Sequential Dependencies in Configuration Operations

Sylvain Hallé, Rudy Deca, Omar Cherkaoui, Roger Villemaire

Université du Québec à Montréal
C.P. 8888, Succ. Centre-ville
Montréal (Québec) CANADA H3C 3P8
halle@info.uqam.ca

Daniel Puche

Cisco Systems inc.
dpuche@cisco.com

RÉSUMÉ *Le déploiement d'un service réseau est sujet à plusieurs dépendances sémantiques et séquentielles. Cependant, un des principaux problèmes des approches de gestion des configurations est l'absence d'un modèle transactionnel qui permettrait aux informations de configuration de conserver leur intégrité durant le processus de configuration. Dans cet article, nous introduisons la notion de dépendance séquentielle et proposons un modèle mathématique basé sur les techniques du model checking permettant de structurer les opérations de configuration. Ce modèle mène au concept « d'état-borne » (milestone state). Nous suggérons par la suite une manière de bonifier le protocole Netconf avec une composante transactionnelle basée sur ces concepts.*

ABSTRACT. *The deployment of a network service is subject to a number of semantical and sequential dependencies. However, one of the main issues with the existing configuration management approaches is the absence of a transactional model, which should allow the network configuration data to retain their integrity during the configuration process. In this paper, we introduce the notion of sequential dependency, propose a mathematical framework based on model checking that allows the structuring of configuration operations leading to the concept of milestone state, and suggest how the Netconf protocol can be enhanced with a transactional component.*

MOTS-CLÉS: *gestion des configurations, Netconf, dépendances sémantiques, dépendances séquentielles, model checking, LTL*

KEYWORDS: *configuration management, Netconf, semantical dependencies, sequential dependencies, model checking, LTL*

1. Introduction

The deployment and configuration of network services is a complex and error-prone task that is subject to constraints at different levels. For instance, *semantical* dependencies between parameters dispersed among multiple configuration operations appear in even the simplest management tasks (Hallé *et al.*, 2004a). Although these dependencies are not currently captured by management protocols such as Netconf (Enns, 2005), it has been shown how tree logics can help in automating their formalising and checking on a given configuration snapshot (Hallé *et al.*, 2004b).

However, even if semantical dependencies can be automatically verified, an important part of the complexity of deploying a service still remains. In general, the configuration operations must be performed in a specific order that is determined by the connected nature of the network, or even by some requirements of the devices' operating system. Therefore, in addition to semantical dependencies, there are *sequential* dependencies that need to be formalised and checked in a smiliar fashion. The importance of checking these sequential dependencies is heightened by the fact that an increasing number of services, such as Virtual LANs and Virtual Private Networks, involve configuration changes on multiple devices at the same time.

While the semantical dependences in network device configurations have been widely debated in the network management literature (Daminaou *et al.*, 2001; Warmer *et al.*, 1999; Crubézy, 2002; Jackson *et al.*, 2000; Hallé *et al.*, 2004b), the sequential dependences have not yet been extensively covered. Among the works on the subject, (Couch *et al.*, 2003) examine a convergent approach to automated configuration and provide an algebraic model of configuration management. According to this model, the managed processes can be decomposed into regions or intents of non-conflicting, stateless actions. Each of these non-commutative regions can then be processed separately. Using this model, procedural processes, which are composed of non-commutative operations, can be redesigned as declarative processes, which are composed of commutative operations. The authors illustrate their approach with examples from file editing.

The transactional aspect of the device configuration process is taken into account by the Netconf configuration protocol (Enns, 2005), which is a new protocol designed for manipulating network device configurations. However, this protocol provides transactional operations at device level, but does not currently have similar operations for the network level.

The purpose of this paper is twofold. First, we raise the question of sequential constraints in network configuration operations and show by a couple of examples that their presence is as common as semantical ones; using concepts borrowed from the field of model checking, we also demonstrate how these constraints can be accurately formalised in Kripke structures by temporal logic formulas. Second, we

define the notion of *milestone* states in a Kripke structure in terms of these mathematical grounds and claim that these states make good candidates for validation, synchronization and rollback points during the deployment of a service, and illustrate how the Netconf protocol could be enhanced by the addition of a likewise transactional component.

The paper is structured as follows. In section 2, we briefly overview the concepts of configuration tree and semantical dependencies and introduce by the means of concrete examples the concept of sequential dependency. We also describe the mathematical framework of model checking and show how sequential dependencies can be modelled by temporal logic. In section 3, we introduce the concept of milestone and apply it to the Netconf protocol. Finally, section 4 shows some experimental results and section 5 concludes with future paths of work.

2. The Sequential Aspect of Network Management

The deployment of a service over a network basically consists in altering the configuration of one or many equipments to implement the desired functionalities. We can presuppose without loss of generality that all properties of a given configuration are described by attribute-value pairs hierarchically organised in a tree structure (Hallé *et al.*, 2004a; Villemaire *et al.*, 2005) like the one shown in Figure 1.

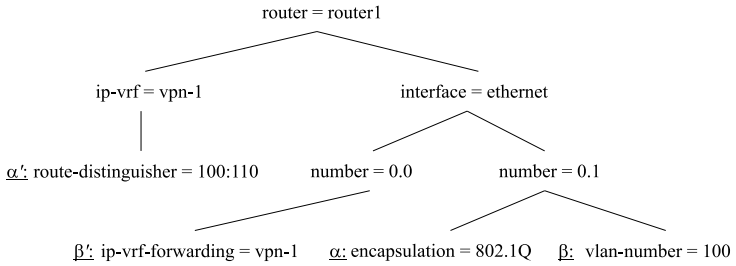


Figure 1. *A sample configuration tree. Nodes labelled α, α', β and β' are not present initially, but are added in the process of deploying the network services given later as examples.*

Possible alterations to the configuration typically include deleting or adding new parameters to the configuration of a device, or changing the value of existing parameters. In most cases, the parameters involved in such modifications are both syntactically and semantically interdependent. For instance, the value of some parameter might be required to depend in a precise way on the value of another parameter; the simplest example of such dependency is the fact that an IP address must match the subnet mask that comes with it. More complex dependencies might

constrain the existence of a parameter to the existence of another. Recent works have shown how such dependencies can be automatically checked by logical tools on a given configuration snapshot (Hallé *et al.*, 2004b).

2.1. *Sequential Dependencies at the Service Level*

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

We illustrate the concept of sequential dependencies by the means of two examples taken from the deployment of network services. For each of these examples, a sequential dependency is extracted and formalised.

2.1.1. *Example 1: Virtual LANs*

A Virtual LAN (VLAN) is a group of devices spanning multiple LAN segments that are configured to communicate as if they were connected to the same wire. Each VLAN works as a completely separate entity that can only be joined by a router. Since VLANs are logically (rather than physically) structured, they are extremely flexible. Among the several protocols designed to this purpose, IEEE 802.1Q (IEEE, 1998) has become the standard.

When configuring a router on a VLAN, the subinterface that is connected to a VLAN *trunk* must be set to support the 802.1Q protocol; since each subinterface is attached to a specific VLAN, the number of this VLAN must also be specified when configuring the trunk. Therefore, configuring a VLAN trunk will have for effect of adding nodes α and β in the configuration tree of Figure 1.

However, 802.1Q frames are designed in a way that they must contain the VLAN number; therefore, encapsulation and VLAN number must be configured toghether. From this simple example, one can deduce this first sequential rule:

Sequential Constraint 1 In a router, the VLAN number must be set at the same time the encapsulation protocol is enabled.

In the case of Figure 1, this means that nodes α and β must be added to the tree in the same step.

2.1.2. *Example 2: Virtual Private Networks*

A VPN is a private network constructed within a public network such as a service provider's network (Rosen *et al.*, 1999). A customer might have several sites, which are contiguous parts of the network, dispersed throughout the Internet and would like to link them together by a protected communication. The VPN ensures the connectivity and privacy of the customer's communications between sites.

Such a service consists of multiple configuration operations; in the case of Layer 3 VPNs, it involves setting the routing tables and the VPN forwarding tables, setting the MPLS, BGP and IGP connectivity on multiple equipments having various roles, such as the customer edge (CE), provider edge (PE) and provider core (PC) routers. An average of 10 paramters must be added or changed in each device involved in the deployment of the VPN.

As an example, for a Layer 3 VPN using MPLS, Figure 1 shows two leaf nodes that must be added, each in its own position, to the configuration tree of a PE router. Node α' corresponds to the creation of the Virtual Routing and Forwarding Tables (VRF) necessary for the proper functioning of the VPN; node β' associates this VRF to a specific interface on the router. Semantically, it is clear that one cannot associate a VRF to an interface before the VRF is created in memory. Therefore, trying to add node β' to the configuration before node α' is created is nonsensical and generates an error. From this situation, we can elicit a second sequential rule:

Sequential Constraint 2 To add node `ip-vrf-forwarding` to a configuration tree, the node `route-distinguisher` must already be present.

Special emphasis must be made on the fact that the node `route-distinguisher` has to be present in the tree *before* node `ip-vrf-forwarding` is added, which rules out the possibility that both nodes be added in a single operation.

2.2. *Formalising Sequences of Configuration Operations*

To formalise the sequences of operations, we first need to introduce some basic concepts taken from the theory of model checking (Clarke *et al.*, 2000). Let *S* be a set of *states* representing a unit situation at a given time. In the context of network configuration, states are labelled trees, as described previously.

We call *transition* from a state s_1 to a state s_2 the structural modifications that transform s_1 into s_2. Formally, transitions can be defined as a subset of tuples $T \subseteq S \times S$; there exists a transition from s_1 to s_2 if and only if $(s_1, s_2) \in T$. The tuple (S, T) forms a directed graph *G* that we call a *Kripke structure*. Figure 2 shows an example of a Kripke structure.

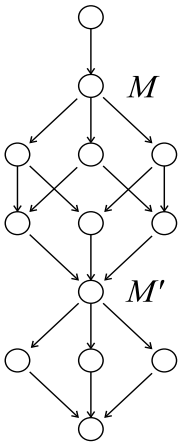


Figure 2. *A Kripke structure with multiple paths from a start state to a target state. Each state represents a labelled tree.*

In the case of the labelled trees we use for modelling device configurations, structural modifications are limited to addition of a labelled node to a leaf, deletion of a leaf node and change in a node's value. These modifications intuitively refer to addition, deletion or modification of a parameter in the configuration of a device.

A *path* is a finite sequence of states $\langle s_1,\ldots,s_n\rangle$ such that, for any s_i, s_{i+1}, there exists a $t \in T$ such that $t=(s_i, s_{i+1})$.

The deployment of a service is a path in such a structure that starts from a given configuration, s_s, and ends at a target configuration s_t. For example, in the case of Figure 1, a possible start state could be the tree without any of α, α', β, β', and a possible target state could be the same tree with all these nodes. A valid deployment sequence could be a sequence of addition of the nodes that respects, among other things, Sequential Rules 1 and 2.

2.3. *Formalising Sequential Dependencies*

The state space generated by spanning all possible transitions between a start and target state is fairly large. For the 4 nodes of Figure 1, there are 24 possible unconstrained paths, and in general, for *n* possible operations, there are *n*! possible paths. We must now restrict our study to acceptable paths —that is, paths that respect the elicited sequential constraints. For this purpose, we use the Linear Temporal Logic (LTL) commonly used in model checking (Clarke *et al.*, 2000).

LTL is a logic aimed at describing sequential properties along paths in a given Kripke structure. Its syntax is based on classical propositional logic, to which modal path operators have been added.

The first such modal operator is **G**, which means "globally". Formally, the formula **G** φ is true on a given path π when, for all states along this path, the formula φ is true. The second modal operator commonly used is **X** ("next"). The formula **X** φ is true on a given path π of the Kripke structure when the next state along π satisfies φ . Finally, a formula of the form **F** φ ("eventually") is true on a path when at least one state of the path satisfies φ . Other modal operators exist, but go beyond the scope of this paper.

A LTL formula is a well-formed combination of the classical $\wedge$ (disjunction), $\vee$ (conjunction), $\neg$ (negation), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence) operators with modal operators. The *atoms* of LTL are the base-level Boolean expressions over which the formulas are built. In the present case, since states are labelled trees, we take the atoms to be formulas themselves, based on a tree logic such as CL (Villemaire *et al.*, 2005).

Equipped with it, it is now possible to formalize sequential constraints into logical formulas. Without delving into further details, suppose that φ_n is a CL formula that is true if and only if node *n* is present in a given configuration tree. Then, the Sequential Constraint 1 presented in the previous section can be translated into the following formula:

Sequential Formula 1

$$\mathbf{G}\ \ \varphi_\alpha \leftrightarrow \varphi_\beta$$

This formula means that in all states of all paths where either α or β exists, the other node must also exist. All temporal rules described earlier can therefore be translated into LTL formulas of that kind. For example, Sequential Rule 2 becomes:

Sequential Formula 2

$$\mathbf{G}\ (\ \neg\varphi_{\alpha'} \rightarrow \mathbf{X}\ (\ \varphi_{\alpha'} \rightarrow\ \neg\varphi_{\beta'}\ ))$$

telling that the presence of node β' implies that node α' is present, and that is must also have been present at least in the previous step in the deployment.

3. Transactional Aspects of the Netconf Protocol

We now show how transactional aspects presented in the previous section can be applied to the Netconf protocol by enhancing it with a transactional component.

3.1. *Overview of the Netconf Protocol*

To send configuration commands to a router, Netconf provides a set of "remote procedure calls" (RPC) and RPC-replies. In a simplified way, an RPC is a block of

XML data whose opening tag contains an identifier that either asks the router to return a portion of its configuration file, or tells it to replace a part of its configuration with a snippet provided by the user and carried in the body of the RPC. Netconf offers other built-in operations, such as commands allowing to lock a part of the router's configuration so that only the current user can modify it, and subsequently unlock it. The RPC-reply is the XML block that is returned to the user.

The Netconf protocol defines a simple mechanism for device management. However, its transactional model, which includes a validation capability, is device-centered, and does not provide a mechanism to ensure the consistency of the sequence of operations with respect to the rules elicited in section 2.

In order to bestow transactional semantics on the update operations of multiple configurations, it is important to determine the optimal points of validation, commitment and roll-back during the update process of the network device configurations.

3.2. *Components and Milestones*

We propose to determine these points by analysing special properties of the Kripke structure induced by the sequential dependencies. To this purpose, we introduce the notion of *milestone state*. A milestone state is a state *m* by which all valid paths must eventually pass. Formally, let ψ be some LTL temporal rule, and τ_m be a CL formula that is true only on state *m*. Then, in a Kripke structure *G*, *m* is such that for every path beginning at the start state and that satisfies ψ, the formula $\mathbf{F}\ \tau_m$ is true.

Milestones can be thought of as unavoidable steps in the path from start to solution, since all acceptable paths must eventually pass by those points, in the order they appear. In the case of Figure 2, we see two milestones, labelled *M* and *M'*.

We argue that milestones are good candidates to divide the modelled process into natural macro-steps of which they are the boundaries; complementary to milestone states are *components*, i.e. sets of states and transitions comprised between two milestone states. The word “natural” is used here, since these milestones emerge from the set of temporal constraints imposed on the lattice. Different temporal constraints generally lead to different milestones.

3.3 *Towards a Transactional Model*

The main advantage of the analysis of the lattice that arises from temporal constraints is that it induces a way of synthesising a protocol for the

implementation of a service. By placing validation checkpoints at milestones, we ensure such checkpoints are placed in semantically sound locations throughout the deployment process. Since these checkpoints reflect the structure imposed by the temporal constraints, they also make good points to roll back in case a failure occurs.

These points are important since they represent optimal places of validation in the flow of operations. Thus, a validation in one of these points can check all or most of the dependences that apply on the multiple flow streams that converge towards the validation point. Moreover, these convergence points are unavoidable during the configuration and, since they concentrate the flow paths, a validation performed at such points provides a maximum extent of coverage for those flows.

Intuitively, we suggest that a validation point be used to validate the operations that are situated along the flow path connecting it to a previous upstream milestone, in which a validation has been already done. If the validation has been successful, the update information generated by the operations is committed. Otherwise, if the validation or the commitment fails, the update information generated by the operations is discarded and the configurations are rolled back to the latest points of successful validation.

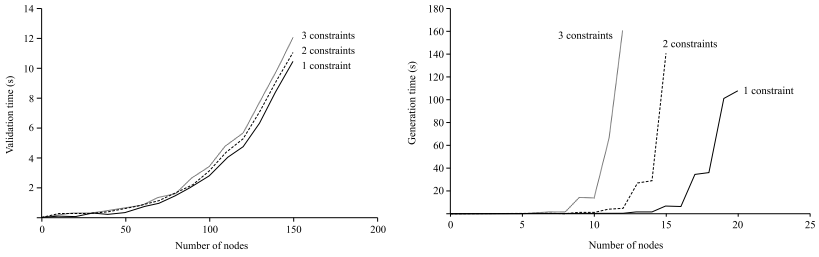
Netconf provides two phases of a successful configuration transaction during a service configuration procedure: preparation and commitment. During preparation, the configurations are retrieved from the network devices. When all the configurations have been retrieved, the edition starts at service level.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

Since the network service update affects multiple device configurations, a two-phase commit is required. The first phase stores the update information on temporary storage and validates it before entering the second phase. If the validation is successful, the update information is transferred onto the real configurations, otherwise this information is discarded. In case of erroneous transfers during the second phase, the configurations are rolled back and the second phase can be resumed.

This semantics can be used with the Netconf configuration protocol to ensure the transactional properties of the service updates on multiple devices. As already mentioned, the Netconf protocol defines transactional operations for device level but does not provide similar operations for network level, i.e. for the multiple device configurations supporting a network service.

Obviously, the higher-level validations may involve multiple devices. For instance, the routing table and the protocol information in a device depend on the network addresses and the protocol information from other devices. Similarly, parameters such as protocol neighbors' IP addresses, autonomous system numbers, protocols' process number and IP addresses must accurately correspond on more than one device, in order for the network service that is deployed over that network to be consistent.

In this case, defining an operation that can validate multiple parameters situated on several devices might be highly recommendable. This multi-device validation operation would replace multiple single-device validation operations and would allow performing complex validation queries directly within the given configuration protocol.



a) b)

Figure 3. a) *Validation time of a deployment sequence in terms of number of nodes to alter and constraints per node.* **b)** *Generation time of a valid deployment sequence in terms of number of nodes to alter and constraints per node.*

4. Experimental Results

Since the structures generated by sevice deployments are Kripke structures and that the sequential formulas can be formalised in LTL, it is possible to submit the problem directly to a model checker like NuSMV (Cimatti *et al.*, 2002). Using this tool, we generated sample deployment sequences and checked that these deployments respected a set of constraints similar to Sequential Rules 1 and 2. For each test, we varied the number of nodes in the sequence and the number of sequential constraints imposed on each nodes. The validation times for these experiments are summarised in Figure 3a. All times given in this section have been calculated on an AMD Athlon XP-M 2200+ running NuSMV 2.1.2 under Cygwin.

One can see that validation times for large sequences of operations (up to 150 nodes) remain under the reasonable bound of 10 seconds, and that augmenting the number of constraints is not the principal factor that makes the computation longer.

Additionally, it is possible to benefit from the counter-example generation mechanism of NuSMV to find a deployment sequence that does not violate any constraint. As a matter of fact, when a LTL property of the form **G** *p* is false, NuSMV provides the user with an execution trace on the Kripke structure for which *p* is false. If *p* is the LTL property one wants to verify on a structure, it suffices to submit the formula **G** ¬*p* for verification. If there exists a trace for which *p* is true, then such a trace is a counter-example for the formula **G** ¬*p*, and therefore NuSMV will display it to the user, giving by the same occasion a valid deployment sequence.

We have conducted experiments with NuSMV on sample deployment sequences with constraints of the same form as Sequential Formulas 1 and 2. We varied the size of the configurations and the number of sequential constraints per node imposed on the structure, and computed the time NuSMV took to provide a correct deployment sequence. The results of these experiments are presented in Figure 3b. Each curve corresponds to the generation time of a valid deployment sequence involving some number of nodes, with 1, 2 or 3 sequential constraints imposed on each *node* —that is, the total number of constraints actually increases with the number of nodes.

As expected, generating a valid sequence is much harder than validating an existing one. Moreover, the number of sequential constraints on each node does matter in this case, and can change a rather simple situation into an untractable one. One can see that, for sequences that involve the addition or modification of about 10 nodes, which is comparable to deployment of a simple VPN on a router, up to three sequential constraints per node can be imposed without the generation time becoming prohibitive.

These findings suggest that model checking is indeed an interesting tool for on-the-fly validation of deployment sequences, and for offline, *a priori* synthesis of valid sequences for network services with a complexity comparable to a Virtual Private Network.

5. Conclusion

In this paper, we have shown how Linear Temporal Logic applied to Kripke structures can accurately formalise sequential constraints in the deployment of network services. Using these model checking concepts, we defined the notion of *milestone* states in a Kripke structure and gave arguments for using these points as validation, synchronization and rollback points during the deployment of a service,and illustrated how the Netconf protocol could be enhanced by the addition of a transactional component based on milestones.

Empirical results on sample network configurations demonstrate the feasibility of validating deployment sequences using model checking tools, and show that

finding a deployment sequence that validates a set of constraints is a computationally hard problem.

The authors plan future work on these concepts in order to further use milestones in a hiearchical decomposition of a service deployment. In such a setting, each component could contain sub-milestones that would further divide a process into sub-steps based on the same principle. Moreover, the current methodology could be extended by considering all possible orderings of operations in a component and eventually reduce the study to one specific ordering, in the same way *partial order reduction* reduces the state space in model checking (Clarke *et al.*, 2000).

References

Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. Proc. International Conference on Computer-Aided Verification (CAV 2002), 359-364. (2002)

Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking, MIT Press (2000)

Couch, A., Sun, Y.: On the Algebraic Structure of Convergence, Proc. DSOM 2003, Springer, 28-40 (2003)

Crubézy, M.: The Protégé Axiom Language and Toolset ("PAL"). Protégé Project, Stanford University (2002) http://protege.stanford.edu/

Daminaou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder policy Specification Language, Proc. Policy'2001, Springer, 29-31 (2001)

Enns, R.: Netconf Configuration Protocol. Internet draft, April 2005. http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-06.txt

Hallé, S., Deca, R., Cherkaoui, O., Villemaire, R.: Automated Verification of Service Configuration on Network Devices. Proc. MMNS 2004, Springer, 176-188 (2004).

Hallé, S., Deca, R., Cherkaoui, O., Villemaire, R., Puche, D.: A Formal Validation Model for the Netconf Protocol. Proc. DSOM 2004, Springer, 147-158 (2004)

IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks. IEEE Standard 802.1Q-1998 (1998)

Jackson, D., Schechter, I., Shlyakhter, I.: Alcoa: the Alloy Constraint Analyzer, Proc. ICSE (2000)

Rosen, E., Rechter, Y.: BGP/MPLS VPNs. RFC 2547 (1999)

Villemaire, R., Hallé, S., Cherkaoui, O.: Configuration Logic: A Multi-site Modal Logic. Proc. TIME 2005, IEEE Computer Society, 131-137 (2005)

Warmer, J., Kleppe, A.: OCL: The constraint language of the UML. Journal of Object-Oriented Programming, May 1999.