Modelling the Temporal Aspects of Network Configurations

Sylvain Hallé[1], Rudy Deca[1], Omar Cherkaoui[1], Roger Villemaire[1] and Daniel Puche[2]

[1]Université du Québec à Montréal, Montréal, Canada
[2]Cisco Systems Inc., Montréal, Canada

Abstract

One of the main issues with the existing management configuration is the absence of a transactional model, which should allow the network configuration data to retain their integrity and consistence during the configuration process. In this paper, we propose a mathematical framework based on lattice theory allowing the structuring of configuration operations leading to the concept of component and validation checkpoint, and present polynomial-time algorithms for studying these structures. We will illustrate the model by an example of two examples of configuration operations: the deployment of a VLAN service through SNMP and the deployment of a VPN service through the Netconf protocol.

1 Introduction

Among other network management functions, configuration management is still mainly accomplished by proprietary means, be it Command Line Interface (CLI), JunOS or TL1. Recent alternatives like SNMPConf, COPS and Netconf have not yet succeeded in bringing a standard configuration solution. This situation is due to numerous causes: security (SNMPv3), absence of an adequate configuration information model, proprietary equipment instrumentation semantics. Another overlooked factor is the absence of a simple transactional model between agents and manag-

ers allowing for the association between management protocol operations (SNMP, COPS and Netconf) and management information.

When configuring a network service that involves multiple equipments, there is an important temporal aspect of complexity of the network service configuration. There are many sequences of configuration commands or operations that must be performed on multiple network elements or equipments, temporal dependencies among these sequences, semantic constraints among their parameters. Moreover, specific groups of commands and parameters belong to the same service or sub-service and must thus be performed together, in an atomic way, even though they affect multiple components or network elements form different network devices. The atomic character of the configuration operations involving a number of parameters is relevant both at device and at network levels.

In this paper, we propose a mathematical framework based on lattice theory allowing the structuring of configuration operations in terms of configuration dependencies. The concepts of component and milestone that we define in terms of paths in the lattice structure help us to simplify the analysis of possible solution paths and provide us with a sound criterion for dividing the deployment of a service into natural macro-steps that serve as validation checkpoints.We will illustrate the model by two examples of configuration operations: the deployment of a VLAN service through SNMP and the deployment of a VPN service through the Netconf protocol, and show preliminary results for validation checkpoints for the latter of these cases.

In section 2, we show by two examples why current management approaches are inedaquate for dealing with the sequential aspect of network configuration. In section 3, the lattice-based mathematical framework for modelling temporal constraints is detailed, and polynomial-time algorithms for studying the resulting structures are presented. Section 4 shows possible applications of this framework and preliminary results obtained for the case of simple Virtual Private Networks, while section 5 concludes and indicates further directions of work.

2 Motivation and Related Work

The deployment of a service over a network basically consists in altering the configuration of one or many equipments to implement the desired functionalities. We can presuppose without loss of generality that all properties of a given configuration are described by attribute-value pairs hierarchically organised in a tree structure [13].

Possible alterations to the configuration typically include deleting or adding new parameters to the configuration of a device, or changing the value of existing parameters. In most cases, the parameters involved in such modifications are both syntactically and semantically interdependent. For instance, the value of some parameter might be required to depend in a precise way on the value of another parameter; the simplest example of such dependency is the fact that an IP address must match the subnet mask that comes with it. More complex dependencies might constrain the existence of a parameter to the existence of another. Recent works have shown how such dependencies can be automatically checked by logical tools on a given configuration snapshot [13].

However, the situation becomes more complex when one wants to actually *deploy* a service from scratch. In addition to constraints on the values of parameters, the dependencies may also impose that the modifications be performed in a specific order. When done in an uncoordinated way, changing, adding or removing components or data that implement network services can bring the network in an inconsistent or undefined state. This fact becomes acutely true in the case where operations must be distributed on multiple network elements, as they cannot be modified all at once. Moreover, while a single inconsistent device can ultimately be restarted when all else fails, there is no such "restart" option when an entire network configuration becomes inconsistent.

However, one of the main issues with the existing management paradigms is the absence of a *transactional model*, which should allow the network configuration data to retain their integrity and consistence during the configuration process.

The network community has proposed different approaches for ensuring the consistence and integrity of the network configuration during the management of network services. The policy-based management using the Ponder language [6] incorporates OCL (Object Constraint Language) [19] to express the dependencies among configuration parameters. Other approaches, based on ontologies [17, 18], use the Protégé Axiom Language and Toolset (PAL) [5] for expressing configuration parameter constraints and queries. Many other constraint languages and tools are also available and can be used to express configuration parameter dependences, such as the Alloy language [16] and the constraint analyser based on it, Alcoa [15]. However, these approaches use constraint languages borrowed from other domains, designed for other purposes. Therefore, they are not adequate to the specifics of network configuration, and they do not tackle the transactional aspect of network configuration.

We will study two examples of configuration management using different paradigms, and stress their weaknesses in this regard. Based on this

evidence, we will show what the transactional model can accomplish and what its benefits are in the area of network configuration.

2.1 VLAN Configuration with SNMP

The deployment of a Virtual Local Area Network (VLAN) [3, 11] on a network involves a number of configuration operations falling into four categories:

- specification of the Virtual Trunk Protocol (VTP) domain and operation mode
- VLAN creation
- port allocation
- trunk creation

However, these operations cannot be performed in any order. Some ordering constraints are imposed.

Clearly, the operations belonging to the first group (VTP) are preparation operations on which the operations of the second group (VLAN creation) rely. In the same fashion, the port allocation cannot be done before the VLAN has been created (for the sake of simplicity, we exclude here the case where the port is reserved in advance or dynamically allocated from a pool). This leads to the formulation of a first set of two temporal constraints:

Temporal Constraint 1 All VTP operations must have been done before any VLAN creation parameter is added to the configuration.

Temporal Constraint 2 All VLAN creation parameters must have been added to the configuration before any port allocation or trunk creation parameters are added.

These two constraints entail that the VLAN be created in an *atomic* way: the name, number and other parameters must be specified together and the editing must be done by one manager at a time. If the configuration were to be modified by means of the command line interface, this atomic property would be achieved by having both number and name parameters mandatory within the same command (for example, the Cisco IOS command vlan <number><name>). A similar reasoning can be done for the other modification operations, leading to more temporal constraints.

However, despite these temporal constraints, the SNMP paradigm [2] allows the parameters to be configured independently and has no semantics for the configuration operations. It does not have a transactional model and thus allows inconsistent evolution of the network configuration. The way SNMP ensures atomicity is by providing an editing buffer for VLAN creation (the `vtpVlanEditingTable`) within the VLAN Management Information Base (MIB) [3]. Only one manager at a time is allowed to own and edit this buffer.

This example illustrates several facts. First, the temporal constraints impose that some of the VLAN parameters be grouped; SNMP does not elegantly enforce this and rather uses an ad-hoc editing buffer mechanism for this purpose.

Second, there are two levels of validation: the first level makes sure that each operation has been correctly made by confirming that the apply buffer operation has succeeded; the second level validates the overall operation and checks whether the VLAN has actually been created.

2.2 Example 2: VPN Configuration with Netconf

In this section we analyse the problems encountered by the Netconf protocol, when dealing with network services that involve multiple equipments and introduce a transactional model that solves these problems.

We illustrate this with an example of an MPLS Virtual Private Network (VPN) service deployment [20]. A VPN is a private network constructed within a public network such as a service provider's network. A customer might have several sites, which are contiguous parts of the network, dispersed throughout the Internet and would like to link them together by a protected communication. The VPN ensures the connectivity and privacy of the customer's communications between sites.

Such a service consists of multiple configuration operations that involve setting the routing tables and the VPN forwarding tables, setting the MPLS, BGP and IGP connectivity on multiple equipments having various roles, such as the customer edge (CE), provider edge (PE) and provider core (PC) routers. In total, a minimum of about 30 parameters must be added or changed in each device involved in the deployment of the VPN. As an example, Figure 1 shows two leaf nodes that must be added, each in its own position, to the configuration tree of a PE router.

Leaf node A is one of the configuration parameters that contributes to the creation of the VPN routing and forwarding tables on the PEs of the service provider. It cannot be added to the configuration of a PE router before the corresponding interface has been configured, which entails, among

other things, the addition of leaf node B. Therefore, one can extract a temporal constraint from this relation:

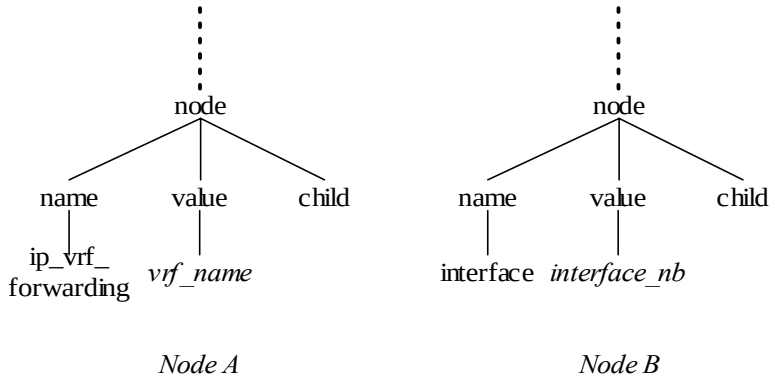

Fig. 1. Two configuration nodes that must be added for deploying a VPN

Temporal Constraint 3 *The node* `ip_vrf_forwarding` *cannot be added to a configuration tree before node* `interface/number` *has been set.*

Similar dependencies can be extracted for many other pairs of nodes among the 30 parameters involved, based on

- the semantic dependencies among the various components and parameters of the configuration;
- the spatial distribution of the configuration components and parameters;
- the choices of topology and technologies (protocols, device roles and types, vendor software, etc.).

These interdependencies imply a logically simultaneous configuration of the respective parameters on all these equipments. Since these equipments are spatially distributed and configuration operations can only be performed sequentially, this goal can only be achieved by "synchronizing" the configurations on different equipments by carefully setting up *validation points* during the configuration procedure.

The Netconf protocol [12] defines a simple mechanism for network device management. However, its transactional model, which includes a Validation capability, is device-centered, and does not provide a mechanism to ensure the consistence of the configuration that involves correlated configuration steps on multiple devices.

Netconf provides two phases of a successful configuration transaction during a service configuration procedure: preparation and commitment.

During preparation, the configurations are retrieved from the network devices. When all the configurations have been retrieved, the edition starts at service level. The validation at this stage ensures that the network configuration is consistent before the proposed modifications required by the service. To ensure the integrity of the configuration edition, the device configurations are locked, edited and subsequently unlocked. When the service edition has been successfully accomplished, the commitment starts. The validation at this stage ensures that the network configuration remains consistent after the respective modification of the network configurations.

Therefore, taken as is, Netconf does not provide any indication as to where and what to validate.

The previous examples have shown that many configuration operations must be done in a specific sequence, others must be performed together notwithstanding the order and others are mutually exclusive. Therefore, we need a clear temporal representation of the operations to be performed, which will describe all the temporal dependencies, indicate the possible procedural order of operation for various groups of configuration parameters on various devices and indicate the optimal temporal order and distribution of these operations.

3 A Theoretical Model

In this section, we present a theoretical study of the temporal issues described in section 2 by providing a theoretical model of the situation.

3.1 States and Transitions

Let S be a set of "states" representing a unit situation at a given time. In the case of network configuration, states are labelled trees described in section 2.

We call *transition* from a state s_1 to a state s_2 the structural modifications that transform s_1 into s_2. Formally, transitions can be defined as a subset of tuples $T \subseteq S \times S$; there exists a transition from s_1 to s_2 if and only if $(s_1, s_2) \in T$. The tuple (S, T) forms a directed graph G that we call a *transition diagram*.

In the case of the labelled trees we use for modelling device configurations, structural modifications are limited to addition of a labelled node to a leaf, or deletion of a leaf node in the tree. These modifications intuitively refer to addition, deletion or modification of a parameter in the configura-

tion of a device. Therefore, it is possible that no transition exists in either way between two given states: this explains why T is only a subset of all possible pairs of states.

A *path* is a finite sequence of states $<s_1, ..., s_n>$ such that, for any s_i, s_{i+1}, there exists a $t \in T$ such that $t = (s_i,s_{i+1})$. The *distance* between two states, noted $\Delta(s_1,s_2)$ is the length of the shortest directed path linking them.

The configuration problem of the previous section becomes in this system the study of all paths that start from a given configuration, s_s, and end at a target configuration s_t. In addition, one might want to find the shortest of such paths.

However, this system, taken as is, is too general for any practical use. In particular, we must make sure that only solutions that progress towards the target are possible.We hence use path constraints to limit our study to sequences of states that have a meaning and are not degenerate.

A solution is to remove all tuples $(s_1,s_2) \in T$ such that $\Delta(s_1,s_t) < \Delta(s_2, s_t)$.

This condition makes sure that the parameters that are actually added are part of the solution, but not of the start state, and that parameters that are removed are part of the start state, but not part of the solution. Any other modification is out of the way of an acceptable solution. This distance restriction also has for effect of removing any loops in the paths.

3.2 Temporal Constraints

Now that G has been trimmed of any nonsensical states and paths, we can add further restrictions by imposing on the remaining transitions the semantic constraints related to the situation we are trying to model.

For example, in order to respect Temporal Constraint 3 in the case of the VPN deployment exposed in section 2.2, we must remove all transitions that lead to states where node A of Figure 1 is added while node B is not present.

More semantic constraints can be added to further trim the state graph from unwanted states and transitions. The remaining paths satisfy to all defined constraints. Intuitively, these so-called *acceptable* paths can be seen as a semantically desirable candidate solution for transforming the start state into the target final state.

3.3 Structuring Operations

In the minimal VPN example described previously, containing only two provider edge and two customer edge routers, the resulting state graph is

composed of over 15,000 states spanning a proportional number of paths. These figures suggest that raw state graphs are far from being meaningful and manageable by hand. However, it is possible to simplify further this graph by studying patterns that can be found in it.

First, we can observe that the remaining graph $G = (S', T')$, induces a partial ordering $\sqsubseteq$ in S' defined in the following way:

$$x \sqsubseteq y \iff (x, y) \in T$$

The tuple $L = (S, \sqsubseteq)$ forms a bounded lattice [8]; the graph G can also be seen as the Hasse diagram of L. This lattice of possible states and transitions on states can then be studied for interesting properties. It is from these properties that a global procedure describing legal transformations to the configurations will be deduced.

Components

The first step is to recognize the presence of *components*, i.e. of closed groups of interweaved actions. For instance, the parameters that contribute to the creation of the VPN routing and forwarding tables on the PEs of the service provider are bound by constraints 1 and 2. They impose that all VRF actions on a router be done before passing on to another router, and that all VRF configuring must be done before going on to another aspect of the configuration.

Therefore, all actions of VRF configuration on a single router may be done in any order, but must all be done before doing anything else. Such set of n actions thus forms a component and appears as a Boolean n-dimensional cube in the Hasse diagram of the lattice, as shown in Figure 2. The left structure shows a component where three actions, α, β and γ, can be performed in any order, resulting in 6 different paths. The right structure shows a similar Boolean cube for 4 different actions.

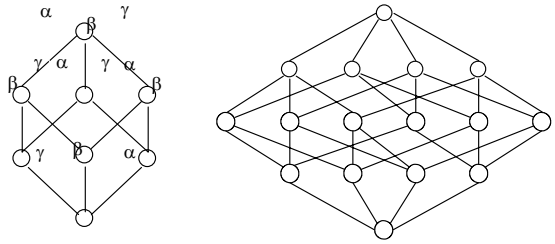

Fig. 2. A closed set of interchangeable operations forms a component

These components act as a “capsule” of operations that must be performed atomically (i.e., that must not be mixed with any other operation).

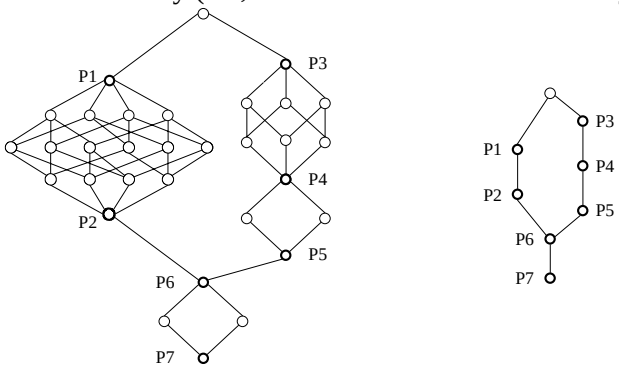

Fig. 3. A complete state graph and its associated reduced state graph

We see that identifying components is an important tool to reduce the complexity of the state graph. Each component has a unique start and end point, and can therefore be assimilated to the single edge linking these two points.

Identification of such components leads to the construction of a reduced state graph where some edges no longer represent a single transition, but rather whole sets of transitions (Boolean cubes) that can be performed in arbitrary order. Figure 3 shows how a reduced state graph can be obtained from a state graph. One can identify 2, 3 and 4-dimensional Boolean cubes that are linked together. For example, points P1 and P2 are the endpoints of a component: all states between P1 and P2 have no contact with any other state. Since these cubes represent components made of swappable actions, they can be identified as such and be identified with their endpoints to form a reduced state graph, as shown on the right part of the picture.

Milestones

The notion of component naturally leads to that of a *milestone*. A milestone is an element $x \in S$ such that for all $y \in S$, either $x \sqsubseteq y$ or $y \sqsubseteq x$. For example, in Figure 3, states P0, P6 and P7 are milestones.

Milestones can be thought of as unavoidable steps in the path from start to solution, since all acceptable paths must eventually pass by those points, in the order they appear. Therefore, milestones are good candidates to divide the modelled process into natural macro-steps of which they are the boundaries. In the case of Figure 3, two macro-steps can be identified: the transition from the start state P0 to state P6, and the transition from P6 to P7. The word “natural” is used here, since these milestones emerge from

the set of temporal constraints imposed on the lattice. Different temporal constraints generally lead to different milestones.

The concept of milestone can also be applied to any sublattice of L. In particular, inside each macro-step, there can be local milestones that may be viewedas natural sub-steps. The process can be recursively repeated and yield a natural, hierarchical decomposition of the whole process into nested natural blocks of swappable operations. Hence, states P1-P5 are sub-milestones, or milestones of order 2.

3.4 Computing Components and Milestones

In order to be efficient, the structures defined in the previous section must be able to be found easily and automatically by means of algorithms. In this section, we provide algorithms for finding milestones and for identifying components, and give an overview of their complexity.

Finding Milestones

The first algorithm we present is aimed at finding milestones in a lattice. Its principle is easy: for a state s, we can mark all states reachable from s by a (depth-first or breadth-first) graph traversal starting at s. Similarly, the set of states from which we can reach s can be computed and marked by a traversal of the transpose graph (going "up" instead of "down"). The chosen vertex is a milestone if and only if all vertices of the lattice are marked at the end of the algorithm.

A brief analysis of this algorithm shows that determining whether a given vertex is a milestone takes no more execution steps than the total number n of vertices in the lattice. Therefore, for finding all milestones, it suffices to repeat the algorithm starting at each vertex; the resulting complexity is therefore in $O(n^2)$.

Finding Components

In the same way as the definition of milestones is linked to that of components, the algorithm for finding components relies on the milestone-finding algorithm. We proceed to the same reachability analysis than in the previous section, except that instead of merely marking a vertex as visited, we explicitly state from which original vertex it was reached. At the end of the process, each node has a list of the nodes from which it is reachable, either forwards or backwards. A vertex m is a milestone if all the vertices in L have m in their list.

Then, for each milestone m_1 and its immediate milestone successor m_2, we consider the subgraph of all points between m_1 and m_2. This subgraph is divided into a number of disjoint sublattices $L_1, L_2, \ldots, L_n$. If one of these L_i has no milestone (which can be easily obtained by analysing the lists a second time), then the sublattice $L_i \cup \{m_1, m_2\}$ is a component.

As we can see, the overall complexity of this procedure depends on the maximum level of nesting where milestones can be found. However, a rough worst case can be calculated by supposing there can be no more nesting levels than there are elements in L. At the first step, it takes $O(n^2)$ operations to find the milestones of the first level. It again takes a time proportional to the square of their size to find all sub-milestones found in these sublattices. However, all nested sublattices found after removing the first-order milestones are disjoint. Therefore, the total time needed to find all second order sub-milestones is again in $O(n^2)$, where n is still the total size of L. Since the nesting level of any component is at most n, the total number of steps required is in $O(n^3)$.

4 Applications

The main advantage of the analysis of the lattice that arises from temporal constraints is that it induces a way of synthesising a protocol for the implementation of a service. By placing validation points at milestones, we ensure such checkpoints are optimally placed in semantically sound locations throughout the deployment process. Since these checkpoints reflect the structure imposed by the temporal constraints, they also make optimal points to roll back in case a failure occurs.

We have succeeded in analysing the deployment of a Virtual Private Network for the basic case of four routers and identified six main milestones. This is helpful in practice. For instance, in an existing tool called NetconfMaker [1], the user must manually set validation point in order to obtain a transactional model on top of the Netconf protocol. Due to the large number of possible solution, this is not an easy task. However, by using the approach presented in this paper, it is possible to feed NetconfMaker with scripts enabling it to proceed automatically to the discovery of these validation points.

The granularity of the configuration components and validation operations depends on how tightly the semantic dependences are coupled within the components and the complexity of these components. For instance, in the case of the VPN example, the BGP component can be split into two subcomponents: the first dealing with the creation of the BGP process and

the second with the neighbour information configuration. Another component refers to the mutual redistribution of the routing information between the IGP, the static routing used, or the connectivity between PE-CEs and the BGP process. If we take into account the initial underlying subservices (establishing the connectivity between PE-CEs, between PE-PCs and MLPS), we obtain six components, to which we add the initial constraints, obtained form the customer and the service provider choices.

One aspect of establishing validation points takes into account the hierarchy existing among the configuration transactions. Establishing the network-level validation points ensures the consistence and integrity of the configuration transactions that involve multiple equipments, roles and configuration parameters.

Moreover, the knowledge of milestones and components for a given service allows for the creation of more structured Management Information Bases (MIBs) and Policy Information Bases (PIBs), where the access mechanisms to configuration parameters could be designed according to the temporal dependencies discovered.

5 Conclusion

In this paper, we have shown by examples that configuration parameters in network devices are subject to syntactical and semantic dependencies which, when deploying a network service, may impose that some of the configuration operations be done in a specific order. We also explained how a mathematical framework using lattice theory can model these ordering constraints. The concepts of components and milestones, defined in terms of paths in the lattice structure, help us to simplify the analysis of possible solution paths and provide us with a sound criterion for dividing the deployment of a service into natural macro-steps that serve as validation checkpoints.

In particular, a deeper study of the implementation of an MPLS VPN in a simple case was found to be divided into six ordered main natural components whose internal configuration operations are mutually swappable. These results are in accordance with the intuitive vision of the deployment of this service.

Further work on this concept can lead to a thorough study of the deployment of a number of network services that could allow us to suggest the location of optimal validation points.

References

1. Cherkaoui O, Bétouret F, Deca R (2004) On the Transactional Issues of the Netconf Protocol. Université du Québec à Montréal, unpublished report.
2. Case J, Fedor M, Schoffstall M, Davin J (1990) Simple Network Management Protocol, STD 15. RFC 1157
3. Cisco SNMP Object Navigator. http://tools.cisco.com/Support/SNMP/
4. Clarke EM, Grumberg O, Peled DA (2000) Model Checking. MIT Press, Cambridge
5. Crubézy M (2002) The Protégé Axiom Language and Toolset ("PAL"). Protégé Project, Stanford University http://protege.stanford.edu/
6. Daminaou N, Dulay N, Lupu E, Sloman M (2001) The Ponder policy Specification Language. In Sloman M, Lobo J, Lupu EC. (eds) Policy'2001, Springer, Berlin Heidelberg New York, pp 29–31
7. D'Antonio S, D'Arienzo M, Pescapè A, Ventre G (2004) An Architecture for Automatic Configuration of Integrated Networks. In NOMS 2004
8. Davey BA, Priestley HA (1990) Introduction to Lattices and Order, Cambridge University Press, Cambridge
9. Deca R, Cherkaoui O, Puche D (2004) A Validation Solution for Network Configuration. In CNSR 2004
10. Deca R, Cherkaoui O, Puche D (2004) Configuration Model for Network Management. In Gaiti D, Galmes S, Puigjaner R (eds) NetCon 2004
11. Draft Standard for Virtual Bridge Local Area Networks, IEEE Draft P802.1Q/D1, May 16, 1997
12. Enns R (2004) Netconf Configuration Protocol. Internet draft, June 2004. http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-03.txt
13. Hallé S, Deca R, Cherkaoui O, Villemaire R (2004) Automated Verification of Service Configuration on Network Devices. In Vicente J, Hutchison D (eds) MMNS 2004, Springer, Berlin Heidelberg New York, LNCS 3271, pp 176-188
14. Hallé S, Deca R, Cherkaoui O, Villemaire R, Puche D (2004) A Formal Validation Model for the Netconf Protocol. In Sahai, A, Wu F (eds) DSOM 2004, Springer, Berlin Heidelberg New York, LNCS 3278, pp 147-158
15. Jackson D, Schechter I, Shlyakhter I (2000) Alcoa: the Alloy Constraint Analyzer, In ICSE 2000
16. Jackson D (2000) Alloy: A Lightweight Object Modelling Notation. Technical Report 797, MIT Laboratory for Computer Science
17. López de Vergara JE, Villagrá VE, Berrocal J (2002) Semantic Management: advantages of using an ontology-based management information meta-model. In HP-OVUA 2002
18. Noy NF (2001) Managing Multiple Ontologies in Protégé-2000. In Fifth International Protégé-2000 Workshop
19. Object Constraint Language (OCL) http://www.omg.org/docs/ptc/03-10-14.pdf
20. Rosen E, Rechter Y (1999) BGP/MPLS VPNs. RFC 2547