CONFIGURATION MODEL FOR NETWORK MANAGEMENT

Rudy Deca[1], Omar Cherkaoui[2] and Daniel Puche[3]
[1,2]University of Quebec at Montreal; [3]Cisco Systems, Inc.

Abstract: As today's networks increase in size and complexity and new network services are deployed, the management becomes more complex and error-prone and the configurations can become inconsistent. To enforce the configuration consistence and integrity, it is necessary to enhance the validation capabilities of the management tools. The Meta-CLI Model presented in this paper captures the dependences among the configuration components and the network service properties and translates them into validation rules. It also translates the device configuration information into tree-like models and checks their integrity and consistence using theses rules.

Key words: network management; network services; integrity and consistence validation; configuration rules; configuration constraints; configuration model.

1. INTRODUCTION

The constant growth of the Internet implies the creation and deployment of an ever increasing number of network services, each of which is becoming more complex in its turn. In this context, the network configuration becomes more difficult and error prone. Some of the causes are the diversity of configuration approaches and information repositories used by the configuration tools.

In plus, the main network configuration approaches, such as the command line interfaces (CLIs),[1,2] the Simple Network Management Protocol (SNMP),[3] the policy-based management (PBM),[4] the NetConf protocol,[5] etc., lack configuration rules that capture the dependences and constraints that characterise the network service configuration and lack adequate logical formalisms that could be applied to the information repositories to validate the configuration integrity and consistence.

These approaches therefore do not take into account the dependences and hierarchy that exist among the device parameters that express the service at device-level, the heterogeneity of the configuration means and the interactions between heterogeneous management and configuration modes.

The Meta-CLI Model proposes a solution for these configuration inconsistencies. Our model captures the features of the CLI, such as the context and parameter dependences of the commands, as well as the service properties into validation rules. It translates the configuration information into trees and validates them using the appropriate rules. Based on the Meta-CLI Model, we have implemented the *ValidMaker* module, and incorporated it in a policy provisioning VPN tool.

This section presents the dependences between the commands and/or parameters that are translated into Meta-CLI concepts and discusses the properties of the network service configurations. Section 2 presents the concepts, operations, functions and properties of the Meta-CLI Model tree structures and the validation rules and procedures that translate and validate the service properties. Section 3 presents the *ValidMaker* tool and Section 4 draws conclusions.

1.1 Configuration Dependences

Several types of dependences exist in the network device configurations.

A. ***Syntactic parameter constraints*** The CLI commands' syntax enforces the constraints regarding the order and number of parameters.

1. *Fixed number of parameters* The number of parameters (which can be mandatory or optional) must be correct, otherwise the command fails.

EXAMPLE The command to configure a primary IP address on an interface requires 2 parameters: the interface *IP address* and the *mask.*

2. *Fixed sequential order of parameters.* In a configuration command or statement, the parameters are ordered.

EXAMPLE In an extended access list, the order of the *IP address* and the *mask* parameters in the above-mentioned command is fixed.

B. ***Parameter and command existence dependences*** Some parameters and commands can only exist in specific circumstances.

1. *The existence of a parameter depends on another*

EXAMPLE In an access list, the *timeout* parameter, which specifies the duration of a temporary access list entry, can exist only if the access list entry is *dynamic* (i.e. has the *dynamic* parameter).

2. *Context dependences* The existence of a parameter or a command depends on the context (mode). Thus, a parameter can only be configured, modified or created in an equipment using a command in a specific configuration context (mode).

EXAMPLE When specifying the *IP address* of an interface, the name of the interface must be specified by a prior command that "opens" it to the user, who can then access and manipulate its resources or parameters (e.g.: IP address, MTU, bandwidth, etc.).

3. ***Result dependence*** The order of some commands can be fixed. In this case, the success of a command depends on the successful completion of a previous one.

EXAMPLE The configuration of a link bundle consists of bundling several similar physical point-to-point links between 2 routers into 1 logical link. By default, at each change in bandwidth in a link bundle, the combined amount of bandwidth used on all active member links is propagated. Optional features are available, by configuring the following parameters.

I. *Automatic propagation*, which sends the bandwidth changes to upper layer protocols for the bandwidth threshold.

II. *Bandwidth threshold*, which sets the value of the threshold. If the actual bandwidth is smaller or equal, it is propagated, otherwise the nominal bandwidth is transmitted.

If we invert the configuration order of these 2 parameters, the threshold will not be set, since it requires the existence automatic propagation.

C. *Value dependences among parameters*

1. *Parameters of the same command are dependent*

EXAMPLE When specifying a classful *IP address*, the net mask must be consistent with the first two bits of the IP address. For instance, the B class IP address starts with the bits *10* and has the mask *255.255.0.0*.

2. *Parameters of different commands on the same device are dependent.*

EXAMPLE An access list is identified by its *number* parameter, which is referenced when the access list is attached to an interface. If we change this ID number in one place, we should change it likewise in the other, lest the functionality is lost. Parameters that reference each other may have the same or different names.

3. *Parameters on different devices are dependent*

EXAMPLE The connectivity between 2 devices requires the IP addresses of 2 interfaces directly connected to be on the same subnet.

D. ***Parameter Hierarchy and Service Composition*** In a configuration, there is a hierarchy of elements from simple to complex, namely from the parameters, going through several levels of aggregation, up to the network services. This grouping expresses the common goal and of the components and the dependences that exist among them.

1. *Grouping parameters under commands.* At the bottom, several parameters can be configured simultaneously using commands or statements, which group them based on logical links.

EXAMPLE An access list entry command specifies several packet parameters to be matched, such as: *source* and *destination addresses*, *direction* of the packet, *protocol*, *port*, whether it is *dynamic* or not, *timeout*, etc. Various dependences among some of these parameters have already been highlighted in the examples in the previous paragraphs § A and § B.1.

2. *Grouping commands under services.* The commands can be grouped as well, if they serve a common goal, i.e. a feature or a service. For instance, an access list is composed one or more access list entries, which are bound by the common access list number.
3. *Network service composition.* A network service can rely on simpler network services, according to a recursive, hierarchical model.

EXAMPLE. A Virtual Private Network (VPN) service requires: a network tunneling, e.g. through LSPs provided by an MPLS protocol, BGP connectivity (e.g. by means of neighbors), for the provider's backbone network, and IGP connectivity between the customer sites and the backbone, e.g. by means of RIP or OSPF protocols.

In complex services, such as BGP MPLS-based VPNs, VLANs, etc., there are multiple dependences at different hierarchical levels.

1.2 Dependences among network service components

Due to their complexity, the dependences can exist at different levels within network services, from parameters to sub-services.

1. *Parameter and command dependences* As already shown, the parameters and commands that compose the services can have their intrinsic, lower-level, dependences, which can be either independent or dependent on the device environment (software and hardware).

EXAMPLE The dependence C.1 is generic, whereas D.2 is command implementation-specific.

2. *Sub-service dependences.* At the top of the service hierarchical structure, there are higher-level dependences among the component sub-services. These dependences are dependent on the service- and network-level information, e.g. network topology, technology, protocols, etc., rather than on the devices, and span multiple equipments.

EXAMPLE If several customer sites use overlapping address spaces and are connected to more than one VPN, the corresponding PEs must ensure traffic isolation among the various VPNs by enforcing specific constraints on their forwarding tables.[6]

These properties are high-level and generic, and need to be transposed into concrete, lower-level properties, by adapting to concrete network and equipment environments, in order to be applicable to the configuration.

For instance, a VPN service can be materialized by a provider tunneling technology such as the Multi-protocol Label Switching (MPLS). The MPLS may run on an IP network and use the multi-protocol Border Gateway Protocol (MP-BGP) for VPN routing advertising among the edge routers and the MP-BGP may use the direct neighbors for configuration on the edge routers. We will explain these concepts and features in the following example.

1.2.1 MPLS VPN Service Example

A VPN is a private network constructed within the network of the service provider, which ensures the connectivity and privacy of customer's communications traffic.[7] For this purpose, the sites (which are contiguous parts of networks) of the customer network are connected to the provider's network through direct links between the interfaces of the devices that are at the edges of these networks, i.e. the Customer Edge device (CE) and the Provider Edge device (PE), as shown in Figure 1 (site 3 has been omitted from this representation, to save space).

The VPN service in the example is configured on the PE routers using the IOS[1] commands, without loss of solution's generality and validity since, as mentioned before, the generic service properties must be transposed into concrete properties by mapping the configuration components to specific CLIs, such as IOS, JUNOS,[2] etc. The provider specifies VPN routing information (by means of the VPN Routing and Forwarding tables – the *VRFs*) on the PE routers' interfaces that are connected to the corresponding CE routers of the customer's sites. (There are many implementations of the VPN service. Our example uses the BGP MPLS-based VPN.) Figure 2(a) presents some highlights from the configuration file of the *PE-1* router, which are explained in the following paragraphs. (A configuration file contains lists of commands, grouped by contexts, which customize the parameters of various configuration components, e.g. interfaces, protocols, security, etc. of network equipments, such as routers and switches.)

Commands 1-4 create and define the VRF *Blue*. This VRF is then assigned by command 8 to the appropriate interface that connects the PE router to the customer's network. The PE router must be directly connected to the CE router of the customer site and learn local VPN routes. Command 5 configures the interface to be used for this router's connectivity inside the provider network and command 6 assigns it an IP address.

Command 10 illustrates the configuration of the OSPF process in the VRF *Blue* and command 11 specifies the network directly connected to the router (and containing the interface IP address configured by command 9). Command 12 ensures that the OSPF process advertises the VPN routing information learned from across the provider's network by means of the BGP protocol, into the CE router.

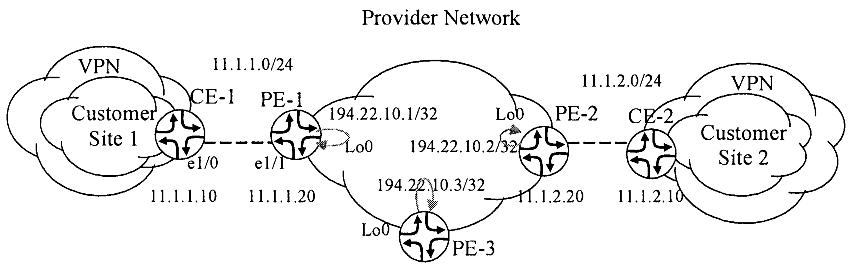


Figure 1. VPN example. Customer sites 1 and 2 are linked through CE routers to PE routers, which communicate over the service provider's network

The PE router exchanges customer and provider VPN routing information with other PE routers from the provider's network using the MP-BGP. Command 13 configures MP-BGP routing process by specifying the local autonomous system and commands 14-19 add the routing information necessary for the connection to other PE routers across the provider's network, i.e. autonomous system, interface type, number and IP address used by the remote PEs. Notice that we use the simplest method to configure the BGP process between PEs, namely the *direct neighbor configuration*.

The BGP needs also its own address family, *vpnv4*, which allows it to carry VPN-IPv4 in order to uniquely identify the addresses of different customers (commands 23, 25) and to advertise the extended community attribute (commands 24, 26).

1.2.2 VPN Configuration Properties

In this example, we can describe many properties, but we will restrict ourselves here to only two properties that the configuration must have with respect to the *neighbor* command and its relationships with other commands (from the same PE router and from other PE routers).

PROPERTY 1 The *address* of the *interface* of a PE router that is used by the BGP process for *PE connectivity* must be defined as BGP process *neighbor* in all of the other PE routers of the provider.

EXAMPLE In Figure 2(a), a *neighbor* with the address *194.22.10.2* is configured by commands 14-16. This corresponds to the IP address of the Loopback0 interface of router PE-2. Conversely, the *PE-1* router's *Loopback0* IP address, i.e. *194.22.10.1*, must be defined as a *neighbor* of the BGP processes of the other PE routers (PE-2, PE-3, etc.).

```
   . . .                                          Cmd
ip vrf Blue                                        1
   rd 100:1                                        2
   route-target import 100:10                      3
   route-target export 100:10                      4
   . . .
interface Loopback0                                5
   ip address 194.22.10.1 255.255.255.255          6
   . . .
interface Ethernet 1/1                             7
   ip vrf forwarding Blue                          8
   ip address 11.1.1.10 255.255.255.0              9
   . . .
router ospf 15 vrf Blue                           10
   network 11.0.0.0 0.255.255.255 area 0          11
   redistribute bgp 100 metric 1                  12
   . . .
router bgp 100                                    13
   . . .
   neighbor 194.22.10.2 remote-as 100             14
   neighbor 194.22.10.2 update-source Loopback0   15
   neighbor 194.22.10.2 activate                  16
   neighbor 194.22.10.3 remote-as 100             17
   neighbor 194.22.10.3 update-source Loopback0   18
   neighbor 194.22.10.3 activate                  19
   . . .
address-family ipv4 vrf Blue                      20
   redistribute ospf 15                           21
   . . .
address-family vpnv4                              22
   neighbor 194.22.10.2 activate                  23
   neighbor 194.22.10.2 send-community extended   24
   neighbor 194.22.10.3 activate                  25
   neighbor 194.22.10.3 send-community extended   26
   . . .
```

a

```
PE-1a:                                                 1
  . . .
  |- ip vrf: Blue                                      2
  |   |- rd: 100:1                                     3
  |   |- route-target                                  4
 . . .    |- import: 100:1                             5
  |       |- export: 100:1                             6
  |- interface                                         7
  |   |- Loopback: 0                                   8
  |   |   |- ip address                                9
 . . .        |- ip: 194.22.10.1                       10
  |   |       |- subnet mask: 255.255.255.255          11
  |   |- Ethernet: 1/1                                 12
  |   |   |-ip address                                 13
 . . .        |-ip: 11.1.1.10                          14
  |           |-subnet mask: 255.255.255.0             15
  |- router                                            16
  |   |-ospf: 15                                       17
  |   |   |-vrf: Blue                                  18
  |   |       |-network                                19
  |   |       |   |-area: 0                            20
 . . .        |   |-ip: 11.0.0.0                       21
      |       |   |-wildcard mask: 0.255.255.255       22
      |       |- redistribute:                         23
      |       |-bgp: 100                               24
      |           |-metric: 1                          25
      |- bgp: 100                                      26
      |   |-neighbor: 194.22.10.2                      27
      |   |   |-remote-as: 100                         28
      |   |   |-update-source: Loopback0               29
      |   |   |-activate: ~                            30
      |   |- neighbor: 194.22.10.3                     31
      |   |   |-remote-as: 100                         32
     . . .    |-update-source: Loopback0               33
          |   |-activate: ~                            34
          |-address-family                             35
              |- ipv4                                  36
              |   |-vrf: Blue                          37
     . . .        |-redistribute                       38
              |   |-ospf: 15                           39
              |-vpnv4                                  40
                  |-neighbor: 194.22.10.2              41
                  |   |-activate: ~                    42
                  |   |-send-community: extended       43
                  |-neighbor: 194.22.10.3              44
                      |-activate: ~                    45
                      |-send-community: extended       46
```

b

Figure 2. Selected commands that configure a VPN service in the configuration file of the provider edge router *PE-1* (a), and the corresponding MCM tree *PE-1a* that models them (b).

PROPERTY 2 The address family *vpnv4* must *activate* and *configure* all of the BGP *neighbors* for carrying only VPN IPv4 prefixes and advertising the extended community attribute.

EXAMPLE In Figure 2(a), the commands 23, 24 activate and advertise the extended community attribute of the *neighbor 194.22.10.2*, configured by the commands 14-16 under the BGP process. We have here an instance of a command whose various parameters are configured in two different contexts.

2. THE META-CLI MODEL

The Meta-CLI Model[8–10] abstracts the configuration information of the devices and network services into tree-like structures and the network services configuration dependences and constraints into rules which it uses to configure network services and to validate their consistence and integrity.

2.1 The MCM Trees and Nodes

The Meta-CLI Model develops the hierarchical architecture of the configuration properties and information into the *MCM tree* concept.

DEFINITION 1 The *MCM tree* is a tree that has its name in the root and the configuration contexts, the command names, and the command parameters of a device configuration in its nodes.

EXAMPLE In Figure 2(b), the *router* (command 13) is a command mode and the *address-family* (commands 20, 22) is its sub-mode. The commands, e.g. *ip address* (command 6), are appended as children or descendants (node 10) of the command modes, e.g. *interface* (command 5, node 7) and sub-modes to which they belong.

DEFINITION 2 An *MCM tree node* is a vertex of an MCM tree and represents a CLI configuration mode, command name or parameter.

The MCM tree node contains *intrinsic information,* such as the *data*, consisting of a *type* (e.g. "subnet mask") and a *value* sub-attributes (e.g. 255.255.255.255), a default value, possible values of the commands/parameters, node operations, etc. and *structural information*. The latter deals with the links and relationships between nodes, such as: child nodes and the *path*, which consists of the *data* of the ancestor nodes starting from the root.

DEFINITION 3 A *node type* represents a class or category of configuration modes, commands or parameters. The type of a node *N* is denoted by *N.type.*

DEFINITION 4 A *node value* represents the value of the command or parameter modeled by the node. The value of a node *N* is denoted by *N.value.* The type of a MCM tree node is unchangeable whereas the value may be changed.

DEFINITION 5 The *node data* represent the *type* and *value* of the node. The data of a node *N* are denoted by *N.data.*

EXAMPLE The type, value and data of node 41 in Figure 2(b) are: *N.type = neighbor, N.value = 194.22.10.2* and *N.data = neighbor:194.22.10.2*, respectively.

2.2 The Validation Rules

The Meta-CLI Model translates the CLI properties and combines them with intrinsic tree properties into validation rules.

DEFINITION 6 A *validation rule* is a condition (constraint) or combination of conditions that one or several MCM trees (and their components) must satisfy.

The validation rules abstract the CLI properties of the commands and configurations.

A few rule examples follow. A first group of simple rules indicate that a node must have a specific attribute.

- A node *N* has a given value *V*, i.e.: N.value = V (1a)
- A node *N* has a given data *D*, i.e.: N.data = D (1b)

EXAMPLE (1a) is *false*, if *N* is node 27 in Figure 2(b) and *V = 194.22.10.1.*

DEFINITION 7 A *node reference* is the purposeful equality of the value or data of two nodes. It represents the conceptual and functional *identity* of some information located in two (or more) nodes.

We may thus have the value reference or data reference:

- A node *N* references the value of node *P*, i.e.: N.value = P.value (2a)
- A node *N* references the type of node *P*, i.e.: N.data = P.data (2b)

The following group of such rules checks whether a sub-tree or a tree *S* contains a node *N* that has a given type *T*, value *V* or data *D*, respectively.

$\exists N \in S$, N.type = T (3a)

$\exists N \in S$, N.value = V (3b)

$\exists N \in S$, N.data = D (3c)

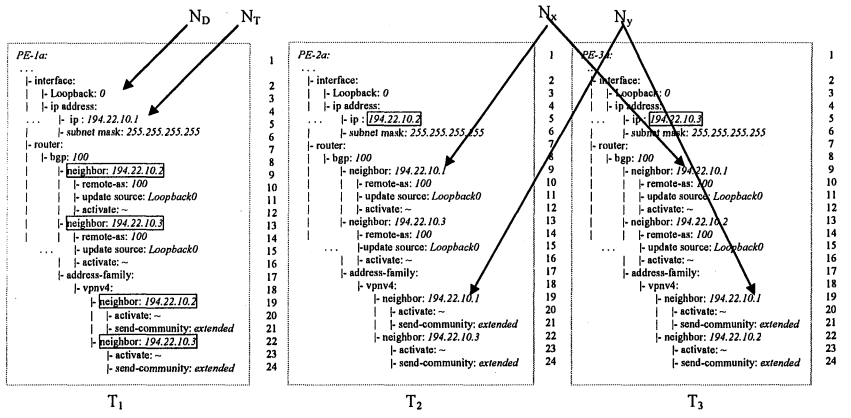
EXAMPLE (3a) is *true,* for *S* being the sub-tree of node 8 and *T = ip*.

Other simple rules are the following:

- A node has an ancestor with a given type, value or data. (4)
- Two (sub)trees S_1 and S_2 contain (at least) a node each, N_1 and N_2, respectively, having the same given value *V*, i.e.:
  $\exists N1, N2, N1 \in S1, N2 \in S2$, N1.value = N2.value (5a)
- Two (sub)trees S_1 and S_2 contain (at least) a node each, N_1 and N_2, respectively, having the same given data *D*, i.e.:
  $\exists N1\ N2, N1 \in S1, N2 \in S2$, N1.data = N2.data (5b)

There are also more complex rules. For instance, the following rule corresponds to the configuration properties 1 and 2, stated in section 2.

- Let *D* be some node data, *T* a node type and $\{T_i \mid i = 1, \ldots, n, (n > 1)\}$, a set of MCM (sub)trees. If a tree T_i has a node N_D of the given data *D*, and N_D has a descendant N_T of the given type *T*, then all of the other trees T_j, contain two nodes N_x and N_y of identical data such that their values are equal to the value of N_T. (6)



```
PE-1a:                                              1
...
|- interface:                                       2
|   |- Loopback: 0                                  3
|   |- ip address:                                  4
...     |- ip : 194.22.10.1                         5
|       |- subnet mask: 255.255.255.255             6
|- router:                                          7
|   |- bgp: 100                                     8
|       |- neighbor: 194.22.10.2                    9
|       |   |- remote-as: 100                       10
|       |   |- update source: Loopback0             11
|       |   |- activate: ~                          12
|       |- neighbor: 194.22.10.3                    13
|       |   |- remote-as: 100                       14
    ...     |- update source: Loopback0             15
        |   |- activate: ~                          16
        |- address-family:                          17
            |- vpnv4:                               18
                |- neighbor: 194.22.10.2            19
                |   |- activate: ~                  20
                |   |- send-community: extended     21
                |- neighbor: 194.22.10.3            22
                    |- activate: ~                  23
                    |- send-community: extended     24
```

```
PE-2a:                                              1
...
|- interface:                                       2
|   |- Loopback: 0                                  3
|   |- ip address:                                  4
...     |- ip : 194.22.10.2                         5
|       |- subnet mask: 255.255.255.255             6
|- router:                                          7
|   |- bgp: 100                                     8
|       |- neighbor: 194.22.10.1                    9
|       |   |- remote-as: 100                       10
|       |   |- update source: Loopback0             11
|       |   |- activate: ~                          12
|       |- neighbor: 194.22.10.3                    13
|           |- remote-as: 100                       14
    ...     |-update source: Loopback0              15
        |   |- activate: ~                          16
        |- address-family:                          17
            |- vpnv4:                               18
                |- neighbor: 194.22.10.1            19
                |   |- activate: ~                  20
                |   |- send-community: extended     21
                |- neighbor: 194.22.10.3            22
                    |- activate: ~                  23
                    |- send-community: extended     24
```

```
PE-3a:                                              1
...
|- interface:                                       2
|   |- Loopback: 0                                  3
|   |- ip address:                                  4
...     |- ip : 194.22.10.3                         5
|       |- subnet mask: 255.255.255.255             6
|- router:                                          7
|   |- bgp: 100                                     8
|       |- neighbor: 194.22.10.1                    9
|       |   |- remote-as: 100                       10
|       |   |- update source: Loopback0             11
|       |   |- activate: ~                          12
|       |- neighbor: 194.22.10.2                    13
|       |   |- remote-as: 100                       14
    ...     |- update source: Loopback0             15
        |   |- activate: ~                          16
        |- address-family:                          17
            |- vpnv4:                               18
                |- neighbor: 194.22.10.1            19
                |   |- activate: ~                  20
                |   |- send-community: extended     21
                |- neighbor: 194.22.10.2            22
                    |- activate: ~                  23
                    |- send-community: extended     24
```

Figure 3. Three MCM trees, *PE-1a*, *PE-2a* and *PE-3a*. The arrows indicate the nodes of the first instance of rule (6).

We may apply this rule to the MCM trees *PE-1a, PE-2a* and *PE-3a*, shown in Figure 3, for *D = Loopback:0* and *T = ip*. The arrows depict the rule instance that has $i = 1$ and $j = 2$ and 3. We see in the tree *PE-1a* that node 5 (corresponding to N_T), of type *ip* is a descendent of command 3 (corresponding to N_D) having the data *Loopback:0* and that the nodes 9 (corresponding to N_x) and 19 (corresponding to N_y), of *PE-2a* and *PE-3a*, respectively, reference the node 5 of *PE-1a*.

A validation algorithm compares the set of value of nodes 5 of *PE-2a* and *PE-3a* with the sets of values of the nodes 9, 13 of *PE-1a*. It also compares the set of data of nodes 9, 13 with the set of data of the nodes 19, 22. This is shown in Figure 3 by the boxes surrounding the respective leaves. Since these set equalities are true, the MCM trees are valid with respect to the first instance of the rule.

3. META-CLI *VALIDMAKER* MODULE FOR CONFIGURATION MANAGEMENT

The Meta-CLI *ValidMaker* module implements the Meta-CLI Model solution for the validation of the network configurations. It provides consistence and synchronization of the configurations achieved with network configuration management tools.

The configurations of the network equipments such as the routers, switches, etc., are provided by configuration management tools, based on the centralized information stored in a network management information base.[11] The configurations can be also changed in a CLI-based mode. The interactions between these heterogeneous modes of operation are managed by the *ValidMaker* module, based on the configuration information from the network equipments, the configuration tools and the network information model.The role of this module is to validate the device configurations in a dynamic and heterogeneous context. Since the changes were neither predicted by, nor included in, the configuration solution provided by the tools that enabled the deployment of the services, the tools cannot intervene in a coordinated fashion. The *ValidMaker* module intervenes to validate, maintain, restore and control the consistence of the network. It can be embedded in each service provisioning tool and ensure thus an error-free, consistent service provisioning.

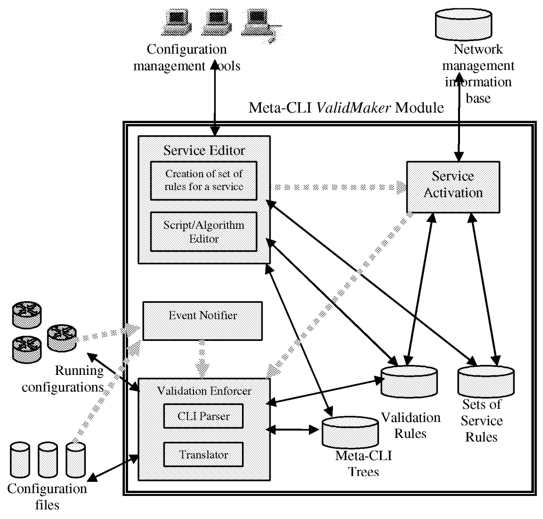


Figure 4. Conceptual View of the Meta-CLI ValidMaker.

The components of the *ValidMaker* module are presented in Figure 4. The *Service Editor* allows the service creator to generate a set of rules, algorithms and scripts, for each service, the *Service Activation* allows the network administrator to activate network services for the specific network equipments, the *Event Notification* monitors the status of the network and the *Validation Enforcer* enforces the rules on the equipment configurations.

4. CONCLUSIONS

The Meta-CLI Model offers multiple possibilities of utilization in various contexts of the configuration management process: service configuration and consistence validation, control of the consistence of data stored in management information bases, network device testing, etc.

This paper presents the application of the Meta-CLI Model to the validation of network service configuration consistence and integrity. The *ValidMaker* module creates validation rules, algorithms and strategies for all of the intermediate steps of the configuration process performed by the service provisioning tools. The validation rules are grouped in sets representing snapshots of the service configuration process and are therefore assigned to validate each configuration step performed by the provisioning tools. When the configuration is complete, the groups of validation rules, algorithms and scripts that are associated with the service remain on standby and intervene upon notification to restore the coherence of the system.

The *ValidMaker* validation solution provides consistence and synchronization of the configurations achieved with network configuration management tools. It may be used in a wide range of contexts, e.g. it can be embedded into the NetConf configuration management operations like *validate*, *commit*, etc.

REFERENCES

1. B. Hill, *Cisco: The Complete Reference* (Osborne, McGraw Hill, 2002).
2. T. Thomas II, R. Chowbay, D. Pavlichek, W. Downing III, L. Dwyer III, and J. Sonderegger, *Juniper Networks Reference Guide* (Addison-Wesley, 2002).
3. RFC 1155–1158, 1901–1908, 2263, 2273, 2573, 2574, http://www.ietf.org/rfc/rfcxxxx.txt.
4. M. Sloman, Policy Driven management for Distributed Systems, *Journal of Network and Systems Management,* Vol.2, No.4, 1994.
5. R. Enns, *NETCONF Configuration Protocol,* Internet-draft, http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-03.txt, June 2004.
6. R. Bush and T. G. Griffin, Integrity for Virtual Private Routed Networks *22nd Annual Joint Conf. of IEEE Computer & Comm. Society* (INFOCOM 2003), San Francisco, USA, 2003.
7. I. Pepelnjak and J. Guichard, *MPLS and VPN Architectures* (Cisco Press, 2001).
8. O. Cherkaoui and R. Deca, *Method and Apparatus for Using a Sequence of Commands Respecting at Least One Rule*, US Patent Application N° 60/458,364, March, 2003.
9. R. Deca, O. Cherkaoui, and D. Puche, A Validation Solution for Network Configuration, *Proceedings of the 2nd Annual Communications Networks and Services Research Conference* (CNSR 2004), Fredericton, NB, Canada, May, 2004.
10. S. Hallé, R. Deca, O. Cherkaoui, and R. Villemaire, Automated Verification of Service Configuration on Network Devices, Article accepted at the *Conference on Management of Multimedia Networks and Services,* (MMNS 2004), San Diego, USA, Oct. 2004.
11. O. Cherkaoui and M. Boukadoum, Managing Network Based VPN with Policies, Paper submitted to *The Telecommunications Journal*, 2003.