serverspec: 宣言的記述で
サーバの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

宮下 剛輔（株式会社paperboy&co./帝京大学）
栗林 健太郎（株式会社paperboy&co.）
松本 亮介（京都大学 情報学研究科 ）

目次

本研究の概要

研究の背景

- システムの大規模・複雑化
  - 科学やビジネス領域における問題の複雑化
  - サーバ設定をコードで記述する管理手法の登場
- 構成管理コードの複雑化
  - 汎用プログラミング言語で構成管理コードを記述
  - Test-Driven Infrastructureの必要性

サーバ構成管理とテスト手法

- 既存サーバ構成管理とテスト手法
  - 構成管理ツールと密な単体テストツール
  - 結合テストツールは「構成管理ツールと密」または「OS毎の違いはテストコードを書く人が考慮」
- 既存サーバテスト手法の問題点概要
  - 構成管理ツールからの独立性
  - OS・ディストリビューション汎用性
  - 双方を満たすテストツールが存在しない

本研究

- 汎用的かつ可読性の高いコードでテスト可能な手法の提案
  - 構成管理ツール独立/OS・ディストリビューション汎用性の双方を満たす
  - OSや構成管理ツールの違いを気にすることなくサーバの状態を容易にテストでき、サーバの運用・管理コストを低減
- 汎用コマンド実行フレームワークと制御テストフレームワークを分離
  - 制御テストフレームワークを容易に変更可能
  - 汎用コマンド実行フレームワークを別用途に応用可能

サーバの構成管理と
テスト手法

サーバの構成管理とテスト手法

- CFEngineからChefへ
  - 1993年にCFEngineが登場
  - 2005年にCFEngineの影響を受けたPuppetが登場
  - 2009年にPuppetの影響を受けたChefが登場
- ChefからTest-Driven Infrastructureへ
  - Chefは構成管理コードを汎用プログラミング言語で記述
  - コードが複雑になりTest-Driven Infrastructureの要請が高まる

従来テスト手法と課題

- Test-Driven Infrastructureにおけるテスト手法の分類
  - 単体テスト/結合テスト/(受け入れテスト)

- 従来テスト手法の問題点
  - 以下のいずれかにあてはまる
  - 特定の構成管理ツールに依存している
  - OS・ディストリビューションの違いをテストを書く人が考慮する必要がある

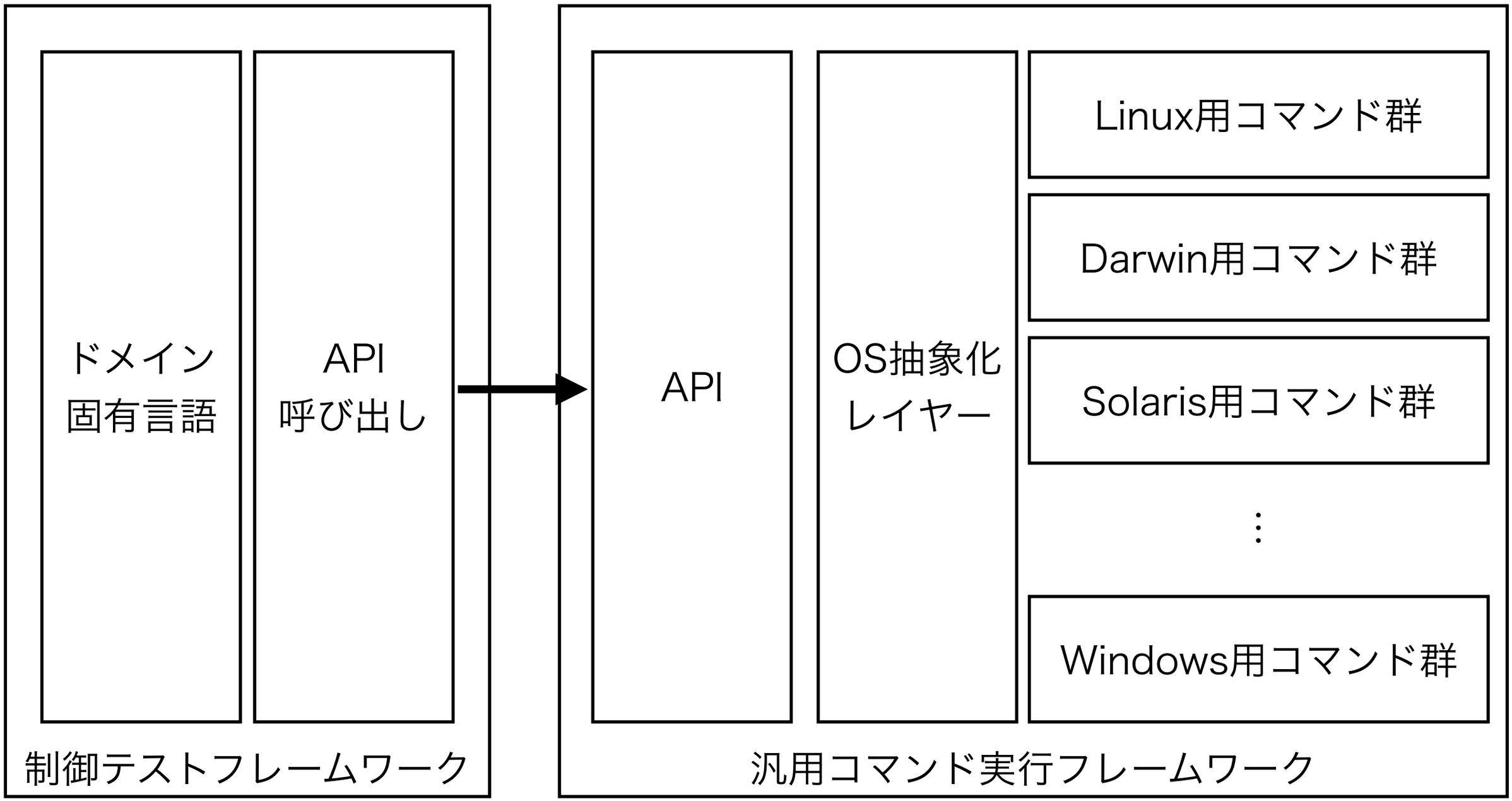
ツール名	テスト種別	構成管理ツール独立性	OS汎用性
ChefSpec	単体	×	○
rspec-puppet	単体	×	○
minitest-chef-handler	結合	×	○
Test Kitchen	結合	×	×
rspec-system	結合	○	×
Cumcuber-Chef	受入	×	○
leibniz	受入	×	○

提案するサーバ
テスト手法

提案するテスト手法

- 汎用コマンド実行フレームワーク
  - 構成管理ツール固有の振る舞いを抽出
  - 振る舞いのテストに特化したAPIを定義
- 制御テストフレームワーク
  - 宣言的かつ自然言語に近い記法で汎用コマンド実行フレー ムワークを操作するための記法の定義
  - 記法内 の各命令と実際に呼び出す汎用コマンド実行フレームワー クのAPIメソッドをひもづけ

提案手法の概要


ドメイン
固有言語
API
呼び出し
制御テストフレームワーク
API
OS抽象化
レイヤー
Linux用コマンド群
Darwin用コマンド群
Solaris用コマンド群
Windows用コマンド群
汎用コマンド実行フレームワーク

提案手法の詳細

- 制御テストフレームワーク
  - 自然言語に近い記法でテストコードが書けるRubyのRSpecを採用
  - RSpecを拡張した記法により、汎用コマンド実行フレームワークのAPIを呼び出す
    - serverspecと命名
- 汎用コマンド実行フレームワーク
  - 制御テストフレームワークから呼び出されるAPI（メソッド）を定義
  - OS抽象化レイヤーでOSやディストリビューションを判別し、適したOS用コマンドを実行
  - specinfraと命名

提案手法によるテストコード例

```
describe package("apache2") do
  it { should be_installed }
end

describe service("apache2") do
  it { should be_running }
end

describe port(80) do
  it { should be_listning }
end
```

提案手法の評価

従来手法のテストコード(1)

```
@test "The package apache2 is installed" {
  dpkg-query -f '${Status}' -W apache2 \
    | grep '^install ok installed$'
}
```

```
@test "The apache2 service is running" {
  service apache2 status
}

@test "Port 80 is listening" {
  netstat -tunl | grep ":80 "
}
```

従来手法のテストコード(2)

```
@test "The package apache2 is installed" {
  pkg list -H apache2
}

@test "The apache2 service is running" {
  svcs -l apache2 | egrep '^status *online$'
}

@test "Port 80 is listening" {
  netstat -an | grep LISTEN | grep "\.80 "
}
```

提案手法によるテストコード

```
describe package("apache2") do
  it { should be_installed }
end

describe service("apache2") do
  it { should be_running }
end

describe port(80) do
  it { should be_listning }
end
```

採用実績と課題

- 実績
  - OSSとして公開されている
  - 任天堂など、採用している企業が既に存在
  - Black Duck Open Source Rookies of the Year 2013 に選ばれた
- 課題
  - 定量的な評価ができていない

まとめ

まとめと今後の予定

- まとめ
  - 汎用コマンド実行フレームワークの定義
  - 宣言的かつ自然言語に近い記法で汎用コマンド実行フレームワークを操作できる制御テストフレームワークを定義
  - 構成管理ツール独立性とOS・ディストリビューション汎用性
- 今後の予定
  - 定量的な評価
  - より良い制御テストフレームワーク実装の模索
  - 汎用コマンド実行フレームワークのテスト以外への応用