A Data Model for Management of Network Device Configuration Heterogeneity

Éric Lunaud Ngoupé, Sylvain Stoesel, Clément Parisot, Sylvain Hallé
Université du Québec à Chicoutimi, Canada
Petko Valtchev, Omar Cherkaoui
Université du Québec à Montréal, Canada
Pierre Boucher
Ericsson Canada

***Abstract*—The management of the configuration of network devices is a complex process, due to both the number of devices and parameters to take into consideration, and most importantly to the widely varying ways in which such parameters can be queried and modified on each device. Each equipment vendor provides its own command-line interface or management protocol where parameters are structured in a different way, and even multiple equipments from the same vendor may need to be interacted with differently. In this paper, we present a generic data model for configuration information of network devices that takes into account vendor and version heterogeneity. Ultimately, a configuration query engine will allow a user to pinpoint a specific, abstract configuration parameter, and be given the proper sequence of commands to query that parameter on a device of given vendor and operating system version.**

I. Introduction

Manufacturers of network devices did not facilitate the automation of network infrastructures configuration management. Each equipment vendor provides its own command line interface or management protocol where parameters are structured in a different way from other manufacturers [2], leaving users with a wider range of incompatible products. Furthermore, software developers for automatic configuration management rushed towards the development of applications oriented for the biggest and the most known manufacturers, which unfortunately led to a proliferation of software that does not support or take into account the equipment of all manufacturers.

Network devices heterogeneity can be divided into hardware and software heterogeneity. Hardware heterogeneity refers to the presence of devices with different capabilities, while software heterogeneity refers to the presence of different operating systems and applications that run on a network. The issue of heterogeneity usually results in interoperability issues, making the use of standards vital.

Interoperability between different areas of network management, heterogeneous systems and various management systems is one of the major requirements of today's complex business services. Another difficulty is the necessity of making configuration changes from time to time, causing a large number of dependencies between systems. We know that the systems do not operate in isolation in a network, so any changes to the configuration of a network service can cause complex and cascades changes at the dependencies of network services. The Sage Research institute found out that 40% of system downtime are due to operational errors and 44% of those errors are due to configuration errors [5].

In recent years, research in configuration management has experienced a rapid evolution. However, this growth has resulted in the emergence of different network management models, with each model conforming to a different set of standards. The aim now is to make these models interoperate in order to overcome the artificial barriers to the heterogeneity of network management devices. Organizations like IETF are trying to standardize the sector by creating models of description which are independent from manufacturers and equipment to be applied in a consistent and standardized way across multiple management domains and multiples type of devices: it is the case of PCIM [4]. Yet, although substantial progress has been made through the establishment of formats and specifications such as SNMP and SMI [6], the heterogeneity problem remains unsolved.

In this paper, we present a generic data model for configuration information of network devices that takes into account vendor and version heterogeneity. A configuration query engine allows a user to pinpoint a specific, abstract configuration parameter, and be given the proper sequence of commands to query that parameter on a device of given vendor and operating system version. The remainder of this article is structured as follows. Section II introduces the concept of configuration heterogeneity, discusses a formal model of device configurations by presenting a new model that can be used by more than one manufacturer to solve the problem of heterogeneity, and presents an implementation under development. Finally, Section III concludes our work.

II. Managing Network Devices

A *device* can be a hardware component such as a router, a switch, a wireless access point or gateway, i.e. physical or virtual hardware used for routing packets in a computer network. Devices have their own internal features and communication interfaces and usually vary depending on the manufacturer for physical devices, or on their version for virtual devices.

A. Configurations and Heterogeneity

The main characteristic of a device is that its default behaviour can be modified dynamically through configuration.

The *configuration* of a device corresponds to a set of parameters and associated values, recorded by the device. These

settings allow network administrators to customize the device, for example by setting the physical interfaces, by applying the rules of data redirection, by adapting the generic behaviour of the device to the network, or by modifying the behaviour of the device so that it fits with previous events.In configuration, editable elements are called *parameters*. Usually, networks devices are configured from the command line. On each device, getting information about their parameters or modifying them is possible by using precise commands.

In some occasions, the problem of interoperability is partially settled because of the existence of vendors that have been enticed to add an extra layer to enable interoperability of network equipment, as has been the case for Cisco and Nortel in the past. Similarly, the problem of using different protocols like SNMP and CMIP has already been resolved; then remains to find interoperability solutions that can occur within a single domain between the use of communication protocols and the development of models. Indeed, it is very possible that two different domains represent the same concept differently. We identify two main types of heterogeneity in the representation of configuration data.

1) Cross-vendor: In the absence of established standards, these controls differ from one vendor to another. The syntax of a command used to display information about a switch or router interface on a Cisco device will not be the same as a Juniper device. Documentations from the two vendors give network administrators the syntax to respect. Thus, configuring a particular device requires knowledge of the manufacturer (in this case, Cisco or Juniper) and the use of the correct syntax in order to properly execute the command.

Vendor	Syntax
Cisco	show interface [intfc]
Juniper	show interfaces [intfc] detail

2) Cross-version: The difference of syntaxes is not only due to vendors' competition. Sometimes, they can take place within the same vendor products, when the latest developed devices contain an updated version of the firmware. Therefore, syntax can vary from one version to another. A good example is provided by Cisco products where commands change as versions of its devices change. The syntax of the command *ip domain list* evolved between version 10 and version 12 of Cisco Operating System (IOS).

- In version 10.0, the syntax is: *ip domain-list name*
- In version 12.2, the syntax is: *ip domain name list* (without dashes)

Unfortunately, a purely syntactic translation model (i.e. a simple string pattern matching) cannot provide a solution; rather, a *semantic* translation is needed to make the concepts of these two areas directly correspond [3].

B. A Formal Model

A common problem to all the approaches mentioned previously is that they are all specific to a particular vendor or version of the device's operating system. As a matter of fact, even approaches aimed at furthering interoperability, such as SNMP and Netconf, actually provide very little in the way of abstracting such differences. In the case of SNMP, each equipment vendor is given its own numerical prefix: for example, all parameters for 3Com devices start with prefix 1.3.6.1.4.1.43, while all parameters for Cisco devices start with 1.3.6.1.4.1.9. Each vendor is then free to organize configuration information in whatever structure it wishes. Hence, even if the SNMP protocol provides a standardized way to query MIB data, the structure of this MIB and the actual queries to write depend on the device's manufacturer. The same can be said of Netconf; despite its RPC-style and the use of XML to send and receive data, again, each vendor is free to structure configuration information in whatever structure is deemed suitable. Therefore, such protocols can be summed up as vendor- (or version-) independent ways of manipulating vendor-dependent data structures. Even the tool closest to our goal, ValidMaker, can only level heterogeneity through a purely syntactical approach, and is hence appropriate to a relatively narrow spectrum of devices.

1) Ontologies: An ontology is an explicit specification of a conceptualization which is an abstract or simplified view of the world one wants to represent for a purpose [1]. Ontology constitutes in itself a representative model of data of a set of concepts in a domain, as well as relations between these concepts. The first objective of an ontology is to model knowledge in a given domain, which can be real or imaginary. It defines a set of primitive representatives that allow modelling a knowledge domain. Primitive representative are generally classes, attributes (or properties), and relations (between the members of the class). The ontology uses the specification language, which is considered as the central element on which it is based. Most of the languages of specifications are base or close to the first-order logic, and thus represent the knowledge in the form of assertion (subject, predicate, objet). These languages() are typically conceived to abstract from data structures and focus on semantics.

Ontologies can be seen as a level of abstraction of models of data, similar in the hierarchical and relational models, but intended to model the knowledge of individuals, their attributes and their relations to other individuals.

The proposed data model is a restricted form of ontology we call *generic model*. It is restricted, since our goal is to rebuild or convert the sequences of commands from a model contrary to ontology that is applied at first to model semantic terms specific to the supplier, and on the other hand, to assist in the automation of the correspondence between the terms in order to create a unified database of the application information such as the management of the various marks of equipment of network can be realized by single gateway. Full-fledged ontologies allow one to perform reasoning on the elements of the ontology, and not merely perform translation of concepts from a domain to another.

2) A Generic Data Model: Generally, a network device such as a switch or router has a set of configurable parameters, stored as a pair *name, values*. These parameters are organized hierarchically, according to the part of the device where they apply. We call these pairs *name, values* **concrete parameters**. They are valuated according to the configuration of a device. They represent a particular feature or a component of the device behavior. Note that the values of each parameter have a specific defined type that can be known in advance.

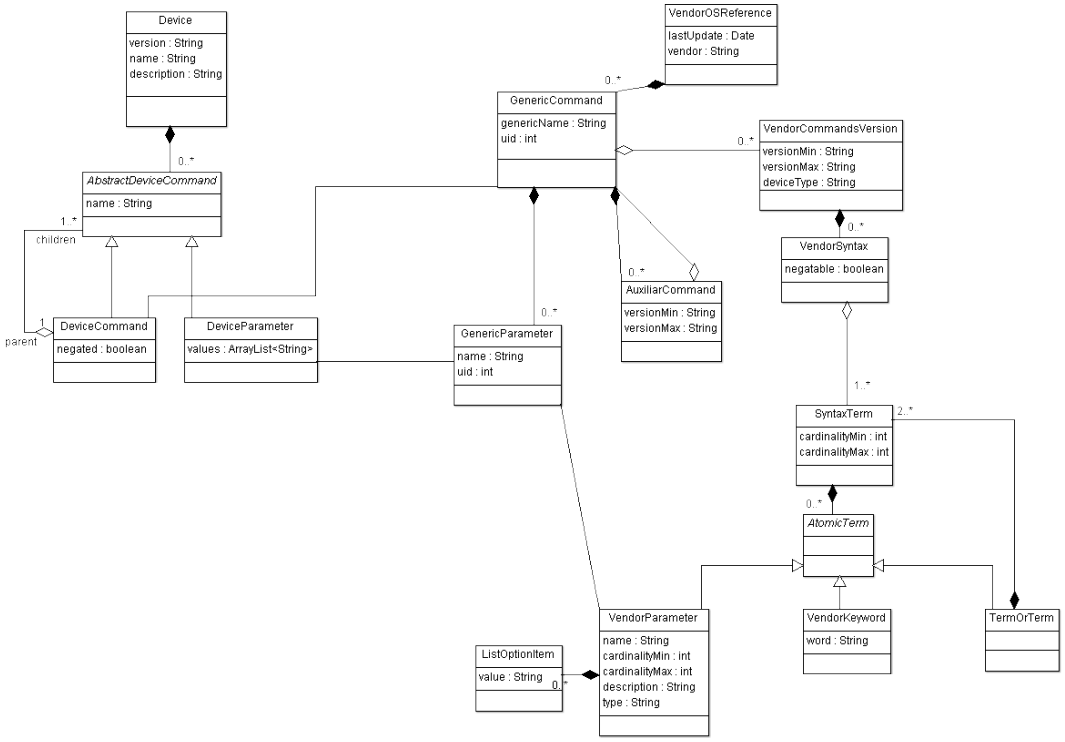

Figure 1: The UML data model for our abstract configuration structure.

From a switch to another, parameters have similar names and a similar hierarchy. Indeed, the internal structure of a switch can vary but its set of parameters remains close. Although the names of specific parameters may vary from one version to another, it is possible to combine two different settings versions with the same functionality. We could do the same grouping parameters devices from different vendors under the same parameter name. We then get a **generic parameter** that would have the same meaning for all devices. This generic parameter consists of a generic name that uniquely identifies all devices. It is associated with a set of concrete parameters that have their own specific name and value set. This data model is used to separate the syntactic and hierarchical aspects of the manufacturers of functional aspects of settings. Whatever the format of the input configuration, it is possible to identify with certainty a specific parameter and values with reference to the generic parameter.

To reflect the hierarchy of parameters of a device, we introduce the concept of commands. A command includes one or more parameters, a name and a set of underlying commands. The data model includes the same way the specific orders of devices in a **generic command**. This allows to manage changes that may affect hierarchical organization of a device to another.

C. Work in Progress

In this section, we describe an implementation in progress, which shall allow network administrators to store devices with their specific parameters, bind them with the generic parameters and find the corresponding syntax for every device.

1) Architecture: A device is described by a set of commands associated therewith. These commands have other underlying commands or parameters. These objects form a hierarchy of parameters and commands that give the device configuration. In fact, each *DeviceCommand* and *DeviceParameter* contains a link to a unique *GenericCommand* and *GenericParameter*. This link allows each device to be described generically. One can thus store an entire device in the form of a tree with nodes for *DeviceParameter* and their associated values and *DeviceCommand*. Each of these nodes will contain a reference to a generic external parameter.

Along with this structure, we have implemented a second solution we named *VendorOSReference*. It can store all the syntaxes used and the parameters required by a vendor for different versions of its OS. We thus find the *VendorParameter* which is the concrete setting device. This parameter is included in a set of other words and form syntax: *VendorSyntax*. Among these terms are found keywords and strings necessary for the proper understanding of the syntax. Optional terms are taken into account in the solution via *TermOrTerm*. We can define several versions of commands by declaring several *VendorCommandsVersion* nodes which regroup the different syntaxes needed to run this command, for each version provided by the vendor. Each node has a reference to the same *GenericCommand* that uniquely identifies it.

The relationship between concrete and generic parameters is done by the chosen data structure. To perform processing on these data, we developed a **Python** data structure based on the **Meta-CLI format**. The implemented program loads the devices data and OS reference from the XML files. It is

then possible to perform processing on the device by browsing nodes as Python objects.

2) Example Usage: To better understand our solution, we'll show an example as a whole. For this example we chose to work with Cisco switches (see Figure 2). We can extract the configuration file via CLI. We assume that the verification form is NFD (Normal Form disjunctive) format. Originally, this Cisco router was configured by CLI commands of Cisco IOS. We present the abstract version of these commands, which characterize the configuration of the device (a).

The abstract device previously written in Meta-CLI (XML) follows a documentation: the reference of the vendor. This reference (see figure 2b) declares all commands and parameters used by an abstract device, these abstract commands and parameters are related to physical commands and parameters of the real device (those provided by the vendor).

3) Construction of the Reference: In order to associate one generic parameter to each concrete parameter, it is necessary to first construct a reference for each operating system command. This will include a *uid* for each parameter. An excerpt of the reference file containing some of the parameters for Cisco's IOS can be found in Figure 2b. We can see there that they have built a node by Cisco command and one node by parameter.

The syntax of the commands is stored in the form of an ordered list of keywords and parameters. Some commands can be negated, as is the case of the *IP address*. This means that it is possible in the Cisco syntax to write *no ip address* corresponding to the lack or absence of IP address on the interface. We see that only one single syntax was defined by command but that it is possible to define several if this one comes to change. There is normally a unique reference that groups the syntax of all the manufacturers and all their versions.

III. Conclusion

In this paper, we have shown how a generic data model can accommodate configuration information for network devices in a vendor-independent fashion. By mapping concrete instances of parameters to generic nodes in a tree structure, querying configuration values for various devices merely amounts to pointing the appropriate parameter in that structure. The mapping between generic and concrete parameters then makes it possible to recreate the command or the sequence of commands required to retrieve the parameter value for a given device. In time, such a generic approach may alleviate interoperability issues in the management of heterogeneous networks by easing out discrepancies between the command line syntax and structure of various devices.

```
<Device version="12" name="Pomerol">
  <DeviceCommand negated="false" name="version" ref_cmd
      ="1">
    <DeviceParameter name="num_version" ref_param="11">
      <Value>12</Value>
    </DeviceParameter>
  </DeviceCommand>

  <DeviceCommand negated="false" name="hostname"
      ref_cmd="2">
    <DeviceParameter name="name" ref_param="12">
      <Value>Pomerol</Value>
    </DeviceParameter>
  </DeviceCommand>

  <DeviceCommand negated="false" name="interface"
      ref_cmd="3">
    <DeviceParameter name="name" ref_param="13">
      <Value>Loopback0</Value>
    </DeviceParameter>
    <DeviceParameter name="option" ref_param="14"/>
    <DeviceCommand negated="false" name="ip address"
        ref_cmd="4">
      <DeviceParameter name="ip" ref_param="15">
        <Value>10.10.10.3</Value>
      </DeviceParameter>
      <DeviceParameter name="mask" ref_param="16">
        <Value>255.255.255.0</Value>
      </DeviceParameter>
      <DeviceParameter name="scndry" ref_param="17"/>
    </DeviceCommand>
  </DeviceCommand>
</Device>
```

(a) Meta-CLI file

```
<VendorOSReference lastUpdate="09/12/2013" vendor="CISCO">
 <GenericCommand genericName="ip address" uid="4">
  <GenericParameter name="ip" uid="15"/>
  <GenericParameter name="mask" uid="16"/>
  <GenericParameter name="secondary" uid="17"/>
  <VendorCommandsVersion versionMin="all" versionMax="all"
      deviceType="all">
   <VendorSyntax negatable="true">
    <SyntaxTerm cardinalityMin="1" cardinalityMax="1">
     <VendorKeyword word="ip address"/>
     <VendorParameter name="ip" guid="15"/>
     <VendorParameter name="mask" guid="16"/>
     <VendorParameter name="secondary" guid="17"/>
    </SyntaxTerm>
   </VendorSyntax>
  </VendorCommandsVersion>
  <AuxiliarCommand versionMin="all" versionMax="all" guid="3"/>
 </GenericCommand>
</VendorOSReference>
```

(b) Cisco Vendor Reference

Figure 2: Device example: Cisco router

References

[1] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquisition*, 5(2):199–220, 1993.

[2] P. Kalyanasundaram and A. S. Sethi. Interoperabifity issues in heterogeneous network. *Journal of Network and Systems Management*, pages 169–193, 1994.

[3] J. E. López De Vergara, V. A. Villagrá, and J. Berrocal. Semantic management: advantages of using an ontology-based management information meta-model. In *Proceedings of the HP Openview University Association Ninth Plenary Workshop (HP-OVUA'2002), distributed videoconference*, pages 11–13, 2002.

[4] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy core information model (RFC 3060), 2001.

[5] D. Thomas and J. Wouter. High-level system configuration. *2D2 edition:1 location:Cambridge, UK date:12-13 May 2008*, 2008.

[6] A. K. Y. Wong, P. Ray, N. Parameswaran, and J. Strassner. Ontology mapping for the interoperability problem in network management. *IEEE Journal on Selected Areas in Communications*, 23(10):2058–2068, 2005.