Vol.2021-IOT-52 No.35
2021/3/2

状況に応じた最適なサーバ構成管理を実現するための ポリシー定義と振る舞い制御を中間言語で分離する手法

宮下 剛輔[1,2,a)]　松本 亮介[1]

概要：サーバ構成管理とは，サーバにインストールすべきソフトウェアや行うべき設定などをポリシーとして定義し，そのポリシーに従ってサーバの振る舞いを制御するプロセスである．したがってサーバ構成管理は，ポリシー定義と振る舞い制御という二つの側面に分解することができる．ポリシー定義は何らかの言語を用いて行うが，どのような言語が最適かは，利用者の好みやスキル，利用者が属する組織の状況などによって異なる．振る舞い制御は，定義されたポリシーに従ってソフトウェアのインストールや設定などを行うが，ポリシーの取得方法，制御プロセスの実行方法など様々で，状況に適した手法や実装を選択する必要がある．ポリシー定義と振る舞い制御が分離されていると，状況に適したポリシー定義と振る舞い制御を個別に選択して組み合わせることで，より適切な形で構成管理が行える．また，実装者は，ポリシー定義の実装のみ，あるいは振る舞い制御の実装のみに注力することができる．しかし，既存の構成管理ツールは，ポリシー定義と振る舞い制御が単一のソフトウェアとして実装されているため，このようなことができない．本研究では，構成管理ツール利用者がより状況に適した形で構成管理を行うため，また，実装者がポリシー定義のみ，あるいは振る舞い制御のみの実装に注力できるようにするため，ポリシー定義と振る舞い制御を中間言語で分離する手法について述べる．

キーワード：サーバ構成管理，中間言語

A method for separating policy definition and behavior control by an intermediate language to achieve optimal server configuration management according to the situation

Gosuke Miyashita[1,2,a)]　Ryosuke Matsumoto[1]

Abstract: Server configuration management is the process of defining the software to be installed on a server and the settings to be made as a policy, and controlling the behavior of the server according to the policy. Therefore, server configuration management can be decomposed into two aspects: policy definition and behavior control. Policy definition is done using some language, and the best language depends on the user's preferences and skills, and the situation of the organization to which the user belongs. In behavior control, software is installed and configured according to the defined policy, but there are various methods of obtaining the policy and executing the control process, and it is necessary to select the method and implementation that are appropriate for the situation. When policy definition and behavior control are separated, configuration management can be performed in a more appropriate manner by selecting and combining policy definition and behavior control that are appropriate for the situation. In addition, implementers can focus only on the implementation of policy definition or behavior control. However, existing configuration management tools cannot do this because policy definition and behavior control are implemented as a single piece of software. In this paper, we describe a method of separating policy definition and behavior control by an intermediate language, so that users of configuration management tools can manage configuration in a more appropriate manner and implementers can focus on implementing only policy definition or behavior control.

Keywords: Server Configuration Management，Intermediate Language

ⓒ 2021 Information Processing Society of Japan

1. はじめに

サーバ構成管理とは，サーバにインストールすべきソフトウェアや行うべき設定などをポリシーとして定義し，そのポリシーに従ってサーバの振る舞いを制御するプロセスである．

サーバ運用管理を必要としない，マネージドなコンテナ実行環境を提供するサービスや，サーバレス・コンピューティングと呼ばれるサービスが普及している．そのため，サーバ構成管理を必要とする現場が少なくなっている．だが，マネージドサービスを利用せずに自前でサーバを運用管理する事業者や，マネージドサービスを提供する事業者にとっては，サーバの運用管理はいまだ必要であり，サーバ構成管理そのものが不要になったわけではない．しかし，サーバ構成管理ツールについては，2012 年に登場した Ansible[1] 以降，これといった発展が見られない．また，Ansible のような既存の構成管理ツールを利用せずに，各事業者が独自の手法や実装でサーバ構成管理を行っている事例 [2][3] も見受けられる．

一方，サービス利用者の利便性や，サービスの信頼性を向上するために，利用者が所持する端末とサーバ間のネットワーク遅延の短縮や，データセンターの災害回復のために，地理的に分散したデータセンターを利用したコンピューティング環境 [4] の重要性が増している．このような地理的に分散したコンピューティング環境では，従来のサーバ構成管理手法や実装が適用できず，別の手法や実装が必要になると考えられる．例えば，モバイルコンピューティングやエッジコンピューティングでは，配備されるサーバの CPU，メモリ，ストレージリソースがそれほど潤沢ではないため，そこで動作する構成管理ツールには，既存のツールよりも大きな制約が課される．また，単一のデータセンター環境と比較し，システムの管理者が中央からすべてのサーバの構成状態を把握することが困難となる．データセンター間のネットワーク遅延が大きく，切断されやすいといった問題もある．さらに，ハードウェアの種類やネットワーク速度・接続の安定性がデータセンター毎に異なる，という状況も考えられる．

サーバの構成管理を必要とする人や現場は少なくなったが，構成管理は依然として必要であり，今後のコンピューティング環境の変化に合わせて発展していく必要がある．Ansible 以降，構成管理ツールの発展が見られないのは，必要とする人や現場が減ったことが一因と考えられる．また，構成管理ツールはその性質上，OS やディストリビューションの違いを吸収して抽象化するために，実装が煩雑である．この煩雑さも，構成管理ツールの発展を妨げる要因と考えられる．

サーバ構成管理は，ポリシーの定義と，定義されたポリシーに基づいたマシンの振る舞い制御という 2 つの側面に分解することができる．ポリシー定義と振る舞い制御が分離されていると，構成管理ツールの利用者は，状況に適したポリシー定義と振る舞い制御を個別に選択して組み合わせることで，より適切な形で構成管理が行えるようになる．また，ツールの実装者は，ポリシー定義の実装のみ，あるいは振る舞い制御の実装のみに注力することができるようになる．しかし，既存の構成管理ツールは，ポリシー定義と振る舞い制御が単一のソフトウェアとして実装されているため，このようなことができない．そこで本稿では，中間言語によって，構成管理のポリシー定義と振る舞い制御を明確に分離する手法について述べる．

本稿の構成を述べる．2 章では，サーバ構成管理の現状と課題について整理する．3 章では，中間言語によりポリシー定義と振る舞い制御を明確に分離する手法について述べ，4 章でまとめとする．

2. サーバ構成管理の現状と課題

2.1 ポリシー定義と振る舞い制御

Burgess ら [5] は，構成管理とは，予め定義されたポリシーとガイドラインに従い，事前に決められたビジネス上の目的を達成するよう，ネットワーク接続されたマシンの振る舞いを制御するプロセスである，と定義している．

構成管理ツールはその名の通り，構成管理を行うソフトウェアである．代表的なものとして，Chef[6]，Ansible，Puppet[7]，SaltStack[8] があり，これらをまとめて CAPS と呼ぶこともある．

Burgess らの構成管理に関する定義から，構成管理はポリシー定義と振る舞い制御という 2 つの側面に分割することができる．

2.2 ポリシー定義用言語

構成管理では，ポリシー定義は何らかの言語を用いて行う．ポリシー定義言語は大別すると，以下の 3 つにわけられる．

- 独自の簡易言語
- YAML[9] や JSON[10] のような標準的な簡易言語
- 汎用プログラミング言語

構成管理ツールの走りである CFEngine[11] や，CFEngine を発展させることで生まれた Puppet は独自の簡易言語を採用している．構成管理ツールの従来の利用対象者であるシステム管理者は，プログラミングを行わない人が多い．また，CFEngine や Puppet の開発当初は，JSON や YAML のような標準的な簡易言語が存在しなかった．その

[1] さくらインターネット株式会社 さくらインターネット研究所
SAKURA Research Center, SAKURA Internet Inc., Akasaka, Chuo-ku, Fukuoka 810-0042 Japan
[2] 合同会社 Serverspec Operations
Serverspec Operations, LLC
[a)] miyashita@serverspec-operations.com

ⓒ 2021 Information Processing Society of Japan

ため独自の簡易言語を採用したと考えられる．

Puppet を発展させることで生まれた Chef は，Ruby[12] という汎用プログラミング言語を採用している．クラウドの普及により，システム管理者だけではなくアプリケーション開発者もサーバインフラを触るようになった．このような人達は，独自の簡易言語や標準的な簡易言語よりも，慣れ親しんでいるプログラミング言語を好む傾向にある．

Ansible は標準的な簡易言語である YAML を採用している．現在，CAPS の中で最も人気があるのは Ansible であり，人気の一因はポリシー定義言語に YAML を採用していることにある．プログラミングを行わないシステム管理者には，YAML のような簡易言語が好まれる傾向にある．また，YAML のような簡易言語は，変数やロジックがないため，記述を簡易にできメンテナンスしやすそうに見える点も，システム管理者に好まれる．ただし，変数やロジックがない，というのはデメリットでもあり，それを補う手法が同時に使われているケースもある．

このように，構成管理用のポリシー定義言語には，独自簡易言語から汎用プログラミング言語へ，汎用プログラミング言語から標準的簡易言語へ，という流れが見られる．

構成管理ツールと同じく Infrastructure as Code[13] にカテゴライズされる他のツールにも，同様なポリシー定義言語の変遷が見られる．

Terraform[14] や AWS CloudFormation[15] という，クラウド上のインフラプラットフォームをコードで管理するツールを例に挙げる．Terraform は，HCL(HashiCorp Configuration Language)[16] と呼ばれる，JSON と互換性のある独自の簡易言語を採用している．AWS CloudFormation は JSON と YAML をポリシー定義言語として採用している．Terraform と同様の立ち位置である Pulumi[17] は，ポリシー定義言語に汎用プログラミング言語を採用している．また，AWS CDK[18] は汎用プログラミング言語で AWS CloudFormation のポリシー定義を行えるようにしたものである．

同じく Infrastructure as Code にカテゴライズされる，コンテナオーケストレーションツールである Kubernetes[19] は，ポリシー定義言語に YAML を採用している．YAML は機能不足なため，その欠点を補うためのツール [20][21] もいくつか出ている．また，Kubernetes with Pulumi[22]，CDK for Kubernetes(cdk8s)[23] など，汎用プログラミング言語で Kubernetes のポリシー定義を行うためのツールも出てきている．

このように，クラウドインフラプラットフォーム管理ツールやコンテナオーケストレーションツールでは，簡易言語から汎用プログラミング言語へ，という構成管理ツールとは逆の流れが発生している．

Infrastructure as Code 全体でポリシー定義言語の変遷を見ると，汎用プログラミング言語から標準的簡易言語へ，標準的簡易言語から汎用プログラミング言語へ，と両者の間で揺れ動いている．

簡易ではあるが機能不足である標準的簡易言語と，機能豊富ではあるが複雑である汎用プログラミング言語，どちらがポリシー定義言語として優れているかは，利用する人のスキルや好み，その人が属する組織の状況，その時の周辺技術のベスプラクティスなどに依存するため，一概に決めることはできない．そのことが，両者の間の揺れに繋がっていると考えられる．

2.3 振る舞い制御

振る舞い制御の手法は大別すると，以下の 3 つに分けられる．

- サーバ/エージェント型
- スタンドアローン型
- エージェントレス型

Chef や Puppet はサーバー/エージェント型の構成をとっており，中央のサーバで管理されたポリシーを各マシンに配布，各マシン上で動作しているエージェントがポリシーを適用し振る舞い制御を行う．また，サーバ/エージェント型としてだけではなく，スタンドアローンで実行することもできる．サーバ/エージェント型は，ポリシーの配布，サーバ/エージェント間の接続や認証，振る舞い制御プロセスの実行方法やタイミングなどを構成管理ツールが持つ機能に任せることができる，というメリットがある．反面，サーバプロセスやエージェントプロセスの監視をどうするか，サーバとエージェントの初期接続時の認証情報をどのように受け渡すか，などといった点を考慮する必要がある．

スタンドアローン型は，中央の管理サーバが存在せず，各マシン上で動作するエージェントがポリシーを適用し，振る舞い制御を行う．中央のサーバが存在しないため，サーバプロセスの監視や，サーバとエージェント間の接続や認証をどう行うのか，といった点については考慮する必要はない．反面，ポリシー定義コードをどのように取得するのか，どのようなタイミングでポリシー適用を行うのか，エージェントプロセスをどのように起動するのか（常駐プロセスとするのか，定期実行するのかなど）を，自ら考えて行う必要がある．

エージェントレス型はスタンドアローン型と異なり，ポリシーを適用して振る舞い制御を行う対象のマシンに，特別なエージェントプログラムを必要としない．そのため，対象マシン上に余分なソフトウェアをインストールしたくない場合に好まれる．Ansible はエージェントレス型で，リモートマシンに対して SSH で接続して振る舞い制御を行うことができる．また，スタンドアローン型のように，対象マシン上で動作して，振る舞い制御を行うこともできる．エージェントレス型もスタンドアローン型同様，ポリシー定義コードをどのように取得するか，どのようなタイ

ミングでポリシー適用を行うのか，エージェントをどのように動かすのかを，自ら考えて行う必要がある．

また，モバイルコンピューティングやエッジコンピューティングの普及に伴い，多様なデバイスへの対応，転送容量削減，メモリ容量節約，実行速度の向上，自律制御といった観点から，別の制御手法や実装が求められることも考えられる．

さらに，現在の構成管理ツールは，記述されたポリシーを逐次解釈して制御を行うインタプリタ型だが，制御用コードを生成して実行するコンパイラ型の方が望ましいケースも考えられる．

このように，ポリシー定義言語と同様，振る舞い制御についても，どれが最適かは一概には決めることはできない．

2.4 多様なポリシー定義言語と振る舞い制御への対応

2.2 節と 2.3 節では，どのポリシー定義言語や振る舞い制御が最適かは状況により異なり，一概には決められないということを述べた．ポリシー定義言語も振る舞い制御も，条件により最適なものが異なる，という前提に立つと，様々なポリシー定義言語や振る舞い制御が存在する，ということは好ましいことである．

しかし，既存の構成管理ツールは，ポリシー定義と振る舞い制御が単一のソフトウェアとして実装されている．そのため，利用者は状況に適したポリシー定義と振る舞い制御を選択して組み合わせて利用することができない．また，ポリシー定義部分だけ，より状況に適した別の手法を採用したい，という場合でも，既存手法ではポリシー定義部分だけではなく，振る舞い制御部分も併せて再実装しなければならない．同様に，振る舞い制御部分だけ，より状況に適した別の手法を採用したい，という場合でも，既存手法では振る舞い制御部分だけではなく，ポリシー定義部分も併せて再実装しなければならない．

この問題を解決するために，次の 3 章では，ポリシー定義と振る舞い制御を中間言語により明確に分離する手法を提案する．

3. 中間言語によるポリシー定義と振る舞い制御の分離

3.1 構成管理ツールの 3 層構造化

Serverspec[24] では**図 1** のように，構成管理ツールによって構成されたサーバのテストを行うフレームワークを 2 層に分離する手法を提案した．Serverspec はテストフレームワークではあるが，ポリシー定義（テストコード）にしたがって，コマンドを実行してサーバのテストを行うという点で，構成管理ツールと同じ挙動をする．図 1 左側の制御テストフレームワーク部分がポリシー定義，右側の汎用コマンド実行フレームワーク部分が振る舞い制御に該当する．そのため，Serverspec の手法は構成管理ツールにも応用できる．

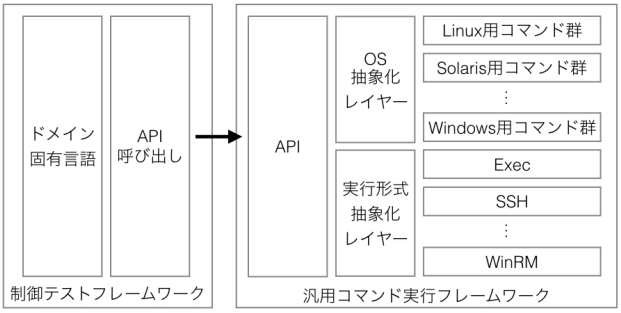

図 1 Serverspec の 2 層アーキテクチャ
Fig. 1 2 layered architecture of Serverspec

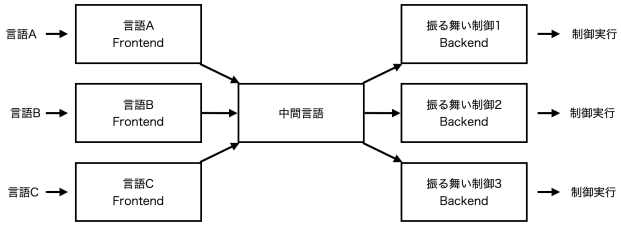

図 2 中間言語によるポリシー定義と振る舞い制御の分離
Fig. 2 Separation of policy definitions and behavior control by an intermidiate language

Serverspec の手法は，ポリシー定義と振る舞い制御をプログラミング言語の関数呼び出しやメソッド呼び出しによって繋ぐ．そのため，実装言語が単一のものに制限され，他の言語で実装されたポリシー定義や振る舞い制御と組み合わせることができない．

そこで**図 2** のように，中間言語を間に置くことで，ポリシー定義言語と振る舞い制御を明確に分離する 3 層モデルを提案する．この方式では，ポリシー定義用フロントエンドが各言語で書かれたポリシー定義を中間言語に変換する．そして，振る舞い制御用バックエンドが，中間言語で記述されたポリシー定義に基づいて振る舞い制御を行う．図 2 右端の制御実行部分は，逐次実行型の場合は中間言語を解釈してそのまま実行することをイメージしているが，そうではなく制御実行コードを吐き出す，といった形も考えられる．

提案方式は LLVM[25] に着想を得ている．LLVM は**図 3** のような 3 層モデルになっており，各種言語用フロントエンドが，その言語で書かれたコードを LLVM IR という中間言語に変換する．その後 LLVM が中間言語に最適化処理などを施し，最終的には各 CPU アーキテクチャ用のバックエンドが，そのアーキテクチャで実行可能なバイナリコードを生成する．

3 層構造にすることで，構成管理ツールの利用者は用途に適したポリシー定義言語と振る舞い制御を組み合わせて利用することができるようになる．また，構成管理ツール

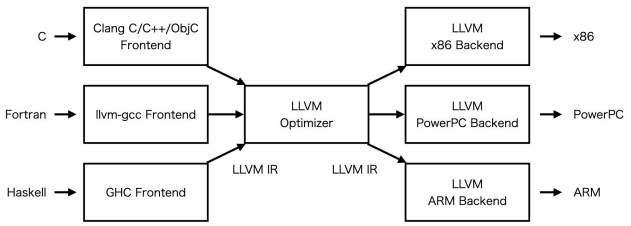

図 3 LLVM の 3 層構造アーキテクチャ
Fig. 3 3 layered architecture of LLVM

の実装者は，ポリシー定義の実装のみ，あるいは振る舞い制御の実装のみに注力することができるようになる．

3.2 中間言語

中間言語は，既存のポリシー定義言語の記述をすべて網羅できるものであると同時に，それ自体がポリシー定義言語として直接使えるものになる．しかし，ポリシー定義言語は人間が読み書きすることを想定して設計されたものであるのに対し，中間言語は必ずしも人間が読み書きする必要がない．そのため，従来のポリシー定義言語とは異なる考え方に基づいて設計する必要がある．

中間言語という統一的な層を設けることによる応用も期待できる．例えば，ポリシー定義言語により記述された構成の正しさを検証する場合，中間言語に変換してから検証を行うことで，ポリシー定義言語がどのようなものであっても，統一的な手法で検証することができる．

中間言語を設けることによるデメリットも存在する．中間言語により，構成管理対象となるマシンの OS やディストリビューションの違いを吸収し抽象化することになるが，違いをすべて吸収することは難しい．また，ポリシー定義言語に汎用プログラミング言語を採用した場合，その記述をすべて中間言語に変換するのは困難である．

4. まとめ

既存の構成管理ツールは，ポリシー定義と振る舞い制御が単一のソフトウェアとして実装されている．そのため，状況に適したポリシー定義と振る舞い制御を選択して組み合わせて利用することができない．また，ポリシー定義部分だけ，より状況に適した別の手法を採用したい，という場合でも，既存手法ではポリシー定義部分だけではなく，振る舞い制御部分も併せて再実装しなければならない．同様に，振る舞い制御部分だけ，より状況に適した別の手法を採用したい，という場合でも，既存手法では振る舞い制御部分だけではなく，ポリシー定義部分も併せて再実装しなければならない．この課題を解決するため，本稿では中間言語を置くことにより，ポリシー定義と振る舞い制御を明確に分離する手法を提案した．

本研究で提案する手法により，構成管理ツール利用者が，状況に適したポリシー定義と振る舞い制御を個別に選択して組み合わせることで，より適切な形で構成管理が行えるようになる．また，構成管理ツール実装者は，より状況に適したポリシー定義の実装のみ，あるいはより状況に適した振る舞い制御の実装のみに注力することができ，効率的に実装できるようになる．

今後の研究の進め方について述べる．

現在はまだ関連研究のリサーチを十分に行えていないので，まずはしっかりとリサーチを行いたい．特に，構成管理に関しては，Burgess, Couch らの先行研究 [5] や，ネットワーク分野での先行研究 [26] が存在するので，これらの関連研究からリサーチを行う．

また，構成管理のポリシー定義言語に関する先行研究もいくつかある [27][28][29] が，構成管理とは別領域，DSL やアスペクト指向言語といった分野についてもリサーチを行う必要があると考える．

本研究の肝は，中間言語をいかにうまく設計できるかにかかっている．ただし，ポリシー定義言語と振る舞い制御の分離を実現する手法は，中間言語を置く以外にもより適切な手法が存在する可能性もある．これについては今後の研究で明らかにしていきたい．

また，本研究の手法では，振る舞い制御実装そのものが持つ煩雑さは解決できていない．これについては別途研究が必要である．

参考文献

[1] Ansible is Simple IT Automation https://www.ansible.com/.
[2] 1000 台規模の Kubernetes クラスタを社内構築　サイボウズが明かすインフラ刷新 (1/2) : 運用を自動で行うツールを内製 - @ IT https://www.atmarkit.co.jp/ait/articles/1909/11/news010.html
[3] OpenStack と Kubernetes を利用したマルチプラットフォームへの CI 環境 - Yahoo! JAPAN Tech Blog https://techblog.yahoo.co.jp/infrastructure/os_n_k8s/
[4] B. Kashif, K. Osman, E. Aiman, and others, Potentials, Prospects in Edge Technologies: Fog, Cloudlet, Mobile Edge, and Micro Data Centers, Computer Networks, Vol. 130, pp. 94-120 2018.
[5] Burgess, Mark, and Alva L. Couch. 2006. “Modeling Next Generation Configuration Management Tools.” In LISA, 13147. static.usenix.org.
[6] Chef: Enabling the Coded Enterprise through Infrastructure, Security and Application Automation https://www.chef.io/
[7] Powerful infrastructure automation and delivery — Puppet https://puppet.com/
[8] Home — SaltStack https://www.saltstack.com/
[9] The Official YAML Web Site https://yaml.org/
[10] JSON https://www.json.org/json-en.html
[11] CFEngine https://cfengine.com/
[12] オブジェクト指向スクリプト言語 Ruby https://www.ruby-lang.org/ja/
[13] Infrastructure as Code https://infrastructure-as-

code.com/
[14] Terraform by HashiCorp https://www.terraform.io/
[15] AWS CloudFormation（テンプレートを使ったリソースのモデル化と管理）— AWS https://aws.amazon.com/jp/cloudformation/
[16] hashicorp/hcl: HCL is the HashiCorp configuration language. https://github.com/hashicorp/hcl
[17] Pulumi - Modern Infrastructure as Code https://www.pulumi.com/
[18] AWS クラウド開発キット アマゾン ウェブ サービス https://aws.amazon.com/jp/cdk/
[19] Kubernetes https://kubernetes.io/
[20] Helm https://helm.sh/
[21] Kustomize - Kubernetes native configuration management https://kustomize.io/
[22] Kubernetes with Pulumi — Pulumi https://www.pulumi.com/kubernetes/
[23] awslabs/cdk8s: Define Kubernetes native apps and abstractions using object-oriented programming https://github.com/awslabs/cdk8s
[24] 宮下剛輔，栗林健太郎，松本亮介，Serverspec: 宣言的記述でサーバの設定状態をテスト可能な汎用性の高いテストフレームワーク，情報処理学会論文誌，Vol.61， No.3，pp.677-686，2020 年 3 月.
[25] The LLVM Compiler Infrastructure Project https://llvm.org/
[26] Heiler, K., and R. Wies. 1996.“Policy Driven Configuration Management of Network Devices” 3 (April): 67489 vol.3.
[27] Cons, Lionel, and Piotr Poznaski. 2002. “Pan: A High-Level Configuration Language.” In Proceedings of the 16th USENIX Conference on System Administration, 8398. LISA ’02. USA: USENIX Association.
[28] Delaet, Thomas, and Wouter Joosen. 2007. “PoDIM: A Language for High-Level Configuration Management.” In Proceedings of the 21st Large Installation System Administration Conference (USENIX LISA’07), 26173. Usenix Association.
[29] Ngoup, ric Lunaud, Clment Parisot, Sylvan Stoesel, Petko Valtchev, Roger Villemaire, Omar Cherkaoui, Pierre Boucher, and Sylvain Hall. 2017. “A Declarative Approach to Network Device Configuration Correctness.” Journal of Network and Systems Management 25 (1): 180209.

ⓒ 2021 Information Processing Society of Japan