

Function Specification Resource Configuration

Ericsson Dynamic Activation 1

FUNCTION SPECIFICATION

Copyright

© Ericsson AB 2017. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

Trademark List

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



Contents

1	Introduction	1
1.1	Purpose and Scope	1
1.2	Target Group	1
1.3	Typographic Conventions	1
2	Overview	2
2.1	Architecture Overview	2
2.1.1	Service Realization	3
2.1.2	Network Abstraction	3
2.2	Key Features	4
2.2.1	Simplified Service Managed by Service Visualization	4
2.2.2	Network Abstraction Engine Provides Vendor Agnostic Solution	4
2.2.3	Simplified Integration by Leveraging on A Model Driven Solution	5
2.2.4	Automatic Generation of Southbound Commands	5
2.2.5	Fault Tolerant Solution with Automatic Rollback	5
2.2.6	Efficient Device Configuration Management	6
2.2.7	Data Synchronization to Remove Data Inconsistencies	6
3	Network Abstraction	6
3.1	Network Abstraction Engine	8
3.1.1	Support for Parallel Commands toward Southbound	10
3.1.2	Rollback Logic	10
3.2	Model Catalog	11
3.2.1	Manage Feature Model	11
3.2.2	Generation of Interface Models	12
3.3	Template Management	13
3.3.1	XML Template	15
3.3.2	YANG Templates	17
3.4	Southbound Interfaces	20
3.4.1	CLI/SSH	20
3.4.2	CLI/Telnet	20
3.4.3	NETCONF	20
3.4.4	HTTP(S)	21
3.4.5	SNMP	22
3.4.6	SDN Adapters	22
3.4.7	Additional Southbound Protocols	22
4	Device Management	23
4.1	Device Repository	23



4.1.1	General Information	23
4.1.2	Access Points	24
4.2	Managing Devices	25
4.2.1	Import of Devices	25
4.2.2	Device Filtering	26
4.3	Management of Device Configuration	26
4.3.1	Revert to Previous Device Configuration	27
4.3.2	Candidate Store	27
4.3.3	Loose Error Handling	28
4.4	Synchronization of Data Models	29
4.4.1	Configuration Validation	29
4.4.2	Discovery of Device Configuration	29
4.4.3	Comparison of Device Configurations	30
4.4.4	Reconcile Data Repository Inconsistencies	30
4.4.5	Synchronization of Replaced Devices	31
5	Service Realization	32
5.1	Service Visualization	32
5.1.1	Filtering of Services	33
5.1.2	Service data in Device Repository	33
5.2	Service Modelling	33
5.2.1	Service Models	34
5.2.2	Tasks	34
5.2.3	Assign Parameters	35
5.2.4	Dependencies	35
5.2.5	Rollbacks	36
	Reference List	37



1 Introduction

This section is an introduction to the document. It contains information about the prerequisites, purpose, scope, and target group for the document. This section also contains explanations of typographic conventions used in this document.

1.1 Purpose and Scope

The purpose of this document is to give detailed description about the Resource Configuration functions in Ericsson™ Dynamic Activation (EDA) and to provide an entry of related documentation. Descriptions of functional behavior for the other features hosted by Dynamic Activation can be found in the respective Function Specification.

For information about Dynamic Activation, refer to *Function Specification Dynamic Activation Execution Environment*, Reference [1].

1.2 Target Group

The target groups for this document are the following:

- Network Administrator
- System Administrator
- Application Administrator
- Network Supervision Administrator
- Application Designer
- Other

For information about the different target groups, see *Library Overview*, Reference [2]

1.3 Typographic Conventions

Typographic conventions are described in *Library Overview*, Reference [2].

2 Overview

Resource Configuration provides an activation solution for Software Defined Networking (SDN), Network Functions Virtualization (NFV) as well as physical devices by providing one common and uniform activation interface. With the introduction of a model driven activation solution it provides a management interface that is independent of network implementation aspects like device types, equipment vendors and type of device (physical, virtual or SDN).

- Service models are responsible for:
 - Service specific business logic
 - Mapping to network features
 - Rollback logic

For example, the service model is used to set up an L3VPN solution.

- Network Abstraction and Implementation provide the following:
 - Model-driven architecture
 - Exposes logical network features. For example, features such as Interface, Multiprotocol Label Switching (MPLS), Border Gateway Protocol (BGP), or Virtual Routing and Forwarding (VRF).
 - Hides network implementation. For example, converts feature request into vendor-specific southbound commands (Cisco, Juniper).
 - Automatically generates southbound commands to Customer Edge (CE) and Provider Edge (PE) routers.

2.1 Architecture Overview

Figure 1 outlines a high-level architecture view of the Resource Configuration solution.

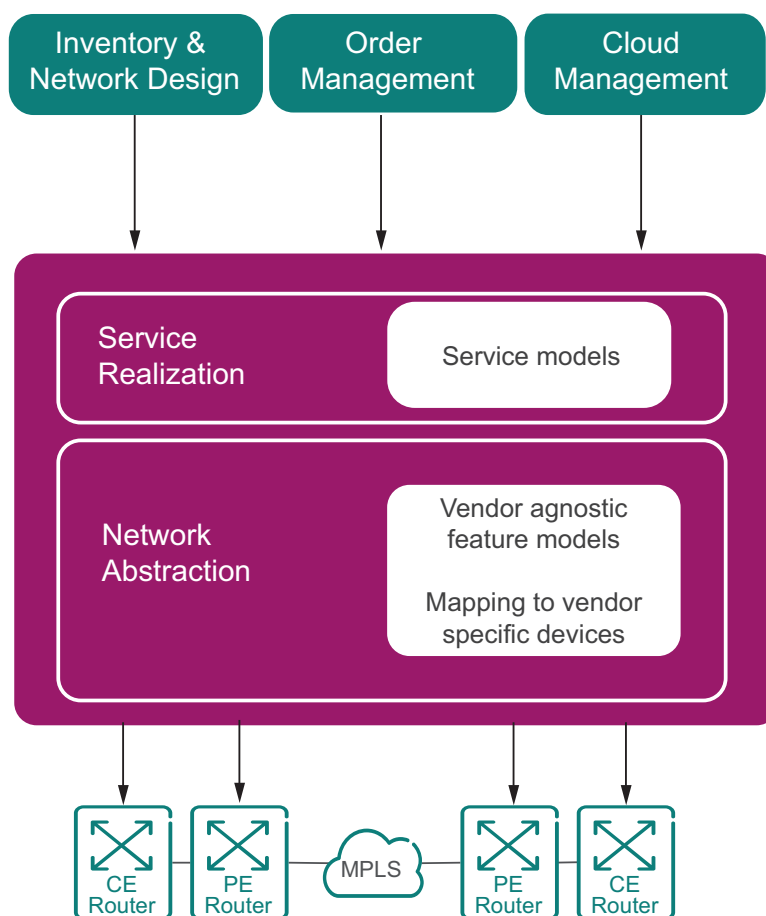


Figure 1 Architecture Overview

2.1.1 Service Realization

The Service Realization layer is based on service models to set up a service, for example an L3VPN service or an FTTH service. It provides the following functionality:

- Service models that contains the service specific business logic with mapping to different features in the network.
- An execution engine that provides rollback support on service level.
- A GUI where it is possible to visualize and manage the deployed service instances.

2.1.2 Network Abstraction

The Network Abstraction layer provides a uniform way to manage features in the devices in a vendor agnostic way. It provides following functionality:



- Model-driven architecture exposing logical network features, for example Multiprotocol Label Switching (MPLS) and Border Gateway Protocol (BGP).
- Vendor templates that hide network implementation and generate vendor-specific southbound commands towards devices, for example Cisco and Juniper routers.
- An execution engine that provides rollback support on feature level.
- A GUI where it is possible to visualize and manage the different devices and their configuration.

2.2 Key Features

This chapter summarizes the key features of Resource Configuration in the Dynamic Activation.

2.2.1 Simplified Service Managed by Service Visualization

The **Service Visualization** GUI shows how a service instance is decomposed into several features instances on a number of devices. It also makes it possible to manage and restore service configurations.

By doing side-by-side comparisons between service configurations at different points in time, it is easy to see how the different service objects has been added, removed or modified over time. This is a very valuable tool when trouble-shooting malfunctioning services.

2.2.2 Network Abstraction Engine Provides Vendor Agnostic Solution

The Resource Configuration solution hides the complexity of the underlying network by abstracting the network functionality into vendor independent and normalized feature models.

The feature model is the abstraction layer that eliminates the need for service designers to understand how different vendors implement specific features in their devices. By providing a generic network abstraction it enables rapid configuration of new services.

The impact on upstream systems is minimized since the network abstraction engine hides network implementation details like device vendors and versions, and instead shows a generic view of the provided network functionality. The generic network features exposed over the northbound interface can be interfaced directly or used as building blocks when creating service logic for different types of services.



2.2.3 Simplified Integration by Leveraging on A Model Driven Solution

The model driven solution in Resource Configuration gives the operators the freedom and independence to define their own provisioning solution via configuration, as well as benefit from Resilience Activation function.

The solution makes it possible for the operators to integrate new features in the network by adding new feature models to the system, and then add vendor templates that define how to set those features up in network devices. All those integration tasks are possible to do on a runtime Dynamic Activation system without writing a single line of code, Instead, the integration is done by simple configuration using the GUI.

Resource Configuration puts the right tools in the hands of the domain expert. By removing the need for handovers between domain experts and programmers, lead time and potential errors caused by the handovers are reduced.

The solution supports both XML schemas and YANG models as a way to import and expose feature models in the solution. This flexibility increases Dynamic Activation's capability to handle the downstream communications dramatically.

The solution provides a model catalog with a number of pre-defined commonly-used-features that are usable as-is by the northbound system, or possible to use as components in service models, which further raises the abstraction layer exposed towards the northbound systems.

2.2.4 Automatic Generation of Southbound Commands

The Resource Configuration solution automatically generates southbound commands based on the feature models and vendor templates in the system. It can generate southbound commands based on the following protocols:

- CLI/SSH
- CLI/Telnet
- NETCONF/SSH
- HTTP(S)
- SNMP

2.2.5 Fault Tolerant Solution with Automatic Rollback

The Resource Configuration solution has automatic rollback support. If any of the southbound commands for a feature request fails, an automatic rollback of all commands related to this request is started. This will leave the device in the same state as it was when the feature request was sent.



The automatic rollback support also covers the service model. If any of the feature requests fails, the service model can trigger a full rollback of all related feature requests. This will leave the network in the same state it was when the service request was sent.

2.2.6 Efficient Device Configuration Management

The Resource Configuration solution provides a repository that keeps track of the devices as well as their configurations. It also stores historical information about the previous configurations.

A GUI is provided where it is possible to view and edit the device configuration. It is also possible to revert back to any previous state of the configuration via this GUI.

The GUI makes it possible to do a side-by-side comparison between device configurations at any two points in time, which makes it easy to see how the different features has been added, removed or modified over time for a specific device. This is a very valuable tool when trouble-shooting malfunctioning devices.

2.2.7 Data Synchronization to Remove Data Inconsistencies

To reduce the risk of inconsistencies between the device repository and the actual device configuration, several functions are provided to find and resolve this type of inconsistencies.

- Automatic check of configuration before feature updates
- Automatic reconciliation of data in the device or the device repository
- Possibility to trigger full device configuration discovery and reconciliation
- Support for getting initial device configuration state

3 Network Abstraction

The Network Abstraction part of Resource Configuration is based on a model driven solution for configuration of devices through their native interfaces (CLI, NETCONF, HTTP, SNMP). It makes it possible to integrate network elements during runtime, without the need of programming. The figure below shows the main component of the Network Abstraction solution.

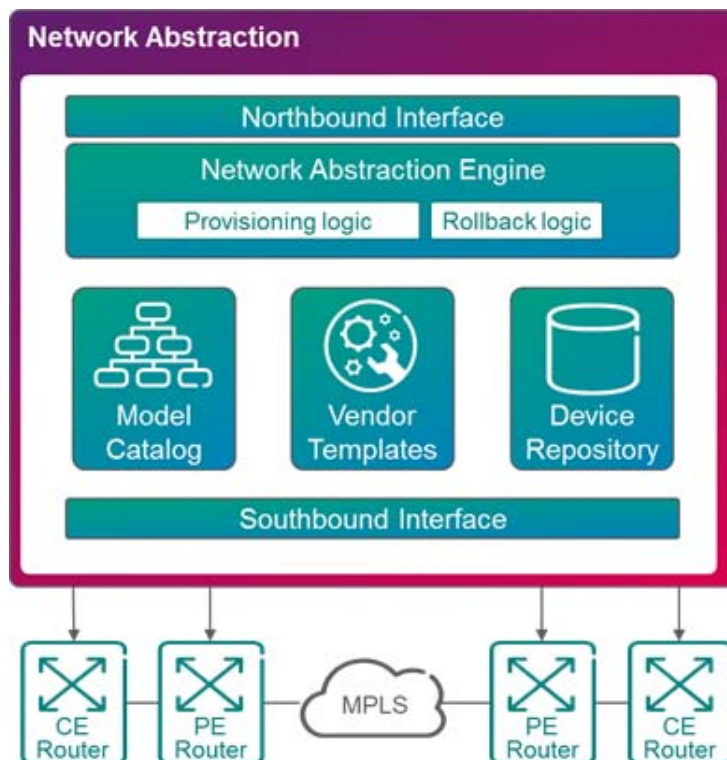


Figure 2 Network Abstraction Components

The solution is based on the following main components:

- **Model Catalog with Feature Models**

A feature model is a logical representation of network functionality, for example, a BGP configuration in a router. The feature is exposed as a Managed Object (MO) in the northbound CAI3G interface. It provides a uniform way of managing this feature, regardless of what vendor and interface is used to communicate with the actual network southbound. The feature model is defined as an XML schema or as a YANG model.

The Resource Configuration solution provides a set of feature model examples for most common network functionality.

- **Vendor Templates**

The vendor template is a configuration file that describes how to setup one or several features in a specific type of device, for example, how to configure the BGP feature in Juniper routers.

- **Device Repository**

The Device Repository keeps track of all the devices that can be managed in the Resource Configuration solution. It stores information like login credentials, device type, and information about the actual device configuration. It also stores historical information about previous

configurations. This makes it possible to visualize configuration changes over time, and revert back to the configuration at any point in time. It is also possible to manage the current device configuration directly from the GUI.

- **Network Abstraction Engine**

The Network Abstraction Engine is using the definition of the feature models in combination with the vendor templates to generate the southbound commands towards the device. It is also responsible for keeping the information in the device repository updated and providing a generic rollback mechanism.

3.1 Network Abstraction Engine

The Network Abstraction Engine coordinates all Network Abstraction steps during activation and configuration of the network features. A prerequisite for this is that the corresponding feature models and vendor templates has been defined.

It is possible to import additional feature models and templates to extend the supported solutions. This is done through the GUI or CLI on the runtime system. It is also possible to edit existing templates from this GUI, as shown in Figure 3.

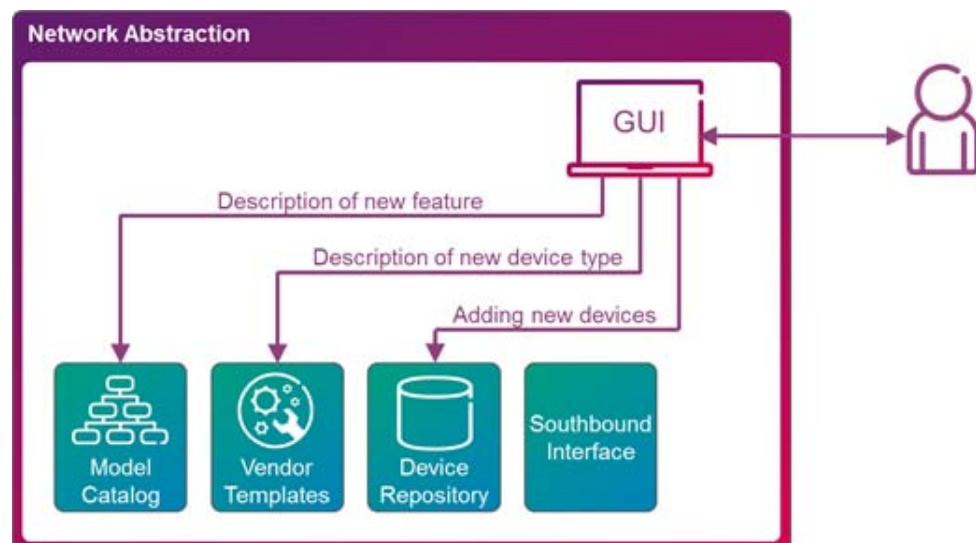


Figure 3 Integrating New Features And Devices

When new models and templates have been imported or updated, they immediately take effect on the runtime system.

Once the models and templates are in place, it is time to register the devices that are to be managed via the Resource Configuration solution. Once this is done, it is possible to start sending configuration requests towards the devices.

Figure 4 illustrates the Network Abstraction process.

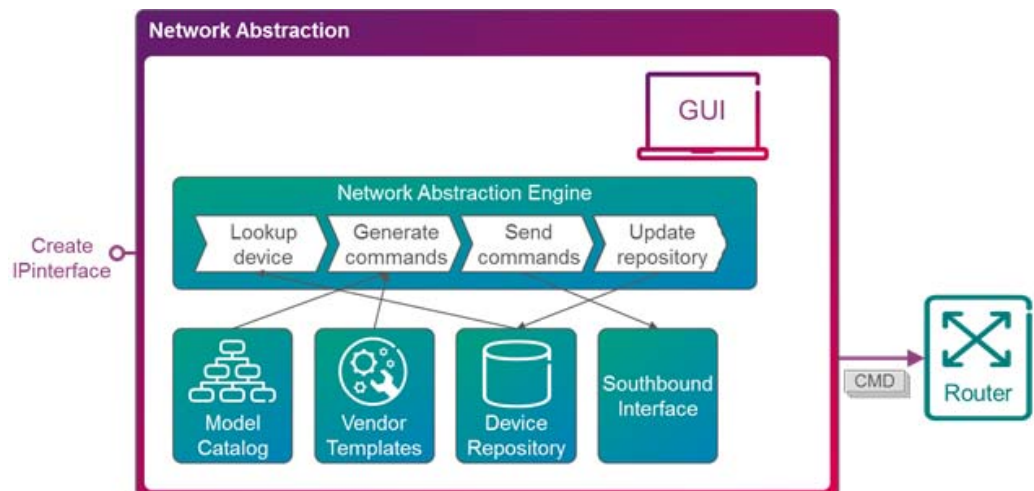


Figure 4 Network Abstraction Process

When the Network Abstraction Engine receives a new request to configure a network feature, like creating a new IP interface in a Juniper router, the following steps take place:

1. **Lookup device:** based on the information in the incoming request, the Network Abstraction Engine looks up information about the device in the Device Repository. This includes for example IP access points, device parameters and a list of templates.
2. **Generate commands:** the Network Abstraction Engine reads the relevant feature model definition and validates the request.
 - Based on the information in the Device Repository, the correct vendor template is retrieved and the corresponding commands are generated.
 - Based on the template protocol type, the matching access point is used to establish a connection towards the device.

Note: One device can be configured with multiple access points using the same template protocol type. In this scenario, the secure protocols always have a higher priority to be chosen.

For example:

- A CLI template can be used towards either an SSH CLI or a TELNET access point, where SSH CLI has a higher priority.
 - An HTTP template can be used towards either an HTTPS or an HTTP access point, where HTTPS has a higher priority.
3. **Send commands:** Network Abstraction Engine sends the commands to devices via southbound interfaces.



4. Update repository: after the commands are sent, Device Repository is updated with the current configuration of the device.

3.1.1 Support for Parallel Commands toward Southbound

The Network Abstraction Engine has the capability to run multiple commands in parallel towards the same device and still keep the data integrity of the device. This is a very important functionality since Dynamic Activation typically handles multiple northbound requests in parallel, and several of those might affect the same device. At the same time Dynamic Activation is a cluster and there might be multiple instances accessing the same device at the same time.

What further complicates the situation is that different devices handle multiple sessions very different. A simple Customer Edge (CE) router might not even allow parallel sessions, while a more advanced Provider Edge (PE) router is built for handling multiple sessions.

When a new device is defined in the Resource Configuration solution, it is possible to configure how much parallelism that will be allowed. In the case of a simple CE router the node might not support parallel updates, and it is then possible to put a lock on device level when updating the device. This means that if one Resource Configuration task is updating the device and another task wants to update the same device it will have to wait until the first update is finished.

When interacting with more advanced nodes, like PE routers, parallel updates are usually allowed. It is then possible to put a lock on feature instance level instead. If a specific feature instance (like for example a specific “EthernetInterface”) is being updated no other updates on this specific interface will be allowed before this task has finished. It is however possible to at the same time update other “EthernetInterface” instances on the device or other features on the device. In this example, the “feature instance lock” protects the device for ending up in unexpected state due to conflicts between updates being done in parallel.

When using the more fine-granular “feature instance lock”, it is also possible to configure the maximum number (up to 100) of parallel transactions that allowed to setup for the device. This limit is a system wide property and applies to all Dynamic Activation nodes in the cluster.

3.1.2 Rollback Logic

The Network Abstraction Engine provides a general rollback support. If a command sequence of 20 CLI commands fails after 15 commands, all rollback will be done on all executed commands to leave the device in the same state as it was in the beginning.

The rollback solution is based on the generation of reverse commands:

- For a create operations the corresponding delete operation will be sent.



- For a set operation a rollback to the previous state is done by sending a new set command with the old parameter values. The old values will be read from the device repository.
- For a delete operation a rollback to the previous state is done by sending a create operation with the original parameters values fetched from the device repository.

The response code from Network Abstraction Engine will indicate if the rollback was successful or not.

3.2 Model Catalog

The Model Catalog keeps track of the feature models, where the feature provides a logical abstraction of network functionality. A typical example is BGP. This feature model is exposed and possible to manage over the northbound interfaces, but it usually wrapped by a service model that makes use of one or several feature models to setup a service (like L3VPN) in the network.

All features in the Model Catalog are exposed as Managed Objects over the northbound CAI3G interface. They are also used for validating the incoming requests.

3.2.1 Manage Feature Model

A GUI is provided where it is possible to view the different models in the system. The models can be expressed as attribute lists (as shown in Figure 5), XML schemas or YANG models.



The screenshot shows the 'Model Browser' interface. On the left, a 'Feature Models' list includes various protocols like Access, BGP (selected), BridgeDomain, DHCPRelayAgent, DHCPSErver, EthernetInterface, EvcBridge, Firewall, GLBP, INNI, IPPIM, Internet, LoopbackInterface, MPLS, NveInterface, OSPF, Policy, SerialInterface, SystemContact, TVConnection, UNI, VPLS, VRF, VRRP, VoD, Voip, Xconnect, and tunnel. The main area displays the 'BGP' model with tabs for 'Data model', 'Interface model', and 'Interface definition'. The 'Attributes' tab is active, showing a table of BGP attributes. Below this, there are sections for 'Imported schemas' and 'Elements'.

Node name	Type	Cardinality
asNumber	unsignedShort	required
groupName	stringType	required
bgpType	stringType	0 or 1
sourceInterface	stringType	0 or 1
routeReflector	stringType	0 or 1
neighbors	complex	0 or 1
neighbor	complex	0 or more
neighborIp	IpAddressType	0 or 1
remoteAs	unsignedShort	0 or 1
redistributes	complex	0 or 1
vrf	complex	0 or 1

Location	Namespace
types.xsd	http://schemas.ericsson.com/scm/Types/

Figure 5 Model Catalog - BGP As Attribute Lists

It is possible to access contents of the Model Catalog via the Model Browser GUI. A feature model can be visualized as an attribute list, an XML schema or a YANG model in the GUI, but all those are just different views of the exact same feature model.

From this GUI, it is also possible to import additional models to the model catalog. It is also possible to disable or delete models from the system. When a model is disabled, it cannot be used for sending create or set commands.

When model is deleted it cannot be used to perform any operations towards a device.

Before deleting a model, user has the possibility to find out if model is currently used by any device.

3.2.2

Generation of Interface Models

Before an imported XSD or a YANG model can be used as a feature model in traffic, it must have an associated interface model defined by an XML schema and a WSDL file.



This is because an XSD or YANG model is a strict data model and does not contain information on what attributes to include in each operation, and the cardinality for each attribute. For example, in an interface model, an attribute might be mandatory in the Create operation, optional in the Set operation and not permitted in the Delete operation. The XML schema and the WSDL file contain this type of information.

The Model Browser GUI provides a interface generation wizard, which makes it possible to add this type of definitions and generate the necessary interface definitions for the northbound systems. The generated schemas and WSDL are also used to validate all incoming requests.

It is also possible to update the interface schema using GUI wizard. This will automatically update the interface schema and WSDL.

When exporting a feature model from the Model Browser GUI, the interface model is included in the exported zip file. Therefore if importing a feature model by using such a zip file, it is not needed to run the interface generation wizard.

3.3 Template Management

The Resource Configuration solution provides a model-driven solution that makes it possible to map the generic features requests to vendor specific commands that should be sent southbound to the devices. The solution is based on the feature models in the model catalog together with vendor templates that defines the structure of the southbound commands.

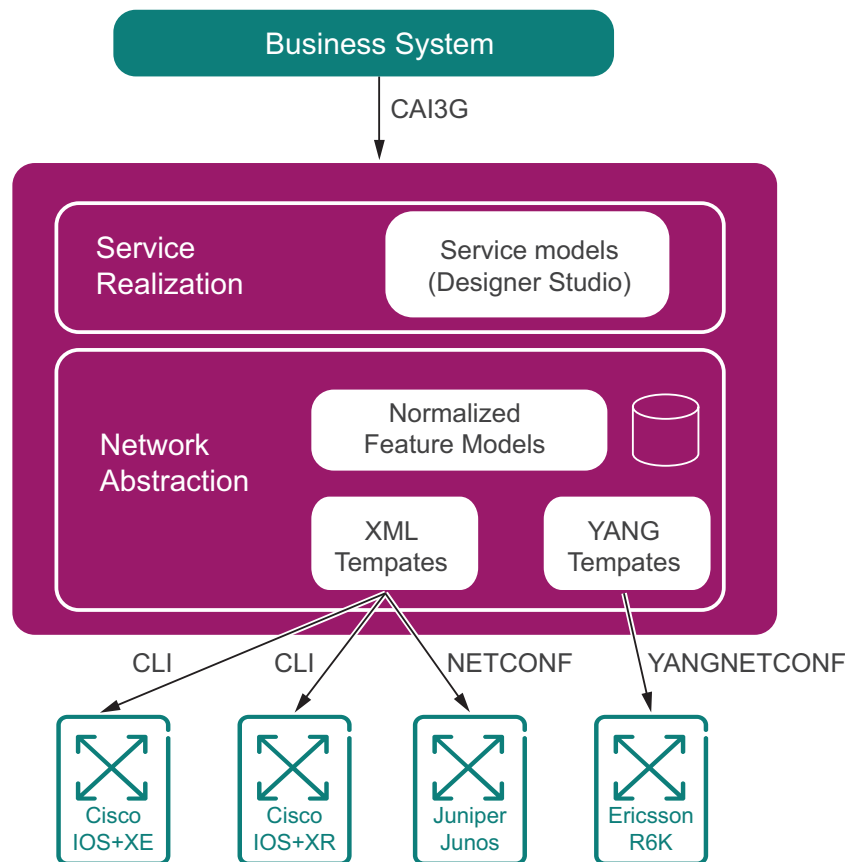


Figure 6 Example of Vendor Templates

A vendor template is a configuration file that describes how to setup one or several features in a specific type of device, for example, how to configure the Loopback interfaces feature in Juniper routers. There are two types of templates:

- **XML Template** – The XML template describes the command sequence to setup a feature in a device and how to map the data from the northbound interface to those southbound commands. Several GUIs are provided to simplify the work with the XML templates.
- **YANG Template** – The generic YANG engine simplifies the integration of all devices that exposes a YANG/NETCONF interface. A GUI is provided where it is possible to map the data in the feature model with the device specific data model, based on the constraints given by the YANG file. The corresponding NETCONF commands are automatically generated.

The XML Template are divided in specific schemas for CLI, NETCONF, SNMP, and HTTP. It is possible to use different protocol types for different operations or features on the same device. This is achieved by using protocol specified-templates and then configuring multiple-access-points on a device.

It is also possible to leverage on both XML templates and YANG templates for the same device type. In general, the YANG templates provides a quicker



integration but they do not provide a full business logic engine like the XML templates. Integration with XML templates takes some more time, but on the other hand they provide much more flexibility.

By combining YANG templates and XML templates, it is possible to get some of the benefits from both technologies. It is for example possible to leverage on the YANG engine when integrating Create and Set support, but use the XML templates for the mapping of Get and Get results.

If a feature has a template realization in both an XML template and a YANG template the Network Abstraction Engine will use the XML template. This makes it possible to overrule YANG template logic with XML template logic when necessary.

Note: The same operation on a feature can exist in multiple templates related to the same device. The Network Abstraction Engine will use the template that is first found in runtime.

It is possible to create and modify vendor templates on the runtime system. This is done via the GUI. When creating a template, the user is prompted to select a protocol type, and then a protocol-specific-schema is loaded. All templates are kept under version control. The generation of southbound commands in the Network Abstraction engine is always using the latest version of a template.

3.3.1 XML Template

Vendor templates in the form of XML templates can be used to generate down-stream commands for devices that exposes CLI, NETCONF, SNMP and HTTP(S) interfaces. This is done by defining the command-response sequence for this type of device and map required parameters to the input data provided from the feature model.

The Template Management GUI simplifies the management of the XML templates of CLI type. It for example provides syntax highlighting of the XML configuration where a color indication immediately tells if for example an XML tag is not properly ended. It is also possible to collapse and expands the different sections of the template.

3.3.1.1 Template Format

The vendor template describes the prompt-response behavior for one or several features defined in the model catalog. It replaces the placeholder for attribute value with the real attribute values provided from the northbound interface.

Figure 7 with below explanations, shows an example of a vendor template.



```
<?xml version="1.0" encoding="utf-8"?>
<cli_template xmlns="http://schemas.ericsson.com/nca/template/">
  <name>IOS_Template</name>
  <description>Communication template for IOS operating system for cisco devices</description>
  <prompt>">"</prompt>

  <definitions>
    <definition>SCM</definition>
  </definitions>

  <feature name="EthernetInterface">
    <!-- EthernetInterface Model version 2.0 -->
    <operation type="create">
      <bindings>
        <val name="interfaceType">
          <xpath>interfaceType</xpath>
        </val>
        <val name="interfaceNumberWithSub">
          <xpath>interfaceNumber</xpath>
        </val>
        <val name="(interfaceNumber,subInterface)">
          <def>
            {
              interfaceNumberWithSub.split("\\.") match {
                case Array(x) => (x, "")
                case Array(x, y) => (x, y)
                case _ => ("", "")
              }
            }
          </def>
        </val>
      </bindings>
      <command>
        <request>
          (subInterface) match {
            case "" => s"interface $interfaceType$interfaceNumber"
            case _ => s"interface $interfaceType$interfaceNumberWithSub"
          }
        </request>
      </command>
      <command>
        <request>
          "" match {
            case "s" => s"$serviceInstanceId $vlanId"
            case _ => ""
          }
        </request>
      </command>
    </operation>
  </feature>

```

a

b

c

d

Figure 7 Vendor Template Example

- a The protocol type definition of the vendor template.
- b The relation to the feature in the model catalog.
- c The vendor template explains the contract how to communicate over this interface as a sequence of requests and prompts.
- d Data values are provided from the request in the northbound interface and mapped to the generated CLI command according to the mapping from service model to the feature model in the model catalog.



The vendor templates are based on XML configuration files with a set of predefined primitives. Those primitives take care of more advanced data management.

The XML templates makes it possible to define configuration interactions with pure configuration, but it also allows more advanced extensions where it is possible to add fragments of Scala code to the templates. This Scala code is compiled together with the rest of the template logic when the template is saved on the system. The possibility to bring in Scala code to the template makes it possible to put quite advanced logic in the templates when needed.

A more complete explanation of the template format can be found in the *Customer Adaptation Guide for Resource Configuration*, Reference [3].

3.3.1.2 Template Definitions

It is possible define a set of type definitions in a separate file and then import this into the definitions of the vendor templates. The purpose is to avoid multiple definitions of common constructs like for example regular expression for an IPv4 address. Moving some of the complexity out from the templates improves the usability at the same time as it avoids potential errors due to multiple definitions of same things.

3.3.1.3 Template Editor

The Template Editor GUI provides an alternative approach to edit templates for CLI interfaces by using GUI elements rather than XML format. The purpose is to simplify making changes to templates and making editing more user friendly. The output from the template editor is still a standard XML template, and it is possible to seamless jump between the XML view and the Template Editor for the same template.

While working in the Template Editor the XML overhead is no longer visible. This makes it possible to focus on the actual southbound commands that should be generated. The necessary XML notations are generated automatically, this includes all the XML tagging as well as the required xPath bindings between the northbound API and the attributes used in the command statements.

All attributes available from the northbound interface or as runtime parameters are provided in a simple drag-and-drop GUI. The command is manually entered in the Request generation field and the relevant attributes are dropped into this section. The corresponding attribute bindings and xPath statements the XML template is generated automatically. This also makes sure that proper attributes are used, preventing errors due to for example misspelling of attribute names.

3.3.2 YANG Templates

A YANG template provides a quicker way to integrate devices that exposes a YANG/NETCONF interface. The solution makes use of a generic YANG engine

to automatically generate the southbound NETCONF commands based on the constraints given by the device specific YANG file.

The YANG template is put on top of this Generic YANG engine to map the device specific YANG model to the normalized feature models. A GUI is provided to simplify this mapping task. This GUI also adds the possibility to add parameter transformations.

A GUI wizard is provided that allows the user to do mapping between the normalized feature models and the device model exposed by the Generic YANG engine. The GUI makes it possible to browse all the parameters of the interface model for the feature and the device specific YANG model, and then simply map the relevant parameters and save the mapping.

Figure 8 shows the how the GUI wizard is used to map attributes related to the “EthernetInterface” for the Create operation. The device specific YANG model is displayed on the right, and it is possible to browse the YANG/NETCONF hierarchies in the same way as browsing a directory hierarchy on the file system.

The YANG model defines a number of constraints that also are visible on the screen. In this example the “name” attribute is a key attribute, “type” is a mandatory attribute and “ethernet” has a “must” constraint (more details are provided when clicking on the icon in the GUI).

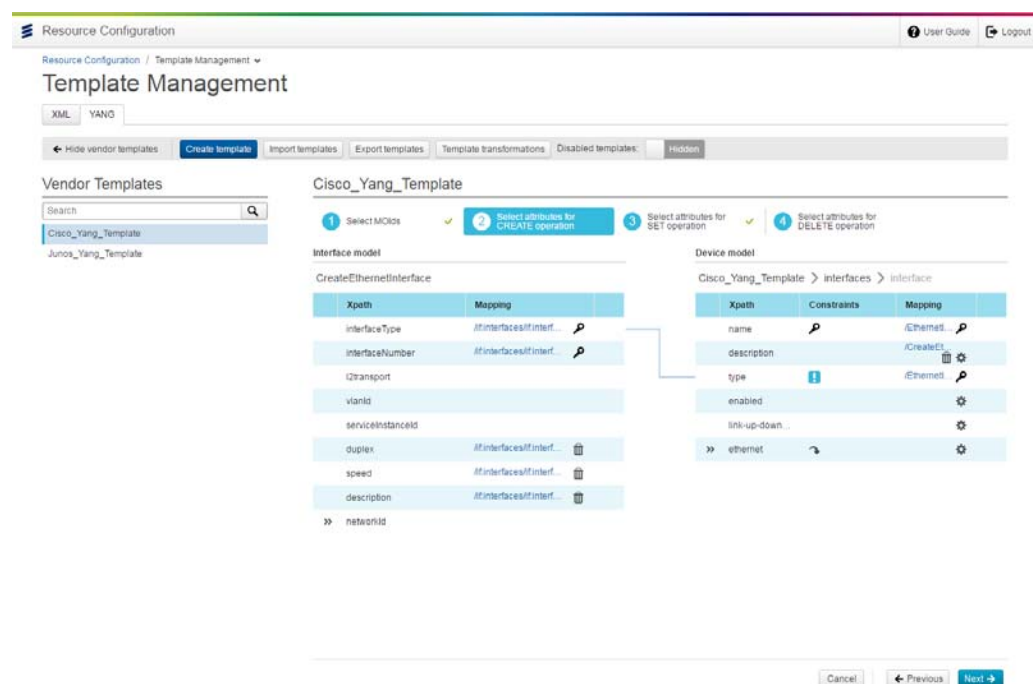


Figure 8 YANG Template GUI Wizard



3.3.2.1 Request Transformations

The YANG engine should be seen as a simple tool for solving simple tasks. It should focus on data mappings, not as an additional engine for creating business logic.

In some cases, it is however necessary to add some basic logic to be able to do a mapping between the normalized feature model and the device specific YANG model. To manage this, it is possible to add data transformation to attribute values.

Below are a couple of examples of mapping problems that can be solved with the introduction of transformations:

- **Appending strings** – Juniper expects router target value to be in the format “target:6500:200”, while other vendors like Cisco only expect “6500:200”. The string “target:” needs to be appended in the Juniper case.
- **Numeric conversions** – R6K uses dotted decimal (0.0.1.2) format for OSPF area, while Cisco and Juniper uses decimal format (258). The normalized northbound interface should use decimal format for all devices and conversions are necessary
- **Data substitution** – “GigabitEthernet” is a valid InterfaceName for a Cisco device, while the corresponding name for a Juniper device would be “ge”.

All those scenarios are possible to handle by adding transformation logic to the attribute mappings. The solution will in the back-end leverage on the XML template engine to resolve the mapping logic, see below for a simple example of a transformer:

```
<transformer xmlns="http://schemas.ericsson.com/ma/transformer/">
  <name>Transformer</name>
  <bindings>
    <val name="appendWithText">
      <function value="value">
        "Appended Text " + value
      </function>
    </val>
  </bindings>
</transformer>
```

Figure 9 Simple Transformer



3.4 Southbound Interfaces

3.4.1 CLI/SSH

A generic SSH adapter is provided to support CLI (Command Line Interface) for managing network devices. Since protocol details are defined in the templates, it is possible to generate commands for virtually any CLI based interface towards a network device.

3.4.2 CLI/Telnet

A generic Telnet adapter is provided to support CLI over Telnet. This is mainly used for integration of legacy devices that lacks support for more modern protocols. Since protocol details are defined in the templates, it is possible to generate commands for virtually any Telnet based CLI interface towards a network device.

3.4.3 NETCONF

Network Configuration Protocol (NETCONF) is an xml based network management protocol standardized by the IETF. It uses Remote Procedure Calls (RPC) for transfer of messages over SSH. It is becoming a standard network management protocol, at the same time as it provides the possibility for a device to register the services it provides. This makes NETCONF a superior technology choice for automated device activation.

A generic NETCONF adapter is provided for management of devices. NETCONF can also be used for handling notification but this is currently not supported by the adapter. The focus of the adapter is to support managing of devices, not monitoring of devices.

Two different NETCONF adapters are available.

- **Netconf-Connector** – This is the preferred NETCONF adapter for all devices that are well aligned with the NETCONF protocol, i.e. following the defined methods in the Netconf RFC.
- **Netconf-Custom-Connector** – This is a less strict NETCONF adapter that can be used for integration of devices that extends the NETCONF interface with own methods.

Note: **Netconf-Custom-Connector** is only applicable for XML template.

3.4.3.1 NETCONF Capabilities

The adapter automatically does a discovery of feature capabilities of the device it should manage in order to understand what technical capabilities it provides.



A typical example of technical capabilities exposed over NETCONF is the support for “data stores”, where each node providing a NETCONF interface can offer different “data stores”.

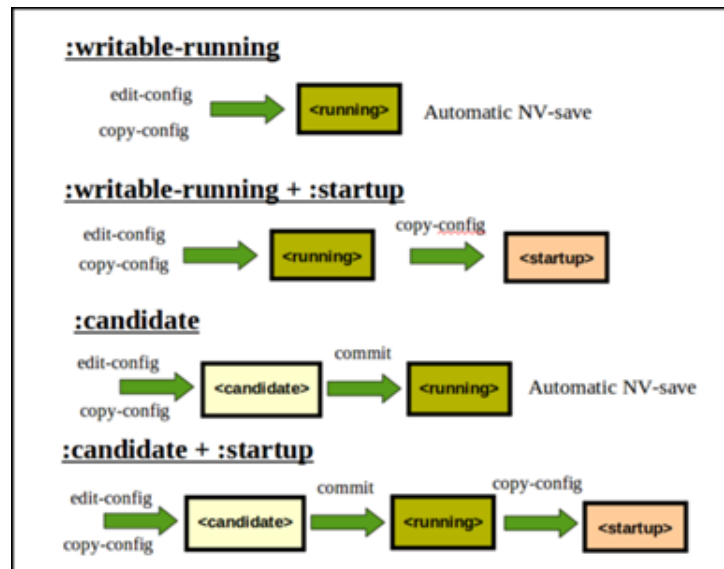


Figure 10

3.4.3.2 Example of NETCONF Data-stores

If a NETCONF node registers a “:writable-running” capability it means that all changes of the configuration immediately affects the configuration used in real traffic `<running>`.

If the node instead registers a “:candidate” capability it means that it has two parallel configurations, one that is being used in traffic (`<running>`) and one that is a pre-step while setting things up (`<candidate>`). When the Resource Configuration solution configures such a node it will first set up all data in the `<candidate>` data store and then commit it into the `<running>` data-store.

If the node registers a “rollback-on-error” capability the Network Abstraction Engine skips the native rollback logic. This means that the node will handle the rollback on feature level itself. The Resource Activation solution will only handle the rollback on service level in this case.

It is first when a discovery of the supported NETCONF capabilities has been done that it is possible to know how to interact with the node and the Network Abstraction Engine will then automatically adapt to the device capabilities.

3.4.4 HTTP(S)

A generic HTTP adapter is provided for managing network devices. It supports HTTP POST, GET and DELETE.



For increased security it also supports HTTPS. When adding a device with access point protocol set to HTTPS, the default behavior is to trust the given device. This security setup can be further strengthened based on trust store and bi-directional authentication. This will require additional parameters possible to set from the GUI.

3.4.5 SNMP

A generic SNMP adapter is provided for managing network devices. It supports SNMP SET and SNMP GET. Further SNMP commands are possible to provide as a Market Requirement Development (MRD).

3.4.6 SDN Adapters

Resource Configuration can be integrated to SDN controllers such as Open Daylight and Floodlight. The integration supports a number of different use cases, like management of the Open vSwitch (OVS) and Openflow management where it possible to create and configure flow policies and also retrieve and manage topology.

SDN is an emerging network architecture where network control is decoupled from forwarding and is directly programmable. Network intelligence is (logically) centralized in software-based SDN controllers, which maintain a global view of the network. As a result, the network appears to the applications and policy engines as a single, logical switch.

The OpenDaylight Project is a collaborative open source project hosted by The Linux Foundation. The goal of the project is to accelerate the adoption of SDN and create a solid foundation for NFV. Ericsson is a platinum member of OpenDaylight.

3.4.7 Additional Southbound Protocols

Additional southbound protocols will be supported in coming releases. If any customer requests support for other protocols prior to the official release date, they will be possible to provide as an MRD.



4 Device Management

4.1 Device Repository

The Device Repository keeps track of all the devices that are possible to manage in the Resource Configuration solution. It also keeps information about the actual device configuration, both current configuration and a record of all previous configurations.

4.1.1 General Information

The Device Repository keeps some general information on the device, like for example the device identifier and the device type. The device type indicates a type of device, like for example “Cisco IOS-XE”. This information is very important since it will be used by the Network Abstraction Engine to figure out what vendor template to use when southbound commands should be generated towards this device.

An important part of the general device information is also the settings for how Resource Configuration should manage the devices and if functionalities like Candidate store, Feature instance lock and Configuration validation should be used for the specific device. More information about those functions are provided in the following chapters.

4.1.1.1 Jump Servers

As part of the general information of a device is the “device category”. It is used as a way to group the different devices into a set of predefined categories. The device category “jump server” has a special purpose and will trigger a special handling.

A jump server is used as a hop to route to a device in a secured network. This means that the jump server is not a device with configuration that should be managed. It is just used to get access to the device that should be managed. The Network Abstraction Engine will login to the jump server and then establish a new session from the jump server to the device that should be managed.

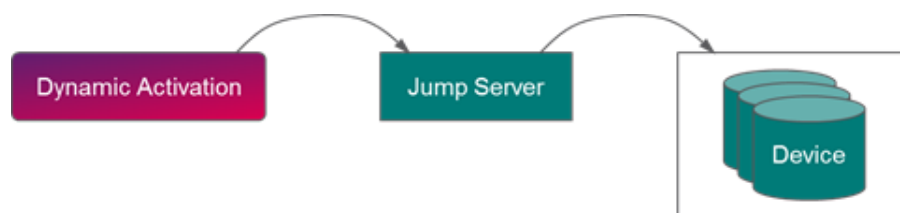


Figure 11 Jump Server

4.1.1.2 Proxy

The device category “Proxy” is used to add an HTTP proxy for devices which are not reachable directly from Dynamic Activation through HTTP/HTTPS interfaces.

The proxy is not a device with configuration to manage. Dynamic Activation uses the proxy only to get access to devices which are to be managed. Such devices must support HTTP connection.

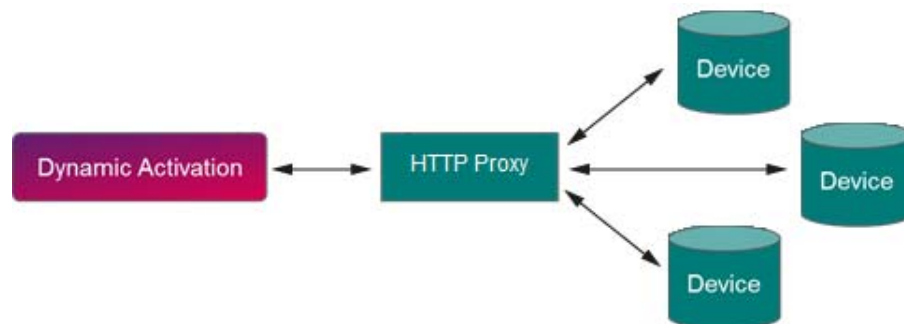


Figure 12 Proxy

4.1.2 Access Points

The Access point defines how the Resource Configuration solution should communicate with the device. The information includes things like IP address, login credentials and supported protocols. All device related passwords are stored encrypted in the Device Repository.

One device can be configured with several access points, but only one for each protocol.

4.1.2.1 Connector Parameters

As part of the device specific information in Device Repository, it is possible to override default configuration of connector parameters, like timeouts, for the connector used to communicate with the device. The connector parameters are possible to manage from the GUI, where the entry fields automatically are rendered based on information the backend connector provides and supports. Default values are provided so the user not always have to configure this when adding devices. The connector parameters are only used when the default value should be replaced for a specific device.

The management solution of connector parameters is generic and it applies to all available connector parameters exposed by the backend. Typical examples of provided parameters are connection timeouts, HTTPS credentials and Telnet parameters.



4.1.2.2 Integration of Devices via EMS

When adding devices that are integrated via an Element Management System (EMS) the access point is the EMS instead of the physical devices.

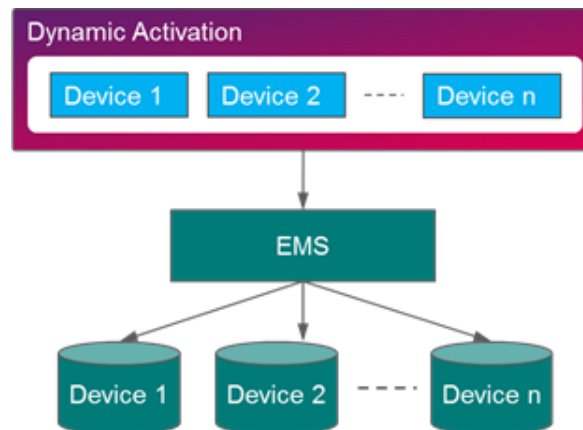


Figure 13 Integration via EMS

All managed devices must be defined as individual devices in the Resource Configuration solution, even when they are integrated via an EMS. There are a number of good reasons for this:

- Avoid losing important Resource Configuration functionality.
- Makes it possible to visualize the different devices separately in the GUI.
- Secures that control of parallel updates happens on the right level.

4.2 Managing Devices

When a new device is added in Resource Configuration, it is necessary to provide some basic information about the type of device and how to communicate with it. A GUI is provided where it is possible to manage the devices. It is also possible to manage the devices using a REST API.

4.2.1 Import of Devices

In broadband activation, it is necessary to interact with a large number of network elements, which means setting up those network elements one by one manually is not realistic. Instead, it requires to be possible to import information about a bulk of devices in the network via a CSV file.

This CSV file is typically generated by an inventory system. If the user is importing a device that already exists, the internal information about it will be update according to the latest information in the import file. This can be used to periodically sync device information in Device Repository with the device information in an external inventory.



4.2.2 Device Filtering

A typical Resource Configuration solution will manage thousands of devices. To simplify management of networks with large number of devices, it is possible to apply filters on what devices to show in the Device Management GUI. It is possible to apply filters on most of the generic device attributes, like for example the "Admin-state", "Device-category", "Type" and devices that has pending actions from Candidate Store or specific features.

It is also possible to filter devices on service type. This means that it is possible to filter on all devices that runs a L3VPN service or a specific instance of the L3VPN service.

4.3 Management of Device Configuration

The GUI provides the capability to browse the current and previous device configurations. By moving the time-slider in the GUI, it is possible to view how device configuration has changed over time. It is possible to update the configuration directly from this GUI. It is also possible to add additional feature to a device directly from the GUI.

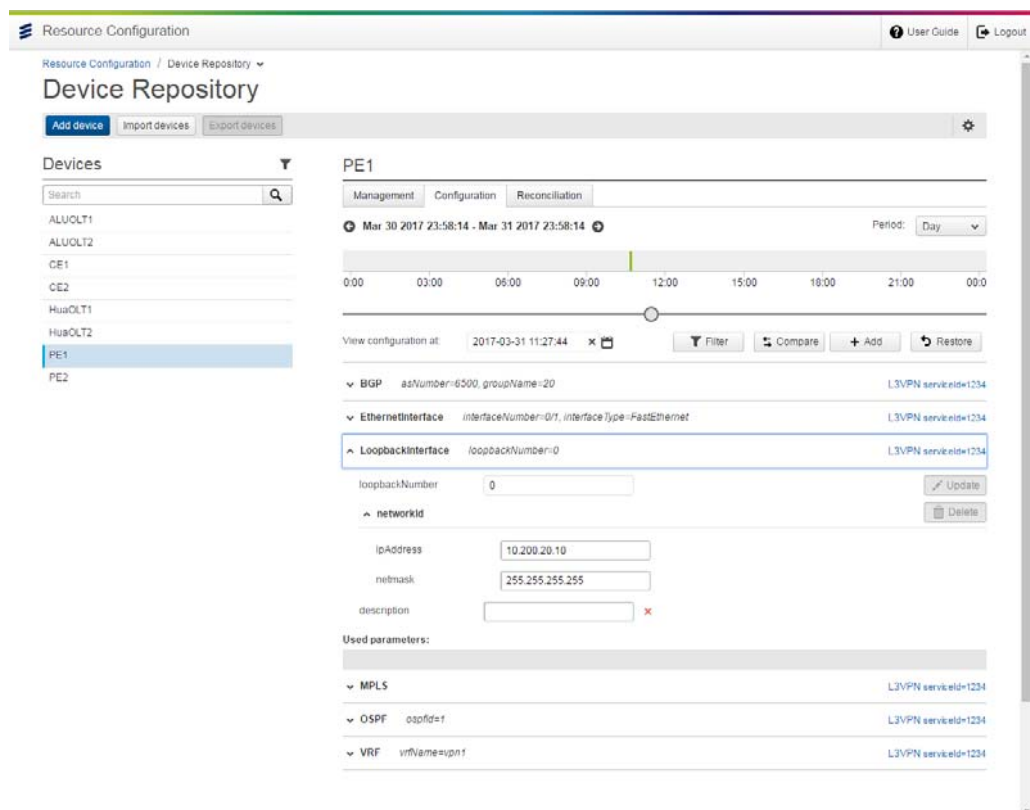


Figure 14 Browsing Device Configuration



4.3.1 Revert to Previous Device Configuration

It is possible to move the time slider to any point in time and click the revert button. If Candidate Store is not used for the device to update, the device is immediately provisioned with the configuration valid at the selected time and the information in the Device Repository is updated accordingly.

Note: It is not possible to restore service affecting features from the Device Management GUI: Restoring data that affects services is only possible from the Service Visualization GUI.

4.3.2 Candidate Store

The candidate store makes it possible to store device configuration in the Device Repository before sending the commands to the actual device. This makes it possible to manually inspect the planned changes and do a manual approval before the changes are committed and sent to the device, as shown in Figure 15.

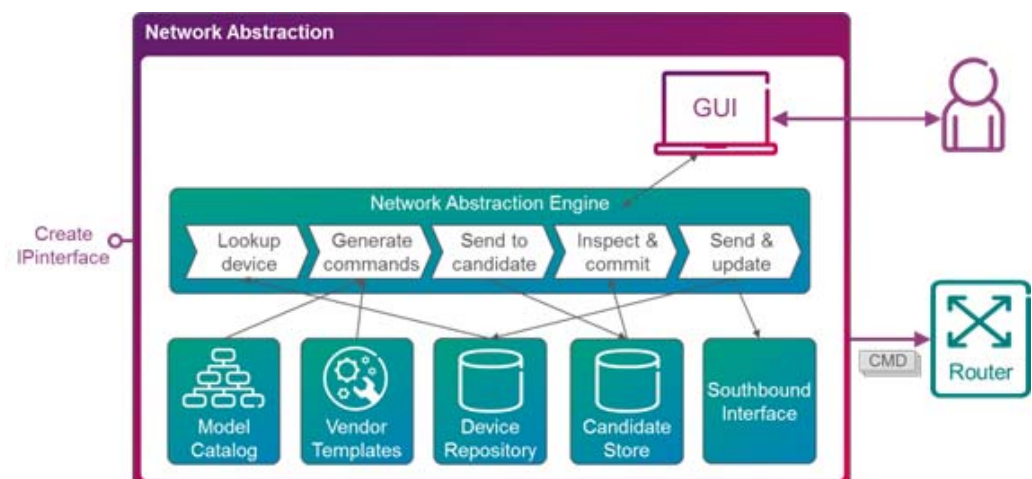


Figure 15 Workflow When Candidate Store Is Enabled

When user enables Candidate Store for a device in the Device Management GUI:

1. **Lookup device:** based on the information in the incoming CAI3G request, the Network Abstraction Engine looks up information about the device in the Device Repository.
2. **Generate commands:** the Network Abstraction Engine reads the relevant feature model definition and validates the request, and then retrieves correct vendor template to generate corresponding commands.
3. **Send to candidate:** the commands are sent to the Candidate Store and stored in the Device Repository.
4. **Inspect & commit:** the user views "candidate" device configuration and southbound commands, and decides to commit them or not.

5. Send & update: if the commands are committed, the Network Abstraction Engine sends the commands to device, and updates the Device Repository with the current device configuration.

Candidate store is enabled with a global setting or per device. A device that has uncommitted changes in candidate store will get an indication in the device list in the GUI. At the same time a new tab is introduced where it is possible to inspect the pending changes and the resulting southbound commands. From this tab, it is also possible to commit the changes.

Since the Candidate store GUI visualizes the resulting southbound commands without sending them to the device, it can also be used to verify new template implementations.

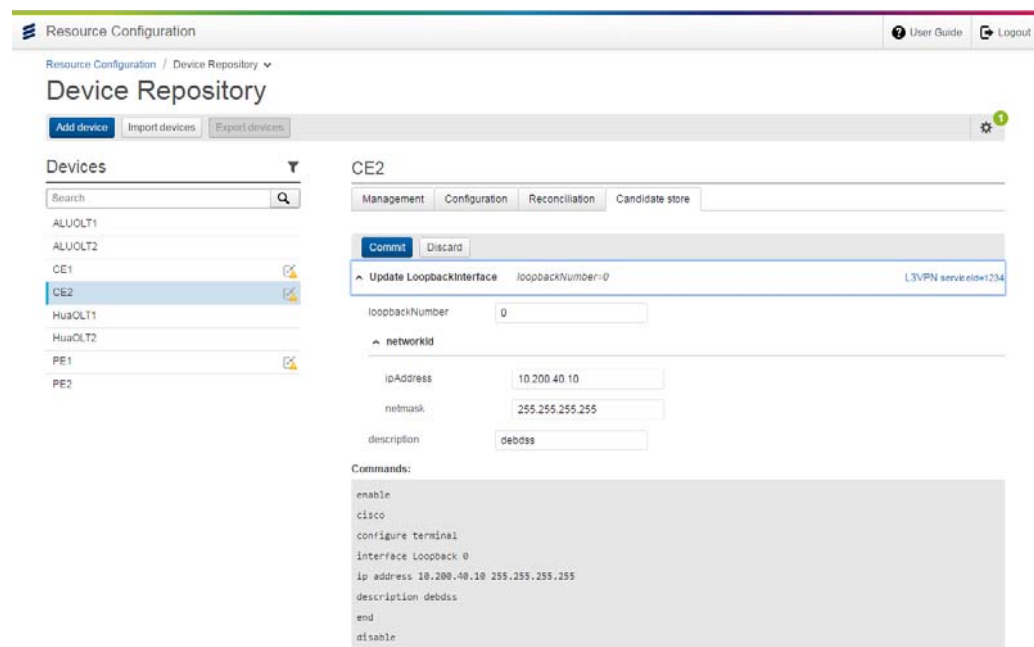


Figure 16 Inspecting Pending Transactions in Candidate Store

4.3.3 Loose Error Handling

The Loose error handling setting can be activated or inactivated in the global settings menu. If Loose error handling is activated, all duplicate requests will be returned with success response and no traffic will be send towards the device. All requests will be logged in the log management GUI.

When loose error handling is enabled all requests below will return a success message without any data sent down to the device:

- A Create request to create a feature that already exists.
- A Delete request to delete a feature that does not exist.



- A Get request to get a feature that does not exist in the device. In this case an empty message is returned instead of an error.

4.4 Synchronization of Data Models

4.4.1 Configuration Validation

Sometimes devices are updated out of control of Dynamic Activation. This typically happens when someone manually logs in to the device and run a set of CLI commands. The information in the Device Repository will then get out of sync with the actual device configuration.

When the configuration responsibility has been delegated to Dynamic Activation, it is very important to not allow manual updates like this, but it typically happens anyway.

To address this problem, it is possible to enable the “Configuration Validation” functionality. If it is enabled an automatic comparison is done of the state of the feature in the Device Repository and the corresponding state in the device, before any changes to the feature is applied. If any differences are found, users can reconcile the issue in one of the following ways:

- **Display error** – The northbound system will be notified in an error response that contains the details of the discrepancies between the configurations. No configuration changes will be applied to the device.
- **Update device** – The device is updated with the stored device configuration from the Device Repository. When the device is in sync with the device repository again the requested configuration changes will be applied to the device.
- **Update repository** – The Device Repository is updated with the current device configuration in the network. When the Device Repository is in sync with the device again the requested configuration changes will be applied to the device.

If the configurations match, there are no discrepancies to reconcile and the device is provisioned with the new configuration.

4.4.2 Discovery of Device Configuration

In the Recourse Configuration solution, it is possible to discover the current configuration of a device and store this information in the Device Repository.

- Get initial device configuration

Without knowing the initial state of a device, the Device Repository lacks the full picture of the configuration of a device. As a consequence of this

the benefits of Resource Configuration functions like configuration verification check will be lost.

To address this problem, it is possible to trigger a discovery of initial device configuration when the device has no device configuration stored in the Device Repository.

- Discovery over REST API

It is also possible to trigger the discovery from a remote system by using REST API. The discovery over REST API can be triggered even if the device has already had a configuration stored in the Device Repository. The stored device configuration will be updated according to the current configuration of the device.

4.4.3 Comparison of Device Configurations

It is possible to compare the device configuration at two different points in time. The Device Management GUI will then compare the device configurations side-by-side highlighting any differences in the feature and parameter setup.

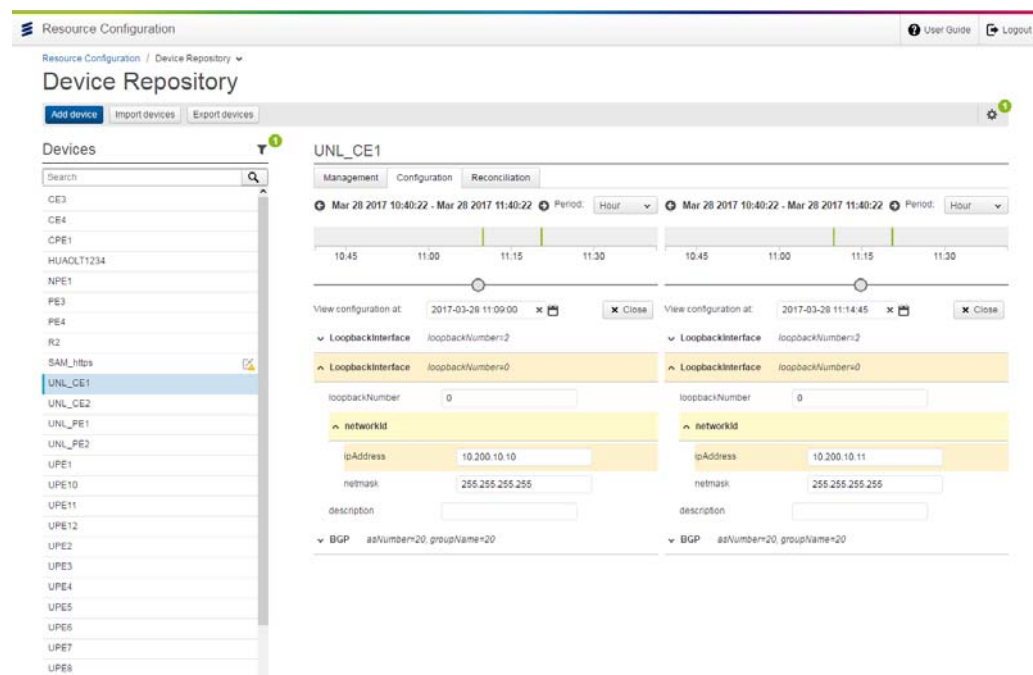


Figure 17 Side-by-side Comparison of Device configurations

4.4.4 Reconcile Data Repository Inconsistencies

In the same way as it is possible to compare device configuration at two different points in time, it is also possible to do a side-by-side comparison of the Device Repository data for a specific device and the actual device configuration



in the network. If any inconsistencies are found they will be highlighted in the GUI and it will be possible to reconcile the device according to the Device Repository, or reconcile the Device Repository according to the current device configuration. It is possible to access device reconciliation from a remote system by using REST API. The reconciliation over REST API will restore the latest device configuration from Device Repository to the device.

Note: It is not possible to reconcile service affecting features. Changes that has an impact on existing service instances has to be managed manually.

4.4.5 Synchronization of Replaced Devices

Since data synchronization is based on the normalized and vendor agnostic feature models Resource Configuration provides support for swapping out devices. Below is a description how a Cisco IOS device is replaced with a Cisco NX-OS device with a minimal impact on Dynamic Activation and the northbound systems.

Note: A prerequisite for this is that the feature models are defined in a normalized and vendor agnostic way. This use case is a good example why it is worthwhile to spend some extra time on the initial data modelling effort.

1. Start a discovery and reconciliation on the device to make sure that the Device Repository has the latest view of the device data.

If any discrepancies are found the option “update the inventory according to the information in the device” should be selected.

2. When the data consistency has been secured, it is possible to swap-out the old device.
3. Set the `Device state` of the old device to **INACTIVE**. This is to prevent new configuration from being sent to the device during the migration.
4. Replace the physical device in the network.
5. Update the `Device type` for the new device in the Device Management GUI. In this example, the device type is changed to a Cisco NXOS device and no longer a Cisco IOS device.
6. If the access point has changed (IP address, port, protocol and credentials) this also has to be updated in the Device Management GUI.
7. After device data is updated, change the `Device state` of the new device back to **ACTIVE**.
8. Start a new discovery and reconciliation process for the device. The Resource Configuration solution will detect that the device lacks all the configuration currently kept in Device Repository. Start the “reconcile device according to inventory” task.



- The device automatically gets updated with the configuration that previously was stored in the IOS device, but this time Resource Configuration generates NXOS commands to setup the configuration.

Note: The Resource Configuration solution will only sync-up information it has the responsibility to manage. It is a model driven solution and it only knows the information as it is defined in the models. This is not a generic backup-restore solution that covers all data in the device.

5 Service Realization

5.1 Service Visualization

The Service Visualization GUI shows how a service instance is decomposed into several features instances on a number of devices.

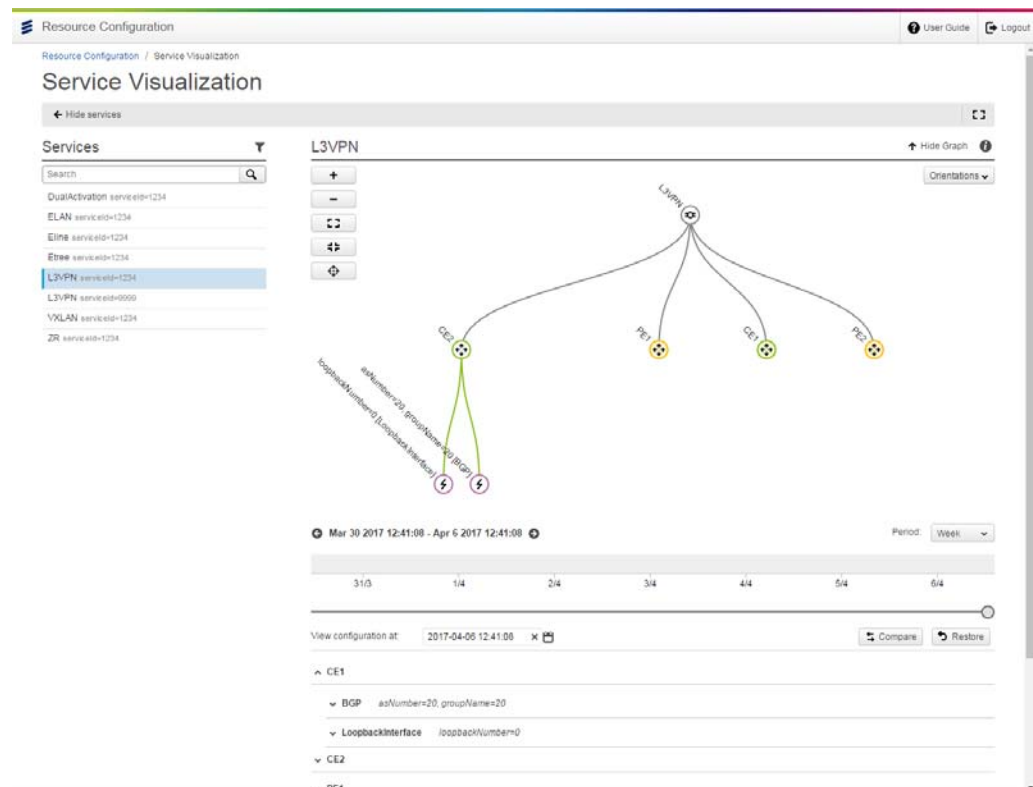


Figure 18 Service Visualization



It is possible to expand the information on the screen to drill down to the individual attribute values of the feature instances that constitute this service instance.

From the Service Visualization GUI, it is also possible to navigate to a particular device or feature to manage the configurations.

In the same way as a time line exist to visualize device configuration updates over time, a similar timeline exists in the service visualization view. This makes it possible to monitor configuration changes on service level, as well as managing and restoring service configurations.

It is also possible to compare service configuration at different points in time in the Service Visualization GUI. The graph will indicate objects that has been added, removed or modified. The detailed view of the differences is highlighted in a color-coded side-by-side view, similar to the existing comparison in the Device Management GUI.

5.1.1 Filtering of Services

Besides searching a part of the service name in the search box, the Service Visualization GUI also supports using filters to find services running on a specific device. When filtering is done, all services that match selected criteria are listed as a search result.

5.1.2 Service data in Device Repository

Each device is represented by a set of instantiated features in the Device Repository, where each feature instance has a deviceID and an optional parentID. The ParentID is populated with a serviceID if there is a service instance defined on top. In this way the Device Repository keeps track of how a service is decomposed into features on different devices. This also makes it possible to understand the service impact of a device configuration.

Based on the service data in Device Repository, it is possible to delete a service with a single CAI3G request using a service identifier. In a similar way, it is possible to get the service data related to a service identifier. The response will contain all the feature instances related to the service instance. As a next step, it is necessary to query for more details for each of those feature instances.

5.2 Service Modelling

The Service Realization back-end executes service logic defined in service models. Those service models are created in the Designer Studio tool. Designer Studio is a graphical tool in Dynamic Activation for designing customer specific service logic. With no programming needed, Designer Studio makes it possible to configure service models based on the existing features exposed by the Model catalog.

For more information, refer to *User Guide for Designer Studio*, Reference [5].

5.2.1 Service Models

A service model specifies the service realization objective. The input for the service model is interface specification for the CAI3G interface that should be exposed northbound, as well as the interface specifications for the feature models in the Model Catalog. It is also possible to integrate all Managed Objects (MOs) in Resource Activation as southbound services.

The service model is configured in the Designer Studio and later deployed in a Dynamic Activation system. When the service model is deployed in Dynamic Activation it will add the corresponding CAI3G interface to the northbound systems.

Note: Service models configured in the Designer Studio require a SW Advanced license to run on the Dynamic Activation.

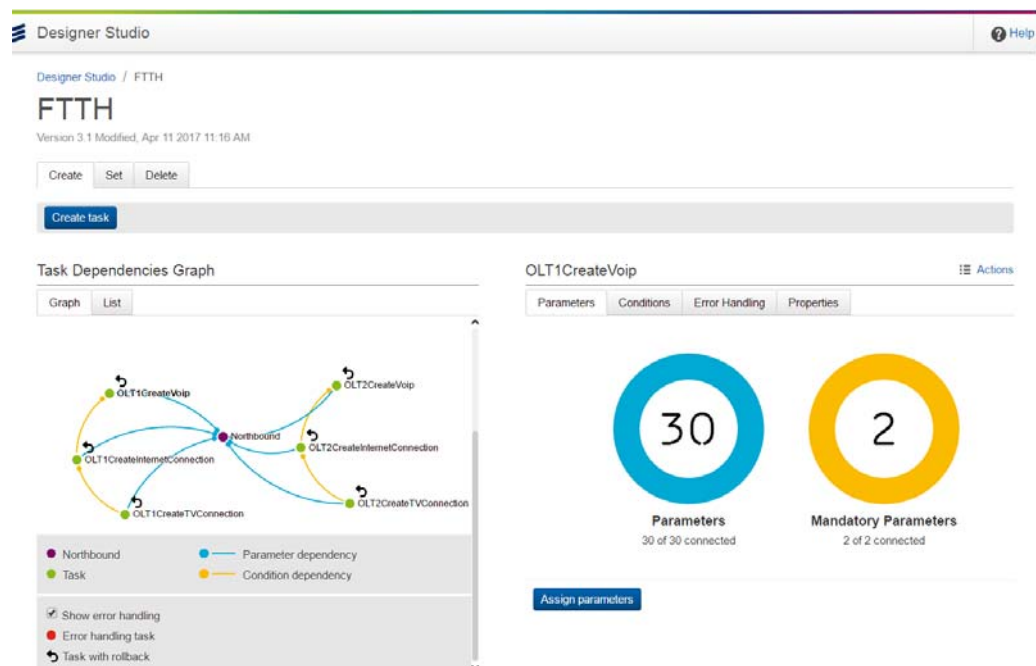


Figure 19 FTTH Service Model in Designer Studio

5.2.2 Tasks

A task is an action to be taken to realize a subset of the objective of a northbound request. A task is an operation (CRUD) of a southbound service and consists of three key attributes: parameters, conditions, and error handling.



5.2.3 Assign Parameters

To provide necessary data for the tasks within a service model, the attributes in the task need to be mapped against the incoming parameters from the northbound interface. It is possible to add own configurations and transformations to the parameter mapping.

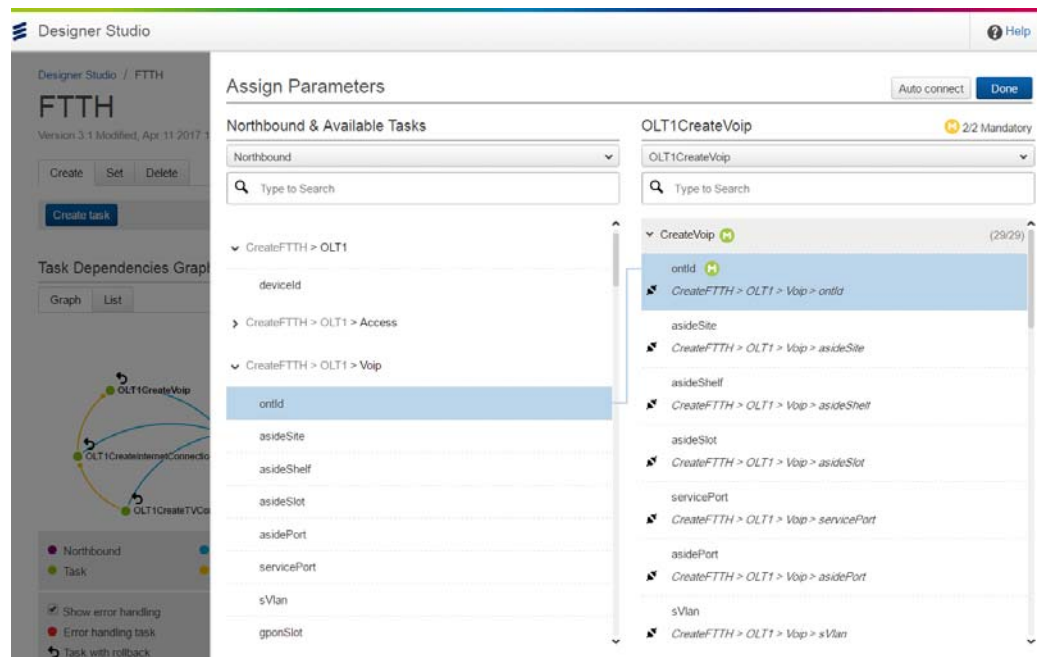


Figure 20 Assign Parameters Example

5.2.4 Dependencies

There are two types of dependencies for a task: parameter dependency and condition dependency. The dependencies influence the execution plan of the service for each northbound request.

A parameter dependency is defined by connecting a task parameter to a source parameter. Often the source is the northbound interface, but it can also be a response of another task.

A condition dependency is defined on per task basis.

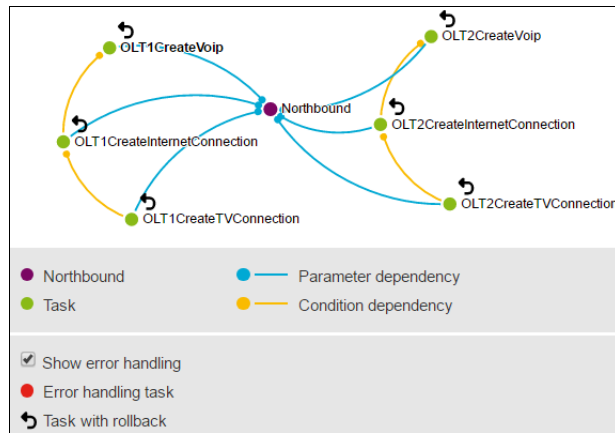


Figure 21 Dependencies in FTTH Service

5.2.5

Rollbacks

Rollbacks are built in into the service realization logic in the Dynamic Activation system. User decides at configuration time in Designer Studio whether rollback is enabled for a task or not. A rollback takes place if any of the other tasks have failed.



Reference List

Ericsson Documents

- [1] *Function Specification Dynamic Activation Execution Environment*, 6/155 17-CSH 109 628 Uen
- [2] *Library Overview*, 18/1553-CSH 109 628 Uen
- [3] *Customer Adaptation Guide for Resource Configuration*, 14/1553-CSH 109 628
- [4] *User Guide for Resource Configuration*, 11/1553-CSH 109 628 Uen
- [5] *User Guide for Designer Studio*, 10/1553-CSH 109 628 Uen