UQAC
Université du Québec
à Chicoutimi

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

COMME EXIGENCE PARTIELLE

DU DOCTORAT EN INFORMATIQUE

PAR

ÉRIC LUNAUD NGOUPÉ

GESTION AUTOMATIQUE DES CONFIGURATIONS RÉSEAUX: UNE APPROCHE DÉDUCTIVE

JUIN 2015

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

La gestion des réseaux informatiques est une tâche de plus en plus complexe et sujette aux erreurs. Les recherches dans le passé ont montré qu'entre 40% et 70% des modifications apportées à la configuration d'un réseau échouent à leur première tentative d'utilisation, et la moitié de ces échecs sont motivés par un problème situé ailleurs dans le réseau. Les opérateurs de réseau sont ainsi confrontés à un problème commun : comment s'assurer qu'un service installé sur le réseau d'un client fonctionne correctement et que le réseau lui-même est exempt de défaut de toute nature ? L'ingénieur réseau a donc à chaque fois qu'un nouveau service sera ajouté au réseau, la responsabilité d'un groupe de périphériques dont les configurations sont gérées individuellement et manuellement. Cette opération vise deux objectifs :

1) Mettre en œuvre la fonctionnalité désirée.
2) Préserver le bon fonctionnement des services existants, en évitant de mettre en conflit les nouveaux paramètres et ceux déjà configurés sur le même réseau.

L'évolution fulgurante du nombre de dispositifs, la complexité des configurations, les besoins spécifiques de chaque service, le nombre même de services qu'un réseau doit

être capable de supporter, et le fait que les données traversent généralement des réseaux hétérogènes appartenant à plusieurs opérateurs, rendent cette tâche de plus en plus difficile. Nous pouvons aisément comprendre la nécessité de nouvelles approches au problème de gestion de configuration réseau.

Au cours de notre étude, nous avons utilisé un formalisme basé sur la logique de configurations qui offre plusieurs avantages, tel que : la vérification efficace et aisée des configurations d'équipements multiples, la séparation claire entre les spécifications de contraintes de configuration et sa validation réelle, mis en relief dans l'outil de configuration et de vérification automatique de configuration appelé ValidMaker. Nous avons aussi présenté un modèle de données génériques pour des informations de configuration des dispositifs réseaux qui prennent en compte l'hétérogénéité des fabricants et de leurs versions. Les concepts tels que Meta-CLI ont été utilisés pour représenter la configuration extraite du dispositif sous forme d'arbre dont les feuilles représentent les paramètres extraits dans le but de pouvoir tester certaines propriétés complexes et d'en déduire les informations restantes. Nonobstant le fait que nos résultats sont basés et validés sur des cas d'utilisation et des configurations matérielles d'une entreprise cible, la méthodologie pourrait être appliquée à des équipements se rapportant à n'importe quel fournisseur de service réseau.

INTRODUCTION

La gestion des réseaux informatiques est toujours un travail pénible, laborieux, sujet aux erreurs et dont la complexité est sans cesse croissante en raison de l'évolution des technologies et du matériel qui entre en compte (Hallé et al., 2005; Delaet et Joosen, 2007). D'une part, les équipements qui forment le réseau doivent se comporter comme un groupe ; cependant, chacune de ces machines est gérée et configurée individuellement. La question fondamentale est la même depuis plusieurs années. Un ingénieur réseaux est responsable d'un pool de dispositifs dont les configurations individuelles sont gérées pour la plupart manuellement. Chaque fois qu'un nouveau service doit être ajouté aux appareils du réseau, il doit s'assurer du réglage parfait et approprié des paramètres de configuration de ces appareils. Cette délicate opération doit viser deux objectifs : mettre en place la fonctionnalité désirée tout en permettant la continuité des services existants. Ceci signifie en particulier que les paramètres de la nouvelle configuration ne doivent pas entrer en conflit avec les paramètres déjà configurés de ces appareils ou ceux d'autres appareils. Imaginez que lors d'un examen en vidéo conférence, après quelques échanges fructueux entre l'étudiant et l'examinateur se trouvant dans une autre ville ou dans une autre université, la communication se coupe. Comment faire pour renouer la communication avec son enseignant ou son étudiant ? Dans ce contexte, les

administrateurs des deux sites et leurs opérateurs réseaux sont confrontés à un problème commun : comment s'assurer qu'un service installé sur le réseau d'un client fonctionne correctement ou que le réseau lui-même soit exempt de tout défaut. On peut donc se poser la question de savoir, comment pourrait-on diagnostiquer automatiquement une erreur de configuration avant l'appliquer les changements ?

Cette problématique soulève donc un pan de voile sur la nécessité de mettre sur pied des modèles efficaces qui doivent être implémentés sur des équipements de réseaux, afin de faciliter le diagnostic et la prise en main rapide des défaillances. Si l'ajout et la prise en compte d'un nouvel équipement réseau pouvait se faire de façon transparente, on estime que la charge de travail des administrateurs réseaux devrait être considérablement réduite. Malheureusement, l'ajout d'un nouvel équipement cause parfois plus de soucis à ces administrateurs qui doivent généralement procéder à des configurations manuelles, et ceci pour chaque équipement déjà présent sur le réseau. De plus, à chaque fois qu'un nouveau service est ajouté au réseau, ils doivent s'assurer que les paramètres de configuration de ces équipements soient réglés sur des valeurs appropriées.

Des recherches menées dans le passé ont montré que 40 à 70% des changements apportés dans la configuration d'un réseau échouent lors de la première tentative et que la moitié de ces changements sont motivés par un problème situé ailleurs dans le réseau (Strassner, 2002). On peut raisonnablement penser que ces chiffres n'ont pas évolué ces dernières années : (Feamster et Balakrishnan, 2005) a révélé plus de 1000 erreurs dans la configuration BGP de 17 réseaux ; (Wool, 2004) a étudié des pare-feux sur le plan quantitatif et a révélé que tous étaient mal configurés d'une façon ou d'une autre. La diversité des équipements réseaux et des contraintes qui leur sont associées font ainsi accroître la complexité de la gestion des configurations ; et comme le mentionnent (Delaet et Joosen, 2007; Campi et Bauer, 2009), parmi tous les problèmes liés aux équipements

réseau, les erreurs de configurations sont les plus difficiles à résoudre. L'objectif des administrateurs de réseaux est de configurer leurs appareils sans aucune erreur. La réduction du nombre d'erreurs peut conduire à une réduction des couts des travaux de maintenance pour les entreprises. L'institut de recherche Sage a découvert que 40% des temps d'arrêt de système sont dus aux erreurs opérationnelles et 44% de ces erreurs proviennent des erreurs de configuration (Pignet, 2007; Delaet et Joosen, 2007).

Ces résultats ont déclenché la conception et la mise sur le marché de plusieurs outils qui, malgré leur utilité, ont fait ressortir d'autres problèmes tels que l'interopérabilité ou l'hétérogénéité du dispositif. Le travail des administrateurs réseau doit donc pouvoir répondre à deux objectifs : la mise en œuvre de la fonctionnalité désirée ainsi que la préservation du bon fonctionnement des services existants. Ces tâches, déjà peu simples, deviennent de plus en plus difficiles à cause de l'évolution fulgurante du nombre des équipements réseaux, la complexité des configurations, les besoins spécifiques de chaque service et le nombre même de services qu'un réseau doit être capable de supporter. Lorsque l'on ajoute à ce tableau le fait que les données traversent généralement des réseaux hétérogènes appartenant à plusieurs opérateurs différents, on comprend pourquoi l'avènement de nouvelles approches au problème de gestion de configuration de réseau est essentiel.

Après avoir présenté les travaux d'autres scientifiques, nous allons nous concentrer sur une nouvelle façon de répondre à ces problématiques et notre solution va s'appuyer sur une approche déductive et structurée basée sur les méthodes formelles, plus particulièrement la logique mathématique et le concept de Meta-CLI. Somme toute la présente thèse rassemble donc les études de quelques aspects de la gestion automatique des configurations ; ses objectifs seront précisés à la fin du chapitre II. Auparavant, au chapitre I, on se propose de familiariser le lecteur avec les concepts de gestion de

configurations réseau et de répondre aux questions essentielles qu'il peut se poser au sujet de l'état de l'art sur la gestion des configurations ainsi que les causes et conséquences des incidents déjà survenus. Nous ferons une description en présentant l'actualité et les risques liés dans la configuration des dispositifs réseaux en présentant quelques exemples des incidents majeurs survenus dans le monde. Par la suite au chapitre 2, nous allons présenter les travaux existants dans ce domaine. Au le chapitre III, nous allons présenter un modèle de données génériques qui prend en compte l'hétérogénéité des fabricants et de leurs versions logicielles. Au le chapitre IV nous allons démontrer comment les contraintes de configuration peuvent être exprimées sous forme d'expressions logiques de premier ordre sur des représentations formelles de configurations comme des structures de données hiérarchiques. Finalement Au le chapitre V, nous proposons un outil qui, enrichi de ces concepts, permet de vérifier et de valider les contraintes et les paramètres avant leur mise en production dans un fichier de configuration quelque soit le fabricant.

Le présent travail tirera une grande partie de sa motivation d'une collaboration d'une grande firme internationale, à la faveur d'un partenariat de recherche financé par le CRSNG [1]. En effet, notre projet aboutira à la mise sur pied d'un outil intégré de raisonnement et de la gestion des configuration pour Ericsson. Pour ce faire nous allons accorder une grande importance à la vérification des contraintes qui est le socle même de la gestion automatique des configurations réseau quel que soit le fabricant, d'où la notion d'intégration.

1. Conseil de recherches en sciences naturelles et en génie du Canada

CHAPITRE 1

LA GESTION DES CONFIGURATIONS

L'évolution fulgurante des réseaux informatiques et Internet a fait accroitre la charge de travail des administrateurs réseaux, occasionnant ainsi un accroissement des ressources humaines consacrées à leur gestion. Les capacités de gestion de réseau ont été poussées à leurs limites et sont donc devenues plus complexes et source d'erreurs. Dans ce premier chapitre, nous allons présenter les incidents majeurs survenus récemment qui sont liés aux problèmes de configurations, ainsi que leurs enjeux au sein des entreprises.

1.1 ÉVÈNEMENTS DE L'ACTUALITÉ

Dans les dernières années, on a assisté à plusieurs incidents liés à des configurations incorrectes des équipements à cause d'une mauvaise vérification des configurations ou tout simplement à cause de la charge importante du travail des administrateurs (Deca et al., 2007).

1.1.1 L'INCIDENT AS 7007

Rappelons d'abord l'incident qui a paralysé le réseau Internet pendant une longue période, 20 minutes pour certains et 3 heures pour d'autres, en avril 1997 aux États-Unis.

D'après Vincent J. Bono, directeur du services réseau chez North American Network Operators Group, le problème a été attribué au routeur central estampillé AS 7007 de la société MAI Network Services en Virginie. Cette société hébergeait les routeurs et les *backbones*[1] devant interconnecter les différents réseaux pour la distribution d'Internet. Malheureusement une mauvaise configuration des routeurs a rendu incorrectes les tables de routage, entraînant rapidement une inondation des routeurs de la société MAI par le trafic réseau. De plus, l'annuaire InterNIC a inscrit par erreur le serveur Internet Exchange comme le propriétaire des tables de routage, c'est à dire qu'il devait diriger les paquets entrants vers leur destination. Ainsi, tous les paquets de l'ensemble du réseau Internet se sont dirigés vers ce seul serveur et, quelque 50 000 adresses sont venues congestionner le réseau, paralysant ainsi tout le réseau Internet des États-Unis et par ricochet ceux des autres réseaux dont il hébergeait aussi le backbone (Bono, 1997).

À la question de savoir pourquoi cette erreur n'a pas été détecté et corrigé, Vincent J. Bono explique qu'une autre erreur de configuration a empêché les routeurs de la société MAI Network Services qui pouvaient corriger le problème de détecter les données erronées. Les tables de routage étaient ainsi considérées comme non optimales, c'est-à-dire pouvant encore accepter une grande quantité de trafic - ce qui n'était malheureusement pas le cas-.

1. Partie centrale d'un réseau d'entreprise, elle permet de connecter entre eux plusieurs sous-réseaux et représente la zone la plus performante et la plus sûr du réseau.

1.1.2 PIRATAGE DU COMPTE ADMINISTRATEUR

Un autre cas de problème de configuration s'est passé en 1992. Une alerte du CERT (Computer Emergency Response Team) avait été lancée suite aux informations indiquant un piratage systématique de mots de passe des clients d'une entreprise de services réseaux (Schaefer, 2003). Bien que les mots de passe soient la cause directe de la panne, l'incident avait débuté par le piratage du compte d'un administrateur réseau de cette entreprise suite à une mauvaise configuration de leur routeur au niveau du cryptage d'un protocole de sécurité. Ceci a permis au pirate d'installer un logiciel d'espionnage afin de s'approprier les comptes et les mots de passe de tous les clients de l'entreprise, qui iraient se connecter sur le réseau distant via leur réseau local. Cet incident malheureux a néanmoins permis d'améliorer le protocole d'identification par mots de passe avec des méthodes modernes de cryptographie (Schaefer, 2003).

1.1.3 PANNE DE GMAIL

Même les plus grands ne sont pas épargnés par les erreurs de configurations. Très récemment la très célèbre entreprise Google a vu sa messagerie Gmail tomber en panne pendant 40 minutes (Journal du Net, 2012). En effet, le service Sync de synchronisation chargé d'harmoniser le navigateur Chrome et certains autres services à partir d'un compte Google a mal fonctionné à cause d'un mauvais réglage d'un paramètre de la *sandbox*[2] affectant ainsi tout le système d'équilibrage des charges ou *load balancing* entrainant ainsi une indisponibilité de la messagerie Gmail, l'espace de stockage Drive,

2. La sandbox Google ou phénomène sandbox désigne une période généralement transitoire pendant laquelle un site est référencé par Google tout en étant maintenu « volontairement »loin dans les résultats, même s'il est particulièrement bien optimisé pour le référencement naturel. Cette période à durée variable (souvent plusieurs mois) est destinée à vérifier que le site est « fiable »en termes de contenus et pratiques pour le référencement.

mais aussi le navigateur Chrome (Journal du Net, 2012).

1.1.4 PANNE CHEZ AMAZON

Contrairement aux autres pannes présentées ici, celle d'Amazon s'est étendue sur une période plus longue, soit quatre jours. Signalée le 21 avril 2011, cette panne avait été causée par une erreur de configuration entrainant une inaccessibilité partielle de leur plate-forme de service de cloud computing (Thibodeau, 2011; Pepitone, 2011).

Cette erreur de configuration a été faite lors de la mise à jour du réseau, laquelle produisit un routage incorrect des paquets. Amazon a expliqué qu'un trafic qui normalement aurait dû aller vers un réseau primaire a été acheminé vers une autre destination de capacité inférieure. Paralysant ainsi l'un de ses cinq sites mondiaux l'empêchant ainsi d'effectuer les opérations de lecture/écriture. Amazon a ainsi vu son pourcentage de disponibilité passer de 41% à 14,6% en seulement 48h (Thibodeau, 2011) (Pepitone, 2011). Suite à cet incident, plusieurs sites web de premier plan ont vu leur temps d'inaccessibilité croître ; c'est le cas de : Quora, Foursquare, Reddit et même du populaire client Twitter HootSuite (Pepitone, 2011).

Les conséquences n'ont pas été des moindres, car le débat sur la maturité même des services de cloud computing s'est réinvité dans toute l'industrie américaine (Thibodeau, 2011; Mi-lung et al., 2004). Selon (Thibodeau, 2011), Amazon n'a pas dit explicitement une erreur humaine a déclenché l'évènement, mais a fait allusion à cette possibilité quand il a écrit que « nous allons vérifier notre processus de changement et accroître l'automatisation pour éviter que ce type d'erreur ne se produise à l'avenir ».

1.1.5 PANNE CHEZ UN FOURNISSEUR DE TÉLÉPHONIE

Nous pouvons aussi citer le cas d'une entreprise de téléphonie qui a vu son système paralysé. En effet, selon l'Agence européenne de la sécurité des systèmes d'information (Enisa), le problème est survenu suite à une erreur de configuration faite par un salarié d'un fournisseur de téléphonie fixe (dont le nom n'a pas été divulgué) qui a attribué une valeur erronée à un paramètre dans un fichier de configuration. L'erreur a empêché des utilisateurs de cette compagnie de téléphonie fixe d'émettre des appels téléphoniques internationaux vers les pays de l'Europe de l'Ouest pendant quatre heures. L'incident a été résolu après une configuration et un réamorçage système (Network et Agency, 2012).

1.2 LES ENJEUX

L'ensemble des erreurs qui viennent d'être présentées ont tous un point commun : une erreur de configuration. Ceci est aussi soutenu par les aveux même du responsable de l'AS 7007

> « MAI's problems stemmed from bad router table information that directed routers operated by Sprint and other ISPs to transmit all Internet traffic to MAI's network »(Network et Agency, 2012).

1.2.1 LE FACTEUR HUMAIN

Les enjeux d'avoir une configuration stable et fonctionnelle sont donc énormes quelque soit le nombre de postes ou d'appareils à gérer. Les conséquences d'une mauvaise configuration sont parfois catastrophiques. De nos jours certaines personnes pensent que pour solliciter un système de configuration il faut d'abord avoir un nombre important

de postes de travail, ce qui est certes vrai comme première exigence, mais pas comme condition essentielle (Campi et Bauer, 2009). Il existe plusieurs causes possibles pour les pannes réseaux, mais les experts s'accordent pour les regrouper en trois catégories à savoir :

— les évènements planifiés de maintenance

— les erreurs système

— les facteurs humains.

Malgré que, les vendeurs d'équipements réseaux se focalisent tous sur les deux premières catégories, les erreurs liées à l'action humaine n'en demeurent pas moins la source la plus répandue (Juniper Networks, 2008).

Plusieurs cabinets ont réalisé des études sur les causes et les enjeux des problèmes de configuration et tous sont unanimes que l'intervention humaine en est le principal facteur. Citons d'emblée le cas de la société Juniper Networks qui dans son livre blanc révèle que 50 à 80% des erreurs de configuration réseau sont causées par le facteur humain (Juniper Networks, 2008).

En décembre 2008 le cabinet de consultation Netcordia lors d'un sondage réalisé chez plus de 450 administrateurs réseau a révélé que 64% des administrateurs pensent que la plus grande menace sur la disponibilité du réseau est interne, c'est à dire causée par l'action humaine.

Citons aussi celui de l'Institut Visible Ops Handbook qui, lors de leurs études sur les processus informatiques, ont pu démontrer que 80% des arrêts non planifiés sont dûs à de mauvaises modifications apportées par les administrateurs ou les développeurs (Bejtlich, 2005). Un autre constat a été fait par l'entreprise *the Enterprise Management Association*, qui rapporte que 60% des erreurs de disponibilité et de performance sont le

résultat d'erreurs de configuration (Handbook, 2011).

Enfin nous pouvons aussi citer la récente étude du Cabinet Gartner qui prévoit qu'avant 2016, 80% des pannes impactant des services critiques seront causées par l'homme et par des erreurs de processus, et plus de 50% de ces erreurs seront causées par le changement de configuration (Colville et Spafford, 2010a) (Colville et Spafford, 2010b).

1.2.2 POUR UNE GESTION AUTOMATISÉE

Il est à considérer qu'on peut avoir un seul serveur, mais que celui-ci puisse héberger des services et des applications de haute importance, dont toute indisponibilité pourrait entrainer des pertes considérables pour l'entreprise. (Campi et Bauer, 2009) avancent qu'avec un système de gestion automatisée on pourrait facilement éviter cela . Dans un autre registre lors de l'absence du principal administrateur la gestion automatisée serait la bienvenue puisque le système pourrait s'auto-diagnostiquer en cas de panne (Campi et Bauer, 2009). Les événements présentés précédemment auraient pu être évités si un système de vérification automatique avait été installé. C'est également le cas de l'incident de 1997 qui avait paralysé internet, un système de configuration et de vérification automatique aurait pu aider à déceler le changement dans le fichier de configuration du routeur. Dans le passé, toute panne du système, fut-elle minime, nécessitait une intervention humaine, ce qui conduisait à la réduction du temps que pouvaient accorder les administrateurs aux autres tâches.

Selon le rapport annuel 2012 de l'Agence européenne de la sécurité des systèmes d'information (Enisa), il y a eu au cours de l'année 2011, 51 incidents significatifs qui ont affectés principalement les réseaux ou les services de communications. Selon ce rapport (Network et Agency, 2012), les erreurs de configuration ont été les plus onéreuses en

temps de travail : des millions d'heures ont été consenties et en termes financiers des millions de dollars avaient été dépensés pour la résolution et le manque à gagner causé par ces incidents.

L'enjeu que revêt une bonne gestion des configurations est donc énorme et demande à être automatisée. L'automatisation de la gestion de configuration, reste donc une préoccupation majeure car le système pourrait lui-même informer de son état. Le souci des utilisateurs d'ordinateurs ou d'équipements réseau n'est-il donc pas d'avoir des équipements qui pourraient comme le corps humain, réagir à toute intrusion, modification ou changement brusque (Burgess, 1998).

Il est donc reconnu que les entreprises gagneraient à avoir des systèmes en parfait état de marche plutôt que de passer plus de temps à essayer par eux même de diagnostiquer manuellement les attaques des virus ou des intrusions ou tout simplement des dysfonctionnements de leurs appareils (Burgess, 1998).

1.3 QU'EST-CE QU'UNE CONFIGURATION ?

Le terme « gestion de configuration »peut être interprété de deux manières.

D'abord celle qui fait référence au suivi du cycle de vie des équipements, appelée en anglais « asset management ». Elle peut être définie comme étant l'ensemble des moyens organisationnels et administratifs mis en place pour obtenir, à tout moment du cycle de vie du produit, une visibilité satisfaisante du produit, au travers de ses caractéristiques physiques et fonctionnelles, dont la démarche est généralement décrite dans des documents (Coulon, 2001).

Ensuite, nous avons celle qui fait référence à la gestion des configurations techniques

des équipements, c'est à dire celle qui permet de valider les configurations et de vérifier le bon fonctionnement des équipements au sein d'un ensemble. D'après le Larousse, ce second aspect de la gestion de configuration peut être défini comme la gestion de toute modification ou réglage de paramètres informatiques en vue de l'optimisation du fonctionnement du système.

Il faut souligner ici que la seconde interprétation bénéficie du concept même de la première, car une bonne gestion des configurations doit être accompagnée d'une documentation bien fournie et bien détaillée afin de permettre une meilleure récupération d'erreurs.

Les évènements présentés soulèvent tous un problème de la configuration des équipements ou d'appareils informatiques, mais qu'entend on par configuration ? Par définition la configuration d'un logiciel, d'un matériel, ou d'un réseau informatique est un ensemble de caractéristiques ou spécifications techniques qui ne dépendent pas du constructeur mais découlent des choix de l'acheteur et de l'utilisateur. Ces spécifications sont donc susceptibles de se différencier même pour des équipements de construction identique.

Ces spécifications comprennent généralement la vitesse du processeur, la quantité de mémoire vive, espace disque dur, et le type de carte vidéo quand on parle d'ordinateur. Mais quand on parle de configuration dans le domaine des réseaux informatiques on fait référence à la modification du fichier de configuration, au réglage de paramètres des équipements réseaux comme les routeurs, les pare-feux et les commutateurs intelligents en vue de leur optimisation pour un meilleur fonctionnement. La configuration fait ressortir la notion de fichier et de paramètre, les fichiers contiennent les paramètres sur lesquels sont basés les configurations afin d'assurer le contrôle et le bon fonctionnement des équipements.

Dans notre cas précis nous allons nous limiter aux systèmes informatiques, principalement à la partie concernant l'infrastructure réseau et ses composantes. Dans cet esprit, on peut donc définir la gestion des configurations comme étant l'ensemble des caractéristiques d'un réseau donné. Ceci inclut autant les caractéristiques physiques telles que la connectique que les caractéristiques logiques telles que les protocoles utilisés, les adresses IP, ainsi que le nom de chaque machine ou équipement (routeurs, pare feu, switch) branchés au réseau. Dans la suite de cette section, nous donnons trois exemples de configurations.

1.3.1 EXEMPLE 1 : CONFIGURATION D'UN ROUTEUR

Comme premier exemple, examinons un fichier de configuration pour la version 11 du système d'exploitation Cisco appélé *Internetwork Opérating System* (IOS) voir figure 1.1.

Chacune des lignes de ce fichier règle l'un des paramètres sur une valeur donnée. Nous en expliquons ici quelques unes.

`no service config` : par défaut, les routeurs Cisco émettent à intervalles réguliers des requêtes TFTP pour vérifier si leur configuration a été modifiée. Ces requêtes sont envoyées à l'adresse de diffusion (*broadcast*) de chaque interface. Hormis le fait que ce trafic est bien souvent superflu, certaines machines réagissent à la réception de la requête et l'inscrivent dans les logs ou sur la console système. Ces traces peuvent constituer une taille gênante (sensiblement 1 mégaoctet par semaine). La commande `no service config` permet de ne plus diffuser de telles requêtes.

`no service tcp-small-servers` et *no service udp-small-servers* :

```
version 11.0
no service config
no service tcp-small-servers
no service udp-small-servers
service password-encryption
!
logging 192.9.200.1
logging facility auth
!
hostname nom_du_cisco
!
enable password 7 XXXXXXXXXXXXXXXXX
!
! Configuration des interfaces Ethernet : 0 pour mon reseau, 1 pour
le reseau du  campus.
!
interface Ethernet0
description -Connexion reseau local-
ip address 192.9.200.254 255.255.255.0
ip access-group 101 out
ip accounting access-violations
no ip proxy-arp
no ip redirects
!
interface Ethernet1
description - Connexion reseau de campus ou Renater -
ip address 192.9.100.1 255.255.255.0
no ip proxy-arp
```

Figure 1.1: Fichier de configuration Version 11 de l'IOS Cisco

Le routeur implante dans l'IOS un certain nombre de services internes, appelés small servers. Parmi ces services, on peut citer `echo` (ports TCP et UDP numéro 7), `discard` (ports TCP et UDP numéro 9), `chargen` (ports TCP et UDP numéro 19), etc. Il est possible pour des utilisateurs mal intentionnés de faire littéralement crouler le routeur sous des requêtes de ce type, jusqu'à ce qu'il devienne incapable d'effectuer normalement les opérations de routage. Les commandes "`no service tcp-small-servers`" et "`no service udp-small-servers`" permettent au routeur d'ignorer les paquets à destination de ces ports et d'éviter ainsi un deni de service.

`service password-encryption` : Par défaut, les mots de passe d'accès aux routeurs Cisco sont stockés en clair dans les configurations. Cette commande permet de ne plus stocker en clair les mots de passe dans les fichiers de configuration.

logging 192.9.200.1 `logging facility auth` : La commande « logging adresse-du-serveur »permet d'utiliser le mécanisme de syslog pour journaliser sur un serveur externe les évènements (arrêts et redémarrages du routeur, les (re)configurations du routeur une trace de tous les paquets ayant satisfaits un élément marqué "log" dans une ACL) importants recensés sur le routeur. La commande « logging facility LOG_FACILITY »permet de préciser la facilité utilisée sur le serveur syslog pour journaliser ces évènements.

`ip access-group n [in|out]` : Cette commande apparait optionnellement dans la déclaration d'une interface, indique que l'ACL numéro n (avec n compris entre 100 et 199) s'applique pour les paquets qui entrent dans le routeur par cette interface.

La commande `access-group n [out]`, qui apparait aussi optionnellement dans la déclaration d'une interface indique que l'ACL [3] numéro n (avec n compris entre 100 et

3. ACL (Access Control Lists) sont des fonctions appliquées à chaque paquet IP transitant à travers le routeur et qui ont pour paramètres : l'adresse IP de l'émetteur du paquet, l'adresse IP du destinataire du paquet, le type du paquet (tcp, udp, icmp, ip), le port de destination du paquet.

199) s'applique pour les paquets qui quittent le routeur par cette interface.

Rappel : les mots-clés *in* et *out* n'existent pas dans les versions inférieures à 10 du firmware de Cisco. Pour ces versions, le fonctionnement est celui de *out*. Pour une même interface, il peut y avoir plusieurs déclarations access-group. En outre, la même commande access-group peut apparaitre dans la déclaration de plusieurs interfaces.

`ip accounting access-violations` : La commande `ip accounting access-violations` apparait optionnellement dans la déclaration d'une interface. Elle indique que le routeur doit conserver une trace de tous les paquets rejetés pour cause de violation d'une ACL associée à cette interface.

`no ip redirects` : C'est une commande qui apparaît optionnellement dans la déclaration d'une interface. Elle indique que le routeur ne doit ni générer, ni accepter de paquets ICMP Redirect.

1.3.2 EXEMPLE 2 : OPENVPN

Nous allons présenter figure 1.2 un autre exemple de fichier de configuration, en l'occurrence celui d'OpenVPN (Raum, 2005) qui est un logiciel permettant de créer un réseau privé virtuel (VPN).

Ce fichier s'interprète de manière similaire. Nous nous permettons d'expliquer brièvement quelques lignes.

— `port` nombre permet de définir le port d'écoute par exemple 5000.

— `dev` permet de définir le type d'interface virtuelle, ici c'est une interface Ethernet virtuelle.

```
FICHIER DE CONFIG : /etc/openvpn/server.conf
version 1.1 : usage of new option "server-bridge" and inactivity control
port 5000
dev tap0
ca /etc/openvpn/cacert.pem
cert /etc/openvpn/openvpncert.cert
key /etc/openvpn/openvpnkey.pem
dh /usr/local/etc/ssl/dh1024.pem
tls-cipher RC4-MD5
la commande server-bridge remplace le bloc de commande suivant :
mode server
tls-server
ifconfig-pool 192.168.0.32 192.168.0.64 255.255.255.0
push "route-gateway 192.168.0.1"
server-bridge 192.168.0.1 255.255.255.0 192.168.0.32 192.168.0.64
persist-tun
persist-key
inactive 3600
ping 10
ping-exit 60
user nobody
group nobody
verb 4
```

Figure 1.2: Fichier de configuration d'OpenVPN

— `tls-cipher RC4-MD5` permet d'utiliser un encryptage qui, ici est RC4-MD5.

— Les deux commandes `persist-tun` et `persist-key` permettent d'utiliser des tunnels persistants pour créer un pont entre les différentes interfaces.

— `ifconfig-pool` permet de définir la plage d'adresse à utiliser.

— verb 4 permet de définir le niveau de log ici le niveau est 4.

1.3.3 EXEMPLE 3 : INTERFACES ETHERNET

Nous présentons un dernier exemple de fichier de configuration cette fois-ci sur les interfaces Ethernet dans un système Linux. Les fichiers de configuration des interfaces contrôlent les interfaces logicielles pour les périphériques réseaux individuels. Lorsque

le système démarre, il utilise ces fichiers pour savoir quelles interfaces il doit utiliser et comment les configurer. Ces fichiers sont en général nommés *ifcfg-<nom>* où nom correspond au nom du périphérique contrôlé par le fichier de configuration. L'un des fichiers d'interface le plus habituel est */etc/sysconfig/network-scripts/ifcfg-eth0*, qui contrôle la première carte d'interface réseau Ethernet ou NIC dans le système. Dans un système comportant plusieurs cartes réseaux, il y a plusieurs fichiers ifcfg-ethX (où X est un numéro unique correspondant à une interface spécifique). Chaque périphérique ayant son propre fichier de configuration, un administrateur peut contrôler la façon dont chaque interface fonctionne. La figure 1.3 présente un exemple de fichier *ifcfg-eth0* pour un système utilisant une adresse IP fixe (Red_Hat, 2013).

```
DEVICE = eth0
BOOTPROTO = none
ONBOOT = yes
NETMASK = 255.255.255.0
IPADDR = 10.0.1.27
USERCTL = no
```

Figure 1.3: Interface utilisant une adresse IP fixe

Signification des paramètres

- `DEVICE` : nom de la carte réseau
- `ONBOOT` : activation de l'interface
- `BOOTPROTO` : type de protocole DHCP ou static. Dans ce cas c'est "static"
- `NETMASK` : masque de sous-réseau
- `IPADDR` : adresse IP liée à la carte.

Les paramètres requis dans un fichier de configuration d'interface peuvent changer en fonction d'autres valeurs. Par exemple dans la figure 1.4, le fichier ifcfg-eth0 pour une

interface utilisant DHCP est différent, car les informations IP sont fournies par le serveur DHCP (Red_Hat, 2013).

```
DEVICE    = eth0
BOOTPROTO = dhcp
ONBOOT    = yes
```

Figure 1.4: Interface utilisant DHCP

1.4 CAUSES

Les configurations sont parfois sujettes à des erreurs qui peuvent occasionner des dysfonctionnements majeurs ou minimes selon la portée du paramètre ou de la valeur mis en cause. Il peut y avoir plusieurs causes pour ces erreurs dans les fichiers de configuration, comme des erreurs de syntaxe, de cohérence d'exécution, de logique, de valeur inappropriée ou non supportée. Nous allons détailler dans la suite chacune.

1.4.1 ERREURS DE SYNTAXE

Les erreurs syntaxiques sont presque toujours liées aux fichiers de configuration. Les erreurs de syntaxe apparaissent lors de l'écriture du code, et peuvent être causées par le non respect des règles, des formats des services implémentés, les erreurs de commandes ou de règles mal écrites. Quand une erreur de syntaxe est détectée tout ou une partie du fichier peut être ignorée et le fonctionnement du matériel ou du logiciel en sera affecté (Buchmann, 2008). Par exemple, il suffit de faire une faute d'écriture sur une variable ou un paramètre comme `port=80` et écrire plutôt `porc=8o` que tout paramètre qui va utiliser la valeur du port peut-être bloqué causant ainsi un risque de dysfonctionnement

ou de plantage de l'équipement en question.

Nous pouvons citer comme cas particulier des erreurs de syntaxe, l'utilisation des valeurs inappropriée ou non supportée. Ce sont des erreurs qui apparaissent lors de l'utilisation d'une valeur ne correspondant ni ne faisant référence à aucun type ni nom de variable déclaré dans l'environnement.

1.4.2 ERREURS DE COHÉRENCE

La gestion de configuration réseau ou matériel est une discipline assez complexe, au vu de la diversité des fabricants d'équipements qui mettent sur le marché une large gamme d'équipements différents les uns des autres bien qu'ayant les mêmes fonctionnalités. Il est primordial de respecter la cohérence lors du paramétrage ou de l'écriture des fichiers de configuration, car l'absence d'erreurs syntaxiques peut cacher une incohérence. Les erreurs de cohérence surviennent généralement lors des cas d'incompatibilité entre les configurations et les appareils. L'incohérence des configurations qui portent sur quelques équipements ne pose pas de problème majeur car la récupération peut se faire rapidement, mais dans un réseau de grande taille cela devient une vraie problématique (Buchmann, 2008; Hallé et al., 2005).

En d'autres termes, le changement de configuration ou de politique sur un équipement peut influer sur l'ensemble ou une partie du réseau par la violation d'une règle déjà établie. On peut imaginer le cas où il faille ouvrir un port pour faire fonctionner une application et pour garantir la sécurité d'un autre équipement du même réseau il faille plutôt le bloquer.

Comme autre exemple d'erreur de cohérence, prenons le cas d'administrateur réseau lors

de l'écriture de son script de mise à jour des configuration d'un type de routeur X, par exemple WiFi Aruba (ayant déjà le même script pour les routeur Cisco). Pour gagner du temps, il copie ce script afin de le modifier pour les routeurs WiFi Aruba, mais il oublie de changer le nom du fichier pour celui des routeurs WiFi Aruba, ceci ne pourra être détecté qu'au moment du fonctionnement du nouveau routeur qui peut afficher des résultats inattendus pendant que les routeurs Cisco peuvent ne plus fonctionner correctement, car leur fichier de configuration est devenu inexistant.

1.4.3 ERREURS D'EXÉCUTION

Les erreurs d'exécution sont celles qui apparaissent lors de l'exécution ou du lancement d'une application ou lors de la mise en route d'un équipement. Elles sont dues à l'impossibilité d'exécution, malgré un code qui semble correct, par la non présence d'erreur de syntaxe. C'est le cas par exemple, où l'on a écrit correctement une instruction permettant de lire dans un fichier afin de récupérer certains paramètres, malheureusement si le fichier est endommagé ou protégé contre la lecture, l'application ne pourra pas exécuter sa fonction de lecture. Généralement on peut résoudre les erreurs d'exécution en réécrivant le code (Microsoft, 2013).

1.5 CONSÉQUENCES DES ERREURS DE CONFIGURATION

Tout dysfonctionnement dans la configuration d'un équipement peut entrainer de graves conséquences sur la sécurité même de l'infrastructure, du routage des paquets et aussi occasionner des pertes financières énormes.

1.5.1 SÉCURITÉ

Toute erreur dans les fichiers de configuration nécessite une attention particulière et une résolution rapide car elle peut constituer une source de problèmes encore plus grave. La sécurité est un souci majeur dans le monde des réseaux, des millions de dollars sont investis par les entreprises pour faire face à cette préoccupation.

Nous pouvons dire à juste titre que le fait d'avoir une mauvaise configuration va inéluctablement jouer sur la sécurité du système tout entier. Quand la sécurité est compromise les risques que courent l'entreprise sont énormes. On peut citer entre autre :

— La panne du système ou corruption des données,

— L'infection virale ou logiciel destructeur,

— Le mauvais usage du système d'information par le personnel,

— L'accès non autorisé par des tiers (y compris des tentatives de piratage),

— Le vol ou la fraude à l'aide d'ordinateurs,

— Le vol ou la divulgation non autorisée d'informations confidentielles.

1.5.2 ROUTAGE

Par définition le routage est l'acheminement des informations d'un réseau à un autre au niveau de la couche 3 du modèle OSI. Le routage revêt une importance capitale dans les communications entre ordinateurs, stations de travail et autres terminaux informatiques (Lalitte, 2003).

Nous savons que pour son fonctionnement le routeur utilise une base de données appelée « table de routage » qui est basée sur les politiques appliquées, et que toute erreur dans

le fichier de configuration peut entraîner un dysfonctionnement du réseau. Lorsque les politiques appliquées dans le nœud d'un réseau (routeur, pare-feu) ne sont pas bien maîtrisées avec des mises à jour régulières par la mise en place des règles strictes que l'administrateur doit respecter pour configurer son équipement, on peut avoir comme conséquence un mauvais routage des paquets dans le réseau. C'est le cas par exemple d'un pare-feu qui au lieu de bloquer laisse plutôt passer les communications suite par exemple à une erreur de syntaxe ou de cohérence. Le dysfonctionnement d'un routeur peut amener certains paquets à prendre des destinations contraires à celles voulues. Avec le dysfonctionnement du routeur on peut aussi être confronté au problème de bouclage ou d'inondation, c'est à dire que le routeur va faire propager un paquet (de données ou de contrôle) dans le réseau entier puisque sa destination n'est plus connue.

1.5.3 COÛTS

Le coût financier des incidents n'est pas la chose la mieux partagée dans les entreprises car il occasionne des manques à gagner financiers énormes, surtout de petite taille. Mais il nous a semblé opportun d'en parler dans notre travail de recherche car il a une influence non négligeable dans toute prise de décision informatique. Selon une étude (Angelo Rossi, 2011) menée par la société *Ponemon Institute* et publiée dans le journal Informatique, 49% des entreprises informatiques ont connu des incidents informatiques. En ajoutant à ces incidents les coûts indirects comme les coûts de résolution de perturbation de l'activité, ils auront coûté plus de 240 000 $ entre mai 2010 et mai 2011.

Dans l'article (Patterson, 2002) l'auteur nous donne une formule pour calculer le coût occasionné par les incidents ou pannes qui peuvent aider les dirigeants ou les responsables informatiques à une meilleure prise de décision.

Selon (Buchmann, 2008) l'estimation du coût moyen pour 1 heure d'indisponibilité (Cm) est égale au coût des employés par heure (Cem) multiplié par le pourcentage des employés affectés par cette coupure (Pe) additionné au revenu moyen par heure (Rm) multiplié par le pourcentage du revenue affectée par cette coupure (Pr).

$$Cm = (Cem \times Pe) + (Rm \times Pr)$$

Le coût des employés par heure est le total des salaires et des avantages de tous les employés par semaine (Ts) divisé par le nombre moyen d'heures de travail fait par mois (Ht).

$$Cem = \frac{Ts}{Ht}$$

Le revenu moyen par heure est le total des recettes de l'institution par mois (Ri) divisé par le nombre moyen d'heures par semaine d'une institution (Nm) (Patterson, 2002).

$$Rm = \frac{Ri}{Nm}$$

Calcul du coût moyen des employés par heure

Si l'on pose :

— C' = Coût des employés par heure.

— T' = Revenu total de l'entreprise sur une semaine.

— A = avantages divers de l'entreprise sur une semaine

— N = Le Nombre moyen d'heure effectué au cours d'un mois.

— Th = Nombre total d'heure sur un mois.

— E = nombre d'employés dans l'institution au cours du mois.

Entreprises	Estimation de perte
Les opérations de courtage	6 450 000 $
Autorisation des cartes de crédit	2 600 000 $
Ebay	225 000 $
Amazon.com	180 000 $
Services d'expédition de l'emballage	150 000 $
Chaîne de télé-achat	113 000 $
Catalogue des centres de vente	90 000 $
Centre de réservation des compagnies aériennes	89 000 $
Service d'activation de cellulaire	41 000 $
Frais de réseau en ligne	25 000 $
Frais de service ATM	14 000 $

Tableau 1.1: Coût d'une heure de panne d'après Kembel (Kembel, 2009)

Alors le coût moyen se calcule ainsi :

$$C = \frac{T' + A}{N} \text{ avec } N = Th * E$$

Le tableau 1.1 présente les résultats de l'étude d'enquête faite dans le journal InternetWeek rapporté par R. Kembel en 2000 sur le coût d'une heure de panne (Kembel, 2009).

Les types d'entreprises considérés dans ce tableau ne sont pas les seuls à perdre des recettes lors d'une panne. Toujours dans cet exemple, on n'a pas tenu compte de la perte de temps occasionnée par les employés qui ne peuvent pas faire leur travail lors d'une panne. Mais le calcul du coût obtenu de cette façon n'est qu'un cas général. Certaines particularités existent, car si votre ordinateur tombe en panne pendant que vous travaillez sur une nouvelle transaction très importante, qui peut dire quel sera le coût final de cette interruption ? De même, si votre site Internet n'est pas accessible quand un client veut passer une commande ou envoyer une demande, ce client pourrait ne jamais revenir.

Au final, on peut voir que les erreurs de configuration peuvent avoir des répercussions profondes et difficiles à quantifier.

CHAPITRE 2

ÉTAT DE L'ART EN GESTION DES CONFIGURATIONS

La gestion des réseaux informatique est toujours un travail fastidieux, laborieux et sujet à des erreurs dont la complexité augmente sans cesse en raison de l'évolution technologique et en fonction des équipements utilisés (Hallé et al., 2005), (Delaet et Joosen, 2007). Dans le chapitre précédent, nous avons présenté plusieurs problèmes ou incidents informatiques liés à une mauvaise configuration d'un équipement réseau dû à l'intervention manuelle de l'homme. L'expérience nous montre que toute intervention manuelle peut être sujette aux erreurs, il serait donc essentiel d'utiliser une machine pour limiter ces incidents. Selon (Gartner, 2012) moins de 20% des entreprises ont une solution de gestion automatisée, pourtant elle présente de nombreux avantages et des outils existent pour l'appliquer.

Dans cette partie nous essayerons de présenter en détail quelques approches qui ont été utilisées pour la gestion des configurations. Ensuite, nous nous focaliserons sur la présentation de quelques protocoles et outils de gestion automatisée des configurations.

Il faut tout d'abord rappeler que ces outils ou systèmes produisent une couche d'abstraction entre ce que l'on souhaite accomplir (c'est à dire les besoins) et la manière dont ces besoins sont réellement implémentés sur les hôtes cibles (Delaet et al., 2011).

Un langage de programmation de bas niveau permet de définir les actions à effectuer et le moteur de configuration des machines du déploiement, de l'exécution et de la génération d'un rapport sur chacune des hôtes cibles.

La plupart des outils de configuration automatique de systèmes fournissent une interface qui permet à l'administrateur de spécifier dans le fichier de configuration toutes les politiques qu'il souhaite appliquer sur ses équipements. Ensuite l'outil utilise cette spécification en entrée et l'applique sur tous les équipements du même type à gérer. Dans la figure 2.1 de (Delaet et al., 2011) nous avons deux agents à savoir :

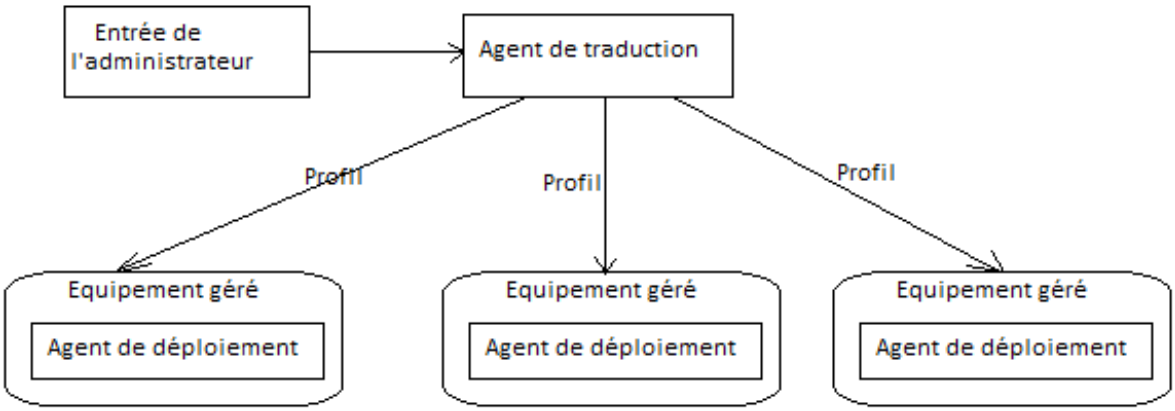

Figure 2.1: Architecture conceptuelle simplifiée d'un outil de gestion de configuration

1. Un agent de traduction qui est le composant chargé de traduire l'entrée de l'administrateur en profil de configuration compréhensible par l'outil.

2. Un agent de déploiement qui est le composant de l'outil qui est chargé d'exécuter les profils de configuration générés.

2.1 APPROCHES ACTUELLES DANS LA GESTION DES CONFIGURATIONS RÉSEAU ET DE LEUR INTÉGRITÉ

Il existe plusieurs approches (Clarke, 2012) pour la résolution des problèmes de configuration soulevés au chapitre précédent. Une approche pourra être privilégiée à une autre suivant plusieurs critères comme la taille du réseau, la récurrence des pannes, les connaissances techniques etc. Dans cette section nous allons présenter les approches existantes en gestion des configurations.

2.1.1 APPROCHE DE GESTION

Gestion manuelle

La résolution manuelle est celle où l'administrateur se charge lui-même d'effectuer toutes les manipulations au moyen d'actions et de commandes qu'il saisit manuellement, de la configuration à la résolution en passant par la détection et le diagnostic. Elle est définie comme étant la mise en place manuelle des paramètres pour le bon fonctionnement des appareils et des services réseaux.

Avantage : elle est adaptée à chaque besoin ; avec ce type de configuration on a toujours une solution sur mesure.

Inconvénient : Problème d'échelle, le fait de répéter plusieurs fois la même chose peut devenir fastidieux et on s'expose ainsi à des erreurs dans le fichier de configuration.

Duplication

La duplication consiste à copier une configuration qui fonctionne bien et à l'appliquer à un autre équipement du même type.

Avantage : Certes, la première configuration peut être fastidieuse (ce qui est normal car on part de zéro) mais les configurations suivantes seront rapides et faciles à déployer. Il y aura donc un gain de temps à partir de la seconde configuration (Clarke, 2012).

Inconvénient : Difficulté à adapter les paramètres à chacun des équipements sur lesquels la configuration est appliquée. De plus, à chaque nouveau changement on sera obligé de dupliquer à nouveau les supports de sauvegarde (les anciennes n'étant plus à jour) pour une nouvelle conservation et les anciens supports détruits suivant un calendrier ou une méthode bien établie et propre à l'entreprise ou tout simplement utiliser la méthode Grand-parent parent enfant [1] La gestion du changement dans les configurations devient ainsi une préoccupation majeure pas toujours souhaitée.

L'autre difficulté réside dans le fait que, bien que l'on applique directement une configuration entière à un équipement, cette opération doit être répétée pour chacun, ce qui va malheureusement alourdir le travail des administrateurs systèmes ou réseaux.

Centralisation

La centralisation consiste à l'aide d'un réseau, de pouvoir gérer les problèmes de configuration à partir d'un point unique qui peut être un serveur, un poste de travail ou un équipement. Cette troisième méthode introduit donc la notion de réseau et propose des solutions aux inconvénients soulevés par les deux précédentes méthodes.

1. Consiste à conserver les trois versions les plus récentes d'un artéfact de configuration.

Avantage : L'administrateur n'aura pas forcément à effectuer des sauvegardes de configuration sur des supports externes, car il y aura un point central où les configurations des équipements seront logées. Le déploiement sera moins fastidieux, l'administrateur va simplement récupérer la configuration d'un équipement X au point central et l'installer directement. L'automatisation couplée à cette méthode permettra un déploiement rapide et plus efficace en facilitant la restauration en cas de panne ou de mise à jour des configurations.

Inconvénient : L'un des désagréments ici est la surcharge du trafic réseau lors du déploiement et de l'exploitation, car sur chaque équipement on doit installer un agent (ManageEngine, 2012), l'autre inconvénient est que la complexité des opérations sera accrue.

2.1.2 APPROCHE DANS LA GESTION DE L'INTÉGRITÉ DES CONFIGURATIONS RÉSEAUX

Les travaux actuels qui traitent de la gestion et de l'intégrité des configurations réseaux peuvent se regrouper en quatre catégories :

1. *La surveillance du réseau*

   Un grand nombre de moyens de gestion de réseau reposent sur des paramètres externes de surveillance tels que la connectivité et le débit pour s'assurer que le réseau fonctionne convenablement. Certains d'entre eux comme le projet Minerals (Le et al., 2006) utilisent des techniques de raisonnement de l'intelligence artificielle et déduisent des manquements dans la configuration du dispositif à partir de ces observations. Malgré le fait que ces solutions permettent de façon relativement efficace de détecter un comportement anormal, les moyens qu'elles offrent pour

retrouver la cause de ce comportement restent très limités. Une autre insuffisance est attribuée au fait que les erreurs sont détectées à posteriori : si des simulations réalistes sont faites avant chaque changement opéré sur le réseau, l'administrateur doit attendre que les configurations erronées soient engagées avant de découvrir que les choses ne vont pas dans le sens souhaité.

2. *Gestion de changement de configuration*

   Cette seconde catégorie de solutions comprend les outils qui gèrent les changements de configuration des dispositifs réseaux. Par exemple le système Really Awesome New Cisco confIg Differ (RANCID) (Networks, 2006) se connecte à chaque routeur d'un réseau, extrait sa configuration et l'envoie à un serveur appelé en anglais *Concurrent Version System* ou (CVS) où les changements peuvent être détectés et suivis. Plusieurs autres outils, tels que le logiciel d'accès libre ZipTie (Castillo, 2006) et le logiciel commercialisé Voyence (Voyence, 2014), offrent des fonctionnalités similaires permettant de comparer des configurations et garder les historiques de changements de configuration. Cependant, même si le fait de détecter des changements dans une configuration est un principe positif, tous les changements ne débouchent pas sur des configurations à problèmes : à cause de nombreuses fausses alarmes peuvent se déclencher. En outre, même si la source de la configuration à problème se rapporte à un paramètre déterminé ayant changé, la cause de l'erreur ne peut être recherchée que manuellement par l'ingénieur réseaux.

3. *Arbres Décisionnels et Systèmes Experts*

   Une autre possibilité de diagnostiquer les problèmes de configuration consiste à décrire et à standardiser pragmatiquement les procédures de résolutions des problèmes de réseau sous forme d'arbres décisionnels décrivant les tests à effectuer et les mesures à prendre par rapport au système. Chaque nœud dudit arbre représente

un test ou une action à effectuer sur le système. Les différents contours de l'arbre apparaissent en fonction du résultat de chaque test jusqu'à ce qu'une solution efficace soit trouvée. A l'origine, l'utilisation des arbres décisionnels requérait une intervention humaine pour des tâches simples ; certains outils mis en place tels que Babble (Couch, 2000) et Snitch (Mickens et al., 2007) peuvent actuellement des scripts pouvant interagir avec des outils de commande en ligne (console) afin d'automatiser des tâches administratives redondantes. Les réseaux bayésiens prolongent ce principe en incorporant les probabilités dans la structure de l'arbre décisionnel. Cette approche a été adoptée par de nombreux projets concrets tels que le BATS (*Bayesian Automated Troubleshooting System*) (Langseth et Jensen, 2003) utilisé par Hewlett-Packard pour résoudre les problèmes d'imprimantes en utilisant la méthodologie SACSO (Jensen et al., 2001). ATSIG1 est un projet de l'UE dont le but était de développer un concept pour l'automatisation des processus de résolution de problèmes, qui a été transformé en produit commercial appelé 2solve ; un outil commercial de résolution de problèmes, *Knowledge Automation System* de Vanguard, utilise la même approche [2]

4. *Règles réactives*

   Plusieurs systèmes s'appuient sur cette idée et sont basés sur des règles sous la forme «si condition, alors réaction» qui permettent la gestion automatisée de systèmes informatiques complexes en déclenchant des scripts définis par l'utilisateur lorsque des conditions précises sont remplies dans le réseau. les principaux outils qui implémentent cette approche sont : cfengine (Burgess, 1995), LCFG (Anderson et Scobie, 2002), PIKT (Osterlund, 2014), Bcfg2 (Desai et al., 2006) et Prodog (Couch et Gilfix, 1999). Ces travaux concernent la configuration d'un réseau d'ordinateurs. Cependant, il est raisonnable de penser que cette approche pourrait s'étendre à la

2. http ://www.vanguardsw.com/products/knowledge-automation-system/cms/

configuration d'un réseau de dispositifs tels que des routeurs et des commutateurs. Cette approche est «réactive» car la partie active d'une règle n'est exécutée que lorsque les conditions décrites dans la partie introductive de la proposition correspondent à la configuration. Ainsi, chaque mauvaise configuration doit aller de paire avec une action corrective. Une question centrale – et toujours ouverte – est de s'assurer que l'exécution d'un script ne déclenche une série inattendue d'événements ne conduisant pas à un point de stabilité. Une étude mathématique des règles réactives est faite dans (Couch et Sun, 2003) où les conditions sont réunies pour que les actions convergent. En outre, (Narain, 2005) explique que pour qu'ils servent de moteurs de détection de mauvaise configuration, les systèmes réactifs doivent être pourvus d'une base de règles qui devraient équivaloir à un encodage procédural de tout le moteur.

5. *Règles Déclaratives*

Une dernière approche consiste à définir les contraintes qu'une configuration doit respecter afin d'être valide. Les règles sont des affirmations sur les dispositifs et ces affirmations sont automatiquement vérifiées avant l'application des changements. Par exemple, (Narain, 2005) utilise des méthodes de proof plans et Prolog pour vérifier des ordres destinés à de larges parties de systèmes informatiques. (Zeller et Snelting, 1997) applique une logique basée sur des pairs d'attributions de fonctionnalités/valeurs sur la production des versions en configuration de programme. (Klarlund et al., 1997) utilise une logique sur les arborescences d'analyse syntaxique pour édicter des principes sur la généralité de la formalisation des configurations. Dans (Bush et Griffin, 2003), une approche formelle à la modélisation des Virtual Private Networks (Réseaux Privés Virtuels) à l'aide d'une logique de premier ordre a été employée pour présenter les propriétés d'isolation de trafic. Dans (Benedikt

et Bruns, 2004), Delta-X, un langage formel pour les contraintes d'intégrité de données, est présenté comme outil de construction des gardiens d'intégrité : un gardien d'intégrité est un morceau de code exécuté avant que la mise à jour d'une donnée ne soit exécutée. Le gardien répond vrai si la mise à jour préservera l'intégrité des données. (Narain, 2005), (Narain et al., 2008) utilise le langage de modélisation Alloy (Jackson, 2002) pour formaliser un ensemble de contraintes afin qu'un VPN fonctionne convenablement. Cet ensemble de contraintes est ensuite converti en une formule Booléenne et envoyé à un solveur de satisfabilité. La solution renvoyée par le solveur peut être reconvertie en modèle Alloy original et constitue une configuration qui respecte les contraintes originales. D'autres outils plus récents tels que COOLAID (Chen et al., 2010b) utilisent une approche de gestion de configuration déclarative similaire. L'approche utilisée par ValidMaker présenté au chapitre 4 rentre dans cette catégorie de solutions. Cependant, ValidMaker se distingue des travaux mentionnés ci-dessus sur plusieurs points. Contrairement au (Rexford et Feldmann, 2001), ValidMaker utilise un langage formel pour entrer des règles de nature plus complexe ; par conséquent, il peut fournir des messages d'erreurs plus détaillés et une validation interactive de la configuration. En outre, les outils actuels manquent de fonctionnalités de recherche de contre-exemple mis en place dans ValidMaker. Ils n'offrent que des questions sur la validité de certaines configurations par lesquelles on ne peut répondre que par oui/non. Le langage formel qu'il propose pour exprimer les contraintes, appelé Configuration Logic, est plus fourni que Delta-X ; il peut servir à modeler des dépendances générales entre les paramètres de configuration et n'est pas lié à un type de dispositif comme dans rcc. Contrairement à l'approche Alloy, ValidMaker n'essaye pas de résoudre le problème général de génération de configuration en satisfaisant à un ensemble de contraintes ; il se concentre plutôt sur le point spécifique portant sur la vérification

qu'une configuration donnée réponde étroitement aux contraintes préalablement établies. Même si les auteurs dans (Narain, 2005), (Narain et al., 2008) suggèrent que la validation peut être indirectement exécutée comme un produit associé, nous verrons plus loin qu'un algorithme dédié à la validation est de loin plus efficace qu'un solveur de critères avec un temps de traitement de l'ordre du millisecondes au lieu de plusieurs minutes.

2.2 PROTOCOLES DE GESTION

Un problème orthogonal aux approches de gestion est celui d'interagir avec les équipements pour leur transmettre les informations de configuration pertinentes. On ne saurait parler de réseau informatique sans faire un arrêt sur les protocoles. De façon simplifiée on peut définir le protocole comme le langage utilisé par les équipements pour communiquer entre eux dans un réseau (Enns et al., 2006, 2011b).

2.2.1 CLI

CLI est un acronyme qui signifie en anglais *Command line interface* qui est une interface homme-machine dans laquelle la communication entre l'utilisateur et l'appareil s'effectue en mode texte (Cisco, 2013a). Son utilisation est simple, l'utilisateur tape une ligne de commande textuelle au clavier pour demander l'appareil d'exécuter une opération. Ensuite, l'ordinateur affiche le texte correspondant au résultat de l'exécution de la commande tapée ou à des questions qu'un logiciel pose à l'utilisateur. Chaque logiciel a son interface utilisateur mais le principe reste le même (Jboss, 2013; Cisco, 2013a).

Aujourd'hui, la plupart des dispositifs ont une interface de ligne de commande encastrée

(CLI) pour des buts de diagnostic de pannes et de configuration. On peut prendre le cas des équipements Cisco, et de son CLI intégré dans le système d'exploitation IOS qui reste une référence de CLI. Certains appellent d'ailleurs le CLI ("Cisco Like-Interface").

Comme exemple d'utilisation de CLI : prenons le cas de configuration des interfaces Ethernet du routeur. Admettons que le nom de l'interface reliée au Poste 1 est th0/0 et celle reliée au Poste 2 est th0/1 et que nous sommes en mode de configuration globale. La figure 2.2 montre les commandes qu'un administrateur doit saisir sur CLI :

```
Pour l'interface th0/0
    Router (config) # interface th0/0
    Router (config-if) # ip address 192.168.1.1  255.255.255.0
                # la ligne précédente permet de configurer l'interface th0/0
    Router (config-if) # no shutdown   Permet d'activer l'interface th0/0
    Router (config-if) # exit Permet le retour au mode config
    Pour l'interface th0/1
    Router (config) # interface th0/1
    Router (config-if) # ip address 10.0.0.1  255.0.0.0
                # la ligne précédente permet de configurer l'interface th0/1
    Router (config-if) no shutdown      Permet d'activer l'interface th0/1
    Router (config-if) exit  Permet le retour au mode config
```

Figure 2.2: Commandes entrées sur CLI

L'accès réseau au CLI a traditionnellement été fait au travers des protocoles TELNET, et SSH plus sécurisé. CLI présente selon (Schoenwaelder, 2003) plusieurs avantages, à savoir :

— Les interfaces de ligne de commande sont généralement orientées sur les tâches, ce qui les rend plus faciles à utiliser pour des opérateurs humains.

— Une séquence enregistrée de commandes peut facilement être reprise.

— De simples substitutions peuvent être faites avec le traitement de texte arbitraire

— Il est nécessaire d'apprendre au moins les parties de l'interface de ligne de commande des nouveaux dispositifs afin de pouvoir créer la configuration initiale. Une

fois que cela est maîtrisé, il devient naturel d'utiliser la même interface ainsi que les abstractions pour l'automatisation des changements de configuration.

— Une interface de ligne de commande n'exige pas d'applications supplémentaires car ses protocoles indispensables TELNET et SSH sont disponibles aujourd'hui sur la plupart des systèmes.

— La plupart des interfaces de ligne de commande fournissent l'aide contextuelle qui réduit la courbe d'apprentissage.

Selon (Schoenwaelder, 2003) il existe aussi quelques inconvénients à savoir :

— Certaines interfaces de ligne de commande n'ont pas de modèle de données commun. Il est très possible que la même commande sur des dispositifs différents d'un même fabriquant se comporte différemment.

— L'utilisation des interfaces de ligne de commande comme interface de programmation est fastidieuse à cause de l'analyse syntaxique.

— Les interfaces de ligne de commande manquent souvent de contrôle approprié de version pour respecter la syntaxe et la sémantique. L'utilisateur est enclin à des erreurs lors de l'écriture des scripts avec des versions différentes des interfaces de ligne de commande entraînant inéluctablement une perte de temps.

— Puisque les interfaces de ligne de commande sont propriétaires, ils ne peuvent pas être utilisés de manière efficace pour automatiser les processus dans un environnement ayant un ensemble hétérogène d'équipement.

— Les installations de contrôle d'accès sont parfois absentes, et quand elles existent sont insuffisantes.

CLI est l'interface la plus simple à réaliser et conserve de nombreux avantages par rapport aux environnements graphiques, notamment sur la précision et la simplicité

d'automatisation des tâches (mode batch) que sur le contrôle à distance, l'uniformité, et la stabilité et surtout consomme peu de ressources.

De plus, CLI offre aussi des moyens à un utilisateur d'afficher des données diverses sur l'état courant du dispositif, utilisant les commandes appelées « show », appelées de cette façon en langage Cisco parce que beaucoup d'opérations de lecture sur une configuration peuvent être transformées en opérations d'écriture en les préfixant par le mot-clé « show ».

2.2.2 FTP

Une des manières les plus simples pour gérer les configurations consiste à transférer la configuration comme des fichiers à travers une connexion FTP. Dans un tel scénario, chaque dispositif agit comme un serveur FTP vers lequel les configurations sont tirées ou poussées comme de simples fichiers et tout changement apporté à la configuration est appliqué immédiatement. Un tel mode de fonctionnement n'offre presque qu'aucune possibilité de vérification d'erreur : un utilisateur peut facilement écraser une configuration qui marche avec un fichier contenant des erreurs de syntaxe ou de fausses valeurs de paramètre, paralysant ainsi le fonctionnement du dispositif. Cependant, il offre souvent une méthode de recours pour agir sur un dispositif, au cas où les méthodes de niveau plus élevé échouent.

Quand on associe CLI et FTP, on peut se connecter directement à l'équipement et lui envoyer une configuration. CLI permet l'approche d'externalisation du modèle de sécurité (SSH) et le côté orienté configuration que monitoring. CLI permet la modification de l'état du réseau, par la configuration de ses composants, mais ne permet pas de détecter les erreurs de configuration lors d'une configuration « running »(en cours de production),

et d'une configuration « startup »lancée au démarrage de la machine. Mais CLI dispose d'une configuration candidate qui permet de travailler sur une configuration hors-ligne de la valider avant de la mettre en production (Zuccarelli et Laouenan, 2010).

2.2.3 SNMP

Comme on vient de le voir, les protocoles TELNET et FTP étaient utilisés pour des connections distantes aux équipements, mais ces deux protocoles n'offraient pas une vue synthétique de l'infrastructure et ne séparaient pas correctement les deux métiers différents que sont la supervision et l'administration réseaux (Enns et al., 2006) d'où la mise sur pied du protocole SNMP.

L'acronyme SNMP pour «Simple Network Management Protocol» est un protocole standardisé par l'IETF [3] qui permet aux administrateurs réseau de pouvoir gérer et de diagnostiquer les problèmes sur les équipements du réseau (Harrington et al., 1999). L'acronyme SNMP est généralement utilisé pour désigner un ensemble de spécifications incluant le protocole lui-même, la définition d'un modèle de l'information, d'une base de données correspondante et enfin des concepts associés.

Il faut noter qu'il existe trois versions de SNMP (Wijnen et al., 2000; Stallings, 1999). La version 1 notée SNMPv1 qui est la plus ancienne, mais toujours utilisée dont le défaut de sécurité a contribué à la mise en place de la version 2 notée SNMPv2 qui essaye de combler cette lacune de sécurité (Willm, 2005). Mais, faute d'un consensus au niveau des groupes de travail de l'IETF, c'est la version intermédiaire et expérimentale connue sous le nom de SNMPv2C (avec C comme nom de communauté) qui est utilisée par la plupart des éditeurs (Interpeak, 2005). Dans cette seconde version la sécurité

3. Internet Engineering Task Force

est encore quasiment nulle car elle reprend tout simplement le modèle de la version 1. Malgré tout, cette version intermédiaire comble tout de même certaines lacunes de la version 1 notamment sur la façon de définir les objets, le traitement des notifications et du protocole lui-même (avec l'ajout d'une commande GERBULK pour minimiser les échanges réseau qui sont particulièrement lourds lors de la récupération de tables avec les commandes GETNEXT de SNMPv1). Cette version intermédiaire n'a pas eu de déploiements importants sur les réseaux, faute de progrès conséquents par rapport à la version originale (Wijnen et al., 2000; Stallings, 1999).

La version 3 notée SNMPv3 apporte essentiellement des fonctions de sécurité en plus de formaliser de façon complète le modèle d'administration SNMP. La version 3 se veut être le standard mais n'est pas encore approuvée par le marché (Willm, 2005). l'un des objectifs de SNMP est de permettre autant que possible l'indépendance entre l'architecture et les mécanismes de certains hôtes ou passerelles particulières (Fedor et al., 1990; Wijnen et al., 1999) (Wijnen et al., 2000; Stallings, 1999).

Comme tout système de gestion de réseau, le protocole SNMP est basé sur trois composants ou entités à savoir :

— le superviseur qui est logé sur les systèmes d'administration,

— les agents qui sont logés sur les systèmes administrés,

— une entité base de données appelée MIB ou Management Information Base qui regroupe et gère les informations.

Pour pouvoir communiquer, ces entités utilisent un ensemble de protocoles, notamment le protocole de transport UDP (Peret et al., 2005; Presuhn et al., 2002).

Le superviseur ou manager est la console qui permet à l'administrateur réseau d'exécuter les requêtes de gestion ; il s'agit d'une machine centrale à partir de laquelle un opérateur

humain peut superviser en temps réel toute son infrastructure réseau afin de diagnostiquer et faire intervenir un technicien pour résoudre les problèmes identifiés (Fedor et al., 1990; Peret et al., 2005; Presuhn et al., 2002).

Les agents sont des entités qui se trouvent au niveau de chaque interface qui permet de connecter l'équipement à administrer à distance, ces agents permettent de récupérer des informations sur différents objets manageables. C'est une application de gestion de réseau résidant dans un périphérique et chargée de transmettre les données locales de gestion du périphérique au format SNMP.

Les objets manageables peuvent être gérés à distance comme ceux contenus dans certains équipements tels que les commutateurs intelligents, hubs, routeurs et serveurs. Les objets manageables peuvent aussi faire référence aux informations matérielles, aux paramètres de configuration, aux statistiques de performance et à d'autres objets qui sont directement liés au comportement en cours de l'équipement en question. Une requête est envoyée à l'aide de la commande `Get` par le gestionnaire pour demander ou modifier les valeurs à l'aide de la commande `Set` d'un objet MIB associé à une ressource. Il est également possible pour un agent d'envoyer des messages non sollicités appelée `Trap` pour avertir le superviseur d'un évènement, par exemple dans le cas où la congestion atteint un certain niveau.

La MIB ou Management Information Base est sans doute l'élément le plus important du protocole SNMP, qui a permis de décrire un grand nombre de composants (réseaux ou autres) de façon standard (Willm, 2005). Les informations gérées par les agents sont structurées par une base de données reprenant toutes les informations utiles concernant l'équipement où est installé l'agent. Cette base de données collectionne toute l'expérience des spécialistes qui ont établi les modèles concernant les sujets qu'ils maîtrisent, ce qui

```
DEFINITIONS ::= BEGIN

    IMPORTS
        mgmt, OBJECT-TYPE, NetworkAddress, IpAddress, Counter, Gauge,
        TimeTicks
          FROM RFC1065-SMI;

    mib     OBJECT IDENTIFIER ::= { mgmt 1 }

    system       OBJECT IDENTIFIER ::= { mib 1 }
    interfaces   OBJECT IDENTIFIER ::= { mib 2 }
    at           OBJECT IDENTIFIER ::= { mib 3 }
    ip           OBJECT IDENTIFIER ::= { mib 4 }
    icmp         OBJECT IDENTIFIER ::= { mib 5 }
    tcp          OBJECT IDENTIFIER ::= { mib 6 }
    udp          OBJECT IDENTIFIER ::= { mib 7 }
    egp          OBJECT IDENTIFIER ::= { mib 8 }
    END
```

Figure 2.3: Extrait d'un fichier MIB

en fait tout l'intérêt pour les administrateurs réseau. La gestion de l'information est représentée avec des objets, un pour chacun des aspects du périphérique de gestion. Le modèle n'est pas un modèle objet : les entités modélisées ne sont pas des objets au sens informatique du terme, mais plutôt un ensemble de variables typées qui peuvent être lues ou mises à jour. Un certain nombre d'astuces sont utilisées ensuite pour permettre de définir des opérations complexes sur ces objets en utilisant les mécanismes définis dans la Structure des informations de gestion (SMI)(Waldbusser et al., 2002a) (Waldbusser et al., 2002b). Il faut noter ici que chaque type d'appareil a son propre MIB, par exemple «Printer MIB» (RFC 1759) ou «UPS MIB» (RFC 1628). La figure 2.3 présente un extrait d'un fichier MIB qui permet de collectionner les informations réseaux (L'adresse du réseau, l'adresse IP etc.). Notre extrait de fichier MIB permet de collectionner les informations réseaux (L'adresse du réseau, l'adresse IP etc.) d'un périphérique. Les

définitions d'objet SNMP qui sont des nœuds de l'arbre (`mgmt`, `system`, `interfaces`, `at`, `ip` etc.) utilisent la clef OBJECT IDENTIFIER et non pas la clef OBJECT-TYPE. NetworkAddress est un type défini qui peut représenter une ou plusieurs familles de protocole Counter, gauge et timeticks représentent des valeurs des syntaxes.

— L'objet mib est attaché à l'objet mgmt avec l'index 1

— L'objet system est attaché à l'objet mib avec l'index 1

— L'objet interfaces est attaché à l'objet mib avec l'index 2

— L'objet at est attaché à l'objet mib avec l'index 3

— L'objet ip est attaché à l'objet mib avec l'index 4

— L'objet icmp est attaché à l'objet mib avec l'index 5

— ...

Il faut préciser ici qu'on n'a pas besoin d'un MIB pour utiliser SNMP ou pour effectuer des requêtes sur des périphériques SNMP mais sans la MIB, on ne pourra pas savoir facilement la signification des données retournées par le périphérique. Dans certains cas, c'est facile comme le nom de l'hôte, l'usage des disques ou les informations d'état des ports. Dans d'autres cas, cela peut être plus difficile et c'est là où une MIB peut être d'une grande aide.

D'après (Fedor et al., 1990) (Pignet, 2007) (Interpeak, 2005) le protocole SNMP est constitué de plusieurs commandes ou messages présentés ci-dessous :

— `Get_Request` : Cette commande, envoyée par le manager à l'agent pour lui de demander une information de la base de données. Celui-ci valide l'information si sa validité est confirmée, renvoie au manager la valeur correspondant à l'information demandée.

— `Get_next_Request` : Elle permet de balayer toute la MIB de l'agent. Cette commande est envoyée par le gestionnaire à l'agent pour lui demander la prochaine information (Il peut s'avérer nécessaire pour l'agent de parcourir toute une liste de variables.) On utilise cette commande à la suite d'une requête «get» afin d'obtenir directement le contenu de la variable suivante.

— `Set_Request` : Cette commande est envoyée par le superviseur à l'agent, dont l'objectif est de définir la valeur d'une variable de l'agent administré. Cela permet des modifications sur le matériel. Un message Set_Request est toujours suivi d'un message Get_Response

— `Getbulk` : Cette commande, est envoyée par le manager à l'agent pour connaître la valeur de plusieurs variables : cela évite d'effectuer plusieurs requêtes Get en série, améliorant les performances (implémenté dans SNMPv2)

— `Trap` : Lorsqu'un événement particulier survient chez l'agent (connexion, modification de la valeur d'une variable donnée, etc...), celui-ci est susceptible d'envoyer ce que l'on appelle une « trap », à savoir un message d'information destiné à la station d'administration : celle-ci pourra alors la traiter et éventuellement agir en conséquence. S'il s'agit par exemple de la coupure d'un lien réseau, cela permet à l'administrateur réseau d'en être immédiatement informé.

— `Inform` : Cette commande est utilisée dans le but d'obtenir une confirmation de la réception et l'analyse après l'envoi d'une « trap» par l'agent elle est implémentée dans SNMPv2 (Interpeak, 2005).

— `Get_Response` : Message envoyé par l'agent au superviseur après l'émission d'une requête. Sans requête au préalable l'agent ne peut pas émettre un tel message

2.2.4 AVANTAGES ET INCONVÉNIENTS

SNMP a été l'une des premières solutions proposées dans son domaine pour effectuer la gestion centralisée des réseaux et cette nouvelle fonction ne s'est pas faite sans difficultés car elle comporte certaines faiblesses et naturellement quelques avantages que nous allons énumérer ici.

Comme avantages, nous pouvons dire que le protocole SNMP permet et facilite une certaine interopérabilité comparativement à ses prédécesseurs, grâce à l'utilisation d'une syntaxe unique pour tous les équipements. Un autre avantage d'utiliser SNMP est son aisance d'implémentation sur un réseau ; grâce à sa conception simple il ne nécessite pas beaucoup de temps de travail ni de compétences particulières (Schrieck, 1998; Willm, 2005).

Comme premier inconvénient pour SNMP, nous pouvons mentionner : la congestion du réseau. En effet, dans les réseaux de grande taille il a été noté que pour obtenir un grand nombre d'informations sur un équipement il faut aussi envoyer un grand nombre de requêtes, ce qui va entraîner une surcharge du trafic SNMP (Schrieck, 1998). Alors que SNMP fournit des performances raisonnables pour la récupération d'une petite quantité de données à partir de plusieurs appareils, il devient plutôt lent lors de la récupération de grandes quantités de données (telles que la table de routage) à partir de quelques appareils (Schoenwaelder, 2003).

SNMP étant basé sur les services du protocole UDP en mode déconnecté, de ce fait il peut arriver que les paquets « Trap » n'arrivent pas à destination. Dans ce cas, le temps d'attente de la réponse va s'écouler et il faudra réémettre la requête. Suivant l'implémentation des agents et la version de SNMP utilisée, si l'authentification échoue

(mauvaise communauté, mot de passe incorrect), l'agent peut ne pas répondre à la requête (Schrieck, 1998; Willm, 2005). Malgré le fait que le temps d'attente de la réponse peut être paramétré dynamiquement, il est possible que le temps défini soit trop court pour permettre le retour de la réponse après un premier échec. Le Manager doit donc surveiller son environnement en procédant à des interrogations régulières de ses agents : c'est ce qu'on appelle le *polling* (Schrieck, 1998) (Pignet, 2007).

SNMP ne permet pas facilement la récupération et la lecture des configurations, deux raisons peuvent l'expliquer : la première est qu'il n'est pas facile d'identifier des objets ou des paramètres de configuration avec SNMP. Pour la seconde, le système de nomenclature est très spécifique et les reconfigurations physiques de dispositifs peuvent ainsi détériorer ou empêcher la capacité d'accéder aux configurations antérieures. (Schoenwaelder, 2003).

Plusieurs modules MIB standardisés n'ont pas une description de procédures de haut niveau. Il n'est donc pas toujours évident à la lecture des modules MIB de savoir comment certaines tâches de haut niveau doivent être accomplies, ce qui conduit à avoir pour un même objectif plusieurs solutions différentes, d'où une augmentation du coût et l'interopérabilité tant recherchée serait ainsi entravée (Schoenwaelder, 2003). En plus, le modèle d'information qui définit les MIB demeure globalement limité malgré les possibilités d'extensions (Schrieck, 1998). SNMP n'est donc pas adapté à la gestion des configurations, et ne permet pas le traitement hiérarchique de l'information de configuration (Wallin et Wikström, 2011) ce qui permettrait de séparer les classes d'objets.

2.2.5 NETCONF

Dans les dernières années, plusieurs solutions ont vu le jour dans le but d'améliorer et de corriger les lacunes dans le domaine de la gestion des configurations réseau. Ces solutions étaient soient partiellement adaptées, spécifiques à chaque fabriquant ou tout simplement orientées vers un seul aspect particulier de la problématique. Il a fallu attendre l'avènement du protocole NETCONF pour voir les choses changer.

En effet, c'est en juin 2002 que l'organisme dénommé Internet Architecture Board (IAB) a tenu une réunion sur la gestion des réseaux (Schoenwaelder, 2003) sous l'égide de l'IETF. À l'issue de cette réunion un groupe de travail a été constitué, afin de trouver des solutions face aux problèmes rencontrés au quotidien par les opérateurs de réseau dans l'exercice de leur travail.

Ce groupe de travail a formulé un certain nombre de recommandations notamment sur les transactions, la restauration, la réduction des coûts de mise en œuvre et la capacité de sauvegarder et restaurer les données de configuration des équipements. Ce groupe de travail a aussi fait ressortir les insuffisances des anciennes solutions telles que SNMP qui présente des lacunes sur l'écriture de données, le manque de couverture complète des capacités des équipements, l'impossibilité de faire la distinction entre les données de configuration et d'autres types de données (Shafer, 2011).

Sur la base donc de ces manquements, un groupe de travail appelé NETCONF a été formé et le protocole qui porte son nom a ainsi vu le jour (Shafer, 2011). NETCONF est donc un protocole proposé par l'IETF, qui fournit un ensemble d'opérations de diagnostic, de manipulation et de suppression de configuration dans les équipements réseau (Halle et al., 2004; Aitken et al., 2012; Enns et Ed, 2006). Il a été conçu principalement pour

couvrir les insuffisances du protocole SNMP ou (Simple Network Management Protocol) et de CLI (Command-Line Interface) dans les mécanismes de configurations réseaux. Il fait donc office de propulseur dans la gestion de configuration des réseaux et est le premier protocole à être totalement indépendant des équipements. D'après (Zuccarelli et Laouenan, 2010) c'est un protocole de gestion qui permet de fournir un moyen d'effectuer la gestion de configuration d'équipement réseau grâce à XML qui permet un mécanisme flexible d'encodage hiérarchique (Mi-lung et al., 2004; Enns et Ed, 2006; Enns et al., 2006; Halle et al., 2004).

NETCONF est donc basé sur le langage XML permettant de coder à la fois les messages du protocole et les données de configuration suivant un modèle appelé YANG (Bjorklund, 2010) que nous présenterons à la section suivante. Il permet, par exemple, à un utilisateur du protocole NETCONF de pouvoir envoyer à un routeur Juniper les mêmes commandes qu'il enverrait à un routeur Cisco : on parle alors d'indépendance de vendeur (Halle et al., 2004; Enns et al., 2011b).

NETCONF structure ses modèles de données de façon à ce qu'ils soient plus proches de l'architecture réelle de la configuration des équipements du réseau. Plus spécifiquement, NETCONF doit pouvoir fournir une interface à l'appareil qui soit la plus proche possible de son interface native. NETCONF prévoit des procédures d'installation, de manipulation, et de suppression de la configuration des périphériques réseaux. Il fonctionne en mode client-serveur. Un serveur ou agent dont le rôle est de recevoir, d'exécuter et de fournir une réponse aux requêtes des clients encore appelés managers. Le `serveur` étant le logiciel résidant sur l'équipement à gérer et le `client` (manager) le logiciel résidant sur la machine centrale qui gère les configurations des équipements (Enns et al., 2006, 2011b). La connexion entre le serveur et le client est appelée session NETCONF. Un appareil doit être capable de supporter au moins une session à la fois.

NETCONF utilise un ensemble d'appels de procédure distante (appelé en anglais RPC ou Remote Procedure Call) pour envoyer des commandes de configuration à un routeur à travers une session en mode connectée et sécurisée (Halle et al., 2004), contrairement à SNMP dont les informations sont envoyées en clair sur le réseau (Trevino et s Chisholm, 2008). D'une façon simplifiée, un RPC est un bloc de données XML dont la balise d'ouverture contient un identifiant qui demande au routeur de retourner une partie de son fichier de configuration, ou de remplacer une partie de sa configuration avec un petit bout de code fourni par l'utilisateur et porté dans le corps de la RPC (Halle et al., 2004).

NETCONF offre d'autres opérations incorporées, comme des commandes permettant de bloquer une partie de la configuration du routeur afin que, seul l'utilisateur courant puisse l'ouvrir et la modifier par la suite.

Le protocole NETCONF est bâti suivant 4 couches à savoir la couche contenu, la couche opération, la couche RPC et la couche du protocole applicatif comme présentées sur la figure 2.4 tirée de (Enns et Ed, 2006).

1. La couche de transport sert à acheminer les messages entre le client et le serveur de façon sécurisée grâce au protocole de communication sécurisé comme SSH.
2. La couche message utilise la procédure RPC pour fournir un mécanisme d'envoi indépendant de la couche transport.
3. Un ensemble d'opérations de base existe et les paramètres sont encodés en XML.
4. Le contenu est spécifique à la représentation de la configuration qui est arbitrairement choisi lors de l'implémentation à l'aide de YANG (Enns et al., 2011b).

Nous présentons ici quelques opérations de base du protocole NETCONF telles que décrites dans la RFC 6241. Selon la RFC 6241, il est important de considérer que toutes

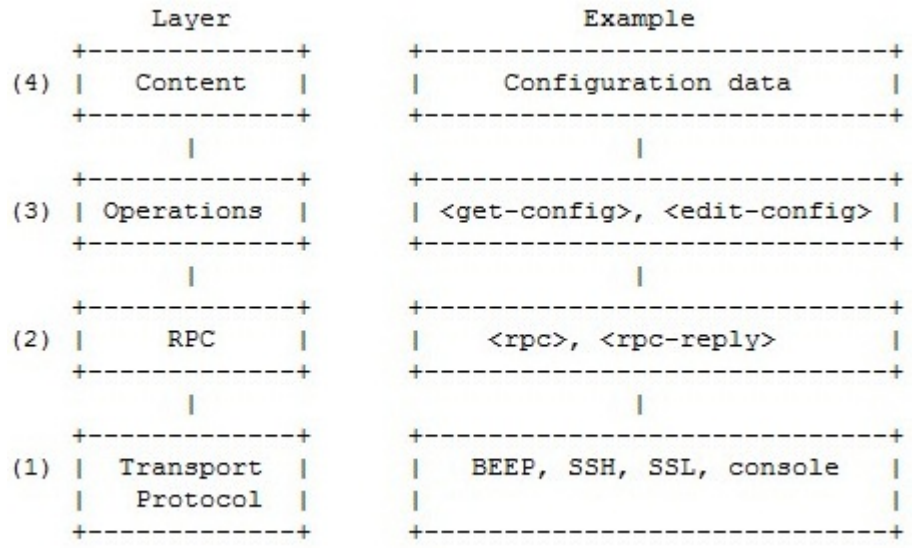

Figure 2.4: Les 4 couches du protocole NETCONF (tirée de la RFC4741)

les opérations ne seront pas forcément un succès et qu'il faille traiter la réponse pour vérifier si elle est positive, par exemple une erreur résultera en une balise <rpc-error> contenue dans la réponse <rpc-reply>.

— ***Get***

Cette opération permet la récupération de la configuration active ainsi que les informations sur le statut de l'équipement. On peut aussi utiliser le paramètre «filter», un filtre permettant de restreindre la partie de la configuration à récupérer (sinon toute la configuration sera envoyée).

— ***Get-config***

L'opération get-config autorise 2 paramètres : le premier, nommé « source », permet de spécifier le nom de la configuration que l'on veut récupérer ;le second se nomme « filter » et est le même utilisé par get.

— ***Edit-config***

Le rôle de cette opération est de charger une partie ou une nouvelle configuration

dans l'équipement. On peut choisir de fusionner, remplacer, supprimer ou créer une configuration. Il est possible de tester l'action que l'on va effectuer en spécifiant le paramètre «test» et en regardant la réponse. Les erreurs sont gérées avec l'option « error-option » où l'on peut spécifier dès qu'une erreur est détectée des actions à effectuer, à savoir l'arrêt de l'opération, la poursuite de l'opération et le «rollback» qui permet de revenir à l'état avant l'opération.

— ***Copy-config***

A la différence d'edit-config cette opération agit sur l'ensemble d'un datastore [4] de configuration, en créant ou remplaçant la totalité de la configuration. En raison de son caractère critique, un appareil peut choisir de ne pas permettre l'application de cette opération sur la configuration courante.

— ***Delete-config***

Permet de supprimer une configuration, à l'exception de la configuration active ou courante qui ne peut être supprimée contrairement à celle utilisée lors du démarrage ou celle permettant de modifier et tester une configuration sans perturber le fonctionnement de l'appareil.

— ***Lock et unlock***

Permet de verrouiller la configuration que l'on passe en paramètre afin qu'aucune autre tentative de modification ne vienne interférer (utilisation par une autre session NETCONF, un administrateur avec l'interface en ligne de commande).Unlock permet de débloquer la session si elle ne l'est pas encore suite à un timeout ou un unlock implicite (coupure de la session brutale par exemple). Seul un client NETCONF qui a verrouillé une configuration peut demander à la déverrouiller.

— ***Close-session***

4. Fichiers contenant chacun une version spécifique de la configuration d'un appareil

Permet de terminer normalement une session NETCONF. Après une requête close-session, le serveur va relâcher les verrous en cours et n'acceptera plus de nouvelles requêtes dans session.

— ***Kill-session***

Permet de terminer une session NETCONF. Toutes les opérations du serveur en cours sont arrêtées, les verrous sont relâchés et si un commit était en cours, un roll back est effectué pour revenir à l'état antérieur.

L'exemple d'échange de messages suivant est tiré de (Schrieck, 1998), il permet de montrer l'utilisation de la balise <rpc>. Dans ce cas, l'absence de l'attribut message-id déclenche l'envoi d'un message d'erreur signalant le problème. Ce message d'erreur est contenu dans une balise <rpc-reply> voir figure 2.5

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
     <source>
        <running/>
     </source>
   </get-config>
</rpc>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <rpc-error>
        <error-type>rpc</error-type>
        <error-tag>MISSING_ATTRIBUTE</error-tag>
        <error-severity>error</error-severity>
        <error-info>
                <bad-attribute>message-id</bad-attribute>
                <bad-element>rpc</bad-element>
        </error-info>
   </rpc-error>
</rpc-reply>
```

Figure 2.5: Utilisation de la balise <rpc> pour signaler une erreur

2.2.6 YANG

YANG est un langage de *modélisation* de données utilisé pour la modélisation des configurations et des données d'état manipulées par le protocole de configuration réseau NETCONF d'où son nom en anglais Data Modeling Language. YANG a été créé par le groupe de travail NETCONF de l'IETF NETMOD spécifiquement pour NETCONF (Bjorklund, 2010).

YANG modélise l'organisation hiérarchique des données comme un arbre dans lequel chaque nœud a un nom et une valeur et peut avoir un ensemble de nœuds enfants. YANG présente une description claire et concise des nœuds ainsi que l'interaction entre ces nœuds (Bjorklund, 2010). Il structure ses modèles de données dans des modules et des sous-modules. Un module peut importer des données d'autres modules externes et inclure ces données dans des sous-modules. Les modèles de données décrit par YANG sont conçus pour être facilement utilisés par NETCONF. Une fois publié, le module YANG agit comme un contrat entre le client et le serveur et chaque partie sait exactement quel sera son rôle. Un client sait comment créer les données valides pour le serveur, et sait également quelles données seront envoyées au serveur. Le serveur à son tour connaît les règles ou contraintes qui régissent ces données et la façon dont elles devraient se comporter (Shafer, 2011). Selon (Bjorklund, 2010), la hiérarchie peut être augmentée, permettant à un module d'ajouter des nœuds de données à la hiérarchie définie dans un autre module. Cette augmentation peut être conditionnelle, avec de nouveaux nœuds apparaissant seulement si certaines conditions sont remplies, ce qui fait de YANG un langage extensible. C'est cette flexibilité qui permet à NETCONF couplé avec YANG une certaine interopérabilité (Bjorklund, 2010). Il faut rappeler ici d'après la RFC2578 et RFC2578, que YANG maintient dans la mesure du possible, la compatibilité avec le

protocole SNMP, SMIv2 (Structure of Management version 2).

Comme NETCONF, YANG vise l'intégration harmonieuse avec l'infrastructure native des équipements. Cela permet aux implémentations de tirer parti de leurs mécanismes de contrôle d'accès existants pour protéger ou exposer les éléments du modèle de données. (Bjorklund, 2010). Selon (Bierman, 2008) c'est YANG qui est utilisé dans la couche de données ainsi que dans la définition des données NETCONF au sein des agents. YANG utilise plusieurs types pour sa structure de données, nous pouvons citer entre autre le type boolean, string, Unint32 etc. YANG dispose d'un ensemble de types intégrés, semblables à celles de nombreux langages de programmation, mais avec quelques différences dues en particulier aux exigences du domaine de gestion. Il faut savoir que si un type qu'on veut utiliser n'existe pas Yang vous permet de le construire vous-même (Bjorklund, 2010). En conclusion nous pouvons dire que YANG possède des qualités ou avantages certaines, il est simple à lire et à apprendre, il est écrit pour NETCONF et la gestion des réseaux, il est extensible et surtout possède une grande communauté technique ouverte aux nouveaux apprenants.

2.3 LA GESTION AUTOMATISÉE

On peut définir la gestion automatique des configurations comme l'ensemble des règles et des moyens destinés à gérer et garantir le suivi, la cohérence de la configuration d'un ensemble fonctionnel et de ses différents composants au cours de son évolution. Le degré d'intervention humain peut être faible (on parle alors de gestion semi-automatique), voire inexistant.

Dans un système de gestion automatique l'administrateur ne doit pas se poser la question de savoir : comment gérer par exemple les fichiers de journaux sur un quelconque

système ?. Certes l'homme est meilleur pour penser, mais la machine est meilleure pour la répétition, ce qui signifie en d'autres termes que l'homme doit s'occuper de la conception de son infrastructure et laisser la machine, par l'intermédiaire de l'outil, s'occuper de l'exécution des procédures (Zamboni, 2012).

Bien sûr l'homme doit s'occuper des tâches manuelles nécessaires une à deux fois, avant de comprendre exactement ce qui doit être fait par lui. Après tout, un ordinateur ne serait pas capable de comprendre par lui-même dans bien des cas par exemple : sélectionner les paramètres que l'on doit mettre dans le fichier de configuration sshd, ou pour écrire un script de sauvegarde d'un poste de travail vers un disque externe à chaque branchement. Ou encore suivre les étapes de mise à jour des fichiers de configuration des nouveaux équipements (Zamboni, 2012). Cependant, si on a complété toutes les étapes, il n'y a donc plus de raison de continuer à le faire manuellement, la machine pouvant répéter ces étapes dans le bon ordre et au bon moment (Zamboni, 2012). D'ailleurs, une des raisons principales qu'ont les administrateurs système pour automatiser la configuration de leurs appareils est d'éviter les erreurs en minimisant les tâches répétitives. D'après (Delaet et al., 2011) un outil de configuration système qui prend en charge la modularisation des morceaux de configuration permet de réduire la répétition dans la spécification de configuration.

Une telle approche présente de nombreux avantages. D'abord, elle permet une reproductibilité de ce qui a déjà été fait et qui est fonctionnel. En d'autres termes, la gestion automatique permet de mettre sur pied une bonne documentation de tous les équipements ainsi qu'une journalisation de tous les incidents déjà survenus permettant ainsi de capitaliser les connaissances. Avec une configuration automatique, les mises à jour des fichiers de configuration, la gestion des différentes versions de ces fichiers ainsi que l'ajout et la configuration d'un nouvel équipement se font de façon automatique et transparente.

Enfin, la gestion automatique permet de fournir des alertes sur les incidents en temps réel et, dans le cadre des systèmes autonomiques permet automatiquement de les résoudre (ManageEngine, 2012).

Aujourd'hui plusieurs normes et recommandations reconnues comme ITIL, ISO 9000 ou encore COBIT insistent sur la mise sur pied d'une gestion de configuration dans les entreprises (Clarke, 2012). Selon (Gartner, 2012) 40% des erreurs dans les centres de données sont des erreurs opérationnelles et la mise en place d'une automatisation pourrait réduire ces erreurs d'au moins 35%.

2.4 OUTILS DE GESTION AUTOMATISÉE DE CONFIGURATION

Selon (Delaet et al., 2011) lorsque l'administrateur décide d'adopter un outil de gestion de configuration, cela implique un investissement important en termes de temps ou d'argent. Mais avant de faire un tel investissement, il a besoin de savoir si l'outil choisi est compatible avec l'infrastructure. Par conséquent, il est primordial de pouvoir connaitre les critères de comparaison des solutions existantes avant toute décision.

Depuis plusieurs années on assiste à l'émergence et la floraison des outils de gestion automatique des configurations. Leur nom exact dépend de leurs concepteurs : certains les appellent « outils de gestion », d'autres « outils de gestion de salles machines », etc. (Zamboni, 2012). Nous les appellerons outils de gestion automatisée de configuration (GAC).

Il convient de préciser ici que la plupart des outils de configuration automatique de système fournissent une interface qui permet à l'administrateur de spécifier le type de configuration qu'il souhaite administrer pour un type d'équipement, ensuite l'outil

utilise cette spécification en entrée et l'applique sur tous les équipements du même type à gérer. Chaque équipement possède un agent de traduction qui est le composant chargé de traduire l'entrée de l'administrateur en profil de configuration compréhensible par l'outil qui permet la communication entre lui et le serveur (Delaet et al., 2011). Cet agent de traducteur est considéré comme le serveur et l'équipement à gérer comme l'hôte ou client selon les outils. Comme présenté dans la figure 2.6 ci-dessous, on distingue deux entités :

— Serveur : c'est l'application installée sur la machine qui héberge les configurations. C'est à partir du serveur qu'on peut pousser un changement de configuration sur les autres équipements du réseau.

— Hôte : chaque hôte va établir une connexion avec le serveur pour valider sa configuration c'est à dire comparer sa version avec celle qui est sur le serveur et s'assurer qu'il n y a pas eu de modification sinon elle met à jour la sienne à partir de la copie logée sur le serveur.

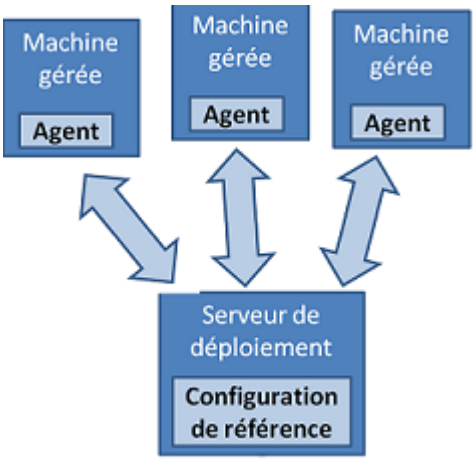

Figure 2.6: Présentation du fonctionnement global d'un outil de gestion automatique de configuration

Après avoir présenté une liste des différents outils existants sur la gestion automatisée des configurations en page 63, nous allons ensuite faire une analyse complète de ces

outils en terme de fonctionnalités et avantages. En annexe 1 nous avons présenté un tableau comparatif tiré des études faites par (Delaet et Vanbrabant, 1999; Narain, 2005).

En termes d'outils nous pouvons citer :

CFEngin	(officiel de CFEngine, 2012)
Alloy	(Jackson, 2002, 2012)
Chef	(Opscode, 2008)
ManageEngine	(ManageEngine, 2012)
Configgen	(Dyson, 2012)
Snitch	(Hander et al., 2012)
Coolaid	(Chen et al., 2010a,b)
2solve	(2operate, 2012)
Edge	(EarthSoft, 2012)
Trigger	(AOL Inc, 2012)
Fabric	(Hansen et Forcier, 2013)
Puppet	(Anderson et Scobie, 2012)
Rudder	(Perron, 2012)
Bcfg2	(Software, 2012)
Ltl Model Checking	(Clavel et al., 2007; Berard et al., 2010) (Clarke et al., 2000)
Config Checker	(Caprica Ltd, 2009)
Whatsconfigured	(Ipswitch, 2013)
IBM Tivoli, Netomata config generator	(netomata, 2010)
Ca Network and Systems Management	(PartnerWorld, 2012)
LCFG	(Anderson, 2008)
HP Serveur automation	(HP France, 2013)
Microsoft Server Center configuration Management	(Microsoft, 2012)
BMC Bladelogic Server automation Suite	(BMC Software, 2012)
Babble	(Couch, 2012, 2000)
Snitch	(Objective Development Software, 2013)
ManageEngine	(Objective Development Software, 2013)
CatTools Kiwi	(CatTools, 2012)
Snitch	(Mickens et al., 2007)

2.4.1 CFENGINE

CFEngine est l'un des outils les plus répandus et le plus populaire parmi les outils de gestion automatisée de configuration (Burgess, 1995). C'est un logiciel libre de gestion automatique de configuration, écrit en langage C, qui utilise pour ses spécifications d'entrées un langage de déclaration dont l'acronyme est DSL. Il utilise un langage de déclaration (Delaet et al., 2011) pour l'entrée de ses variables, ce qui lui permet de déclarer au préalable ses contraintes ou spécifications sur l'état de l'infrastructure, et lors de son exécution compare ainsi la configuration désirée et celle présente sur les équipements (Schönwälder et al., 2008).

Au point de vu architectural CFEngine fonctionne en mode client-serveur. Il est supporté par la plupart des systèmes existants de Windows à Unix en passant par Linux. Sur le client ; on trouve l'agent cf-agent qui va analyser et traiter les différentes configurations. On peut l'apparenter à un interpréteur. Sur le serveur on a deux agents importants ; cf-servd et cf-key :

— cf-servd : c'est lui qui a la responsabilité du partage des fichiers.

— cf-key : c'est lui qui permet de sécuriser au minimum les flux de données, et fonctionne à la manière du keygen d'OpenSSH.

Au point de vu fonctionnement, CFEngine permet de déployer des configurations sur les équipements d'un parc informatique, de synchroniser des fichiers sur des serveurs hétérogènes (différents Unix, Linux et Windows) et d'envoyer des commandes sur ces derniers. CFEngine permet donc à l'administrateur de s'affranchir de certaines tâches tout comme les autres outils de GAC, mais son principal atout est de regrouper et d'organiser les traitements selon les classes d'objets (Burgess, 1995).

L'un des avantages de CFEngine est aussi qu'il permet de contrôler et de coordonner la

distribution de charges des équipements ce qui n'est pas le cas de bien des outils. C'est une solution de gestion de configuration réseau qui permet l'automatisation des tâches d'administration, les notifications des sauvegardes des configurations d'équipement et des changements au sein de ces configurations. Il fournit également un moyen facile de pousser un changement de configuration sur tous les équipements du réseau (Zamboni, 2012). CFEngine peut-être utilisé quelque soit la taille de l'entreprise autant sur de petits réseaux que sur des réseaux de grande taille.

CFEngine est conçu de façon à ne pas lancer en boucle les commandes afin de ne pas saturer le système. Il est généralement exécuté périodiquement sur les hôtes. Lors de son démarrage, il récupère une instance de la définition des politiques ou des contraintes établies au niveau du serveur central (celui qui héberge les politiques établies) et tente d'adapter son système local à la politique définie.

CFEngine contient aussi un utilitaire (démon) appelé cfenvd, qui collecte les statistiques locales afin de détecter un comportement anormal (Schönwälder et al., 2008). Enfin CFEngine contient deux algorithmes qui analysent les données pour détecter les anomalies (Burgess, 2002).

Selon (Chuche, 2007), pour résoudre ce problème et permettre l'utilisation d'un seul fichier de configuration pour plusieurs systèmes, CFEngine implémente la notion de classes. Celles-ci permettent d'exécuter certaines parties des scripts sous certaines conditions définies par l'environnement ou par l'utilisateur, à la manière du '*if*' des langages de programmation. La figure 2.7 présente un fichier de configuration, composé de trois sections : control, shellcommands et tidy.

La première section permet de configurer le comportement de cfagent et en l'occurrence de fixer l'ordre d'exécution des actions : shellcommands et tidy.

```
# cat /var/cfengine/inputs/cfagent.conf
                control:
                actionsequence = ( shellcommands tidy )
                shellcommands:
    "/usr/bin/id"
                tidy:
   /tmp pattern=*~ recurse=inf age=2
```

Figure 2.7: Exemple d'un fichier de configuration

La deuxième section indique qu'il faut lancer la shellcommands (commande externe à CFEngine, à la manière de "system" en Perl) /usr/bin/id.

La troisième section stipule qu'il faut lancer l'action tidy (suppression) sur le répertoire tmp et sur les fichiers dont le nom se termine par ($\tilde{}$) $\sim$ (pattern=$*\tilde{}$), vieux de plus de deux jours (age=2), et en analysant également les sous-répertoires (recurse=inf). Pour exécuter ce fichier, il suffit de faire :

```
# cfagent -q -K
  cfengine::usr/bin/id: uid=0(root) gid=0(root) groups=0(root)
```

2.4.2 CATTOOLS KIWI

CatTools est une application qui permet d'automatiser la gestion des configurations des appareils tels que les routeurs, les commutateurs et les pare-feu (CatTools, 2012). C'est donc une solution de gestion de configurations d'équipements réseaux qui peut être utilisé pour plusieurs types de fabricants en occurrence Cisco, 3Com, Dell, Enterasys, Extreme, Fonderie, HP, Juniper et les équipements Nortel. Parmi les nombreuses tâches que peut effectuer cet outil, nous pouvons citer :

— L'exécution des sauvegardes des fichiers de configuration et l'envoi instantané d'emails de confirmation à l'administrateur.

— L'émission des commandes via TELNET ou SSH à de nombreux périphériques à la fois.

— La modification et les notifications des sauvegardes des configurations d'équipement et des changements au sein de ces configurations à des intervalles de temps prédéfinis.

— Le changement simultané des mots de passe des périphériques réseaux.

— L'automatisation des tâches d'administration

Il fournit également un moyen facile de pousser un changement de configuration sur tous les équipements de réseau. Au point de vue architectural, comme la plupart des autres outils de configuration, CatTools possède une partie serveur et une partie client appelé *Hôte*. Malheureusement CatTools est principalement orienté vers les systèmes d'exploitation Microsoft. Pour l'écriture de ses scripts personnalisés il utilise le langage VBscripts (langage de script de Visual Basic) (CatTools, 2012). CatTools offre une possibilité de créer son propre type de périphérique personnalisé et les fichiers de script de ce périphérique ne doivent pas être pris en charge par l'un des types prédéfinis. Selon l'entreprise, CatTool utilise les protocoles TELNET, SSH 1 et 2 pour se connecter et a un serveur TFTP incorporé pour pousser ses configurations sur les équipements.

L'utilisation de CatTools se fait en quatre étapes comme présentées ci-dessous :

1. If faut d'abord définir ses options de configuration telle que le courrier électronique en utilisant le menu de configuration. (En utilisant le menu Options / Setup)
2. Entrer les détails pour au moins un périphérique réseau. (En utilisant l'onglet Devices)

3. Créer une activité et associer un ou plusieurs équipements à cette activité (en utilisant l'onglet Activités)
4. Exécuter l'activité. Ceci peut être accomplie en utilisant soit le bouton "Run now" bouton ou le planificateur. Ces étapes sont détaillées dans le document d'aide officiel de CatTools (CatTools, 2012).

Son fonctionnement est aussi simple. Pour chaque nouvel équipement on doit d'abord le configurer manuellement ensuite copier le fichier de configuration sur le serveur. Après un intervalle de temps défini par l'administrateur CatTools va comparer les deux versions de sa configuration ; par défaut cette comparaison se fait au démarrage de CatTools.

CatTools utilise le concept de variables qui elles, peuvent être appliquées à certaines activités ou à des propriétés de champs. Les variables peuvent être utilisées dans les commandes pour saisir un paramètre ou modifier les activités de configuration. Les variables peuvent également être utilisées au sein des noms de fichiers définis comme les propriétés des activités, normalement pour les rapports ou des données capturées. Ces variables sont réglées à des valeurs concrètes dès que l'activité est lancée. En voici quelques exemples :

```
\%ctDeviceName variable contenant le nom de l'équipement
\%ctGroupName  variable contenant le nom du groupe
\%ctHostName variable contenant l'adresse d'hôte
```

CatTools utilise aussi des méta-commandes qui peuvent être utilisés au même niveau que les commandes, mais peuvent aussi servir à modifier les fichiers de configuration. Voici un exemple d'utilisation de la méta-commande %ctDB, pour mettre à jour dans le cadre d'une activité un champ directement dans une table de la base de données CatTools :

```
\%ctDB:tablename:fieldname:new field value
```

Pour ajouter un nouvel équipement dans CatTools selon (CatTools, 2012), deux fichiers sont nécessaires :

1. Le dossier type d'appareil (fichier .ini), qui définit les éléments suivants :
   - nom du type de dispositif,
   - ID du périphérique qui est l'identifiant de l'équipement,
   - les champs de valeurs par défaut de l'interface utilisateur qui sont affichés sur l'équipement lorsqu'on ajoute ou lorsqu'on met à jour l'équipement.
2. Le fichier de script du dispositif (.txt), qui contient le code spécifique du type de dispositif devant permettre à CatTools de se connecter à l'appareil, d'entrer et sortir des différents modes, d'effectuer différentes activités (sauvegardes de configuration, par exemple, envoyer des commandes CLI, modifier la configuration) et d'analyser les commandes et les données de sortie pour produire un rapport (CatTools, 2012).

Le format de fichier d'importation des périphériques doit commencer par une en-tête qui définit les champs. Les lignes qui suivent l'en-tête sont traitées comme des données, comme le montre la figure 2.4

Name	Type	Group	HostAddress	Filename	Model	ConnectVia	Telnet	TelnetPort
Cisco Router 2600	Cisco.Router.General	Test Group	127.0.0.1	Cisco_Router_2600	Cisco 2600	Direct connect	Telnet	23

Figure 2.8: Format de fichier périphérique

Au minimum, le fichier doit contenir les 3 domaines suivants :

• Type (Le type de périphérique CatTools)

• Nom (Un nom unique pour le périphérique)

• HostAddress (L'adresse IP ou le nom d'hôte de l'appareil)

Selon l'entreprise, CatTools est un logiciel graphique dont les activités représentent le cœur de son utilisation, en effet ce sont elles qui servent à automatiser les actions.

Une activité est une tâche interne qu'effectue CatTools sur un ou plusieurs périphériques du réseau qui ont été introduit dans la base de données de CatTools (CatTools, 2012). Elle peut être programmée pour s'exécuter à des moments précis à l'aide de la minuterie ou être exécutée directement. Son exécution permet de lancer le client qui va interroger le périphérique afin d'obtenir les informations nécessaires pour compléter la tâche. Elle va ainsi collectionner toute l'information retournée et l'analyser ou tout simplement la stocker pour une utilisation ultérieure.Un rapport sur les résultats sera envoyé par e-mail (CatTools, 2012).

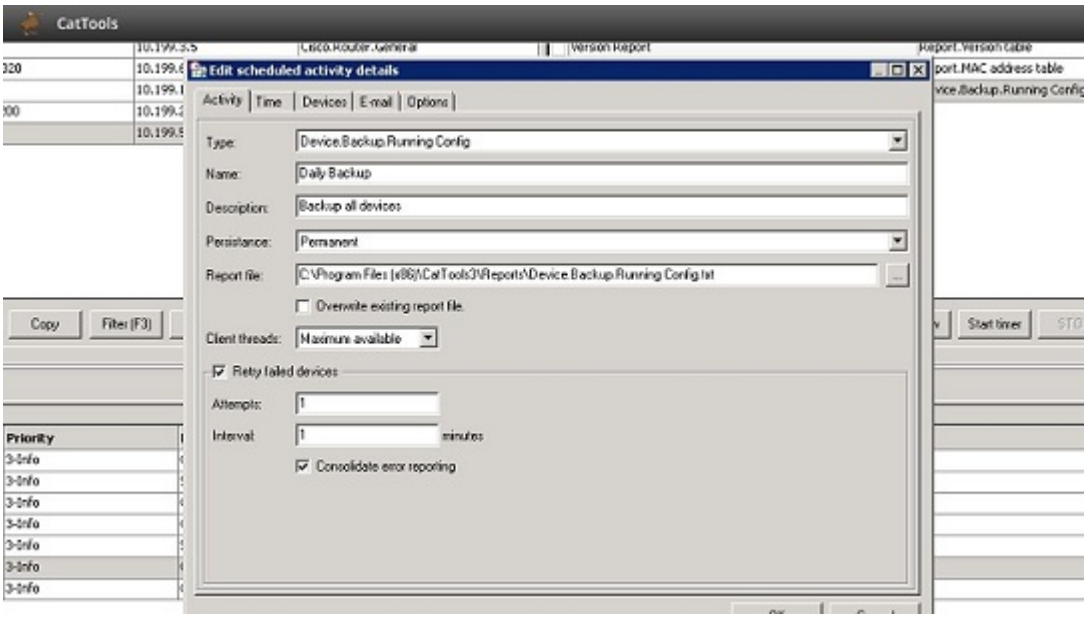

Figure 2.9: Une capture d'écran de CatTools

Nous présentons ici un exemple d'activité de sauvegarde des configurations de tous les équipements en cours d'exécution dans le réseau. Pour ce faire, l'on doit :

— Ajouter une nouvelle activité de type d'équipement appelé. **Device.Backup.Running Config**

— Associer l'ensemble des périphériques, on peut le faire en mode graphique en cliquant sur le bouton « Sélectionner tout » sur l'onglet Périphériques

— Cliquez sur « OK » pour enregistrer l'activité

— Cliquez sur « Exécuter » pour lancer l'activité.

La progression de l'activité peut être contrôlée en utilisant soit le journal d'information ou le panneau d'affichage.

Sa configuration proprement dite est simple : après avoir défini au préalable les options de configurations dans le fichier de configuration, on saisit ou on importe les détails de l'équipement réseau dans la base de données ensuite, on crée les activités prévues pour ce périphérique dans la base de données et enfin on exécute ou on active la minuterie de programmation pour démarrer l'activité à une date ultérieure. Un exemple de fichier de configuration de CatTools est proposé en annexe2

2.4.3 PUPPET

Puppet est un outil de gestion de la configuration automatique orienté vers les systèmes d'exploitation de type serveur. Il permet le déploiement à distance des configurations sur un ensemble de serveurs en très peu de temps.

Au niveau architectural, Puppet fonctionne en mode client/serveur, il se décline donc en deux parties : un serveur (appelé Puppetmaster) et un client.

Le serveur permet de créer et de modifier les fichiers de configuration sous forme de manifeste [5], le serveur écoute les connexions des clients, puis leur signale quel manifeste ils devraient avoir.

Au niveau du client, on installe un programme qui s'exécute sur chaque machine sous le

5. Le manifeste est un extrait de code qui explique à Puppet ce qu'il doit faire, par exemple comment un équipement devrait être configuré, ou encore quel élément (paquet, utilisateurs, fichiers et autres ressources) doit être présent, et ce qu'ils doivent contenir.

contrôle de Puppet et se connecte à *Puppetmaster* pour obtenir son manifeste suivant un intervalle de 30 minutes (chronomètre par défaut) (Anderson et Scobie, 2012) (Arundel, 2010). Au niveau du client, nous avons aussi *facter* qui est un outil de remontée d'informations sur le système.

Puppet utilise le langage déclaratif DSL de Ruby pour installer et exécuter les scripts sur le serveur ou pour distribuer une spécification de configuration réutilisable pour la gestion de certains sous-systèmes ou dispositifs (Delaet et al., 2011). Ces spécifications de configuration, peuvent être appliquées soit directement sur le système, ou alors regroupées dans un catalogue et distribuées sur le système cible par l'intermédiaire d'un paradigme comme l'API REST (Arundel, 2010) [6], et l'agent utilise les spécifications fournies par le serveur pour faire mettre en application les spécifications contenues dans le manifeste. Ce langage inclut toutes les principales caractéristiques de langage de haut niveau comme les fonctions, les instructions conditionnelles, l'héritage et d'autres concepts sans avoir besoin de spécifier les commandes du système d'exploitation (comme *apt*, *yum* ou encore *rpm*). Cette fonction permet d'avoir une cohérence dans les paramètres de configurations de Puppet et favorise leur meilleure lisibilité et leur réutilisation (Arundel, 2010; sparksupport, 2010).

Dans le but d'alléger le réseau et d'optimiser les ressources, il est important dans la gestion des configurations de pouvoir regrouper les configurations des équipements ayant des paramètres similaires (Delaet et al., 2011). Dans cet ordre idée, Puppet utilise le concept de classe. Ces classes peuvent inclurent d'autres classes permettant ainsi de créer des hiérarchies. L'utilisation des outils externes permet ainsi à Puppet de pouvoir assigner dynamiquement ces classes. Puppet utilise ce mécanisme pour distribuer les

6. Representational State Transfer (REST) REST est un style d'architecture pour les systèmes distribués.

configurations sur les équipements (Delaet et al., 2011). Par exemple, les équipements ont besoin de la configuration du client DNS, du mécanisme d'authentification et du système de partages de fichiers. Ces besoins peuvent être exprimés dans un seul module et être utilisés par tous les équipements au lieu de les réécrire dans chaque fichier de configuration.

Au niveau du fonctionnement : Le serveur principal (master) de PUPPET stocke toutes les configurations des clients et chaque client va contacter le serveur via le port 8140 (par défaut) pour récupérer sa configuration. Lorsque le client compile les configurations du serveur, il peut afficher les messages d'erreur s'il y a des erreurs de syntaxe dans les définitions de configuration. Nous pouvons le vérifier sur le serveur de Puppet et le fichier journal du client (sparksupport, 2010; Arundel, 2010). Avant d'utiliser ou installer Puppet, nous avons besoin d'installer des dépendances. Premièrement, nous devons installer Ruby et les fichiers de bibliothèque communs (XML, SSL, etc) ainsi que Facter, qui est un autre projet de Ruby qui rassemble toutes les informations système. Facter sera installé chez tous les clients Puppet. Le serveur de Puppet récupère chez Facter les paramètres de configuration du client et d'autres détails spécifiques au système (Arundel, 2010).

PUPPET tout comme CFEngine permet la distribution des charges contrairement à Chef ou à d'autres outils de configuration, mais cette distribution des charges est séquencée en phase avec l'exportation des ressources collectées entre les équipements gérés (Delaet et al., 2011; Anderson et Scobie, 2012). Une grande différence entre Puppet et la plupart des autres outils, est que les configurations de Puppet sont idempotentes, c'est à dire qu'elles peuvent être exécutées plusieurs fois sans risque.

Selon (Chawla, 2012) le serveur (puppetserver) exige qu'un manifeste soit en place avant

qu'il ne puisse fonctionner. La figure 2.10 présente un exemple de manifeste qui demande à Puppet de créer le dossier "/tmp/testfile" sur le client.

```
Puppet :  vim /etc/puppet/manifests/site.pp
 Créer "/tmp/testfile" s'il n'existe pas.
class test_class {
   file { "/tmp/testfile":
      ensure => present,
      mode   => 644,
      owner  => root,
      group  => root
   }
}
```

Figure 2.10: Un exemple de manifeste

2.4.4 CHEF

Chef est un outil de configuration automatique, spécialisé pour les machines tournant sous Linux, mais pouvant aussi fonctionner sur des machines Windows. Il permet un déploiement facile des serveurs et des postes de travail ainsi que de certaines applications quelque soit la taille du réseau et l'emplacement de l'équipement dans le réseau ou dans une infrastructure en nuage (Opscode, 2013).

Au niveau architectural Chef utilise le concept d'applications distribuées, et est basé sur l'utilisation du langage impératif DSL de Ruby pour installer et exécuter les scripts sur le serveur ou pour distribuer une spécification de configuration réutilisable pour la gestion de certains sous-systèmes ou dispositifs. Chef possède aussi une interface Web pour gérer son infrastructure. Voici une capture d'écran :

Chef, tout comme les autres outils de configuration implémente un mécanisme de regroupement propre à lui pour gérer les configurations des équipements. C'est ainsi qu'il

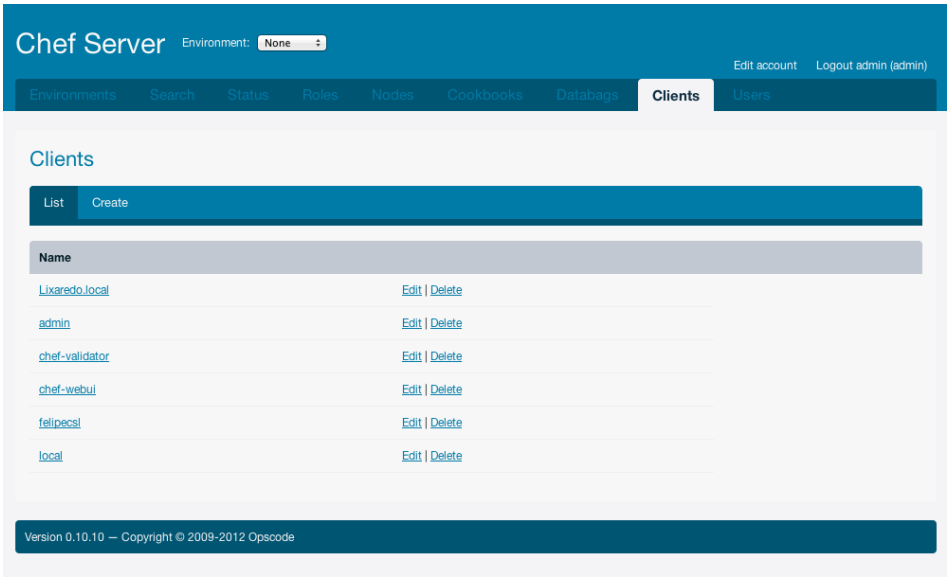

Figure 2.11: Une vue du serveur Chef avec la liste des clients

offre deux types de regroupements à savoir : les regroupements statiques ; c'est-à-dire qui sont prédéfinis dès le départ et les regroupements basés sur les requêtes c'est à dire sur les nouvelles politiques à appliquer au cours du fonctionnement de l'outil (Delaet et al., 2011; Metz, 2011) raison pour laquelle Chef ne peut pas réinstaller un service qui existe déjà car le serveur de Chef est informé de son existence.

Le fonctionnement de Chef repose sur des définitions abstraites (connu sous le nom de livres de cuisine ou « cookbooks »qui contiennent les recettes) qui sont écrites en Ruby et sont gérées comme du code source [7] (Delaet et al., 2011). Chaque définition décrit comment une partie bien spécifique de l'infrastructure devrait être construite et gérée. Ensuite Chef applique de façon automatique ces définitions sur les équipements, comme spécifié, aboutissant ainsi à une infrastructure entièrement automatisée. Lorsqu'un nouveau serveur est en ligne, la seule chose que Chef a besoin est de savoir dans quel livre de recette aller chercher la définition pour l'appliquer au nouvel équipement

7. C'est la description de ce que l'on voudrait que chacune des parties de notre infrastructure fasse.

(Opscode, 2013, 2008).

Une infrastructure de Chef se compose des principaux éléments suivants : un serveur, au moins un poste de travail (Client) et tous les nœuds qui vont être configurés et gérés par Chef.

Le serveur de Chef agit comme un concentrateur de données de configuration, il stocke pour les rendre disponibles, les « cookbooks », les politiques à appliquer sur ces « cookbooks »ainsi que les métadonnées qui décrivent chaque nœud et enfin contient l'inventaire de tous les nœuds disponibles (Opscode, 2013). Le serveur Chef conserve toujours une copie de l'état des nœuds à la fin de chaque exécution.

Le poste de travail est l'endroit à partir duquel les « cookbooks »ou recettes sont créés et où les contraintes sont définies. Les données présentes dans le poste de travail sont synchronisées avec celles du serveur de Chef. Ce qui permet aussi de synchroniser les données qui sont téléchargées vers le serveur de Chef.

Un nœud peut être n'importe quel serveur physique ou virtuel qui est configuré pour être maintenu par un client Chef. Chaque nœud contient un client Chef qui effectue les différentes tâches d'automatisation que chaque nœud a besoin, par exemple donner des détails de configurations tels que les recettes, les templates (modèles) et des fichiers de distributions (Opscode, 2013).

Les recettes et les « cookbooks »représentent l'unité fondamentale de la gestion de configuration et de la distribution des contraintes dans l'outil Chef. En général, les « cookbooks »et les recettes sont écrits et gérés à partir du poste de travail et déplacés au niveau du serveur de Chef, et sont ensuite envoyées aux nœuds par le client Chef pendant chaque exécution de l'outil.

```
apache_site Definition
# definition de la nouvelle ressource
define :apache_site, :enable => true do
include_recipe "apache2"
if params[:enable]execute "a2ensite #{params[:name]}" do
   command "/usr/sbin/a2ensite #{params[:name]}"
   notifies :restart, resources(:service => "apache2")
   not_if do
                ::File.symlink?("#{node[:apache][:dir]}/sites-enabled/#{params[:name]}") or
                ::File.symlink?("#{node[:apache][:dir]}/sites-enabled/000-#{params[:name]}")
          end
   only_if do ::File.exists?("#{node[:apache][:dir]}/sites-available/#{params[:name]}")
          end
   end
else
  execute "a2dissite #{params[:name]}" do
  command "/usr/sbin/a2dissite #{params[:name]}"
  notifies :restart, resources(:service => "apache2")
  only_if do ::File.symlink?("#{node[:apache][:dir]}/sites-enabled/#{params[:name]}")
          end
  end
 end
end
```

Figure 2.12: Exemple de création d'une nouvelle ressource

Chaque « cookbook »définit un scénario, par exemple tout équipement doit installer et configurer MySQL. Chaque « cookbook »contient en plus de la liste de tous les composants qui doivent exécuter ce scénario, d'autres éléments comme, les valeurs des attributs qui sont définies sur les nœuds. Les « cookbooks »contiennent également les définitions, les fichiers de distribution, les bibliothèques, les recettes, les ressources personnalisées ,les modèles, les métadonnées sur les recettes (y compris les dépendances), les contraintes des versions, les plateformes prises en charge etc. Les définitions permettent la création et la collections de ressources réutilisables. Les bibliothèques permettent d'étendre Chef et/ou de fournir l'aide sur le code de Ruby. Les recettes spécifient quelles ressources sont à gérer et l'ordre dans lequel ces ressources seront appliquées (Opscode, 2013).

La figure 2.12 présente un exemple de création d'une nouvelle ressource appelé `apache_site` à partir d'un fichier de définition utilisant une ressource existante apache2 (Opscode, 2013). Une fois créée, la définition peut être utilisée en la plaçant dans une recette

comme dans le code ci-dessous où on a procédé à l'activation et la désactivation de la ressource.

```
# Enable my_site.conf
apache_site "my_site.conf" do
enable true
end
# Disable my_site.conf
apache_site "my_site.conf" do
enable false
end
```

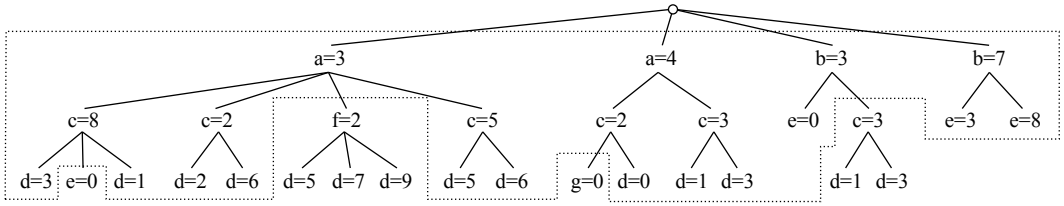

Figure 2.13: Une partie d'une arborescence de configuration méta-CLI. Noms et valeurs des paramètres sont abstraites

2.4.5 ALLOY

Alloy est un solveur et un outil de configuration basé sur la logique du premier ordre. Il permet de vérifier si une contrainte ou une politique à appliquer est valide dans un domaine d'interprétation donnée.

Alloy permet, à partir de plusieurs types de raisonnement sur les exigences des réseaux, d'accomplir plusieurs tâches de configuration (Narain, 2005). Celles-ci comprennent la synthèse de configuration, le diagnostic d'erreur de configuration, leur résolution, l'ajout et la suppression des composants, et enfin l'exigence de vérification. Cependant, un tel raisonnement est actuellement bien élaboré.

Au niveau de sa composition architectural, Alloy est en même temps un langage et un outil d'analyse (Jackson, 2012). Sa partie langage est chargée de structurer les spécifications logiques. Alloy utilise un langage déclaratif c'est-à-dire qu'il peut décrire l'effet d'un comportement sans donner son mécanisme. Ceci permet aux modèles très succincts et partiels d'être construits et analysés. Ce langage est basé sur la logique du premier ordre qui permet d'exprimer différents types de contraintes (Jackson, 2012; Dennis et Seater, 2012).

Sa partie outil d'analyse, considéré comme un compilateur (Dennis et Seater, 2012) permet de trouver les solutions à des formules logiques. Cette partie est aussi appelée solveur de contraintes qui fournit la possibilité d'une vérification et d'une simulation entièrement automatiques des modèles (Jackson, 2012; Dennis et Seater, 2012).

Au niveau fonctionnel, Alloy est basé sur le concept de la recherche de modèles valides. En utilisant la logique du premier ordre, il est capable de traduire les spécifications en expressions booléennes qui peuvent être automatiquement analysées par des solveurs SAT (Narain, 2005; Jackson, 2012). Ainsi, lorsqu'on donne des formules logiques à Alloy, son analyseur va essayer de trouver un modèle qui le satisfait. En d'autres termes, Alloy essaye de trouver des contre-exemples au modèle. En combinant ces contre modèles avec ceux des modèles qui peuvent satisfaire les exigences, Alloy fourni un modèle rigoureusement exact (Jackson, 2012; Narain, 2005). Il est donc utilisé pour construire un solveur d'exigences qui prend en entrée un ensemble de composants réseau et des exigences appliquées sur leur configuration et détermine les configurations qui répondent à ces exigences (Narain, 2005). Au fil des ans, Alloy est devenu un langage de modélisation structurelle complet capable d'exprimer des contraintes structurelles complexes ainsi que les comportements (Jackson, 2012) en une expression simple de logique mathématique.

Tous les problèmes sont résolus dans le périmètre défini par l'utilisateur qui délimite la taille des domaines et permet ainsi de minimiser le problème en le réduisant en une simple formule booléenne (Dennis et Seater, 2012).

Généralement Alloy ne peut pas automatiquement prouver qu'un modèle est valable, mais peut être utilisé pour vérifier si un modèle est correct dans un périmètre fini ou jusqu'à une certaine taille du domaine (Dennis et Seater, 2012). Comme dit plus haut la construction d'un modèle est un préalable à l'utilisation d'Alloy. La figure 2.14présente l'exemple d'un modèle (Dennis et Seater, 2008) dont les mots clés sont expliqués ci-dessous.

— `Fact` : c'est un mot clé utilisé pour forcer certains éléments comme des contraintes ou des prédicats à être vrai dans le modèle.

— `Root` : représente la racine des répertoires ou dossiers.

— `sig` : signifie ensemble qui permet de déclarer les types d'objet.

— `assert` : est un mot clé qui signifie qu'un objet ou une contrainte doit être vrai en raison du comportement du modèle.

— `sig FSObject` : définit un ensemble dénommé FSObject, représentant l'ensemble de tous les objets de système de fichiers (les deux répertoires et fichiers) dans le système de fichiers. file signifie ficher dans le système de fichiers

— `Dir` : signifie répertoire ou dossier dans le système de fichier.

— `Ione` : indique simplement qu'il existe 0 ou 1 parent *dir* pour chaque FSObject. En d'autres termes sachant que tous les objets du système de fichiers n'ont pas tous de parent, c'est ce mot clé qui indique si un objet du système de fichiers a un parent ou pas.

— `Parent` : c'est la relation qui lie FSObject à Dirs (répertoires). Dans Alloy chaque

objet du système de fichiers a un champ parent qui est un répertoire.

— Check : permet à Alloy de rechercher automatiquement des contre-exemples aux affirmations.

Nous allons nous appuyer sur les travaux de (Narain, 2005) et des configurations de la figure 2.15 pour présenter un exemple concret d'utilisation d'Alloy.

— Conf1 : Les trois lignes déclarent trois types d'objet : le routeur, le sous-réseau et l'interface. Le dernier type a deux attributs, le châssis (de type routeur) auquel il appartient, et le réseau (de type sous-réseau) sur lequel il est placé. Ces attributs modélisent les paramètres de configuration d'une interface.

— Conf2 : Le Prédicat `spec` définit une spécification dont nous essayerons de trouver le modèle. C'est une conjonction de trois contraintes. La première affirme que pour chaque routeur x il y a une interface y dont le châssis est x, c'est-à-dire que chaque routeur a au moins une interface. La seconde affirme que deux interfaces non-égales sur le même routeur ne peuvent être placées sur le même sous-réseau. La troisième affirme que notre réseau contient un routeur, deux sous-réseaux et deux interfaces. Ce sont les composantes que nous voulons configurer.

— Conf3 : Permet de produire le modèle (valeurs des paramètres de configuration) apparaissant dans la Conf4.

— Conf4 :La première ligne affirme que la valeur du châssis pour interface_0 est router_0 et la valeur de châssis pour interface_1 est router_0. Même chose pour la seconde ligne. Alloy crée automatiquement des instances d'objets tels que router_0, sous-réseau_0, et sous-réseau_1. Il faut noter qu'Alloy n'a pas placé interface_0 et interface_1 sur le même sous-réseau à cause de la seconde contrainte. D'autre part, si nous enlevons cette contrainte, Alloy produit une solution supplémentaire apparaissant dans Conf5 dans laquelle interface_0 et interface_1 sont placés sur

```
// Un objet système de fichiers dans un système de fichiers
sig FSObject { parent: lone Dir }
// Un répertoire dans un système de fichier
sig Dir extends FSObject { contents: set FSObject }
// Un fichier dans un système de fichier
sig File extends FSObject { }
// Un répertoire est le parent de son contenu
fact { all d: Dir, o: d.contents | o.parent = d }
// Tous les objets du système de fichiers sont soit des fichiers ou des répertoires
fact { File + Dir = FSObject }
// Il existe une racine
one sig Root extends Dir { } { no parent }
// Le système de fichiers est connecté
fact { FSObject in Root.*contents }
// Le chemin d'accès contenu est acyclique
assert acyclic { no d: Dir | d in d.^contents }
// Maintenant le vérifier pour  le domaine 5
check acyclic for 5
// Le système de fichiers a une racine
assert oneRoot { one d: Dir | no d.parent }
// Maintenant le vérifier pour le champ d'application 5
check oneRoot for 5
// Chaque objet fs est la plupart du temps dans un répertoire
assert oneLocation { all o: FSObject | lone d: Dir | o in d.contents }
//  Maintenant le vérifier pour le champ d'application 5
check oneLocation for 5
```

Figure 2.14: Un exemple de modèle pour d'Alloy

```
sig router {}
sig subnet{}
sig interface {chassis: router, network: subnet}
```

Conf 1: Déclaration avec types d'objet.

```
pred spec ()
{all x:router | some y:interface | y.chassis = x}
{no disj x1,x2:interface |
        x1.chassis=x2.chassis && x1.network = x2.network}
{#router=1 && #subnet=2 && #interface=2}
```

Conf 2: Une spécification avec trois contraintes.

```
run spec for 1 router, 2 subnet, 2 interface
```

Conf 3: Commande alloy pour créer un modèle.

```
chassis : = {interface_0 -> router_0, interface_1 -> router_0}
network : = {interface_0 -> subnet_1, interface_1 -> subnet_0}
```

Conf 4: Resultat de la commande de Conf 3.

```
chassis : = {interface_0 -> router_0, interface_1 -> router_0}
network : = {interface_0 -> subnet_0, interface_1 -> subnet_0}
```

Conf 5: solution avec moins de contrainte.

Figure 2.15: Alloy par l'exemple

le même sous-réseau.

2.5 LACUNES OBSERVÉES

Le domaine de la gestion automatisée des configurations a connu ces dernières années un développement certain avec l'apparition de nombreux outils et études dans le domaine. Malgré toutes leurs fonctionnalités, ces outils présentent néanmoins un certain nombre de lacunes qui peuvent en limiter l'utilité. Nous présentons ici les principales.

2.5.1 ORIENTÉ VERS LES POSTES DE TRAVAIL PLUTÔT QUE SUR LES ÉQUIPEMENTS RÉSEAUX

Force est de constater que la plupart d'outils de gestion de configuration sont orientés vers les postes de travail et les serveurs : le domaine des réseaux n'ayant que très peu d'outils dédiés. Ceci peut s'expliquer par le fait que la plupart des types de poste de travail peuvent communiquer entre eux, le développement d'un outil pour un type de poste ne demande que quelques modifications pour pouvoir supporter ou communiquer par un autre type de poste. Il y a donc une large clientèle pour la gestion des postes de travail.

2.5.2 MANQUE D'INTEROPÉRABILITÉ

Si le domaine des réseaux constitue le parent pauvre de la gestion automatisée des configurations, c'est aussi à cause de la non-interopérabilité entre les équipements. Les configurations des équipements sont tributaires des fabricants.

Les interfaces pour envoyer les configurations ne sont pas compatibles, ainsi que les paramètres des fichiers de configuration. La plupart des outils qui existent utilisent un langage de bas niveau ce qui ne permet pas une séparation entre les contraintes et le code. En d'autres termes l'utilisation d'un langage de bas niveau ne permet pas une abstraction des caractéristiques techniques entre les équipements de fabricants différents (Delaet et al., 2011). Pourtant l'utilisation des langages de haut niveau, permettrait de mettre sur pied des outils de gestion automatisée de configurations sans tenir compte de la façon dont fonctionne le matériel. Certaines solutions comme Config Differ (RANCID) (Networks, 2006) permettent de travailler sur des équipements de plusieurs fabricants

différents, il est spécialement orienté vers les équipements réseaux plus précisément ceux de Cisco mais permet également de monitorer les équipements d'autres fabricants comme Juniper, Redback NASs, Alteon,HP etc. Il faut aussi relever que la plupart des systèmes d'exploitation (OS) des équipements réseaux sont apparentés au système d'exploitation Unix ou Linux ce qui augmente aussi l'interopérabilité entre ces équipements.

2.5.3 IMPÉRATIF VS. DÉCLARATIF

La majeure partie des outils utilise un langage impératif qui décrit explicitement les manipulations à effectuer, mais le but recherché par ces manipulations ne permet pas de façon définitive de mieux cerner le problème. Dans les langages impératifs on cherche à décrire « le comment »des choses c'est à dire la solution sans s'intéresser à la source, contrairement aux langages déclaratifs qui eux cherchent à décrire le « quoi ». Par exemple pour le développement d'un site web, les langages déclaratifs vont décrire ce que contient une page (texte, titres, paragraphes, etc.) et non comment les afficher (positionnement, couleurs, polices de caractères, etc.). Les langages déclaratifs cherchent à décrire ce que l'on voudrait faire plutôt que de spécifier la façon dont on veut faire quelque chose. Si on sait comment on veut faire quelque chose on peut l'appliquer à d'autre types de choses, mais si on s'intéresse à la façon de faire quelque chose cela réduit le champ d'application.

2.5.4 GESTION ET VALIDATION

Bien que les outils présentés permettent une détection relativement efficace du comportement anormal des équipements réseau. Ils offrent un support limité pour la recherche des causes d'erreurs possibles qui pourraient aider à revenir aux paramètres de configuration

de départ. Un autre inconvénient est que les erreurs sont détectées à posteriori, c'est à dire après le dysfonctionnement. A moins que des simulations réalistes soient effectuées avant chaque modification apportée au réseau, il faut attendre que les configurations erronées soient mises en exécution avant de s'apercevoir que quelque chose ne fonctionne pas comme prévu. Il y a un manque de validation des configurations avant toute configuration, ce qui permettrait de gagner du temps. La plupart des outils sont basés sur la gestion des fichiers de configuration et non sur leur validation préalable.

Certains équipements permettent de conserver l'historique des fichiers de configuration en vue d'un rollback pour récupérer une ancienne configuration qui fonctionnait bien : c'est le cas de RANCID. Mais cette solution présente des inconvénients majeurs car même si la détection des changements dans une configuration représente un bon principe, tous les changements n'entraînent pas toujours des erreurs de configuration. Pour cette raison, de nombreuses fausses alarmes peuvent être déclenchées. En outre, même si la source d'une mauvaise configuration est causée par un paramètre spécifique qui a changé, la raison de l'erreur doit toujours être comprise et résolue manuellement par l'administrateur réseau.

La plupart des outils présents sur le marché n'en sont pas capables, il s'agit pourtant d'une nécessité étant donné l'évolution des équipements réseau.

CHAPITRE 3

MODÈLE DE CONFIGURATION GÉNÉRIQUE (META-CLI)

Comme nous l'avons vu, la gestion des dispositifs de configuration de réseaux est un processus complexe du fait du nombre de dispositifs et des paramètres à prendre en considération et, plus important, de la façon très variée dont ces paramètres peuvent être interrogés et modifiés sur chaque dispositif. Chaque fabricant offre sa propre interface de ligne de commandes ou protocole de gestion dans lequel les paramètres sont organisés de manière distincte, de même les différents appareils du même fabricant nécessiteraient de pouvoir communiquer entre eux. Dans le présent chapitre, nous présentons un modèle de données génériques pour des informations de configuration des dispositifs de réseaux qui prennent en compte l'hétérogénéité des fabricants et de leurs versions. Un moteur de requête de configuration permet à l'utilisateur d'identifier un paramètre de configuration spécifique, abstrait et d'obtenir la séquence appropriée de commandes pour interroger ce paramètre sur un dispositif d'un fabricant donné et d'une autre version de système d'exploitation. Le contenu de ce chapitre a fait l'objet d'un article qui a été accepté et sera présenté à la conférence IM 2015.

3.1 GESTION DES DISPOSITIFS RÉSEAU

Un dispositif peut être une composante matériel telle qu'un routeur, un switch, un point d'accès sans fil ou une passerelle. En d'autres termes c'est du matériel physique ou virtuel utilisé pour acheminer des paquets dans un réseau informatique. Ils possèdent leurs propres caractéristiques internes et leurs interfaces de communication et dépendent habituellement des fabricants pour ce qui est des dispositifs physiques ou de leur version pour les dispositifs virtuels. La principale caractéristique d'un dispositif est que son comportement par défaut peut être modifié dynamiquement par configuration. La configuration d'un dispositif correspond à un ensemble de paramètres et de valeurs associées enregistrés par le dispositif. Ces réglages permettent aux administrateurs réseaux de personnaliser le dispositif, par exemple en paramétrant les interfaces physiques, en appliquant les règles de redirection des données, en adaptant le comportement générique du dispositif au réseau, ou en modifiant le comportement du dispositif de telle sorte qu'il corresponde aux évènements précédents. En matière de configuration, les éléments éditables s'appellent les paramètres.

3.1.1 CONFIGURATIONS ET HÉTÉROGÉNÉITÉ

D'habitude, les dispositifs de réseaux sont configurés à partir de la ligne de commande. Sur chaque dispositif, obtenir des informations sur les paramètres ou les modifier est possible en utilisant des commandes précises.

Le problème d'interopérabilité s'est résolu sur un plan général,car la plupart des fabricants d'équipements ont été forcés d'ajouter une couche supplémentaire pour permettre l'interopérabilité des appareils de réseau. C'est le cas de Cisco NORTEL. Le problème

d'utilisation de certains protocoles comme SNMP et CMIP a déjà trouvé une solution ; il reste à trouver des solutions aux problèmes d'interopérabilité qui peuvent survenir dans un même domaine entre l'utilisation des protocoles de communication et le développement de modèles. La question suivante sera : « que se passerait-il, si deux domaines différents représentaient le même concept mais de manière différente ? » Une traduction purement syntaxique ne peut apporter une solution mais plutôt une traduction sémantique serait nécessaire pour faire correspondre directement les concepts de ces deux domaines, d'où la contribution de l'ontologie. L'ontologie offre toutes les constructions nécessaires pour ajouter les informations sémantiques représentées (Vergara et al., 2002).

Hétérogénéité inter-fabricant

En l'absence de normes reconnues, ces commandes diffèrent d'un fabricant à un autre. La syntaxe d'une commande utilisée pour afficher les informations sur l'interface d'un commutateur ou d'un routeur sur un dispositif Cisco ne peut pas être la même que sur un dispositif Juniper. La documentation mise à disposition par les deux fabricants indique aux administrateurs réseaux la syntaxe à utiliser. Ainsi, la configuration d'un dispositif particulier requière la connaissance du fabricant (dans notre cas : Cisco et Jupiter) et l'utilisation de la syntaxe appropriée afin d'exécuter correctement la commande. Le tableau ci-dessous présente deux commandes de deux fabricants permettant d'afficher la configuration de l'interface, le statut et les statistiques.

Fabricant	Syntaxe
Cisco	show interface[intfc]
Juniper	show interfaces[intfc] detail

Hétérogénéité inter-version

La différence de syntaxe n'est pas seulement due à la concurrence entre fabricants. Elle peut parfois être observée entre les produits du même fabricant lorsque les dispositifs les plus récemment développés contiennent une version mise à jour du programme. Par conséquent, la syntaxe peut varier d'une version à une autre. Ceci est bien illustré par les produits Cisco dont les commandes changent lorsque les versions de ses dispositifs changent. Par exemple, la syntaxe de la commande ip domain a évolué de la version 10 à la version 12 du Système d'Exploitation de Cisco (IOS).

— Dans la version 10.0, la syntaxe est : `ip domain-list name`

— Dans la version 12.2, la syntaxe est : `ip domain name list` (sans les tirets)

Etat de l'art sur l'interopérabilité et les structures de données dans la gestion de configuration

Le taux élevé de modifications des contraintes fait en sorte que la configuration et la gestion de réseaux deviennent de plus en plus complexes. Cette complexité est en outre influencée par l'ampleur croissante, des comportements imprévisibles des logiciels ainsi que la diversité en termes de dispositif de réseau et leur logiciel. Afin de résoudre ce problème de complexité, plusieurs programmes de gestion de configuration ont été proposés, dont certains ont déjà été énumérés au chapitre précédent. Alors que les outils comme CFEngine et bcfg2 utilisent une syntaxe qui n'est pas très différente de celle de la configuration manuelle du système, les outils tels que Puppet et LCFG permettent la modélisation de la dépendance entre les configurations de dispositifs. Mais aucun des outils mentionnés ci-dessus ne fonctionne sous des spécifications de contraintes qui

pourraient être appliquées à ces dispositifs (Delaet et Joosen, 2007). La modélisation de la dépendance, de même que la gestion efficace des contraintes par un langage de haut niveau peut aider les administrateurs réseaux à résoudre les problèmes d'interopérabilité et d'hétérogénéité.

Les outils tels que PoDIM (Delaet et Joosen, 2007) qui est un langage de haut niveau pour la gestion de configuration qui se compose d'un langage de règles et d'un modèle de domaine extensible comme celui de ValidMaker (Hallé et al., 2012) autorisent l'abstraction des programmes à travers l'utilisation des concepts ou des langages de haut niveau. PoDIM tente de résoudre le problème posé par la complexité de la gestion de configuration due à la multiplicité des systèmes hétérogènes à travers l'utilisation de mécanismes permettant de spécifier de multiples contraintes croisées. En outre, PoDIM permet la spécification de haut niveau pour une infrastructure informatique dans son ensemble sans aller dans les détails de la syntaxe du langage afin de ne pas être lié à un système ou appareil déterminé

En plus, PoDIM permet à l'administrateur d'exprimer des contraintes telles que «un de mes serveurs doit être configuré comme un serveur web» et sur « chaque sous-réseau, il doit y avoir deux serveurs DHCP »en lieu et place des affectations de paramètres. Par exemple, l'administrateur de réseau peut instaurer des règles telles que «un serveur web doit utiliser un numéro de port supérieur à 1024». Le logiciel d'exécution de PoDIM prend en entrée un ensemble de règles et essaye de satisfaire ces règles en créant des objets et des paramètres de réglage des objets. En procédant ainsi, il génère une configuration pour chaque dispositif géré indépendamment du fabricant de l'équipement. PoDIM utilise alors des outils de gestion de configuration existants tels que CFEngine pour deployer la configuration générée sur tous les dispositifs gérés. Le principe de base du logiciel d'exécution de PoDIM stipule que chaque objet du monde réel doit être simulé

dans le système. Les différentes classes d'objet sont définies dans le modèle de domaine (Delaet et Joosen, 2007). Cependant, PoDIM ne permet pas la vérification de ses règles ni des fichiers de configuration de ses dispositifs. Ceci peut conduire les utilisateurs à rencontrer des problèmes d'état de fonctionnement stable rendant ainsi impossible le fait d'atteindre un état stable et générer une configuration invalide. PoDIM est un outil pour techniciens dont l'utilisation requiert, des connaissances en Eiffel.

Contrairement à PoDIM, ValidMaker (Hallé et al., 2012) est un outil de gestion de configuration de réseaux qui repose sur une représentation abstraite des paramètres et des contraintes. L'outil poursuit deux objectifs principaux. D'abord, il met à niveau l'hétérogénéité des dispositifs à travers de multiples plateformes en offrant une représentation commune des informations de configuration appelée Meta-CLI qui est un arbre de paires valeur-attribut. Des services de réseaux définis peuvent être mis en place par la création des modèles d'instances de configuration appelés services génériques. Deuxièmement, à l'aide du langage formel appelé Configuration Logic (CL), ValidMaker permet à l'ingénieur de réseaux d'entrer des contraintes personnalisées sur des paramètres de configuration et de les vérifier automatiquement.

Le modèle de configuration proposé par le modèle Meta-CLI place les informations dans des modèles de structure arborescente qui déterminent les opérations qu'il faut faire pour extraire, modifier, enlever et ajouter les données sans considération du fabricant du dispositif réseau. Par conséquent, l'outil ValidMaker que nous présenterons dans un autre chapitre, extrait le fichier de configuration du dispositif sous forme de fichiers Meta-CLI qui peuvent être traduits à leur tour en configurations exécutables et envoyés aux dispositifs dans un format approprié selon le système d'exploitation du vendeur du matériel et suivant le numéro de leur version (Hallé et al., 2012). La configuration de dispositifs réseau tels que les routeurs et les switches peuvent être représentés sous forme

d'arbre dans laquelle chaque noeud est une paire composée d'un nom et une valeur. Cet arbre représente la hiérarchie de paramètres inhérents à la configuration de ces dispositifs (Hallé et al., 2004). Les structures méta-CLI utilisées dans la représentation interne des configurations dans ValidMaker utilisent également ce type de modèle sous forme d'arbre.

3.2 MODÈLE FORMEL DE CONFIGURATIONS DE PÉRIPHÉRIQUES RÉSEAUX

Le problème commun à toutes les approches mentionnées au chapitre précédent est qu'elles sont toutes spécifiques à un fabricant défini ou à une version du système d'exploitation du dispositif. En fait, même les approches dont l'objectif est de promouvoir l'interopérabilité, telles que SNMP et Netconf, offrent plutôt très peu de possibilités d'abstraire ces différences. Dans le cas de SNMP, chaque fabricant d'appareil reçoit son préfixe numérique propre : tous les paramètres pour les dispositifs 3com commencent avec le préfixe 1.3.6.1.4.1.43 alors que tous les paramètres pour les dispositifs Cisco commencent avec 1.3.6.1.4.1.9., chaque fabricant est ensuite libre de structurer ses informations de configuration tel que bon lui semble. Par conséquent, même si le protocole SNMP offre un moyen standard d'interroger les données de MIB, la structure de ces MIB et l'écriture réelle des requêtes dépend du fabricant du dispositif. La même chose peut être dite de Netconf ; en dépit de son style RPC et de l'utilisation de XML pour envoyer et recevoir des données, une fois de plus, chaque fabricant de dispositif est libre d'organiser ses informations de configuration selon ce qu'il juge approprié. Par conséquent, de tels protocoles peuvent être considérés comme des moyens indépendants des fabricants de manipuler des structures de données dépendantes du fabricant.

3.2.1 ONTOLOGIE

Une ontologie est une spécification bien detaillée d'une conception d'une vue abstraite ou simplifiée du monde que nous voulons représenter pour un but précis (Gruber, 1993). Confère représentation de la fauve et de la flore sauvage présenté ci bas. L'ontologie constitue en elle-même un modèle représentatif de données d'un ensemble de concepts dans un domaine, ainsi que des relations entre ces concepts. Le premier objectif de l'ontologie est de modéliser un ensemble de connaissances dans un domaine donné, qui peut être réel ou imaginaire, il définit un ensemble de primitives représentatives qui permettent de modéliser un domaine de connaissance. Les primitives représentatives sont généralement des classes, des attributs (propriétés) ou des relations (relations entre les membres de la classe). L'ontologie utilise le langage de spécification considéré comme l'élément-clé sur lequel il se base (Gruber, 2009). La majorité des langages de spécification sont basés ou proches de la logique du premier ordre et représentent ainsi la connaissance sous la forme d'une affirmation (sujet, prédicat, objet). Ces langages sont généralement conçus pour extraire les structures de données et en mettant l'accent sur la sémantique. Dans le contexte des systèmes de bases de données, l'ontologie peut être considérée comme un niveau d'abstraction de modèles de données semblable aux modèles hiérarchiques et relationnels, mais qui vise à la modélisation des connaissances par exemple si l'on applique l'ontologie sur les individus on va rechercher leurs attributs et leurs relations avec d'autres individus (Gruber, 2009). En fin de compte, une ontologie est un réseau sémantique qui se compose d'un ensemble de concepts décrivant un domaine particulier. Ces concepts sont liés les uns aux autres d'abord par la hiérarchie de concepts, ensuite par la sémantique. Pour construire une ontologie,au moins trois concepts doivent être pris en compte :

— la détermination des agents passifs et actifs.

— Leur condition fonctionnelle et contextuelle.

— les changements possibles apportés à leurs objectifs limités.

(Antoniou et van Harmelen, 2003) et (Lapalme, 1999) montrent un exemple d'ontologie décrivant une partie de la faune et de la flore sauvage.

1. Un Animal est une classe.
2. Une Plante est une classe, mais différente de l'Animal
3. Un Arbre est une sous-classe d'une Plante.
4. Une Branche est une partie d'un arbre.
5. Une Feuille est une partie d'une branche
6. Un Herbivore est un Animal qui ne mange qu'une Plante ou une partie d'une plante.
7. Un Carnivore est un Animal qui mange un autre Animal.
8. Un Lion est un Carnivore qui ne mange que des Herbivores.
9. Une Plante appétissante est une Plante qui est mangée par un Herbivore mais aussi par un Carnivore
10. Une Girafe est un Herbivore qui ne mange que des Feuilles.

Dans notre travail, nous avons préféré utiliser le concept de modèle générique qui est une autre forme d'ontologie, car notre objectif est de reconstruire ou de convertir les séquences de commandes d'un modèle contrairement à l'ontologie qui est appliquée premièrement à la modélisation des dépendances de configuration et d'autre part d'apporter une contribution à l'automatisation de la correspondance entre les termes en vue de créer une base de données unifiée des informations d'application de telle sorte que la gestion des différentes marques d'équipements réseaux puisse se faire par une simple passerelle.

3.2.2 UN MODÈLE DE DONNÉES GÉNÉRIQUE

Généralement, un dispositif réseau tel qu'un commutateur ou un routeur possède un ensemble de paramètres configurables, stockés comme une paire nom/valeur. Ces paramètres sont organisés hiérarchiquement, en fonction de la partie du dispositif à laquelle ils s'appliquent.

Paramètre Concret

Une paire nom/valeur représente les paramètres concrets. Les paramètres concrets sont liés à l'implémentation des fonctionnalités au sein du composant réseau. Pour chaque version du composant, un nouveau paramètre générique est défini qui possède un identifiant différent. Afin de retrouver ce paramètre concret, on l'associe directement à un paramètre générique qui lui fournira un identifiant générique. Le paramètre concret possède sa propre syntaxe et ses fonctions d'accès et de modification. De la même manière que les paramètres génériques, un paramètre concret est relié éventuellement à des paramètres concrets enfants. Cette hiérarchie peut être différente de celle des paramètres génériques et peut être différente d'une version de périphérique à une autre. Ces paramètres regroupent des informations utiles à la configuration. On retrouve un identifiant, un nom de paramètre, une syntaxe associée, la valeur ou l'ensemble de valeurs prise par ce paramètre.(Parisot, 2014) Ils sont estimés en fonction de la configuration du dispositif. Ils représentent une caractéristique particulière ou une composante du comportement du dispositif. Il faut noter que les valeurs de chaque paramètre sont d'un type bien défini qui peut être connu d'avance.

Paramètre Générique

D'un switch à un autre, les paramètres ont des noms similaires et une hiérarchie similaire. En effet, la structure interne d'un commutateur peut varier mais l'ensemble de paramètres reste fermé. Malgré le fait que les noms de paramètres spécifiques peuvent varier d'une version à une autre, il est possible de combiner deux versions de paramètre avec la même fonctionnalité. Nous pouvons faire la même chose en regroupant les paramètres des dispositifs de différents fabricants sous le même nom de paramètres. Nous obtenons alors un paramètre générique qui aurait la même signification pour tous les dispositifs. Ce paramètre générique se compose d'un nom générique qui identifie de manière unique tous les appareils (dispositifs). Il est associé à un ensemble de paramètres concrets qui ont leur propre nom spécifique et leur valeur fixée. Ce modèle de données est utilisé pour séparer les aspects syntaxiques et hiérarchiques des fabricants d'aspects fonctionnels de configuration. Quel que soit le format de la configuration d'entrées, il est possible d'identifier avec certitude un paramètre et les valeurs spécifiques en se référant au paramètre générique.

Les paramètres génériques définissent les paramètres de configuration d'un switch indépendamment de son implémentation. Ils ont des valeurs qui peuvent être accédées et modifiées à travers des fonctions génériques. On définit à travers eux, une interface de gestion unifiée des configurations. Chaque paramètre générique a un identifiant qui lui est propre et qui ne dépend pas de la version utilisée ou du constructeur. On peut donc utiliser cet identifiant pour construire nos requêtes et accéder à des paramètres précis de la configuration. Ces paramètres génériques sont organisés de façon hiérarchique sous forme d'arbre. En effet, de la valeur d'un paramètre générique peut dépendre de la valeur d'autres paramètres génériques. Les paramètres concrets sont au contraire liés à

l'implémentation des fonctionnalités au sein du composant réseau. Pour chaque version du composant, un nouveau paramètre générique possèdent un identifiant différent est défini. Afin de retrouver ce paramètre concret, on l'associe directement à un paramètre générique qui lui fournira un identifiant générique.

Le concept de Commandes

Pour tenir compte de la hiérarchie des paramètres d'un dispositif, nous introduisons le concept de commandes. Une commande comporte un ou plusieurs paramètres, un nom et un ensemble de commandes sous-jacentes. Le modèle de données inclut de la même manière des commandes spécifiques de dispositifs dans une commande générique. Ceci permet de gérer les changements qui peuvent influer sur l'organisation hiérarchique d'un dispositif à un autre, (Parisot, 2014).

3.3 MISE EN ŒUVRE ET EXPÉRIMENTATION

3.3.1 IMPLÉMENTATION

Pour réaliser cette approche, nous avons choisi de mettre en pratique une solution permettant à l'administrateur réseau de stocker les dispositifs avec leurs paramètres spécifiques. Ensuite de les faire correspondre à des paramètres génériques et de trouver la syntaxe correspondant aux dispositifs. Un dispositif est décrit par un ensemble de commandes qui lui sont associées. Ces commandes possèdent d'autres commandes ou paramètres sous-jacents. Ces objets forment une hiérarchie de paramètres et de commandes qui donnent la configuration du dispositif. En fait, chaque commande de l'appareil et chaque paramètre de l'appareil contient un lien vers une commande

générique unique et à un paramètre générique. Ce lien permet à chaque dispositif d'être décrit de manière générique. On peut stocker ainsi un dispositif entier sous forme d'arbre avec des nœuds pour les paramètres et les commandes du dispositif ainsi que leurs valeurs associées. Chacun de ces nœuds contiendra une référence à un paramètre générique externe.

Parallèlement à cette structure, nous avons mis en place une deuxième solution que nous avons baptisée *VendorOSReference*. Elle peut stocker toutes les syntaxes utilisées et les paramètres exigés par un fabricant pour les différentes versions de son système d'exploitation. Nous avons ainsi trouvé *VendorParameter* qui est le paramètre concret du dispositif. Ce paramètre est inclut dans un ensemble d'autres mots et forme la syntaxe *VendorSyntax*. Parmi ces termes se trouvent des mots-clés et des chaînes nécessaires à la bonne compréhension de la syntaxe. Des termes optionnels sont pris en compte dans la solution via *TermOrTerm*. Nous pouvons définir plusieurs versions de commandes en déclarant plusieurs nœuds de *VendorCommandsVersion* qui regroupent les différentes syntaxes indispensables pour exécuter cette commande, pour chaque version fournie par le fabricant. Chaque nœud fait référence au même *GenericCommand* qui l'identifie de manière unique. La relation entre les paramètres concrets et génériques est établie par la structure de données choisie. Pour exécuter le traitement de ces données, nous avons développé une structure de données en Python basée sur le format Meta-CLI. Le programme mis en place charge les données des dispositifs et la référence OS des fichiers XML. Il devient donc possible de faire le traitement sur le dispositif en navigant sur les nœuds comme des objets Python.

```
# alias m in the nodes
aliasM = Alias ("m")
uid_m = 106
# condition m !="0.0.0.0"
condition1 = Condition ("!=")
condition1 . setTerms ( AtomicAliasTerm ( aliasM ), AtomicConstantTerm (",! 0.0.0.0 "))
# node
node2 = ForAllNode (uid_m , [], [ condition1 ], aliasM )#no child
node1 = ForAllNode (0, [ node2 ], [], None )#no conditions and no aliases
46
# Formula tree
formulaA = LogicFormulaTree ([ node1 ])# just one node at root
```

Figure 3.1: Formule logique

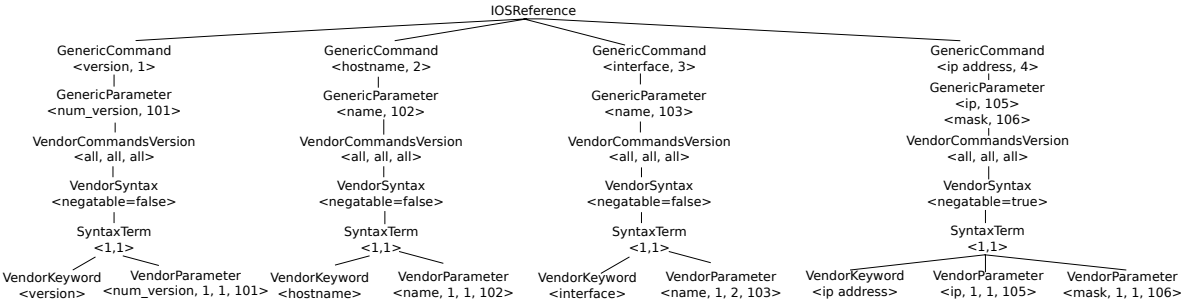

Figure 3.2: Cisco IOS Reference sous forme d'arbre

3.3.2 EXEMPLE D'UTILISATION

Pour que notre solution soit bien comprise, nous allons présenter un exemple dans son entièreté. Pour ce faire, nous avons choisi de travailler avec les commutateurs Cisco (voir figure 3.6). Nous pouvons extraire le fichier de configuration via CLI. Nous supposons que la forme de vérification est le format NFD (Normal Form Disjunctive). Il convient de rappeler que mettre une formule CL sous Forme Normale Disjonctive (FND) consiste à convertir en une disjonction de clauses conjonctives. Exemple l'expression suivante est FND $(A \wedge B) \vee (\neg C \wedge D)$ tandis que celle-ci ne l'est pas $(A \wedge B) \vee \neg(\neg C \wedge D)$.

Au départ, ce routeur Cisco a été configuré par les commandes CLI de Cisco IOS (figure 3.6 (a)). Nous présentons également la version abstraite de ces commandes qui caractérisent la configuration du dispositif (figure 3.6 (b)).

1. *Hypothèses et scénarios* : notre scénario consiste à vérifier la définition appropriée des masques du sous-réseau sur tous les dispositifs en utilisant Meta-CLI. Le dispositif abstrait précédemment écrit en Meta-CLI (XML) respecte une documentation : la référence du fabricant. La figure 3.5 donne un exemple de référence d'un fournisseur où toutes les commandes et paramètres utilisés par un dispositif abstrait sont déclarés. Ces commandes et paramètres abstraits sont reliés aux commandes et paramètres physiques du dispositif réel (ceux fournis par le fabricant).

2. *Construction de la référence* : Afin d'associer à chaque paramètre concret un paramètre générique, il est nécessaire de construire d'abord une référence. Elle comportera un *uid* pour chaque paramètre. Le fichier de référence contenant les paramètres de Cisco IOS Reference se trouve à la figure 3.5. Une forme sommaire de l'arbre se trouve à la figure 3.2. Nous pouvons y noter qu'ils ont construit un nœud par commande Cisco et un nœud par paramètre. La syntaxe des commandes est stockée sous une forme ordonnée de mots-clés et de paramètres. Certaines syntaxes comme l'adresse IP permettent la négation. Ceci signifie qu'il est possible d'écrire, grâce à la syntaxe Cisco, *pas d'adresse IP* s'il y a un manque ou une absence d'adresse IP sur l'interface. Nous voyons que seule une syntaxe a été définie par la commande mais qu'il est possible d'en définir plusieurs si celle-ci venait à changer. Il y a normalement une référence unique qui regroupe les syntaxes de tous les fabricants et toutes leurs versions. La formule logique peut se décliner ainsi : $[\forall \text{mask}{=}m] \neg (m{=}\text{'0.0.0.0'})$

   Il est donc nécessaire de vérifier que tous les masques de sous-réseau sont différents de l'adresse 0.0.0.0. Cette formule a été écrite sous le format NFD. Nous pouvons donc la convertir par exemple en langage Python. Pour cela, nous utiliserons les

éléments suivants :

— Alias : Nous définissons un alias qui représentera m dans notre formule.

— condition :La condition d'inégalité entre m et et '0.0.0.0' ainsi que l'opérateur ($\neg$).

— ForAllNode : Cet objet représente la condition $\forall$ de la formule. Nous lui attachons la condition, l'alias m et l'uid (l'identificateur dans la référence) du paramètre mask, donc ici uid aura une valeur de 106.

— LogicFormulaTree : Nous construisons un arbre du modèle logique de ce noeud. Le résultat de cette étape donne le code de la figure 3.1 et la Formule Logique sous forme d'arbre est présenté par la figure 3.4.

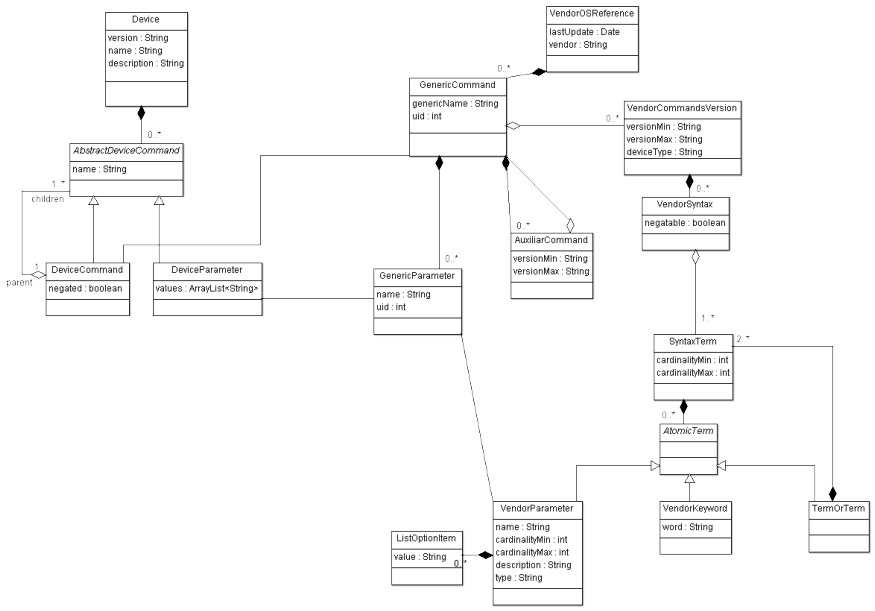

Figure 3.3: Le modèle de données UML pour notre structure de configuration

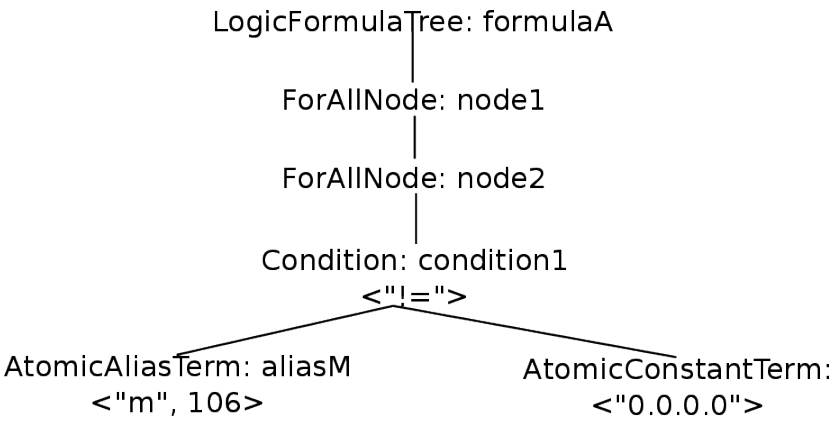

Figure 3.4: Formule logique sous forme d'arbre

```
<VendorOSReference lastUpdate="09/12/2013" vendor="CISCO">
  <GenericCommand genericName="version" uid="1">
      <GenericParameter name="num_version" uid="11"/>
      <VendorCommandsVersion versionMin="all" versionMax="all" deviceType="all">
          <VendorSyntax negatable="false">
              <SyntaxTerm cardinalityMin="1" cardinalityMax="1">
                  <VendorKeyword word="version"/>
                  <VendorParameter name="num_version" guid="11"/>
              </SyntaxTerm>
          </VendorSyntax>
      </VendorCommandsVersion>
  </GenericCommand>
   <!--    ....    -->
  <GenericCommand genericName="interface" uid="3">
      <GenericParameter name="name" uid="13"/>
      <GenericParameter name="option" uid="14"/>
      <VendorCommandsVersion versionMin="all" versionMax="all" deviceType="all">
          <VendorSyntax negatable="false">
              <SyntaxTerm cardinalityMin="1" cardinalityMax="1">
                  <VendorKeyword word="interface"/>
                  <VendorParameter name="name" guid="13"/>
                  <VendorParameter name="option" guid="14"/>
              </SyntaxTerm>
          </VendorSyntax>
      </VendorCommandsVersion>
  </GenericCommand>
  <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>
  <!--    ....    -->
  <GenericCommand genericName="ip directed-broadcast" uid="6">
      <GenericParameter name="access-list-number" uid="19"/>
      <CiscoCommandsVersion versionMin="10" versionMax="11" deviceType="all">
        <CiscoSyntax negatable="true">
          <SyntaxTerm cardinalityMin="1" cardinalityMax="1">
            <CiscoKeyword word="ip directed-broadcast"/>
            <CiscoParameter name="access-list-number" guid="19"/>
          </SyntaxTerm>
        </CiscoSyntax>
      </CiscoCommandsVersion>
      <CiscoCommandsVersion versionMin="12.0" versionMax="all" deviceType="all">
        <CiscoSyntax negatable="true">
            <SyntaxTerm cardinalityMin="1" cardinalityMax="1">
                <CiscoKeyword word="ip directed-broadcast"/>
                <CiscoParameter name="extended-access-list-number" guid="19"/>
            </SyntaxTerm>
        </CiscoSyntax>
      </CiscoCommandsVersion>
      <AuxiliarCommand versionMin="all" versionMax="all" guid="3"/>
  </GenericCommand>
</VendorOSReference>
```

Figure 3.5: Exemple de référence d'un fournisseur, Référence Cisco IOS

```
!
version 12.0
!
hostname Pomerol
!
interface Loopback0
ip address 10.10.10.3 255.255.255.255
ip router isis
```

(a) Fichier de configuration d'origine

```
<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 negated="false" name="ip router isis" ref_cmd="5">
   <DeviceParameter name="area-tag" ref_param="18"/>
  </DeviceCommand>
 </DeviceCommand>
</Device>
```

(b) Equivalent Meta-CLI file

Figure 3.6: Un exemple simple de configuration de l'appareil Cisco

CHAPITRE 4

COMMENT OPTIMISER LA RÉCUPÉRATION DE LA CONFIGURATION : VIRTUALISATION ET ÉVALUATION SÉLECTIVE

Le développement d'un réseau affecte fréquemment trois aspects : la gestion de ses dispositifs, la bande passante et la charge de travail des administrateurs réseau qui doivent configurer ces dispositifs. Dans Le présent chapitre nous proposons une méthode de récupération d'une partie d'un fichier de configuration en vue de sa correction. Pour ce faire, nous nous servons des concepts de programmation de haut niveau pour l'extraction de la configuration de l'équipement. Nous avons utilisé des concepts tels que les formules logiques ou encore Meta-CLI pour représenter la configuration extraite du dispositif afin de pouvoir tester certaines propriétés complexes d'un ou plusieurs dispositifs et déduire ainsi les informations restantes. Ces formules logiques sont en forme d'arbre contenant des paramètres extraits du dispositif. Ainsi notre algorithme doit seulement récupérer les paramètres nécessaires pour le calcul de la formule logique. Le contenu de ce chapitre a fait l'objet d'une publication voir ((Éric Lunaud Ngoupé et al., 2014))

4.1 VERS UNE VIRTUALISATION SÉMANTIQUE DES CONFIGURATIONS

Nous savons que les informations de configuration de dispositifs réseau sont stockées dans une structure de données interne, qui peuvent être accessible par divers moyens. Cependant, l'organisation avérée des paramètres, et la manière concrète de les atteindre, ne reflètent pas nécessairement une vue sémantiquement exacte des opérations de configuration dans toutes les situations. Dans ce paragraphe, nous proposons l'utilisation des règles de dépendance des configurations afin de fournir une virtualisation sémantique des paramètres. Nous montrerons ensuite comment cette virtualisation peut être superposée avec un minimum d'interférences sur les protocoles de configuration actuelle, en prenant en exemple le protocole Netconf. Toujours dans cette partie nous prenons une approche opposée à la question de la restauration d'un équilibre sémantique dans les configurations. Au lieu d'espérer un «good-for-all», pour l'organisation des paramètres nous suggérons plutôt l'utilisation de multiples structures «locally optimal» qui fournissent chacune une représentation fragmentée efficace de la configuration bien adaptée pour un seul service ou une tâche de gestion. Ces structures appelées « sous arbres virtuels », sont logiquement des sous-ensembles superposés d'une structure de configuration originale.

Historiquement, les structures de données des appareils réseau ont commencé comme un tout petit ensemble bien organisé de commandes et paramètres. Cependant, l'ajout de nouvelles fonctionnalités et services a au fil du temps conduit à une complexification progressive de ces structures. Il était très souvent nécessaire d'établir la compatibilité avec les versions antérieures du produit. Par conséquent aucune réorganisation n'était recommandée. Le résultat final est que la plupart des structures de configuration pour les périphériques actuels ont dépassé à plusieurs reprises leur coquille initiale et ont ainsi perdu leur cohésion originale. C'est le cas des paramètres qui gèrent par exemple

un service unique qui peuvent être dispersés à plusieurs endroits et souvent mélangés à l'intérieur des opérations de configuration d'autres paramètres provenant d'autres services. On peut citer l'exemple des réseaux privés virtuels (VPN), qui modifient en même temps la configuration des tables de routage, des systèmes d'adressage IP et d'autres paramètres et le plus souvent soumis aux dépendances sémantiques. La tendance actuelle tournée vers la virtualisation du réseau et son découplage de dispositifs physiques rend le problème encore plus complexe. Il n'est même pas garanti que pour tous les cas de figure, la construction d'une nouvelle configuration à partir de zéro conduirait à de meilleurs résultats. La recherche d'un modèle de données de configuration universel a commencé il y a longtemps avec des initiatives comme *Management Information Base (MIB)* (MacFaden et al., 2003), *Directory Enabled Networking (DEN)* (Strassner, 1999) et *Common Information Model* (cim, 2000). Le fait qu'une telle unification et des efforts de standardisation aient proliféré au fil des années donne matière à réflexion.

4.1.1 RELATIONS SÉMANTIQUES DANS LES VUES DE CONFIGURATION

Nous présentons ici une forme générale d'arbre appelé « arbre de configuration »qui peut être utilisée pour modéliser des structures hiérarchiques (ligne de commande et XML), ainsi que des dépendances de service entre les paramètres de ces structures.

Vues de Configuration sous forme d'arbre

Une vue de la configuration est l'organisation structurelle des informations de configuration à l'intérieur d'un dispositif, et l'ensemble des opérations fournies pour accéder et modifier cette structure. Traditionnellement, les dispositifs réseau offrent un nombre limité de vues. La première et la plus simple de ces vues est la manipulation directe des

fichiers de configuration, une opération fastidieuse et source d'erreurs qui nécessite de l'ingénieur réseau une connaissance approfondie de l'organisation du fichier, de la syntaxe et des exigences implicites. Bien que ces fichiers puissent regrouper les paramètres en sections, l'organisation des informations est surtout linéaire et offre généralement peu de structure.

L'utilisation d'une interface de ligne de commande (CLI) fournit un second point de vue améliorée de la configuration. L'accès aux paramètres de configuration se fait par l'utilisation de commandes qui permettent d'afficher et de modifier plusieurs paramètres à la fois. La syntaxe et l'organisation hiérarchique de ces commandes dans des modes et sous-modes permet une plus grande structuration de l'information. Un niveau plus élevé de fonctionnalité est offert en exécutant la syntaxe de base et la validation des paramètres. La modification fortuite des paramètres est réduite en divisant l'espace de configuration dans les modes et sous-modes avec chacun un nombre limité de valeurs accessibles. Des efforts plus récents ont conduit à des protocoles comme Netconf (Enns et al., 2011a) qui offrent des capacités de gestion de configuration vers SOAP via des messages en format XML.

Dans une session typique de Netconf, codée en XML les appels de procédures distantes (RPC) sont envoyés par une application à un dispositif, qui envoie à son tour une réponse RPC en donnant ou en reconnaissant la réception d'un ensemble de données d'une configuration XML complète ou partielle. Afin de parvenir à une telle communication normalisée, on a vu que le projet Netconf actuel définit un ensemble d'opérations de base qui doivent être pris en charge ou supporté par des devices ou dispositifs réseau. Parmi ces opérations, *get-config* récupère tout ou une partie d'une configuration spécifiée d'une source dans un format donné, *edit-config* charge la totalité ou une partie d'une configuration spécifiée à la configuration cible indiquée. La configuration est organisée

en une hiérarchie de paramètres assez précise (le plus souvent spécifique au fournisseur) qui prend la forme d'un document XML. Comme expliqué dans (Hallé et al., 2004), la configuration des dispositifs réseau tels que les routeurs et les commutateurs peuvent être représentés sous forme d'arbre où chaque nœud est une paire composée d'un nom et d'une valeur. Cet arbre représente la hiérarchie de paramètres inhérents à la configuration des périphériques. Cette structure fournit un modèle général pour les deux transformations tant pour les structures modernes de CLI que pour les structures d'XML supportées par Netconf. La figure 4.1 montre une partie d'un tel arbre de configuration. Sans aucune précision, un *get-config* récupère l'ensemble de la configuration. Cependant, le présent projet Netconf définit un procédé de récupération ou de filtrage d'une partie de la configuration que l'on peut appeler le filtrage de sous-arbres. Cette méthode est obligatoire dans tous les appareils Netconf ; il nécessite, une section <filtre> qui est une représentant d'un sous-document sur lequel la configuration doit être filtrée et, éventuellement, un espace de noms pour cette section.

Il y a, cependant, d'autres moyens de filtrer une configuration comme l'utilisation du langage XML appelé XPath (Bray et al., 2001). XPath est un langage d'arbres qui permet d'indiquer des emplacements arbitraires dans une structure. A cet égard, XPath est beaucoup plus souple que le filtrage de sous-arbre pour obtenir et définir les parties d'une configuration, comme dans EnSuite (Cridlig et al., 2005) une plateforme de gestion des réseaux basée sur Netconf. D'autres suggestions ont été d'utiliser un sous-ensemble sur mesure de XPath appelé LocationPath, le Language de liaison d'XML dénommé XLink (DeRose et al., 2001) et enfin le Protocole d'Accès de Configuration d'XML (XCAP) (Rosenberg, 2007). On a cependant eu une crainte, que XPath (ou un de ses sous-ensembles) ne soit un langage trop riche pour l'intégrer dans les périphériques réseau dont la priorité ne devrait pas être de traiter des données de contrôle et de

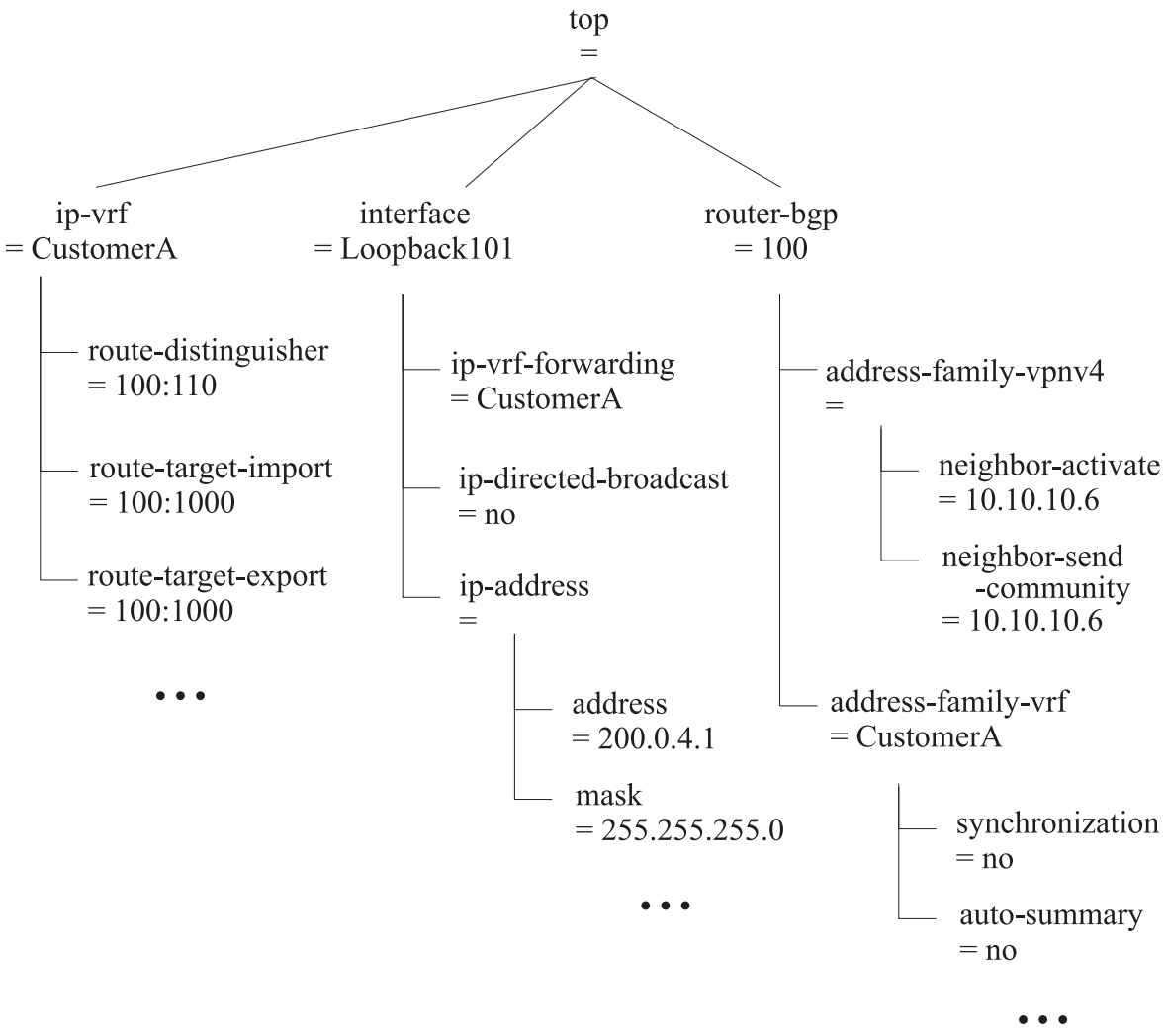

Figure 4.1: Une vue partielle d'une arborescence de configuration

gestion.

Dépendances sémantiques

Quelque soit le langage de sélection utilisé, les paramètres et les composantes de la configuration affectée par un service présentent des dépendances précises. Toutes ces dépendances doivent être étudiées et capturées par les modèles de gestion, afin de fournir des solutions efficaces. La logique de configuration (CL) (Hallé et al., 2005) a été développée spécifiquement pour exprimer les propriétés sur les arbres de configuration. Les formules de la logique de configuration utilisent les connecteurs booléens classiques de la logique des prédicats : $\wedge$ ("et"), $\vee$ ("ou"), $\neg$ ("non"), $\rightarrow$ ("implique"), à laquelle deux quantificateurs spéciaux sont ajoutés. Le quantificateur universel, identifié par "[]", indique un chemin délimité par des point-virgules dans l'arbre et qui impose qu'une formule soit vraie pour tous les nœuds à l'extrémité de ce chemin (le quantificateur existentiel se comporte de la même façon). D'autres approches formelles, telles que l'utilisation d'ontologies (Hui et Wuhan, 2006), (Cleary et al., 2005), ont été utilisées pour modéliser les dépendances de configuration. Par conséquent, le but de cette partie n'est pas de promouvoir la logique de configuration (CL) ou tout autre langage d'arbre comme un formalisme requis pour la gestion de configuration, mais son objectif est de montrer comment les dépendances de configuration peuvent être formellement exprimées et analysées.

Pour illustrer les dépendances sémantiques, nous prenons le cas du réseau privé virtuel (VPN). Un service de VPN (Rosen et Rekhter, 1999) est un réseau privé construit au sein d'un réseau public tel que le réseau d'un fournisseur de services. La majorité de la configuration d'un VPN est réalisée dans les routeurs placés à la frontière entre le client

et le réseau d'un fournisseur. Du côté du client, ces routeurs sont appelés *customer edge (CE) routers*, et du côté de l'opérateur, ils sont appelés *provider Edge (PE)*. La Figure 4.1 montre la partie d'une configuration de routeur pour un réseau privé virtuel. Dans la configuration d'un VPN, le *Virtual Routing and Forwarding Table* (VRF), nom spécifié pour la connectivité entre le routeur du côté fournisseur (PE) vers le routeur du côté client (CE) et le nom VRF configuré sur l'interface PE pour le lien CE doivent être compatibles. Cela met en lumière une première dépendance sur deux paramètres dans la configuration : la valeur du nœud `ip-vrf` et la valeur de `l'IP-vrf-forwarding` sous une interface spécifique.

Ceci n'est qu'un exemple de la propriété du réseau qui doit être validée sur les configurations des routeurs. En réalité, chaque service réseau impose des dizaines de ces contraintes, et à son tour des dizaines de différents services peuvent coexister sur un seul appareil. Dans (Hallé et al., 2004), des contraintes de service supplémentaires ont été décrites et formalisées.

Effets de valeurs sémantiquement dépendantes

Nous savons qu'une tâche de gestion de configuration devrait permettre de réduire le risque d'erreurs et les effets secondaires injustifiés. Pour atteindre cet objectif, la vue de configuration employée pour effectuer cette tâche doit réduire au minimum le nombre d'opérations distinctes exigées et ne fournir à l'utilisateur que les paramètres externes nécessaires (ou le moins de paramètres possible). La logique derrière ces exigences est que si les opérations les plus superflues et les informations de configuration sont mises à la disposition d'un utilisateur, cela augmente les chances que les opérations inutiles soient invoquées et les paramètres indépendants modifiés, ce qui perturberait effectivement la

cohérence de la configuration globale.

Par conséquent, dans un cas d'utilisation idéale, une tâche de configuration devrait consister à la récupération et à la modification appropriée d'une partie de la configuration, et à la mise à jour de cette partie modifiée dans le fichier de configuration. Si la vue de configuration est efficiente et la partie extraite logiquement et sémantiquement d'un seul bloc, on n'est pas tenu de vérifier l'intégrité de l'ensemble de la configuration : il suffit de s'assurer que le bloc est correct. La cohérence globale de la configuration est maintenue en effectuant des séquences de petites opérations atomiques qui préservent chaque cohérence locale.

Toutefois, la configuration du réseau fonctionne rarement de cette façon. Au lieu de cela, les paramètres à modifier lors de l'ajout d'une nouvelle fonctionnalité à un réseau ou tout simplement leur réparation sont souvent dispersés à travers différentes hiérarchies. Du point de vue de la CLI, ceci est illustré par la nécessité de passer d'un mode à l'autre pour la configuration d'un service réseau. Les conséquences d'utiliser une vue de configuration sous-optimale pour une tâche de configuration donnée sont nombreuses :

1. *Augmentation du temps de traitement et de communication* : Ceci devient particulièrement vrai dans des environnements gérés à distance à l'aide de protocoles tels que Netconf. Chaque demande supplémentaire à une partie d'une configuration nécessite au moins une paire de requête/réponse de RPC pour chaque dispositif, ce qui augmente la charge sur le réseau causée par le contrôle et la gestion du trafic. En outre, les demandes supplémentaires conduisent à leur tour à un traitement supplémentaire dans l'appareil, puisque chaque RPC exige la collecte de la configuration, la construction de l'arbre XML basé sur elle, ensuite le filtrage (plusieurs fois) de cet arbre pour produire l'ensemble de résultats souhaités.
2. *Augmentation du risque d'incohérence* : L'exemple de la section 4.1.1 a prouvé que

plusieurs paramètres placés à différents endroits dans une structure de configuration peuvent être dépendants les uns des autres, c'est-à-dire que la modification d'un paramètre pourrait entrainer la modification de certains autres afin de conserver la cohérence. Si l'accès à ces paramètres sémantiquement liés ne peut être fait qu'avec plus d'une requête et opération d'édition, le système court le risque qu'un utilisateur exécute uniquement une des modifications et laisse le système dans un état incohérent. Il affecte également la capacité de CLI de valider l'intégrité de la configuration au moment de quitter un sous-mode de configuration. Actuellement, ceci est impossible car les paramètres de configuration dans un mode unique forment rarement un ensemble d'un seul bloc qui peut être examiné de façon indépendante pour vérifier la validité.

3. *Perte d'atomicité dans les opérations de configuration* : En raison de ce facteur précédent, il est bien possible d'utiliser les opérations comme *get/set* pour annuler une action sans toutefois permettre de ramener le système à son dernier état cohérent. Ceci a pour effet de rendre maladroites les opérations de restauration *« commit »* et *« rollback »* en impliquant un mécanisme non systématique de points de contrôle de configuration. Notez en outre que (Hallé et al., 2005) ont montré que les dépendances concernant les paramètres sur différents sous-arbres ne peuvent pas être exprimés en XPath 1.0. Elles requièrent l'utilisation de quantificateurs qui peuvent seulement être trouvés dans le langage de requêtes XML (XQuery) (Boag et al., 2005), qui est un langage encore plus expressif. Par conséquent, en restaurant la cohérence et l'atomicité du côté client et en permettant des requêtes complexes pour obtenir et définir les paramètres de configuration dans une opération en bloc, ceci exigerait au minimum les capacités de XQuery à l'intérieur d'un dispositif. Malheureusement cette fonctionnalité est très peu susceptible d'être mise en œuvre.

4.1.2 VIRTUALISATION DES INFORMATIONS DE CONFIGURATION

Sur le plan de la construction, la structure arborescente trouvée dans les hiérarchies de CLI et XML est déjà un regroupement d'information en familles de paramètres sémantiquement liés. Cependant, elle n'est pas suffisante pour toutes les tâches possibles de configuration. Nous proposons donc de superposer sur un arbre donné plusieurs arborescences virtuelles. Nous prenons donc une approche orthogonale comme celle décrite dans (Cridlig et al., 2005), où au lieu de permettre les appels des requêtes des multiples parties disjointes d'un arbre à la fois, on utilise à la place une correspondance à un arbre virtuel.

Les sous-arbres virtuels

Pour ce faire, nous suggérons l'utilisation de sous-arbres virtuels. Un sous-arbre virtuel est simplement une structure arborescente qui se superpose à l'arbre, d'origine, et dont le contenu reflète les relations sémantiques entre les paramètres. Chaque nœud du sous-arbre virtuel correspond à un nœud de l'arbre source. Nous supposons que les sous-arbres virtuels préservent la relation de condition de parent originale. Dans ce cas particulier, la construction d'un sous-arbre virtuel revient simplement à prendre les nœuds de l'arbre original et à les lier selon la hiérarchie d'origine. Pour construire un sous-arbre virtuel pour un service réseau spécifique, un algorithme simple peut être conçu. Cet algorithme regroupe d'abord toutes les formules de la logique de configuration (CL) exprimant les contraintes liées à ce service, et ensuite sélectionne tous les nœuds qui apparaissent le long des quantificateurs dans chaque formule CL.

Considéront le service VPN représenté dans la section 4.1.1. Dans le cas de la formule

qui y est présentée, on voit trois quantificateurs, et donc trois chemins sont présents. L'algorithme sélectionne tous les nœuds (peu importe leurs valeurs) dont le nom est *ip-vrf* ou l'interface qui se trouve directement sous le nœud supérieur, et tous les nœuds dont le nom est ip-vrf-*forwarding* directement en-dessous d'un des nœuds de l'interface précédemment sélectionné. L'ajout de nouvelles contraintes au service VPN, comme cela a été fait dans (Hallé et al., 2004), conduirait à plus de formules logiques de configuration (CL) et par conséquent à une plus large sélection des nœuds pour le sous-arbre virtuel VPN. Il est important de noter que l'algorithme de sélection inclut seulement les nœuds qui sont explicitement mentionnés dans le chemin d'un des quantificateurs ; en particulier, il n'inclut pas automatiquement le sous-arbre sous un nœud sélectionné, à moins que chaque nœud de ce sous-arbre soit également mentionné dans l'un des quantificateurs.

Il peut y avoir beaucoup de sous-arbres virtuels définis au-dessus d'une simple structure arborescente originale. Dans un contexte XML, des sous-arbres virtuels peuvent être simplement ajoutés sur une structure originale en marquant, pour chaque nœud, à quel sous-arbre virtuel il appartient. Ceci peut être réalisé en assignant un espace de nom aux nœuds concernés. Par exemple, on pourrait facilement ajouter l'espace de noms « service : vpn »(ou n'importe quelle autre chaîne d'ailleurs) aux nœuds XML de l'arbre d'origine pour les marquer individuellement en tant qu'élément du sous-arbre virtuel de service de VPN. Comme indiqué dans le cahier des charges de nommage XML (Bray et al., 2001), plusieurs préfixes d'espaces de noms peuvent être déclarés comme attributs d'un élément unique : il est donc possible de superposer un certain nombre de sous-arbres virtuels d'une manière transparente au-dessus d'une structure déjà définie. L'interrogation d'un sous-arbre virtuel se fait de la même manière que l'interrogation d'un arbre original. Il peut y avoir beaucoup de sous-arbres virtuels définis au-dessus d'une simple structure arborescente originale. Dans un contexte XML, des sous-arbres

virtuels peuvent être simplement ajoutés sur une structure originale en marquant, pour chaque nœud, à quel sous-arbre virtuel il appartient. Ceci peut être réalisé en assignant un espace de nom aux nœuds concernés. Par exemple, on pourrait facilement ajouter l'espace de noms « service : vpn »(ou n'importe quelle autre chaîne d'ailleurs) aux nœuds XML de l'arbre d'origine pour les marquer individuellement en tant qu'élément du sous-arbre virtuel de service de VPN. Comme indiqué dans le cahier des charges de nommage XML (Bray et al., 2001), plusieurs préfixes d'espaces de noms peuvent être déclarés comme attributs d'un élément unique : il est donc possible de superposer un certain nombre de sous-arbres virtuels d'une manière transparente au-dessus d'une structure déjà définie. L'interrogation d'un sous-arbre virtuel se fait de la même manière que l'interrogation d'un arbre original.

Avantages de la méthode

Les avantages de l'utilisation du groupement virtuel de données de configuration sont multiples. Tout d'abord, on peut interroger une configuration du périphérique réseau en suivant la syntaxe *get-config* d'origine en utilisant le filtrage de sous-arbre. Il suffit de formuler sa requête en demandant l'ensemble de la configuration et en filtrant uniquement par l'espace de noms. Bien que les dépendances de service soient exprimées en arbre de logique formelle, aucun XPath, XQuery ou logique de configuration n'est nécessaire, du côté client comme du côté serveur, au moment où la requête est traitée. Par conséquent, la méthode proposée se dégrade correctement sur les systèmes qui ne supportent pas les arbres virtuels : dans ce cas, les espaces de noms peuvent simplement être ignorés, et on revient à un mécanisme standard de récupération de la configuration.

Un autre avantage est qu'il fournit une meilleure granularité pour le contrôle de sécurité

et d'accès ; selon l'identité du demandeur de requête, seulement un certain nombre de sous-arbres virtuels peuvent être rendus disponibles, et tout paramètre qui ne relève pas de ces grappes ou groupes sélectionnés est automatiquement exclu de toute demande ou réponse que cet utilisateur fait sans aucun contrôle explicite sur le périphérique réseau. Par exemple, l'utilisateur Alan Smith pourrait être autorisé à accéder à des espaces de noms « service : ACL »et « service : vlan »et ne serait pas en mesure de voir un nœud ailleurs.

La méthode fournit un découplage entre la représentation interne d'information et la méthode d'accès. Au lieu qu'un dispositif rassemble l'ensemble de la configuration et filtre ensuite (plusieurs fois) l'arbre XML afin de produire l'ensemble de résultats, le dispositif ne peut exécuter que des opérations pré-écrites qui rassemblent et construisent l'arbre XML requis. Si l'arbre virtuel est intelligemment construit, il réduira en même temps le nombre d'appels émis en donnant toutes les informations nécessaires en un seul appel. La méthode peut également mettre en garde contre les effets secondaires potentiels dans d'autres services ; en construisant un arbre virtuel pour un ensemble de services dépendants, il suffit de regarder à quels autres sous-arbres virtuels un nœud particulier est également joint. Enfin, il faut remarquer que la méthode proposée ne prévoit pas trop de charge sur le dispositif : l'analyse des dépendances de chaque service peut être calculée a priori et chaque arbre virtuel peut être défini de façon statique et rester valable jusqu'à ce que les dépendances soient modifiées. Dans un contexte Netconf, à la limite, seulement un support minimal de XPath est nécessaire du côté de l'appareil (le cas échéant), puisque toutes les déclarations complexes nécessaires à l'obtention des parties spécifiques d'une configuration peuvent être prédéfinies.

4.2 EXACTITUDE DE CONFIGURATION D'UN DISPOSITIF DE RÉSEAU

Dans cette partie, nous définissons d'abord formellement le concept de configuration de dispositif réseau et présentons ensuite la notion d'exactitude de configuration en donnant des exemples de contraintes sur des paramètres de configuration à partir d'un service réseau existant.

4.2.1 DISPOSITIFS ET CONFIGURATIONS

Ce que nous définissons comme un dispositif peut être une composante matériel telle qu'un routeur, un commutateur, un point d'accès sans fil ou une passerelle. C'est donc un matériel physique ou virtuel utilisé pour diriger des paquets dans un réseau informatique. Ce dispositif a ses propres caractéristiques internes et des interfaces de communication et dépend d'habitude du fabricant pour des dispositifs physiques ou de leur version pour les dispositifs virtuels. La caractéristique principale d'un dispositif est que son comportement par défaut peut être modifié de façon dynamique par configuration. La configuration d'un dispositif correspond à un ensemble de paramètres et de valeurs associés, enregistrés par le dispositif. Ces réglages permettent de personnaliser le dispositif par exemple en :

— Réglant les interfaces physiques,

— Appliquant les règles de redirection de données,

— En adaptant le comportement générique du dispositif au réseau, ou en modifiant le comportement du dispositif pour qu'il corresponde aux événements précédents.

Les éléments éditables dans une configuration sont appelés des paramètres. Ces paramètres sont des éléments utilisés pour personnaliser le comportement d'un dispositif pour un réseau donné. Les valeurs des paramètres peuvent être numériques (par exemple,

une adresse IP ou un numéro de masque de sous-réseau), une chaine de caractère (par exemple, le nom d'une interface d'Ethernet) ou être plus complexes (par exemple, l'état d'un port).

La forme d'une configuration dans le dispositif dépend de son fabricant. Par exemple, le fabricant Cisco utilise un fichier batch contenant « Command Line »Interface" (l'Interface de Ligne de commande-CLI-) des commandes et des valeurs associées. Ce fichier est hiérarchisé, puisque certaines commandes permettent au dispositif de pouvoir changer de mode ou de sous mode d'entrer ou d'en sortir (un contexte particulier, comme la configuration d'une interface spécifique). Plus généralement, nous considérerons sans perdre de vue la généralité que nous pouvons réduire la configuration de chaque fabricant de dispositif (Cisco, Juniper, OpenFlow, etc) à une configuration générique en format XML (ou celui appelé "Meta-CLI" vu précédemment), comme le montre le schéma 3.6. Cette uniformisation des configurations ne fait pas partie du champ d'étude du présent chapitre car cela a déjà été étudié dans des travaux précédents (Hallé et al., 2012).

4.2.2 EXACTITUDE DE CONFIGURATION

Les contraintes sont utilisées dans le processus de configuration interactif pour prendre en compte les dépendances entre les caractéristiques de plusieurs ou du même fabricant (SAP, 2013). Les paramètres de configuration d'un dispositif ne peuvent pas prendre n'importe quelle valeur ; pour qu'une configuration soit correcte, un certain nombre de règles doivent être appliquées selon les services qui apparaissent. Les contraintes décrivent habituellement les conditions à remplir pour une configuration cohérente. Les contraintes peuvent être utilisées, par exemple, pour s'assurer qu'un conflit dans la configuration n'apparaisse lorsque la cohérence de la configuration n'est plus respectée.

Nous illustrons ceci en utilisant un exemple simple basé sur la configuration d'un Réseau local Virtuel (VLAN), généralement gérée par des commutateurs de réseau et qui permettent à un réseau d'être divisé en segments logiques. Chaque port d'un commutateur peut être assigné à un VLAN donné ; les ports qui sont assignés au même VLAN peuvent communiquer à la Couche 2 du modèle OSI tandis que les ports non assignés au même VLAN requièrent une communication à la Couche 3. Tous les commutateurs qui doivent partager la Couche 2 pour la communication intra-VLAN doivent être connectés par un lien appelé chemin réseau (trunk), qui joint deux interfaces, une sur chaque commutateur, et ces interfaces devraient être encapsulées dans le même mode. IEEE 802.1Q (IEEE, 2003) et VTP (Cisco, 2013b) sont deux protocoles connus pour des chemins réseau VLAN.

1. Deux dispositifs ne peuvent avoir la même adresse pour plus d'une interface. Puisque les adresses IP peuvent être manuellement configurées pour chaque interface sur un dispositif, des conflits entre adresses peuvent survenir. Ceci peut faire en sorte qu'une des interfaces partageant la même adresse paraisse défectueuse et cause des interruptions dans les communications.
2. Chaque dispositif doit être un client VTP ou un serveur VTP et un seul serveur peut exister. Cela signifie que pour chaque commutateur où VTP est activé sur le mode client, il doit exister un commutateur où VTP est activé sur le mode serveur. En d'autres termes ces contraintes signifient : pour tout noeud de racine avec le nom "device" et la valeur s1, il existe un noeud sous "device = s1" ayant pour nom "vtp mode" et pour valeur x, tel que x est égal à "client", ou qu'il existe un noeud sous "device = s1" ayant pour nom "vtp mode" et pour valeur x, tel que x est égal à "server".
3. Tous les dispositifs doivent être dans le même domaine VTP. Ici nous pouvons

utiliser le prédicat appelé SwitchesInSameVTPDomain qui signifie que pour chaque couple de dispositifs par exemple s1 et s2, le prédicat doit être vrai.

Ceci n'est qu'un petit échantillon des contraintes de configuration qui peuvent être obtenues. La plupart des contraintes imposent des restrictions sur des valeurs de paramètres et que plusieurs fois, les valeurs à l'intérieur d'un dispositif sont reliées aux valeurs d'autres paramètres dans d'autres dispositifs. (Hallé et al., 2012) donne une présentation plus détaillée des VLANs.

4.3 ÉVALUATION SÉLECTIVE DES CONTRAINTES DE CONFIGURATION

L'analyse des protocoles et des outils de gestion présentés dans les chapitres précédents soulève un certain nombre de questions. D'abord, très peu de ces solutions abordent l'exactitude de configuration. Des protocoles de gestion permettent des modifications de configurations d'un seul dispositif à la fois et peuvent tout au plus exécuter la vérification syntaxique de base des modifications sur les paramètres à appliquer. Les outils de gestion, de leur côté, permettent l'automatisation de tâches diverses sur des dispositifs multiples, mais offrent très peu en termes d'expression et de vérification de l'exactitude d'une configuration selon des règles établies. De toutes les solutions, seul ValidMaker (et, dans une moindre mesure, Puppet et Chef) offre de telles fonctionnalités.

Hélas, tous ces outils s'attendent à ce que les configurations soient gérées dans un emplacement centralisé. Dans le cas de ValidMaker, la vérification de l'exactitude de la configuration est faite localement sur une copie du réseau complet. À moins que les modifications du réseau entier ne soient gérées que par ValidMaker, la configuration complète de chaque dispositif doit être extraite chaque fois qu'une vérification est exécutée pour éviter des problèmes de synchronisation. Ceci, à son tour, entraine une

consommation indésirable de la bande passante, au point où les données de contrôle occupent une portion inacceptable de la bande passante totale.

La bande passante consommée pour une telle procédure peut être évaluée par un calcul simple ; les réseaux de 1,000 dispositifs ne sont pas rares et les configurations non compressées peuvent excéder la taille du NVRAM du dispositif (128 Ko est une valeur typique) jusqu'à un facteur de 3 ; ceci indique que la vérification de la configuration revient à extraire presque 400 Mo de données chaque fois. À moins que les modifications du réseau entier ne soient gérées par le même outil centralisé, la configuration complète de chaque dispositif doit être extraite chaque fois qu'une vérification est exécutée pour éviter des problèmes de synchronisation. Cette question a longtemps été soulevée ; les débats sur les capacités d'extraction de configuration du protocole Netconf d'il y a une décennie mentionnaient déjà : « un mécanisme de filtrage moins puissant entraînerait plus de trafic dans le réseau et augmenterait ainsi la charge du réseau et de charge CPU, puisque les gestionnaires devraient extraire plus de données que nécessaire si le filtrage requis ne peut pas être fait du côté de l'agent. » [1]

Dans cette section, nous présentons une stratégie qui vise à réduire la quantité de données de configuration qui doit être transférée à un lieu centralisé pour des besoins de vérification. Nous appelons cet ensemble de techniques l'évaluation sélective des configurations, puisqu'ils sont semblables dans l'esprit au concept du même nom dans les langages de programmation (Henderson et al., 1976; Friedman et Wise, 1976).

1. `http://psg.com/lists/netconf/netconf.2004/msg00448.html`

4.3.1 CHOIX DE SOUS-ENSEMBLES DE CONFIGURATION AVEC NETCONF

L'application récursive des règles sémantiques présentées dans le Tableau I offre un algorithme concret pour l'évaluation de n'importe quelle formule de CL sur n'importe quelle configuration. Chaque fois qu'un quantificateur doit être évalué, des valeurs pertinentes, compte tenu des variables actuellement définies, sont interrogées sur la configuration grâce à la fonction d'estimation ou d'évaluation ν. Chaque valeur de ce type engendre une nouvelle sous-formule devant être évaluée de façon récursive. Malheureusement, tel que souligné plus haut, un tel algorithme suppose l'accès aléatoire à n'importe quelle partie de la configuration à tout moment. Considérons par exemple l'expression suivante :

$$[a = x_1]\,[b = x_2]\,[a = x_1\,;\, c = x_3]\,[b = x_2\,;\, d = x_4] \qquad x_1 = x_2 \rightarrow x_3 = x_4$$

Appliquer la sémantique du Tableau I demandera d'abord d'interroger toutes les valeurs du paramètre a ; pour chaque valeur de ce type, on interrogera alors toutes les valeurs du paramètre b, suivi par toutes les valeurs du paramètre c sous le a précédemment interrogé. Finalement, toutes les valeurs du paramètre d sous b seront interrogées et les valeurs extraites seront comparées. Supposant qu'ils sont égaux, l'algorithme va revenir en arrière pour chercher une nouvelle valeur du paramètre d sous b et finalement une nouvelle valeur du paramètre c sous a, et ainsi de suite. Clairement, ceci est seulement faisable si on peut aller chercher rapidement ces différentes parties de la configuration, ce qui exclut l'ouverture et la fermeture d'une connexion à un dispositif chaque fois qu'une nouvelle valeur doit être obtenue. Par conséquent l'accès local à la configuration semble être la seule option raisonnable.

Cependant, la connaissance de la propriété formelle à évaluer peut être utilisée à bon escient afin de réduire la quantité d'informations qui doit en réalité être téléchargée de chaque dispositif. Une première étape dans l'évaluation sélective d'une configuration pour des buts de vérification consiste à filtrer les parties d'une configuration globale qui ne sont pas pertinentes pour la formule CL à vérifier. Dans l'exemple précédent, clairement, aucun chemin d'accès dans l'arbre d'un dispositif autre que a, b, a/c et b/d n'a d'impact sur le résultat de l'algorithme d'évaluation. Par conséquent, on peut seulement télécharger les parties de la configuration correspondant aux chemins pertinents. Dans un tel arrangement, l'algorithme fonctionne toujours d'une façon centralisée, mais fonctionne sur une version élaguée de la configuration globale où seules les parties pertinentes de l'arbre global ont été maintenues.

Il s'avère que les protocoles de gestion de réseau offrent exactement cette sorte de fonctionnalité. En particulier Netconf, qui selon la RFC6241, offre ce qui s'appelle le filtrage de sous-arbre XML, qui est un mécanisme qui permet à une application de choisir précisément les sous-arbres XML à inclure dans une réponse de message RPC *get* et *get-config*. Il est offert un petit ensemble de filtres pour l'inclusion et pour une correspondance exact de contenu sélectionné, qui permet certains mécanismes de sélection utile (mais aussi très limité). Conceptuellement, un filtre de sous-arbre est composé de zéro ou plus d'élément de sous-arbre, qui représentent les critères de sélection de filtre (Enns et al., 2006). À chaque niveau de confinement à l'intérieur d'un sous-arbre, l'ensemble de noeuds apparentés est logiquement traité par le serveur pour déterminer si son sous-arbre et son chemin d'accès d'éléments à la racine sont inclus dans la sortie du filtre. Une première possibilité consiste à choisir des noeuds selon l'espace de noms auquel ils appartiennent. Si l'attribution d'espaces de noms aux noeuds est faite soigneusement, les noeuds peuvent être choisis d'une manière très flexible, tel que suggéré dans (Hallé

et al., 2012).

Un noeud de sélection est un noeud de feuille vide à l'intérieur d'un filtre. Il représente un filtre de « sélection explicite »sur le modèle de données sous-jacentes. La présence de tout noeud de sélection dans un ensemble de noeuds apparentés emmènera le filtre à sélectionner le(s) sous-arbre(s) indiqué(s) et supprimera la sélection automatique de l'ensemble tout entier des noeuds apparentés dans le modèle de données sous-jacentes, comme dans le bout de code suivant :

```
<filter type="subtree">
<top xmlns="http://example.com/schema/1.2/config">
<configuration_file/>
</top>
</filter>
```

Finalement, un "noeud de correspondance de contenu" est un noeud de feuille abritant un contenu simple. Il peut être utilisé pour le choix d'une partie ou de tous ses nœuds apparentés (frères) pour le rendement du filtre et il représente un filtre de correspondance exacte sur le contenu d'élément du noeud de feuille. Les contraintes suivantes s'appliquent aux noeuds de correspondance de contenu voir schéma 4.2

```
<filter type="subtree">
  <top xmlns="http://example.com/schema/1.2/config">
      <config_file>
        <parameter>
          <name_para>ipadress</name_para>
        </parameter>
      </config_file>
  </top>
</filter>
```

Figure 4.2: noeuds de correspondance

Dans cet exemple, les noeuds <config_file> et <parameter> sont des noeuds de retenue (confinement), et <name_para> est un noeud de correspondance de contenu. noeud

$$
\begin{array}{rcl}
([\overline{p} = \overline{x}\,; p = x] : \varphi) \wedge \psi & \Leftrightarrow & [\overline{p} = \overline{x}\,; p = x] : (\varphi \wedge \psi) \\
([\overline{p} = \overline{x}\,; p = x] : \varphi) \vee \psi & \Leftrightarrow & [\overline{p} = \overline{x}\,; p = x] : (\varphi \vee \psi) \\
(\langle \overline{p} = \overline{x}\,; p = x \rangle : \varphi) \wedge \psi & \Leftrightarrow & \langle \overline{p} = \overline{x}\,; p = x \rangle : (\varphi \wedge \psi) \\
(\langle \overline{p} = \overline{x}\,; p = x \rangle : \varphi) \vee \psi & \Leftrightarrow & \langle \overline{p} = \overline{x}\,; p = x \rangle : (\varphi \vee \psi) \\
\neg\langle \overline{p} = \overline{x}\,; p = x \rangle : \varphi & \Leftrightarrow & [\overline{p} = \overline{x}\,; p = x] : \neg\varphi \\
\neg[\overline{p} = \overline{x}\,; p = x] : \varphi & \Leftrightarrow & \langle \overline{p} = \overline{x}\,; p = x \rangle : \neg\varphi \\
([\overline{p} = \overline{x}\,; p = x]\,\varphi) \rightarrow \psi & \Leftrightarrow & \langle \overline{p} = \overline{x}\,; p = x \rangle\,(\varphi \rightarrow \psi) \\
(\langle \overline{p} = \overline{x}\,; p = x \rangle\,\varphi) \rightarrow \psi & \Leftrightarrow & [\overline{p} = \overline{x}\,; p = x]\,(\varphi \rightarrow \psi) \\
\varphi \rightarrow ([\overline{p} = \overline{x}\,; p = x]\,\psi) & \Leftrightarrow & [\overline{p} = \overline{x}\,; p = x]\,(\varphi \rightarrow \psi) \\
\varphi \rightarrow (\langle \overline{p} = \overline{x}\,; p = x \rangle\,\psi) & \Leftrightarrow & \langle \overline{p} = \overline{x}\,; p = x \rangle\,(\varphi \rightarrow \psi)
\end{array}
$$

Tableau 4.1: Réécriture des règles pour la forme normale prenex dans CL, où φ et ψ sont des expressions arbitraires

apparenté de <name> n'est spécifié (et donc aucun noeud de retenue ni de sélection), tous les noeuds apparentés de <name_para> sont renvoyés dans le résultat du filtre.

4.3.2 FILTRAGE SUPPLÉMENTAIRE GRÂCE À L'ÉVALUATION LOCALE

Considérons la formule suivante :

$$
\begin{array}{rcl}
\varphi \wedge (\psi \vee \psi') & \Leftrightarrow & (\varphi \wedge \psi) \vee (\varphi \wedge \psi') \\
\varphi \vee (\psi \wedge \psi') & \Leftrightarrow & (\varphi \vee \psi) \wedge (\varphi \vee \psi') \\
\neg(\varphi \vee \psi) & \Leftrightarrow & \neg\varphi \wedge \neg\psi \\
\neg(\varphi \wedge \psi) & \Leftrightarrow & \neg\varphi \vee \neg\psi \\
\neg\neg\varphi & \Leftrightarrow & \varphi
\end{array}
$$

Tableau 4.2: Réécriture des règles pour la forme normale disjonctive dans CL, où φ, ψ et ψ' sont des expressions arbitraires.

$$\langle \mathrm{a} = x_1 \rangle \, [\mathrm{b} = x_2] \, \langle \mathrm{a} = x_1 \, ; \mathrm{c} = x_3 \rangle$$

$$[\mathrm{a} = x_1, \mathrm{c} = x_3 \, ; \mathrm{d} = x_4] \, \langle \mathrm{b} = x_2 \, ; \mathrm{e} = x_5 \rangle$$

$$x_1 = x_2 \wedge x_3 > x_1 \wedge x_4 = x_3 \wedge x_5 > x_2$$

Pour cette formule, le filtrage de sous-arbres de base correspond dans la figure 2.13 au seul envoi des noeuds inclus dans la zone en pointillés de l'arbre. Il est possible, cependant, de filtrer encore plus la quantité de données devant être envoyée au validateur centralisé en exécutant une meilleure analyse plus fine des valeurs réelles qui pourraient demander à être comparées avec d'autres. Cette méthode est présenté ci-dessous.

1) *Réécriture de PNF et DNF* : la première étape est directe et consiste à récrire l'expression de CL sous la forme normale prenex (Mendelson, 1997). Ceci a pour effet de produire une expression logiquement équivalente, mais où tous les quantificateurs apparaissent au début de la formule. Le Tableau Le Tableau 4.1 liste les règles de réécriture qui sont appliquées de manière répétée jusqu'à l'obtention de la forme prenex.

La deuxième partie de cette étape consiste à réécrire la partie non-quantifée de l'expression sous la forme normale disjonctive (Disjunctive Normal Form) (DNF). Par le résultat des manipulations précédentes, la partie non-quantifiée de la formule est concentrée en toute fin d'expression et peut être transformée en DNF en appliquant de manière répétitive un certain nombre de règles de transformation décrites dans le Tableau 4.2.

Le résultat de cette étape est une expression de la forme suivante :

$$Q_1 Q_2 \ldots Q_q (e_{1,1} \wedge \cdots \wedge e_{m_1,1}) \vee \cdots \vee (e_{n,1} \wedge \cdots \wedge e_{m_n,n})$$

Où tous les quantificateurs sont présents au début de la formule, suivis par une expression de la forme $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n$. Chaque φ_i est lui-même un terme de la forme $e_1 \wedge e_2 \wedge \cdots \wedge e_m$, où e_i est une expression de la forme $x_i \star i_j$ pour un certain opérateur binaire $\star$. Par les règles du Tableau 4.1, cette expression peut alors se réécrire comme suit :

$$(Q_1 Q_2 \ldots Q_q(e_{1,1} \wedge \cdots \wedge e_{m_1,1})) \vee \cdots \vee (Q_1 Q_2 \ldots Q_q(e_{n,1} \wedge \cdots \wedge e_{m_n,n})) \tag{4.1}$$

Nous avons maintenant atteint un point où la contrainte originale a été décomposée en un certain nombre « d'alternatives »dont chacune est une formule quantifiée en PNF composée seulement de conjonctions de comparaisons booléennes entre les variables. La valeur de la contrainte globale peut être obtenue à partir de la disjonction booléenne de chaque alternative. Puisque chaque alternative peut être traitée indépendamment des autres, nous décrivons le traitement d'une alternative unique dans les lignes qui suivent.

2) *Identifier des chaînes de dépendances entre variables* : la deuxième étape doit identifier les dépendances entre les variables de l'expression qui peuvent être évaluées localement. Ceci est fait en analysant premièrement la partie quantifiée de l'expression et en extrayant toutes les chaînes maximales de quantificateurs $Q_1 Q_2 \ldots Q_n$ tel que Q_{i+1} prolonge le chemin d'accès défini dans Q_i par un segment. Ces quantificateurs n'ont pas besoin de se suivre directement dans la formule originale, cependant ils doivent arriver dans le bon ordre. Pour chaque chaîne, toutes les contraintes impliquant des variables apparaissant dans la chaîne sont alors ajoutées, produisant un ensemble de formules de CL partielles contenant toutes les contraintes locales de chaque chaîne.

Dans nos expressions exemples, deux chaînes peuvent être extraites, avec leurs contraintes respectives, à savoir :

$$[\mathrm{a} = x_1]\ \langle \mathrm{a} = x_1\,;\, \mathrm{c} = x_3 \rangle\ \langle \mathrm{a} = x_1, \mathrm{c} = x_3\,;\, \mathrm{d} = x_4 \rangle \qquad x_3 > x_1 \wedge x_4 = x_3$$

$$et \qquad [\mathrm{b} = x_2]\ \langle \mathrm{b} = x_2\,;\, \mathrm{e} = x_5 \rangle\ x_5 > x_2$$

3) *Identifier des dépendances inter-chaînes* : à la fin de l'étape 3, il est possible que quelques conditions entre les variables n'aient été incluses dans aucune expression de chaîne. Dans notre exemple, c'est le cas pour $x_1 = x_2$, puisqu'elle compare les valeurs des variables entre deux chaînes distinctes. Pour préparer chaque expression de chaîne pour le traitement local, nous annotons chaque expression en identifiant les variables sur lesquelles une dépendance inter-chaînes existe. Dans notre exemple présent, les variables x_1 et x_2 seraient marquées comme telles.

4) *Évaluer localement* : l'ensemble complet d'expressions de chaînes est ensuite envoyé à chaque dispositif, qui doit les évaluer localement. Cette évaluation revient simplement à construire un arbre et/ou pour chaque chaîne, tel qu'indiqué dans la figure 4.3. Cet arbre est alors débarrassé de tout noeud et branches qui ne satisfont pas une des conditions booléennes attachées à cette chaîne. Dans l'arbre précédent, les branches pour lesquelles une condition est évaluée à faux sont en pointillés. Par exemple, dans l'arbre le plus à gauche, le noeud "8" est éliminé parce qu'aucun de ses enfants ne satisfait la condition $x_4 = x_3$, tandis que le noeud "4" est éliminé parce qu'il viole la condition $x_3 > x_1$.

Cette étape garantit une retro-compatibilité pour les dispositifs qui ne prennent pas en charge l'évaluation locale de contraintes. Au lieu d'exécuter un filtrage complet de sa configuration, un dispositif peut choisir simplement de filtrer la configuration en gardant

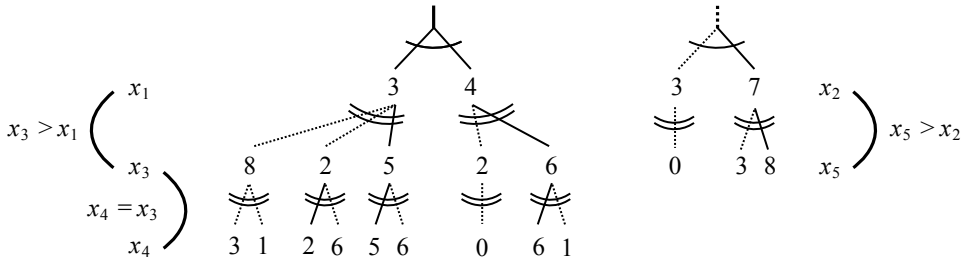

Figure 4.3: 4.3 : l'arbre et-ou pour la configuration de la figure 2.13, pour les deux chaînes de dépendance identifiées plus haut. Les arcs simples indiquent des conjonctions, les arcs doubles indiquent des disjonctions.

seulement les chemins d'accès apparaissant dans les quantificateurs, sans évaluation des conditions booléennes. Dans un cas extrême, le dispositif peut simplement ignorer les expressions de la chaîne et retourner l'ensemble de sa configuration. Dans les deux cas, exécuter un filtrage plus grossier (brute) de la configuration n'a aucun impact sur la validité du résultat final de l'algorithme.

5) *Transmettre l'arbre qui en résulte et évaluer* : Dans la dernière étape, chaque dispositif renvoie au validateur centralisé la configuration filtrée résultant de son traitement local des expressions de chaîne. Dans les arbres de la figure 4.3, ceci correspond seulement aux chemins d'accès, en commençant par la racine, qui n'est pas en pointillés. On peut voir que la fraction de l'arbre qui doit effectivement être envoyée est beaucoup plus petite qu'un simple filtrage de l'arbre basé sur les chemins d'accès apparaissant dans les quantificateurs ; dans la première chaîne, seul les sous-arbres 3-5-5 et 4-6-6 est requis, tandis que dans la deuxième chaîne, seul le sous-arbre 7-8 est envoyé.

Les arbres résultants sont alors fusionnés et le validateur centralisé continue à évaluer la formule CL originale sur cette arborescence fusionnée. Le résultat final est que le validateur centralisé exécute une évaluation normale de la propriété à vérifier, mais sur une copie expurgée de la configuration globale ne contenant que les paramètres dont le résultat global dépend.

Intuitivement, on peut voir que les valeurs de paramètre éliminées localement sur chaque dispositif n'ont aucun impact sur le résultat global, puisqu'elles font en sorte que l'une des conditions de l'expression soit évaluée à faux. Donc, même si de telles valeurs ont des dépendances avec d'autres variables dans d'autres chaînes (et par conséquent probablement dans d'autres dispositifs), elles ne peuvent pas rendre l'expression globale vraie et peuvent par conséquent être ignorées sans aucun risque. Notez que ceci n'est possible seulement que si l'expression originale est composée uniquement de conjonctions de conditions atomiques, une hypothèse que nous pouvons supposer à cause de la transformation en PNF et DNF à l'étape 1.

Comme optimisation supplémentaire, si on suppose que tous les dispositifs ont évalué les conditions intra-chaînes, alors ces conditions peuvent être retirées de l'expression (plus précisément, remplacées par la constante *vrai*) après évaluation au validateur centralisé. Dans un tel cas, seul le sous-ensemble de l'arbre et/ou contenant des variables soumises à des dépendances inter- chaînes doit être envoyé. Dans notre exemple, ceci correspond seulement aux noeuds $a = 3$, $a = 4$ et $b = 7$.

On peut voir le potentiel de réduction de données nécessitant d'être transférées à un emplacement centralisé. Dans notre exemple, l'envoi d'une configuration complète exigerait la transmission de 30 noeuds ; l'application du filtrage d'un sous-arbre simple réduit ce nombre à 21. Le calcul de l'arbre et-ou et l'évaluation des conditions le réduit encore à 8 et l'envoi des valeurs soumises aux dépendances entre chaînes seulement le réduit à 3.

4.4 EXPÉRIMENTATION

Afin d'évaluer la faisabilité de cette approche et mesurer son impact sur la quantité de données de configuration à être transférées, nous avons mis en œuvre l'algorithme décrit dans la partie 4.3.2 et l'avons testé sur des échantillons d'arbres de configuration. Les résultats de ces expérimentations sont détaillés dans cette section.

4.4.1 DÉTAILS DE LA MISE EN OEUVRE

L'algorithme est mis en œuvre en Python et est accessible au public [2]. Le diagramme de classe UML est présenté dans la figure 4.4 et décrit la structure de données utilisée pour représenter les chaînes de dépendance. La composante principale est le Central Validation, qui gère l'ensemble du processus de vérification d'une formule logique pour les dispositifs d'un réseau donné. La formule abstraite à calculer est formatée comme un arbre (LogicFormulaTree) et stockée dans le CentralValidation. Chaque Dispositif reçoit une copie de cette formule.

La méthode valuate() de la formule est utilisée pour estimer les noeuds de l'arbre de manière récursive, en parcourant le dispositif correspondant. Il est important de dire que seules les valeurs strictement nécessaires à la vérification de la formule sont extraites du dispositif. S'il reste des noeuds sans valeur ni enfant, ils sont éliminés de l'arbre de la formule, afin de minimiser les demandes en temps et en espace. Les noeuds essentiels sont ceux dont les noms sont mentionnés dans la formule et particulièrement dans les conditions de cette formule (voir l'exemple complet ci-dessous).

La méthode compute() est utilisée pour évaluer les noeuds selon leur nature : NodeAnd

2. `https://github.com/sylvainhalle/MetaConfig`

(pour tous : chaque enfant doit répondre vrai), NodeOr (existe : un enfant répondant vrai est suffisant), Node (neutre, utilisé pour les feuilles de l'arbre : seule la condition de ce noeud doit répondre vrai) ; les conditions attachées à ces noeuds sont aussi calculées. Les noeuds dont les conditions sont fausses sont supprimés. Par conséquent, ceci est aussi une fonction récursive dont le résultat est un booléen. S'il y a interdépendances entre des noeuds ou entre des dispositifs éloignés, le résultat est un arbre qui contient toutes les valeurs acceptées des termes interdépendants. Dans ce cas, le Central Validation récupère une copie de tous les arbres de formule logique et analyse les interdépendances s'il y en a.

La mise en œuvre d'Alias est utilisée pour assigner une référence à un noeud. La Condition est représentée comme un lien entre 2 AbstractTerms et un opérateur. Ceux-ci peuvent être des AtomicAliasTerm (valeur assignée à un noeud via un Alias), AtomicConstantTerm (une valeur constante) ou une autre condition simple (une structure récursive).

Comme entrée pour notre programme, on a une formule CL et des configurations de dispositifs multiples exprimées sous forme de Meta-CLI. La sortie est une valeur booléenne représentant le résultat de l'évaluation de la contrainte sur des dispositifs multiples.

4.4.2 RÉSULTATS

Nous avons effectué plusieurs tests de l'algorithme tel que décrit ci-dessous. Le dispositif utilisé pour le test est un arbre de configuration fictif avec des valeurs simples, semblables

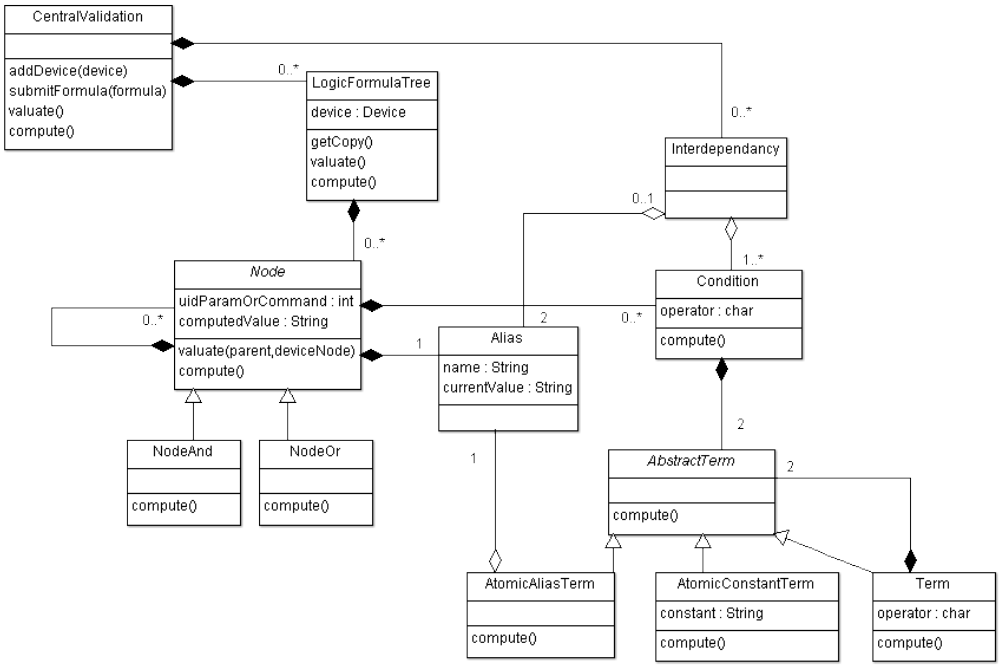

Figure 4.4: Diagramme de classe pour la mise en œuvre d'une évaluation sélective

à celui de la figure 2.13. La première formule CL testée sur cette configuration est :

$$[d = x]\ [d = x\,;\, a = y]\ y \leq 0$$

Le calcul de cette formule doit retourner vrai pour un dispositif donné. Cette formule est stockée sous forme d'arbre en mémoire et envoyée au dispositif. Il contient une référence à une commande d et au paramètre a dans le fichier Meta-CLI d'un dispositif donné. La condition logique $y \leq 0$ est attachée au paramètre a du noeud.

La quantité de données envoyée par le central aux dispositifs correspond à un petit arbre. La réponse du dispositif est calculée localement. Pour vérifier la formule, il évalue seulement un type de commande et un type de paramètre. Dans cet exemple, il compare

seulement les valeurs 0 et -1 d'un paramètre sous la commande d de sa configuration. Il requiert de petites quantités de données par rapport à tous les paramètres contenus dans les divers fichiers de configuration. La taille des données extraites de la configuration pour la validation de cette formule de 3 noeuds sur un dispositif contenant n noeuds est égale à 4 valeurs : 2 pour a et 2 pour d.

Nous avons aussi testé 3 autres formules :

$$[d = x]\,[d = x\,;\, a = y]\,[d = x\,;\, b = z]\; y < z$$

$$\langle d = x\rangle\,\langle d = x\,;\, a = y\rangle\,\langle d = x\,;\, b = z\rangle\; z = 3 \wedge y = 0$$

$$\langle d = x\rangle\,\langle d = x\,;\, a = y\rangle\,[e = w]\,[e = w\,;\, c = z]\; y = -1 \wedge z = 3$$

Nous avons aussi créé plusieurs fichiers de configuration de dispositif tel que détaillé ci-dessous. Des ensembles de fichiers contiennent de 1 à 10 fichiers avec 5 à 500 nœuds générés aléatoirement. Pour chaque formule de CL et chaque configuration, nous avons exécuté l'évaluation de la configuration en utilisant tant l'approche d'évaluation sélective (simplifiée) décrite dans la partie précédente, que l'approche « totale »qui requiert le téléchargement de la configuration complète de chaque dispositif avant le début du calcul. Dans chaque cas, nous avons mesuré la quantité de données requises pour être envoyé au validateur centralisé pour le calcule du résultat.

	C1	C2	C3
Nombre de dispositif	2	5	10
Nombre de nœuds par dispositif	5	100	500

Les résultats que nous avons obtenus sont présentés dans le Tableau 4.3. Dans les deux scénarios, nous avons compté comme une seule unité de bande passante chaque noeud

Approche	Selective				Total			
Formule	1	2	3	4	1	2	3	4
C1	8	12	12	12	10	10	10	10
C2	20	30	30	30	500	500	500	500
C3	40	60	60	60	5000	5000	5000	5000

Tableau 4.3: Echange de données dans les dispositifs

d'arbre transmis par un dispositif au validateur centralisé. Nous pouvons voir que la quantité de données échangées dans l'approche sélective est beaucoup plus faible que dans l'approche totale, parfois la différence s'évalue en facteur de 100. Une exception est la configuration C1, dont la petite taille, combinée avec la surcharge de rassemblement des chaînes de dépendance vers et en provenance des dispositifs, entraîne une consommation de la bande passante légèrement plus élevée pour l'approche sélective. Cependant, on peut voir que l'approche sélective est très efficace aussitôt que le nombre de dispositifs et la taille des configurations atteignent un seuil minimal.

CONCLUSION GÉNÉRALE

De nos jours les réseaux informatiques ont une grande influence sur le niveau et la qualité de vie de la population mondiale et plus précisément sur celle des canadiens. Compte tenu de la géographie du Canada, les réseaux informatiques constituent une sorte d'autoroute, qui devient ainsi une partie essentielle de l'infrastructure nationale, permettant la création de richesses et le maintien de notre niveau de vie. Cette situation ne pourra perdurer que si les entreprises informatiques continuent d'accentuer leurs recherches en développement.

Au terme de nos travaux, nous avons pu démontrer que les formalismes comme la logique de configuration est essentiel pour la validation des configurations réseaux avant leur déploiement. L'intégration de CL dans l'outil de configuration ValidMaker montre qu'une logique suivant la structure hiérarchique familière aux ingénieurs réseaux offre un cadre naturel et efficace pour exprimer et vérifier les propriétés de configuration du réseau. Les résultats expérimentaux montrent que cette approche est facile à aborder en pratique. La fonctionnalité d'exploration du contre-exemple interactif permet aux ingénieurs réseaux d'utiliser efficacement la configuration logique et de résolution des problèmes avant le déploiement sur un réseau. Sur la base des résultats prometteurs

obtenus sur des cas d'utilisation initiaux, des extensions sur l'outil sont en cours et prendront en compte la nature décentralisée des informations de configuration et la validation des arbres de configuration incomplets.

Nous avons aussi pu démontrer, comment un modèle de données générique peut accueillir des informations de configuration pour des dispositifs réseaux sans dépendre des spécifications du fabricant. Le mappage des instances de paramètres concrètes aux noeuds génériques sur une structure arborescente permet de recréer la commande ou la séquence de commandes nécessaire pour extraire la valeur du paramètre sur un dispositif donné. Avec le temps, une telle approche générique devrait réduire le nombre de difficultés rencontrées dans le domaine de l'interopérabilité et dans la gestion des réseaux hétérogènes en réduisant les écarts entre la syntaxe de ligne de commande et la structure des dispositifs divers. Nous avons pu démontrer que très peu de solutions offrent des capacités de vérification de configurations selon des règles définies par l'utilisateur et que ceux qui le font supposent un accès arbitraire à une copie locale de la configuration complète du réseau. Ceci, à son tour, provoque une consommation élevée de la bande passante, puisque la configuration du réseau entier doit être déposée à un emplacement central chaque fois qu'un contrôle de justesse (exactitude) doit être effectué. Nous avons donc présenté une solution stratégique de traitement basée sur le concept d'évaluation sélective, qui essaye de récupérer le moins d'informations possible de chaque dispositif pour un traitement centralisé. Cette stratégie est basée sur le fait que les contraintes de configuration peuvent être exprimées formellement dans un langage appelé Configuration Logic et que les expressions de CL peuvent être analysées pour concevoir des règles de filtrage à appliquer sur chaque dispositif. Les premiers résultats empiriques ont montré que cette technique offre la possibilité de réduire considérablement la quantité de données à récupérer de chaque dispositif, tout en préservant toujours les mêmes garanties sur la

justesse (exactitude) de la configuration.

Notre travail se prête à plusieurs extensions et optimisations. Par exemple, une plus grande élimination des arbres logiques et/ou produits par chaque dispositif pourrait être calculée localement en croisant plusieurs arbres et/ou qui partagent des variables quantifiées, aboutissant à une économie accrue de la bande passante. L'identification des paramètres de configuration qui sont pertinents pour le résultat Booléen d'une règle pourrait aussi être utilisée pour créer des alarmes qui seraient déclenchées à chaque fois qu'un paramètre présent dans un arbre et/ou pré-calculé changerait de valeur. L'interrogation des valeurs de configuration des différents périphériques revient simplement à indiquer ou à pointer le paramètre approprié dans cette structure.

Somme toute, cette thèse a constitué une partie importante dans la réalisation d'un outil de gestion automatique des configurations réseau pouvant être utilisé en entreprise.

ANNEXES

4.4.3 ANNEXE1 : PRÉSENTATION DES PROPRIÉTÉS DE QUELQUES OUTILS

Propriétés de spécification

	Cfengine3	Pantin	BMC BladeLogic Server Automation Suite	Chef	Générateur Config Netomata	Microsoft Server Center Configuration Manager	CA Réseau et Systèmes de Management	Bcfg2	LCFG	HP Server Automation	IBM Tivoli System Automation for Multiplatforms
Spécification paradigme											
Type de langage	Déclaratif	Déclaratif, impératif	Impératif	Impératif	Déclaratif, impératif	Impératif	Déclaratif, impératif	Déclaratif	Déclaratif	Impératif	Déclaratif
Type d'interface utilisateur	Ligne de commande	Interface graphique et ligne de commande	Interface graphique et ligne de commande	Interface graphique et ligne de commande	Ligne de commande	Interface graphique	Interface graphique	Ligne de commande	Ligne de commande	Interface graphique	Interface graphique et ligne de commande
Mécanismes d'abstraction utilisés	- les fichiers de configuration, - instances dépendant de l'implémentation, - configurations d'instance	- les fichiers de configuration, - instances dépendant de l'implémentation	- les fichiers de configuration, - instances dépendant de l'implémentation	Instances dépendant de l'implémentation	Les fichiers de configuration	Les fichiers de configuration	- les fichiers de configuration, - instances dépendant de l'implémentation, - configurations d'instance	Les fichiers de configuration	Instances dépendant de l'implémentation	Les fichiers de configuration	Instances dépendant de l'implémentation
Mécanismes de modularisation											
Type de regroupement	-Regroupement statique, - requête basée sur les groupes, - groupes hiérarchiques	Regroupement statique, - requête basée sur les groupes, - groupes hiérarchiques	-Regroupement statique, - requête basée sur les groupes, - groupes hiérarchiques	Regroupement statique, - requête basée sur les groupes	Regroupement statique	Regroupement statique, - requête basée sur les groupes,	Les groupes hiérarchiques	Regroupement statique, - groupes hiérarchiques	Regroupement statique, - requête basée sur les groupes,	Regroupement statique	Regroupement statique
Modules de configuration	oui	oui	oui	oui	aucun	aucun	oui	aucun	oui	aucun	oui
Modélisation des relations											
Arité	n: n; 1:n; 1:1	1 :n; 1 :1	1 :1	n :n	1 :1				n: n; 1:1		n: n; 1:n; 1:1
Contraintes	contraintes génératives										contraintes génératives
Granularité	Relation d'instance, de paramètres – instance, relation entre paramètres	Relations d'instance	Relations d'instance	Relations d'instance	Relations d'instance				Relations d'instance, relations entre paramètres		Relation d'instance, de paramètres – instance, relation entre paramètres

Propriétés de déploiement

Propriété	Cfengine3	Pantin	BMC BladeLogic Server Automation Suite	Chef	Générateur Config Netomata	Microsoft Server Center Configuration Manager	CA Réseau et Systèmes de Management	Bcfg2	LCFG	HP Server Automation	IBM Tivoli System Automation for Multiplatforms
Évolutivité	> 10000	1000 à 10000	> 10000	inconnu	inconnu	inconnu	inconnu	<1000	1000 à 10000	inconnu	inconnu
Workflow	Coordination des changements distribués	Coordination des changements distribués	Coordination des changements distribués			Soutien aux politiques (règles) organisationnelles	Soutien aux politiques (règles) organisationnelles			Coordination des changements distribués	Coordination des changements distribués
Architecture de déploiement											
Agent de traduction	Gestion fortement distribués	Gestion centralisée	Gestion faiblement distribués	Gestion centralisée	Gestion centralisée	Gestion faiblement distribuées	Gestion faiblement distribuées	Gestion centralisée	Gestion centralisée	Gestion centralisée	Gestion centralisée
Mécanisme de distribution	Pull, Push	Pull, Push	push	Pull, Push	aucun	Pull, Push	push	pull	pull	Push	push
Plate-forme supportée	BSD, AIX, HP-UX, Linux, Mac OS X, Solaris, Windows	BSD, AIX, HP-UX, Linux, Mac OS X, Solaris	AIX, HP-UX, Linux, équipements réseaux, Solaris, Windows	BSD, Linux, Mac OS X, Solaris, Windows	Equipements réseaux	Windows	AIX, HP-UX, Linux, Mac OS X, équipements réseaux, Solaris, Windows	BSD, AIX, Linux, Mac OS X, Solaris	Linux	AIX, HP-UX, Linux, Équipements réseaux, Solaris, Windows	AIX, Linux, Solaris, Windows

Spécification des propriétés de gestion

Propriété	Cfengine3	Pantin	BMC BladeLogic Server Automation Suite	Chef	Générateur Config Netomata	Microsoft Server Center Configuration Manager	CA Réseau et Systèmes de Management	Bcfg2	LCFG	HP Server Automation	IBM Tivoli System Automation for Multiplatforms
Convivialité ou Facilité d'utilisation											
Facilité d'utilisation	moyen	moyen	facile	Pas facile	Pas facile	facile	facile	Pas facile	moyen	facile	facile
Prise en charge des tests de spécifications	mode de simulation	mode de simulation, d'intégration et de test d'essai		Intégre les essaie de test				mode de simulation	mode de simulation		
Suivi de l'infrastructure	surveillance intégrée	interaction avec la surveillance externe	surveillance intégrée		interaction avec la surveillance externe	surveillance intégrée	surveillance intégrée	surveillance intégrée	surveillance intégrée, l'interaction avec la surveillance externe	surveillance intégrée	surveillance intégrée
Gestion des versions	référentiel externe	référentiel externe	référentiel externe	référentiel externe, référentiel intégré	référentiel externe	référentiel intégré	référentiel intégré	référentiel externe	référentiel externe		référentiel externe
Documentation des spécifications	documentation structurée	la documentation est structurée et à usage libre	documentation à usage libre	la documentation est structurée et à usage libre	documentation à usage libre			documentation à usage libre	documentation à usage libre	d documentation à usage libre	documentation à usage libre
L'intégration avec l'environnement	caractéristiques d'exécution	bases de données externes et les caractéristiques d'exécution	bases de données externes	bases de données externes, les caractéristiques d'exécution		bases de données externes, les caractéristiques d'exécution	caractéristiques d'exécution	caractéristiques d'exécution	caractéristiques d'exécution		caractéristiques d'exécution
Contrôle d'accès	basic	basic	Hiérarchique	basic		Hiérarchique	Hiérarchique	basic	basic	Hiérarchique	Hiérarchique

Soutenir

Propriété	Cfengine3	Pantin	BMC BladeLogic Server Automation Suite	Chef	Générateur Config Netomata	Microsoft Server Center Configuration Manager	CA Réseau et Systèmes de Management	Bcfg2	LCFG	HP Server Automation	IBM Tivoli System Automation for Multiplatforms
documentation disponible	Documentation des processus, -livre de référence, -Tutoriel	Documentation des processus, -livre de référence, -Tutoriel	Documentation des processus, -livre de référence, -Tutoriel	Référence et Tutoriel	Référence et Tutoriel	Documentation des processus, -livre de référence, -Tutoriel		Documentation des processus, -livre de référence, -Tutoriel	Documentation des processus, -livre de référence, -Tutoriel		Documentation des processus, -livre de référence, -Tutoriel
Support commercial	Formation, le développement soutenu par une entreprise, il y a aussi le support en ligne	Formation, le développement soutenu par une entreprise, il y a aussi le support en ligne	Formation, le développement soutenu par une entreprise, il y a aussi le support en ligne	Formation, le développement soutenu par une entreprise.	le développement soutenu par une entreprise	Formation, le développement soutenu par une entreprise, il y a aussi le support en ligne	Formation, le développement soutenu par une entreprise, il y a aussi le support en ligne			Formation, le développement soutenu par une entreprise, il y a aussi le support en ligne	Formation, le développement soutenu par une entreprise, il y a aussi le support en ligne
Communauté	5	5	1	4	1	4	3	2	1		5
Maturité	5	3	5	2	1	3	5	4	5	5	4

4.4.4 ANNEXE2 : EXEMPLE DE FICHIER DE CONFIGURATION DE L'OUTIL CATTOOLS

```
Current configuration:
!
version 12.0
no service pad
service timestamps debug uptime
service timestamps log datetime localtime show-timezone
service password-encryption
!
hostname SS-C29-1
!
aaa new-model
aaa authentication login default group tacacs+ line none
aaa authentication enable default group tacacs+ enable none
aaa accounting exec default start-stop group tacacs+
aaa accounting commands 15 default start-stop group tacacs+
aaa accounting system default start-stop group tacacs+
enable password 7 xxx

clock timezone EST -5
clock summer-time EDT recurring 2 Sun Mar 2:00 1 Sun Nov 2:00
spanning-tree uplinkfast
!
ip subnet-zero
no ip domain-lookup
!
interface FastEthernet0/1
 description Usager
 no logging event link-status
 duplex full
 speed 10
 no snmp trap link-status
 spanning-tree portfast
!
interface FastEthernet0/2
 description Usager
 no logging event link-status
 no snmp trap link-status
 spanning-tree portfast
!
interface FastEthernet0/3
 description Usager
 no logging event link-status
 duplex full
 speed 10
 no snmp trap link-status
 spanning-tree portfast
interface FastEthernet0/4
 description Usager
 no logging event link-status
 duplex full
 speed 10
```

```
 no snmp trap link-status
 spanning-tree portfast
!
interface FastEthernet0/5
 description Usager
 no logging event link-status
 duplex full
 speed 10
 no snmp trap link-status
 spanning-tree portfast
!
interface FastEthernet0/6
 description Usager
 no logging event link-status
 duplex full
 speed 10
 no snmp trap link-status
 spanning-tree portfast
!
interface FastEthernet0/7
 description Usager
 no logging event link-status
 duplex full
 speed 10
 no snmp trap link-status
 spanning-tree portfast
!
interface FastEthernet0/8
 description Usager
 no logging event link-status
 duplex full
 speed 10
 no snmp trap link-status
 spanning-tree portfast
!
interface FastEthernet0/9
 description Usager
 no logging event link-status
 duplex full
 speed 10
 no snmp trap link-status
---
---
---
---
---
---
```

BIBLIOGRAPHIE

2000. DSP0111 Common information model (CIM) core model, version 2.4.

2operate. 2012. http://www.2operate.com/index.php?mact=News,cntnt01,detail,0&cntnt01articleid=27&cntnt01returnid=59.

Aitken, P., B. Claise, et G. Muenz. 2012. « Configuration data model for the IP flow information (RFC 6728 ».

Anderson, P. 2008. *LCFG : A Practical Tool for System Configuration*. Usenix Association.

Anderson, P. et A. Scobie. 2002. « Lcfg : The next generation », p. 321–333.

———. 2012. « Puppet documentation », *Puppet Labs documentation*.

Angelo Rossi. 2011. « CoÛt de la non-sÉcuritÉ informatique », *ponemon.org/library*.

Antoniou, G. et F. van Harmelen. 2003. Web on tology language :owl. http://www.cs.vu.nl/~frankh/postscript/OntoHandbook03OWL.pdf.

AOL Inc. 2012. http://trigger.readthedocs.org/en/latest/.

Arundel, J. 2010. Puppet tutorial for Linux : Powering up with Puppet. http://bitfieldconsulting.com/puppet-tutorial.

Bejtlich, R. 2005. http://www.amazon.ca/The-Visible-Ops-Handbook-Implementing/dp/0975568612.

Benedikt, M. et G. Bruns. 2004. « On guard : Producing run-time checks from integrity constraints ». In *AMAST*, p. 27–41.

Berard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, et P. Schnoebelen. 2010. *Systems and Software Verification : Model-Checking Techniques and Tools*. Springer Publishing Company, Incorporated, 1st édition.

Bierman, A. 2008. Welcome to netconf central. http://www.netconfcentral.org.

Bjorklund, M. 2010. « Rfc 6020 : Yang - a data modeling language for the network configuration protocol (netconf) », *Internet Engineering Task Force (IETF)*.

BMC Software. 2012. http://www.bmc.com/products/bladelogic-server-automation/server-management.html.

Boag, S., D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, et J. Siméon. 2005. XQuery 1.0 : An XML query language, W3C working draft.

Bono, V. J. 1997. « 7007 explanation and apology », *North American Network Operators Group*.

Bray, T., D. Hollander, A. Layman, et R. Tobin. 2001. Namespaces in XML 1.0, W3C recommendation.

Buchmann, D. 2008. « Verified network configuration », *Universität Freiburg (Schweiz), thesis*.

Burgess, M. 1995. « A site configuration engine », *Usenix*, p. 309–337.

———. 1998. « Computer immunology », *LISA*, p. 88–96.

———. 2002. « Two dimensional time-series for anomaly detection and regulation in adaptive systems ». In *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems : Operations and Management : Management Technologies for E-Commerce and E-Business Applications*. Coll. « DSOM '02 », p. 169–180, London, UK, UK. Springer-Verlag.

Bush, R. et T. Griffin. 2003. « Integrity for virtual private routed networks », *INFOCOM*, p. 293–298.

Campi, N. et K. Bauer. 2009. *Introducing the Basics of Automation*. Apress.

Caprica Ltd. 2009. Configchecker. `http://www.configchecker.com/`.

Castillo, R. 2006. « Ziptie network inventory framework : Enabling the next era of network management tools », *ACM SIGCOMM*, p. 10.

CatTools, S. 2012. Cattools help. `http://www.kiwisyslog.com/help/cattools/mnu_filedbimportdevicefrmtab.htm`.

Chawla, S. 2012. Centralized management with puppet on centos 5. `http://www.mylinuxtips.info/centralized-management-with-puppet-on-centos-5/linuxtipstutorials/224/`.

Chen, X., Y. Mao, Z. M. Mao, et J. V. der Merwe. 2010a. « Declarative configuration management for complex and dynamic networks », p. 6 :1–6 :12.

Chen, X., Y. Mao, Z. Morley, et J. V. der Merwe. 2010b. « Declarative configuration management for complex and dynamic networks ». In *Proceedings of the 6th Inter-*

national COnference. Coll. « Co-NEXT '10 », p. 6 :1–6 :12, New York, NY, USA. ACM.

Chuche, N. 2007. « cfengine - un outil pour l'administrateur système », *Linux Magazine 95*.

Cisco. 2013a. Command line interface.

———. 2013b. Configuring VTP. `http://www.cisco.com/en/US/products/hw/switches/ps708/products_configuration_guide_chapter09186a008019f048.html`.

Clarke, E. M., O. Grumberg, et D. A. Peled. 2000. *Model checking*. The MIT Press ; 1 edition (Dec 20 1999).

Clarke, J. 2012. « Gestion de configuration : Une best practice reconnue mais difficile à mettre en oeuvre », *Normation*. `http://www.solutionslinux.fr/animation_87_168_2466_p.html?cid=1382`.

Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, et C. Talcott. 2007. *All About Maude - A High-Performance Logical Framework*. Springer Berlin Heidelberg. `http://link.springer.com/chapter/10.1007/978-3-540-71999-1_13`.

Cleary, D., B. Danev, et D. O. Donoghue. 2005. « Using ontologies to simplify wireless network configuration ». In *In : Proceedings of formal ontologies meet Industry*. Department of Computer Science, NUI Maynooth, Ireland.

Colville, R. J. et G. Spafford. 2010a. « Configuration management for virtual and cloud infrastructures », *Gartner*. `http://www.rbiassets.com/getfile.ashx/42112626510`.

———. 2010b. « Top seven considerations for configuration management for virtual and cloud infrastructures », *Gartner*. `http://www.gartner.com/id=1458131`.

Couch, A. et Y. Sun. 2003. *On the Algebraic Structure of Convergence*. Coll. Brunner, M. et A. Keller, éditeurs, Coll. « *Self-Managing Distributed Systems* ». T. 2867, série *Lecture Notes in Computer Science*, p. 28–40. Springer Berlin Heidelberg.

Couch, A. L. 2000. « An expectant chat about script maturity », *LISA Conference*, p. 15–28. `https://www.usenix.org/legacy/events/lisa2000/full_papers/couch/couch_html/index.html`.

Couch, A. L. 2012. The Babble scripting tool. `http://www.cs.tufts.edu/~couch/babble/`.

Couch, A. L. et M. Gilfix. 1999. « It's elementary, dear Watson : Applying logic programming to convergent system management processes », *Lisa 1999*, p. 123–138.

Coulon, A. 2001. « http ://www.adeli.org/webfm_send/182 », *ADELI*.

Cridlig, V., H. Abdelnur, J. Bourdellon, et R. State. 2005. « A NetConf network management suite : ENSUITE ». In Thomas Magedanz,Edmundo R. M. Madeira, Petre Dini, éditeur, *5th IEEE International Workshop on IP Operations and Management, IPOM 2005, Barcelona, Spain, October 26-28, 2005. Proceedings*, p. 152–161.

Deca, R., O. Cherkaoui, Y. Savaria, et D. Slone. 2007. « Constraint-based model for network service provisioning », *Annals of telecommunications*, p. 847–870.

Delaet, T. et W. Joosen. 2007. « Podim a language for high-level configuration management », *Conference LISA 2007*, p. 261–273. `http://static.usenix.org/events/lisa07/tech/full_papers/delaet/delaet_html/`.

Delaet, T., W. Joosen, et B. Vanbrabant. 2011. « A survey of system configuration tools », *Usenix Association*, p. 14. `http://static.usenix.org/event/lisa10/tech/full_papers/Delaet.pdf`.

Delaet, T. et B. Vanbrabant. 1999. Sysconfigtools. http://distrinet.cs.kuleuven.be/software/sysconfigtools/tool.

Dennis, G. et R. Seater. 2008. Tutorial for alloy analyzer 4.0. http://alloy.mit.edu/alloy/tutorials/online/frame-FS-1.html.

———. 2012. Alloy analyzer 4 tutorial session 2 : Language and analysis. http://alloy.mit.edu/alloy/tutorials/day-course/s2_language.pdf.

DeRose, S., E. Maler, et D. Orchard. 2001. XML linking language (XLink) version 1.0, W3C recommendation.

Desai, N., R. Bradshaw, et C. Lueninghoener. 2006. « Directing change using bcfg2 », *Lisa 2006*, p. 215–220.

Dyson, D. 2012. Configgen - .net configuration file generator. http ://configgen.codeplex.com/.

EarthSoft. 2012. Edge v5.5 - configuration - configuration plugin tool. http://help.earthsoft.com/default.asp?W2220.

Enns, R., M. Bjorklund, J. Schœnwælder, et A. Bierman. 2006. « Network working group », *RFC 4741*. http://www.rfc-editor.org/rfc/rfc4741.txt.

———. 2011a. Netconf configuration protocol, IETF RFC 6241.

Enns, R., M. Bjorklund, J. Schoenwaelder, et A. Bierman. 2011b. « Network configuration protocol (netconf) (RFC 6241) ».

Enns, R. et Ed. 2006. « Netconf configuration protocol », *RFC 4741*.

Feamster, N. et H. Balakrishnan. 2005. « Detecting bgp configuration faults with static analysis ». In *Proceedings of the 2Nd Conference on Symposium on Networked Systems*

Design & Implementation - Volume 2. Coll. « NSDI'05 », p. 43–56, Berkeley, CA, USA. Usenix Association.

Fedor, M., M. L. Schoffstall, et J. Davin. 1990. « An architecture for describing SNMP management frameworks (RFC 1157) ».

Friedman, D. P. et D. S. Wise. 1976. « Cons should not evaluate its arguments ». In *ICALP*, p. 257–284.

Gartner. 2012. « Data-center-automation », p. 14.

Gruber, T. 2009. « Ontology », *Encyclopedia of Database Systems*, p. 1963–1965.

Gruber, T. R. 1993. « A translation approach to portable ontology specifications », *Stanford University*, p. 199–220.

Hallé, S., O. Cherkaoui, et P. Valtchev. 2012. « Towards a semantic virtualization of configurations ». In *NOMS*, p. 1268–1271.

Halle, S., R. Deca, , O. Cherkaoui, R. Villemaire, et D. Puche. 2004. « A formal validation model for the netconf protocol », *Conference DSOM 2004*, p. 147–158.

Hallé, S., R. Deca, O. Cherkaoui, et R. Villemaire. 2004. « Automated validation of service configuration on network devices », *7th IFIP/IEEE International Conference, MMNS 2004*, p. 176–188.

Hallé, S., E. L. Ngoupé, G. Nijdam, O. Cherkaoui, P. Valtchev, et R. Villemaire. 2012. « Validmaker : A tool for managing device configurations using logical constraints », *Network Operations and Management Symposium (NOMS) 2012 IEEE*, p. 1111–1118.

Hallé, S., R. Villemaire, et O. Cherkaoui. 2005. « Configuration logic : A multi-site modal logic », *12th International Symposium on Temporal Representation and Reasoning (TIME 05)*, p. 131–137.

Handbook, V. O. 2011. Misconfigurations Have Major Impact on Performance.

Hansen, C. V. et J. Forcier. 2013. Fabric 1.6.2 documentation. `http://docs.fabfile.org/en/1.6/`.

Harrington, D., B. Wijnen, et R. Presuhn. 1999. « An architecture for describing SNMP management frameworks », *RFC 2571*.

Henderson, P., J. H, et M. Jr. 1976. « A lazy evaluator ». In *Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles on Programming Languages*, p. 95–103. ACM.

HP France. 2013. Server automation. `http://www8.hp.com/fr/fr/software-solutions/software.html?compURI=1172939#.UWtZLKJUHE0`.

Hui, X. et D. X. Wuhan. 2006. « A common ontology-based intelligent configuration management model for ip network devices ». DBLP, http ://dblp.uni-trier.de.

IEEE. 2003. 802.11Q : Virtual bridged local area networks standard. http ://standards.ieee.org/getieee802/download/802.1Q-2003.pdf.

IIander, T., H. Toivonen, et P. Smolander. 2012. Snitch-data management for in-field measurements. `http://www.stuk.fi/julkaisut_maaraykset/en_GB/stuk-ttl-flyers/_files/88555479804019685/default/Flyer_007_SNITCH.pdf`.

Interpeak. 2005. Simple Network Management Protocol.

Ipswitch. 2013. http://www.whatsupgold.com/fr/products/whatsup-gold-plugins/whatsconfigured/index.aspx.

Jackson, D. 2002. « Alloy : A lightweight object modelling notation », *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 2, p. 256–290. http ://doi.acm.org/10.1145/505145.505149.

———. 2012. Alloy : a language and tool for relational models. http://alloy.mit.edu/alloy/faq.html.

Jboss. 2013. Jboss application serveur 7. https://community.jboss.org/wiki/CommandLineInterface?_sscc=t.

Jensen, F. V., U. Kjærulff, B. Kristiansen, H. Langseth, C. Skaanning, J. Vomlel, et M. Vomlelová. 2001. « The SACSO methodology for troubleshooting complex systems », *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 15, no. 4, p. 321–333. http://dx.doi.org/10.1017/S0890060401154065.

Journal du Net. 2012. « La panne de gmail due à une "mauvaise configuration" », *Journal du net*.

Juniper Networks. 2008. « What's behind network downtime ? », *White Paper*, p. 3–10.

Kembel, R. W. 2009. *Fibre Channel a Comprehensive Introduction*. Coll. « The fibre channel consultant series ». Northwest Learning Associates, Incorporated. http://books.google.ca/books?id=T1s6QwAACAAJ.

Klarlund, N., J. Koistinen, et M. I. Schwartzbach. 1997. « Formal design constraints », p. 370–383.

Lalitte, E. 2003. « Le routage », *site web de framip*.

Langseth, H. et F. V. Jensen. 2003. « Decision theoretic troubleshooting of coherent systems », *Reliability Engineering and System Safety*, vol. 80, p. 49–62.

Lapalme, G. 1999. `http://www.iro.umontreal.ca/~lapalme/ift6281/OWL/EtapesCreationOntologie.html`.

Le, F., S. Lee, T. Wong, H. S. Kim, et D. Newcomb. 2006. « Minerals : Using data mining to detect router misconfigurations », *ACM SIGCOMM*, no. CMU-CyLab-06-008, p. 293–298.

MacFaden, M. R., D. Partain, J. Saperia, et W. F. Tackabury. 2003. Configuring networks and devices with simple network management protocol (SNMP). (RFC 3512).

ManageEngine. 2012. « Deviceexpert - logiciel de gestion des configurations réseau », *site internet de ManageEngine*.

Mendelson, E. 1997. *Introduction to Mathematical Logic, Fourth Edition*. Springer.

Metz, C. 2011. The chef, the puppet, and the sexy it admin. `http://www.wired.com/wiredenterprise/2011/10/chef_and_puppet/`.

Mi-lung, C., C. Hyoun-Mi, H. J. Won-Ki, et J. Hong-Taek. 2004. « Xml-based configuration management for ip network devices », *Communications Magazine, IEEE*, p. 84–91.

Mickens, J., M. Szummer, et D. Narayanan. 2007. « Snitch : Interactive decision trees for troubleshooting misconfigurations », *SysML*, p. 1–6.

Microsoft. 2012. Microsoft cloud et serveurs d'entreprise. `http://www.microsoft.com/france/serveur-cloud/system-center/default.aspx`.

———. 2013. « Types d'erreurs (visual basic) », *Visual Studio 2012*.

Narain, S. 2005. « Network configuration management via model finding », *LISA 2005*, p. 155–168.

Narain, S., G. Levin, S. Malik, et V. Kaul. 2008. « Declarative infrastructure configuration synthesis and debugging », *J. Network Syst. Manage.*, vol. 16, no. 3, p. 235–258.

netomata. 2010. Netomata config generator (ncg). `http://www.netomata.com/tools/ncg`.

Network, E. et I. S. Agency. 2012. « Annual incident reports 2011 », *ENSIA Annual Incident Reports 2011*.

Networks, S. 2006. « RANCID - really awesome new cisco config differ ».

Objective Development Software. 2013. `http://www.obdev.at/products/littlesnitch/index.html`.

officiel de CFEngine, S. 2012. What is cfengine ? http ://cfengine.com/.

Opscode. 2008. Opscode chef gives your it infrastructure the speed, flexibility and efficiency you need to compete in the digital economy. http ://www.opscode.com/chef/.

———. 2013. `http://docs.opscode.com/chef_overview.html`.

Opscode. 2013. `https://wiki.opscode.com/plugins/viewsource/viewpagesrc.action?pageId=13173077`.

Osterlund, R. 2014. PIKT. http ://www.pikt.org/.

Parisot, C. 2014. « Optimisation de la récupération des configurations réseau ». Mémoire de maîtrise, Université de Lorraine and UQAC.

PartnerWorld, I. 2012. Ca nsm (ca network and systems management). https://www-304.ibm.com/partnerworld/gsd/solutiondetails.do?&solution=38673&lc=en.

Patterson, D. A. 2002. « A simple way to estimate the cost of downtime », *In Sixteenth Systems Administration Conference (LISA 02), Berkeley, CA, USA. Usenix*, p. 185–188.

Pepitone, J. 2011. Amazon EC2 outage downs Reddit, Quora.

Peret, N., P. Sidler, et F. X. Marseille. 2005. Administration des réseaux Principes et modèle ISO-SNMP.

Perron, N. 2012. Rudder - it automation made easy ! http://www.rudder-project.org/foswiki/.

Pignet, F. 2007. « Réseaux informatiques : Supervision et administration », *Collection Expert*, p. 177–179.

Presuhn, R., S. Waldbusser, M. Rose, K. McCloghrie, et J. Case. 2002. « Management information base (mib) for the simple network management protocol (snmp) », *RFC 3418*.

Raum. 2005. http://linbox.free.fr/passerelle_simple/chapitres.php?chapitre=9.

Red_Hat. 2013. https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s1-networkscripts-interfaces.html.

Rexford, J. et A. Feldmann. 2001. « Ip network configuration for intradomain traffic engineering », *IEEE Network*, p. 46–47.

Rosen, E. C. et Y. Rekhter. 1999. « BGP/MPLS VPNs (RFC 2547) », *RFC 2547*, p. 25.

Rosenberg, J. 2007. The extensible markup language (XML) configuration access protocol (XCAP), RFC 4825.

SAP. 2013. « Réseaux logiques et contraintes ». In *of Lecture Notes in Computer Science*, p. 1. sap.

Schaefer, M. 2003. « Courte information sur le systeme unix et les reseaux teléinformatiques », p. 69.

Schoenwaelder, J. 2003. « Overview of the 2002 iab network management workshop (RFC 3535) », p. –.

Schrieck, V. V. D. 1998. « Conception d'un langage flexible de définition de politiques de routage bgp », *UNIVERSITE CATHOLIQUE DE LOUVAIN*.

Schönwälder, J., V. Marinov, et M. Burgess. 2008. « Integrating Cfengine and SCLI managing network devices like host systems », *Salvador Bahia Conference*, p. 1067–1070.

Shafer, P. 2011. « An architecture for network management using netconf and yang (RFC 6244) ».

Software, E. 2012. `http://trac.mcs.anl.gov/projects/bcfg2/`.

sparksupport. 2010. `http://www.sparksupport.com/blog/puppet-configuration-management-tool`.

Stallings, W. 1999. « SNMP, SNMPv2, SNMPv3, and RMON 1 and 2 ».

Strassner, J. 1999. *Directory Enabled Networks*. New Riders Publishing.

———. 2002. Bridge to ip profitability. http://www.intelliden.com/library/GlobalOSS_BridgetoIP45.pdf.

Thibodeau, P. 2011. Amazon cloud outage was triggered by configuration error.

Trevino, H. et s Chisholm. 2008. « Netconf event notifications (RFC 5277) ».

Vergara, J. E. L. D., V. A. Villagrá, et J. Berrocal. 2002. « 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*, p. 11–13.

Voyence. 2014. « "voyence" ».

Waldbusser, S., K. McCloghrie, J. Case, et R. Presuhn. 2002a. « Management information base (mib) for the simple network management protocol (snmp) », *RFC 3418*.

Waldbusser, S., M. Rose, D. Perkins, J. Case, et K. McCloghrie. 2002b. « Management information base (mib) for the simple network management protocol (snmp) », *RFC 2578*.

Wallin, S. et C. Wikström. 2011. « Automating network and service configuration using NETCONF and YANG », p. 22–22. http://static.usenix.org/event/lisa11/tech/full_papers/Wallin.pdf.

Wijnen, B., R. Presuhn, et D. H. R. 2571). 1999. « Architecture for SNMP frameworks ». http://www.ietf.org/rfc/rfc2571.

Wijnen, B., S. Routhier, D. Levi, et R. Frye. 2000. « Coexistence between version 1, version 2, and version 3 of the internet-standard network management framework (RFC 2576) ».

Willm, O. 2005. Administration de réseaux informatiques : protocole SNMP. http://www.techniques-ingenieur.fr/base-documentaire/technologies-de-l-information-th9/architecture-des-systemes-et-reseaux-42303210/administration-de-reseaux-informatiques-protocole-snmp-h2840/.

Wool, A. 2004. « A quantitative study of firewall configuration errors », *IEEE Computer*, p. 62–67.

Zamboni, D. 2012. « Automated system administration for sites of any size. learning cfengine 3 », *O'Reilly Media*, p. 194.

Zeller, A. et G. Snelting. 1997. « Unified versioning through feature logic », *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 4, p. 398–441.

Zuccarelli, L. et M. L. Laouenan. 2010. « Protocole d'administration netconf », *Tecom Lille*.

Éric Lunaud Ngoupé, S. Hallé, , O. Cherkaoui, S. Stoesel, C. Parisot, P. Valtchev, et P. Boucher. 2014. « A lazy evaluation strategy for assessing network device configuration correctness », *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, p. 190–193. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6923136&punumber%3D6921524%26filter%3DAND%28p_IS_Number%3A6923102%29%26pageNumber%3D2.