   - Don’t care about the success.
3. `redirect.registerDevice(mac, serverName)`

The Ugly: Fanvil / take 2

1. `redirect.addServer(serverName, provisioningUrl)`
   - ~~The **`serverName`** and **`provisioningUrl`** actually are the same~~
   - The **`serverName`** is the MAC Address
   - Don't care if the Server already exist
2. `redirect.deRegisterDevice(mac)`
   - Don't care about the success.
3. `redirect.registerDevice(mac, serverName)`

The redirect URL exceeds the maximum length of **`ServerName`**

The Ugly: Fanvil / take 3

1. `redirect.deleteServer(mac)`
   - Don't care about the success.
2. `redirect.addServer(serverName, provisioningUrl)`
   - ~~The **`serverName`** and **`provisioningUr`** actually are the same~~
   - The **`serverName`** is the MAC Address
   - ~~Don't care if the Server already exist~~
3. `redirect.deRegisterDevice(mac)`
   - Don't care about the success.
4. `redirect.registerDevice(mac, serverName)`

A server can't be overwritten, so we have to delete it first.

The Ugly: Fanvil / take 4

1. `redirect.deleteServer(mac)`
   - Don't care about the success.
2. ~~`redirect.addServer(serverName, provisioningUrl)`~~
   - ~~The **`serverName`** and **`provisioningUr`** actually are the same~~
   - ~~The **`serverName`** is the MAC Address~~
   - ~~Don't care if the Server already exist~~
3. `redirect.deRegisterDevice(mac)`
   - Don't care about the success.
4. `redirect.registerDevice(mac, serverName)`

We can use `redirect.addServer()`... 😭

The Ugly: Fanvil / take 4

`redirect.addServer()` don't let you to configure some aspect of the groups, like:

- Priority of the configurations source, Fanvi defaults:
  1. DHCP
  2. UPnP
  3. RPS
- Force the phone to apply the configuration after the reboot

We have to use `redirect.addMaterialServer()...`

The Ugly: Fanvil / take 4

3.10 redirect. addMaterialServer

You can add a global configuration server which is used to register devices to

3.10.1 XMLRPC signature

redirect.addMaterialServer(array)

3.10.2 Parameter:

array: ['cfgName=Test', 'cfgApUsername = ', 'cfgApPassword = ', 'cfgApKey = ',
'cfgApKeyGen=', 'cfgApSave=0', 'cfgDhcpOpt=66',
'cfgPnpEnable =0', 'cfgPnpSrv =224.0.1.75', 'cfgPnpPort=5060', 'cfgPnpProt=0',
'cfgPnpInterval =1', 'cfgPfSrv=', 'cfgPfName=',
'cfgPfProt=1', 'cfgPfInterval=1', "cfgPfMode=0", 'cfgTrEnable=0', 'cfgTrType=1',
'cfgTrAcsSrv=0.0.0.0', 'cfgTrAcsUser=admin', 'cfgTrAcsPw=admin','cfgTrTlsVersion=0',
'cfgTrAutoLogin=0', 'cfgTrPeriod=3600', 'cfgStunEnable=0', 'cfgStunSrvAddr=0.0.0.0',
'cfgStunSrvPort=3478', 'cfgStunLocPort=30000']

Note: all configurations are not nessary except for cfgName.

The Ugly: Fanvil / take 4

1. `redirect.deleteServer(mac)`
   - Don't care about the success.
2. `redirect.addMaterialServer([Array])`
   - `Array:`
     - `cfgName=mac`
     - `cfgPfMode=1` (Apply the configuration after reboot)
     - `cfgDhcpOpt=false` (Disable the DHCP provisioning)
     - `cfgPnpEnable=false` (Disable the PnP provisioning)
     - `cfgPfProt=[1,2,4,5],cfgPfSrv=domain,cfgPfName=config_path` (The redirect URL)
3. `redirect.deRegisterDevice(mac)`
   - Don't care about the success.
4. `redirect.registerDevice(mac, serverName)`

nethesis

Falconieri

Falconieri characteristics

- Open source (AGPL v3)
- Single GoLang binary
- Easily deployment with provided ansible role.
- Created with "*12 factor app*" in mind
- Stateless
- Easily vertically and horizontally scalable

APIs

PUT /providers/:provider/:mac

Path variables

- ***provider***: Name of the remote provider.
- ***mac***: Mac address of the device, represented in the EUI-48 IEEE RA

Query parameters

- ***crc***: mac address CRC code, only valid with Gigaset provider.

Body

A JSON object with the url field:

- ***url***: URL of configuration server.

Usage

```
Usage of ./falconieri:

  -c string

  Path to configuration file (default "/opt/falconieri/conf.json")
```

Configurations

Falconieri can be configured in two way:

- JSON file
- Environment Variables

The configuration passed via environment variables **take the precedence**.

Falconieri JSON configuration

```
{
  "providers": {
    "snom": {
      "user":"user",
      "password": "password",
      "rpc_url":
"https://secure-provisioning.snom.com:8083/xmlrpc/",
       "disable": false
    }
}
```

Falconieri TODOs

- Client authentication
- Configuration of a list of devices
- More deployment strategy: RPM, DEB, Docker, HELM ecc..
- Deletion APIs?

Every Pull Request, enhancement, critique are very welcome!

https://github.com/nethesis/falconieri

Tancredi

An IP phone provisioning engine

- **Falconieri** help the phone to find the configuration
- **Tancredi** build the actual phone configuration

https://nethesis.github.io/tancredi/

Some statistics

Provides availability

Statistics over the last 90 days (updated on 06/08/2020)

Yealink	~99.7%
SNOM	~99.6%
Fanvil	~99.3%
Gigaset	~95.0%

Obtained by configure a MAC address every 2 minutes and checks the operation result.

Falconieri Nethesis' installation

The Leopard project was released as **Private Beta** on **26/03/2020** and released as **General availability** on **06/07/2020**.

	Unique Registered Phones
Private Beta	537
General Availability	977

Updated on 06/08/2020

Thanks for listening!

Questions?

Matteo Valentini

Developer at Nethesis

 Amygos

 _Amygos

 Matteo Valentini

 matteo.valentini@nethesis.it