ワークフローエンジンを用いた インシデント切り分け自動化

大阪市立大学大学院工学研究科
電子情報系専攻
瓦井 太雄

研究背景

- ネットワーク運用管理者の負担増加
  - ネットワークを構成する機器の種類が増え，ベンダーも多様化

  → 連携する情報インフラではそれぞれの専用管理ツールによる一元管理が困難
  - SDN や NFV などの仮想化技術が広く活用
    - OTT や IoT などサービスの拡大
    - 多種多様化する機器，種々のトラヒック処理に対応
    - インターネット設計の容易さと短いサービス開発サイクル

  → 専用の OS により行っていたルータ等のネットワーク機器の運用管理に対し，より広範な知識および技能が要求

前提目標：障害時間を極力短くし，影響を最小限に抑える

障害検知 → 障害切り分け → 復旧処理 → 復旧報告

研究背景

- 人を介した現在の運用管理における問題
  - システムの複雑化により，障害の切り分け調査や復旧にかかる時間が増加
    （平均切り分け時間約 70 分）
  - 処理の複雑化により人が介在するステップ数が増加し，ヒューマンエラーのリスク
  - 監視項目の増加により運用管理者の状況把握が限界
    - インシデント発生が一般利用者の通報により発覚する事故が多発
    - 障害検知から原因特定までの時間が増加

例）Microsoft Azure /ストレージと仮想マシンへの接続障害[2]

表：Google Cloud Platform の障害切り分け時間毎の件数[1]

障害切り分け時間	該当件数
～ 30[min]	35
30[min] ～ 1[h]	9
1[h] ～ 2[h]	10
2[h] ～ 4[h]	4
4[h] ～	3
不明	19
合計	80

インシデント対応の自動化が求められる

[1]:https://status.cloud.google.com/summary
[2]:https://azure.microsoft.com/ja-jp/status/history/

関連研究

- モデルベースの障害切り分け自動化

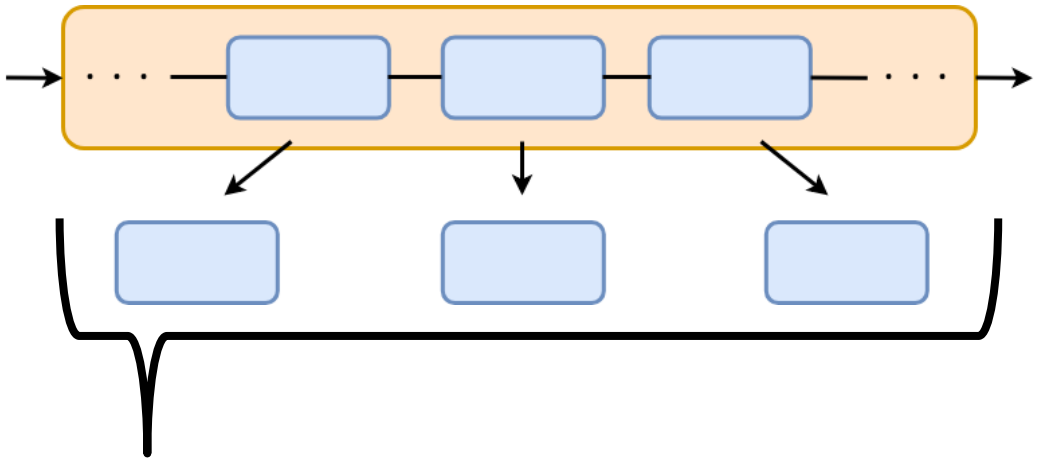
	自動化方法	課題
ワークフロー アプリケーション	事前に想定されるインシデントに対する障害切り分けのための操作を連結してワークフローを実装	インシデント全てに対し膨大な処理操作からワークフローを作成するコストの高さ
ニューラル ネットワーク	複数の変数に対し設定した閾値と入力情報を比較．どのパターンに当てはまるか判別	正常でありながら異常を示す擬似情報の曖昧さや不確実性を減らすため，指数的に増える計算量

- 新たなインシデントに対する障害切り分けフローの作成や計算処理のコストが高い

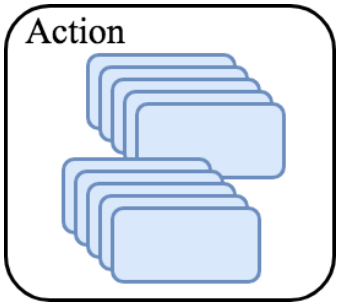
研究目的

- 課題
  - 膨大な処理を総当たりで行うことは無駄な処理時間を生む
  - 障害が発生している箇所の候補だけでは原因の特定が困難
- 目的
  - 障害切り分けを予めワークフローを作成したり，ニューラルネットワークを用いて計算しパターンに分類するモデルベースではなくインシデントの発生情報をもとに自動で原因特定のための処理順序を構成することを目標とする
- メリット
  - 人の手でされてきた障害切り分けの手順作成をアルゴリズム化することで新たなインシデントへの対応フロー作成のコストが下がる
  - 複数の障害切り分けフローの中で，共通する処理の結果を再利用することで処理の効率化

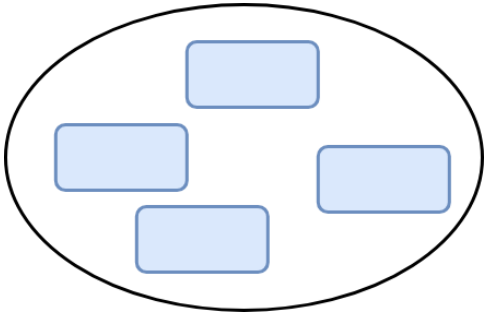
提案手法



①調査項目のまとまりごとに
分解 (モジュール化)

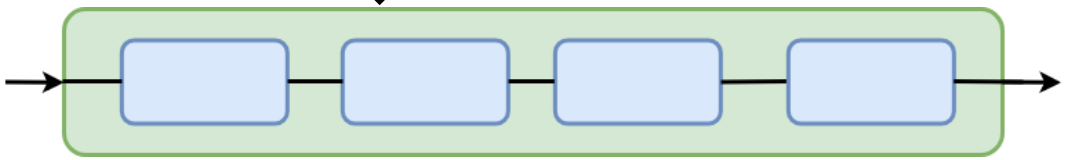


インシデント発生



③モジュールをピックアップ

②操作(モジュール)を蓄積



④フローを作成

提案手法 ~処理の選定~

- 目的：最小限の入力情報で必要十分なモジュールを選定する
- 入力：
  - 障害が発生しているサービス
  - 障害発生機器 IP
  - サーバ機器 IP
- 方法：
  - 障害が発生しているサービスの情報とモジュール同士の依存関係を使用
  - モジュール群の中からサービスレイヤよりも下位レイヤのモジュール，サービスに関係のあるモジュールをピックアップ
  - サービスに関係のあるモジュールが依存するモジュールも加える
  - サービスはプロトコルレベルで区別

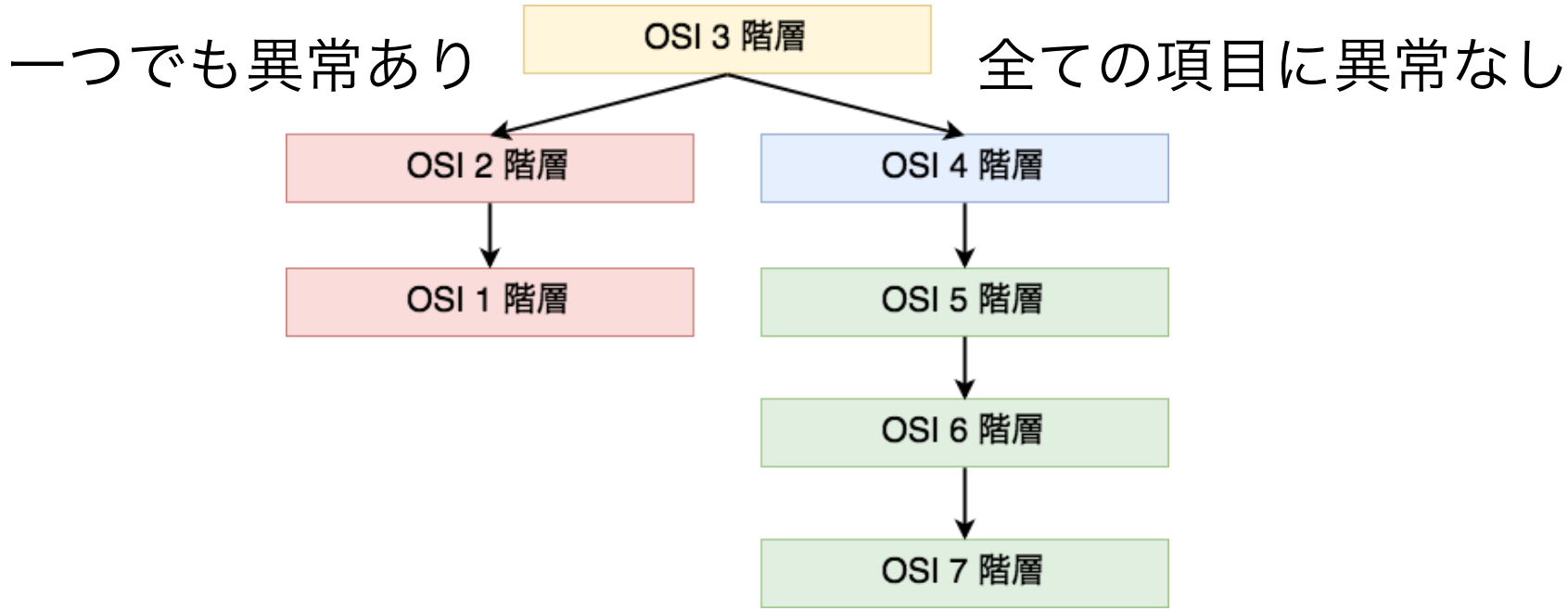
提案手法 ~処理の順序決定~

- 目的：障害切り分け（原因特定）のための処理順序を決める
- 方法：目的達成のため処理順序の決定指標は以下の通りに定め，これらに基づき決定する（上から順に優先度の高い指標とする）

1. OSI 参照モデルにおけるプロトコル階層
   ↓
2. モジュール間の依存関係
   ↓
3. 参照回数/要因回数

提案手法 ~処理の順序決定~

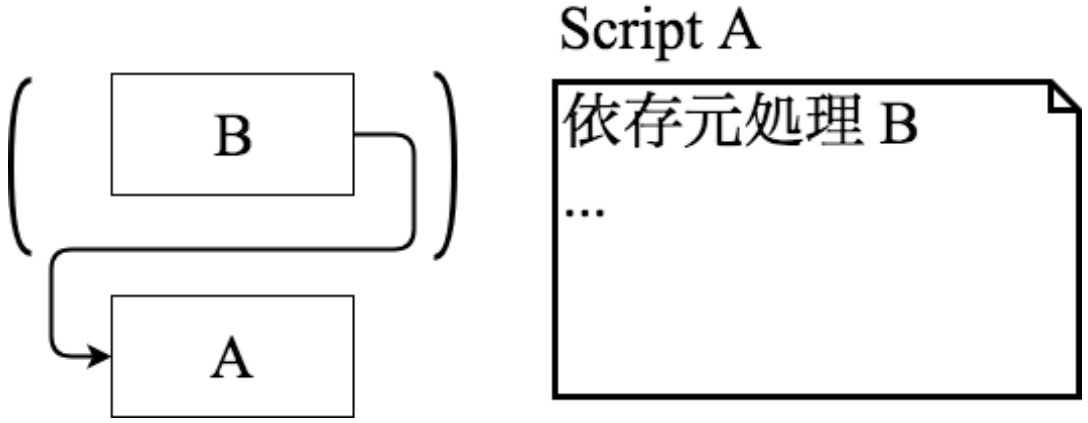
- OSI レイヤ：障害が発生しているレイヤよりも上位にはエラーが出る → 下位レイヤから調査をする
- 障害原因箇所の 7 割以上はソフトウェア（OSI 3 階層以上）の問題[3] → 3 階層を最優先に



[3]:https://www.janog.gr.jp/meeting/janog15/data/11-unyo-matsujun.pdf

提案手法 ~処理の順序決定~

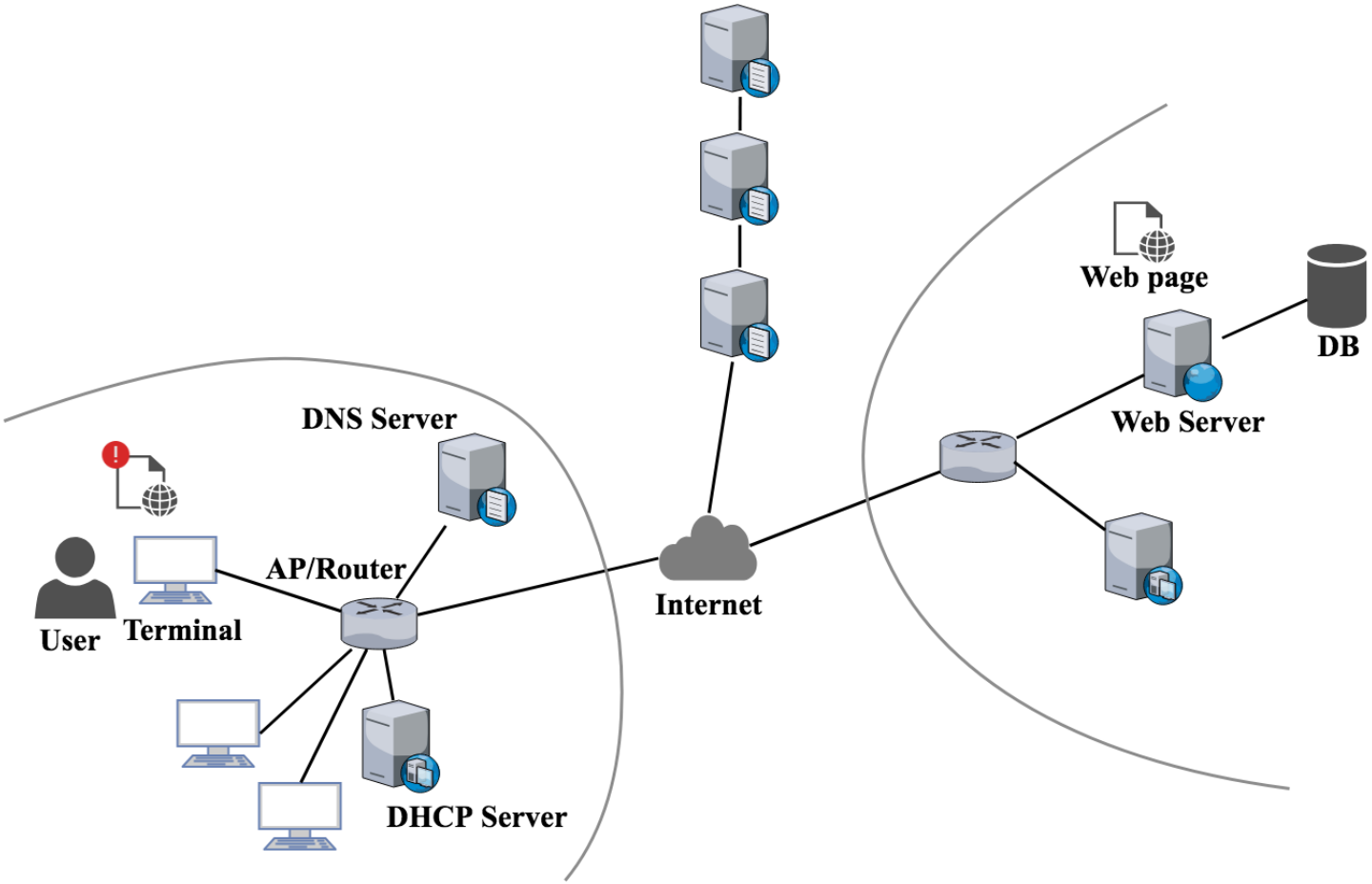
- 依存関係：処理の属性値に依存関係を含めることで切り分けに必要な処理を漏れることなくピックアップする
- 例) 処理 A が処理 B に依存（B の終了が A の開始条件）



- 参照回数/要因回数
  - 実行順序に制約が無い処理の順序決定に用いる
  - 実行後に更新することで，環境毎の障害の傾向・特徴に適応する
  - フローの実行後に各処理の参照回数/要因回数を更新する

実装

- 想定する環境は Web ページの提供

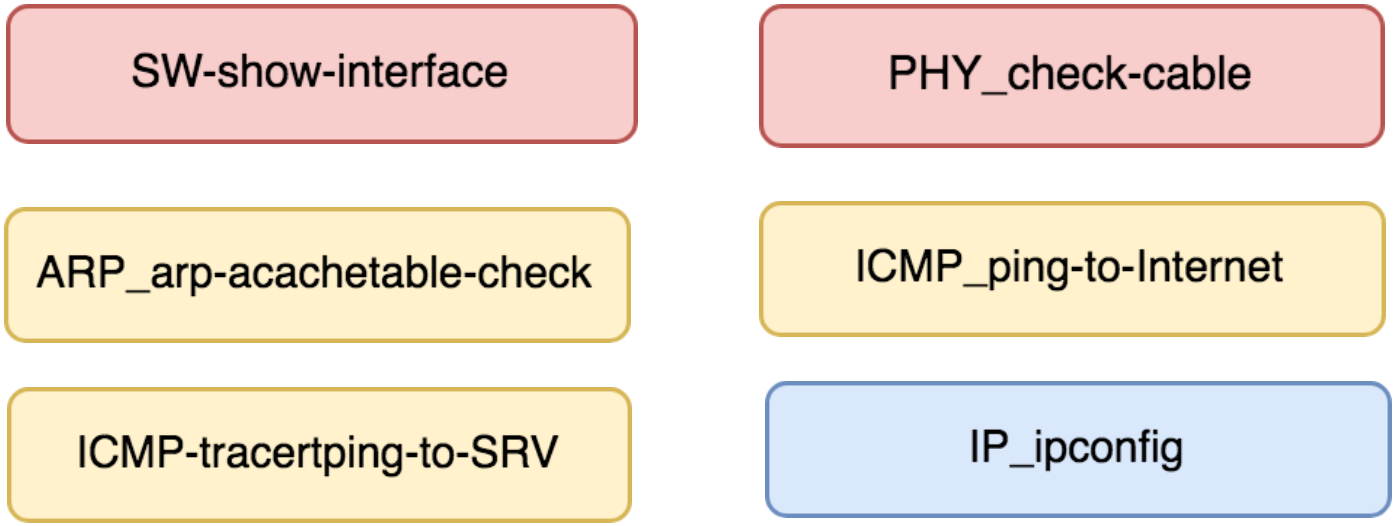


実装

モジュール名	プロトコル	依存モジュール	参照回数	要因回数	検査項目
PHY_check-cable	PHY	ー	0	0	ケーブルに異常がない
SW_show-interface	SW	ー	0	0	インターフェースに異常がない
ARP_arp-acachetable-check	ARP	ー	0	0	AP/ルータが稼働している
ICMP_ping-to-Internet	ICMP	ー	0	0	クライアントがインターネットにつながる
ICMP_tracertping-to-SRV	ICMP	ー	0	0	サーバ IP に疎通確認が取れる
IP_ipconfig	IP	ー	0	0	TCP が正常に作動している
DNS_dig-x-IPDN	DNS	ー	0	0	IP から DN が取得可能
DNS_dig-ns-NSREC	DNS	DNS_dig-x-IPDN	0	0	DN から取得する NS レコードが正常値
HTTP_curl-status-code	HTTP	DNS_dig-ns-NSREC	0	0	HTTP ステータスコードが正常

実装

- 入力プロトコル情報のレイヤ（アプリケーション層）より下位レイヤのモジュールをピックアップ



実装

- 入力プロトコル情報の調査モジュールをピックアップ

SW-show-interface

PHY_check-cable

ARP_arp-acachetable-check

ICMP_ping-to-Internet

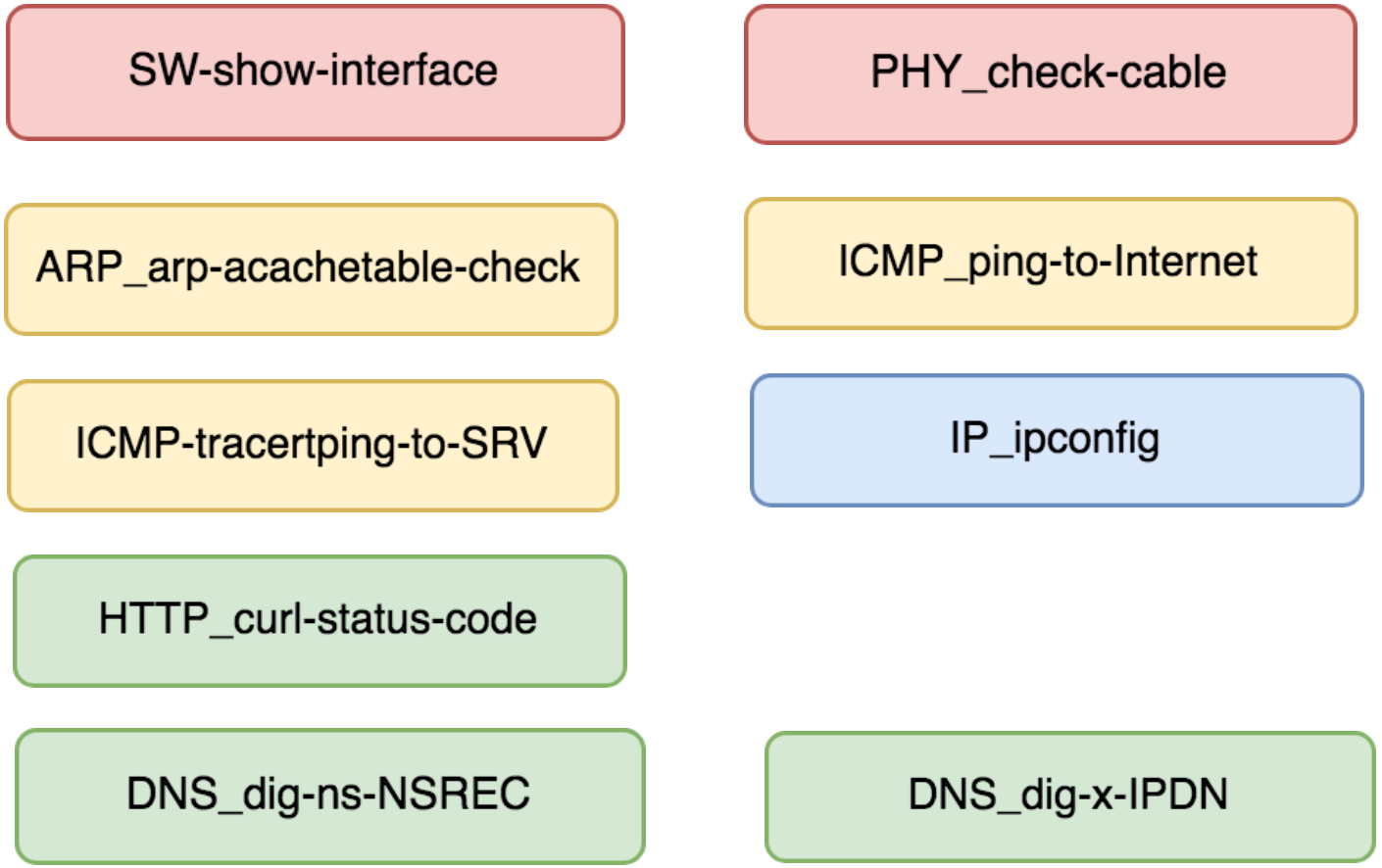
ICMP-tracertping-to-SRV

IP_ipconfig

HTTP_curl-status-code

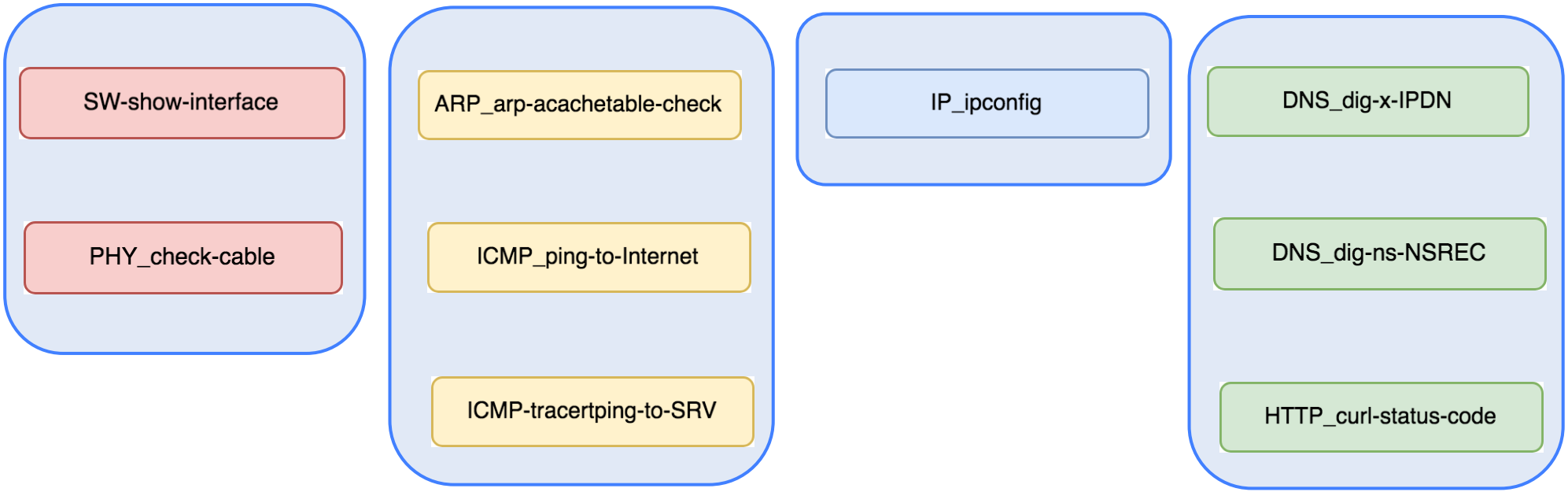
実装

- 入力プロトコル情報の調査モジュールが依存するモジュールピックアップ

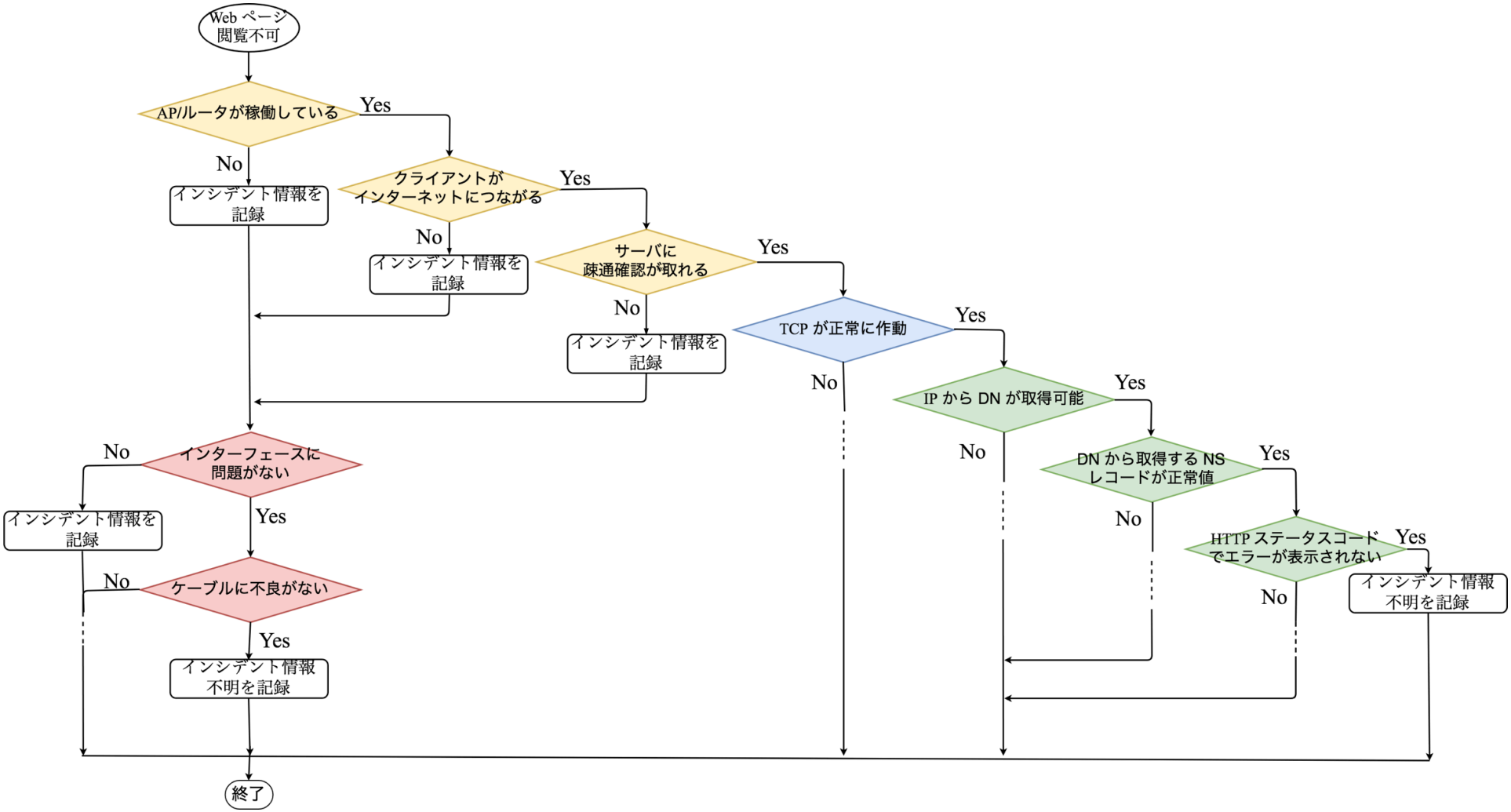


実装

- レイヤごとにモジュールの実行順序を決定

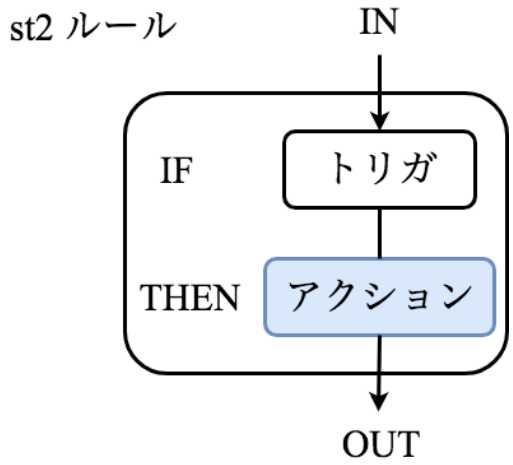


実装

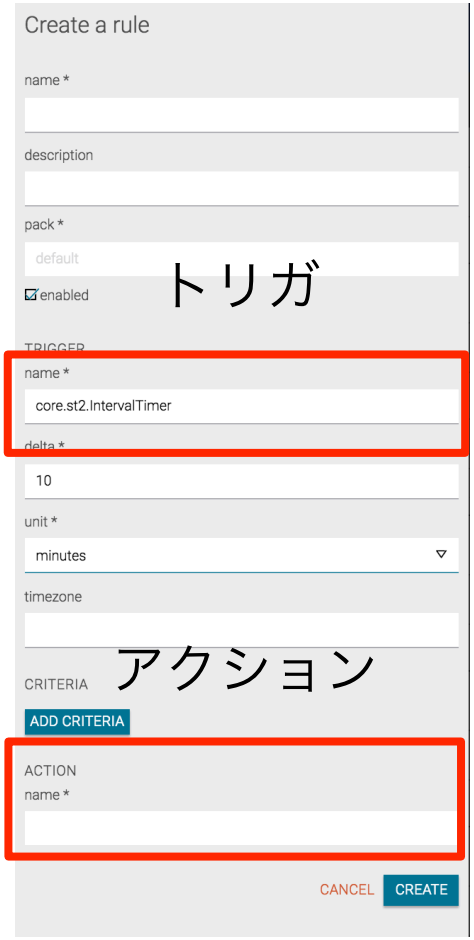

Web ページ
閲覧不可
AP/ルータが稼働している
Yes
No
インシデント情報を
記録
クライアントが
インターネットにつながる
Yes
No
インシデント情報を
記録
サーバに
疎通確認が取れる
Yes
No
インシデント情報を
記録
TCP が正常に作動
Yes
No
IP から DN が取得可能
Yes
No
DN から取得する NS
レコードが正常値
Yes
No
HTTP ステータスコード
でエラーが表示されない
Yes
No
インシデント情報
不明を記録
インターフェースに
問題がない
No
Yes
インシデント情報を
記録
ケーブルに不良がない
No
Yes
インシデント情報
不明を記録
終了

StackStorm(st2)

- ユーザや他の機器が実行した操作に応じて
  ワークフローを実行するイベント駆動型の
  ワークフローエンジン
- 最小構成はルール（1 トリガ +1 アクション）
- モジュールはアクションとして作成
- 切り分けフローは ActionChain の定義ファイル
  （YML）で作成



ルール作成画面



実装

- 仕様
  - 障害が発生しているレイヤを調べるためにプロトコルと階層番号の対応をテーブルで保持
  - モジュールが必要な属性値はプロトコル，依存処理，参照回数，要因回数．これらも DB のテーブルで保持

Table 1

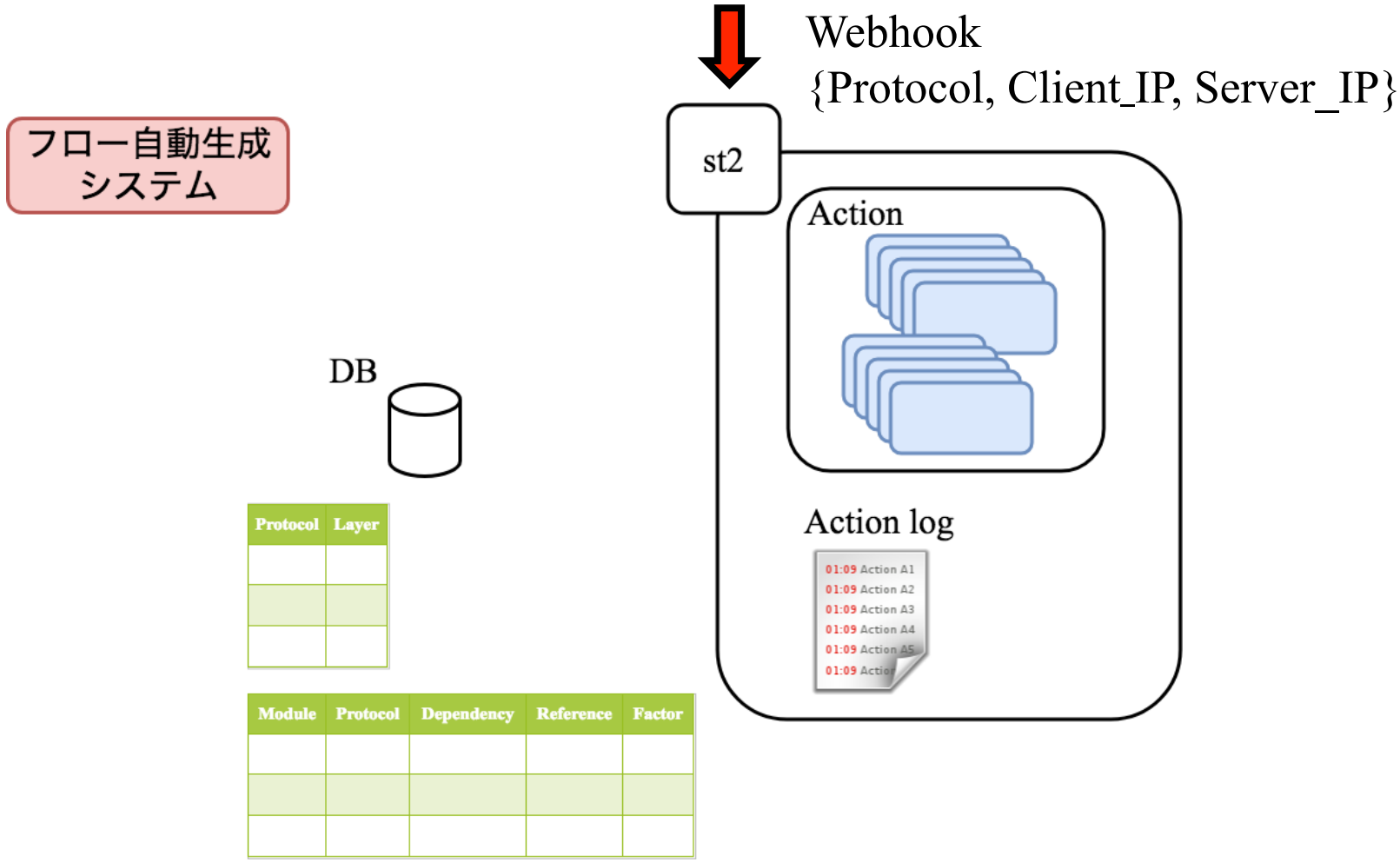
Protocol	Layer
ICMP	3
HTTP	7
⋮	⋮

Table 2

Module	Protocol	Dependency	Reference	Factor
A	DHCP		10	1
B	HTTP	A	5	2
⋮	⋮	⋮	⋮	⋮

実装

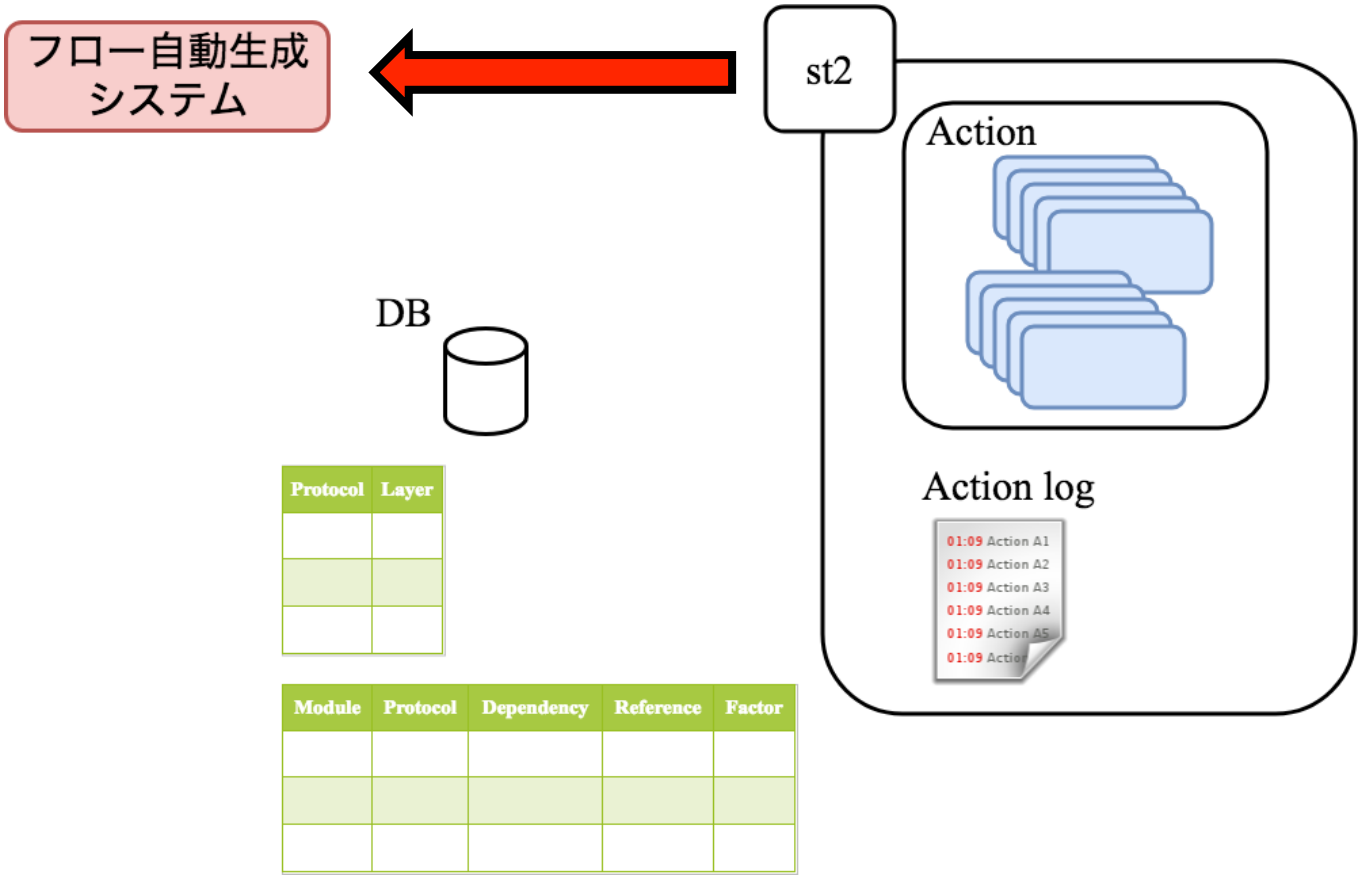
- webhook でインシデント発生情報が st2 に渡る



実装

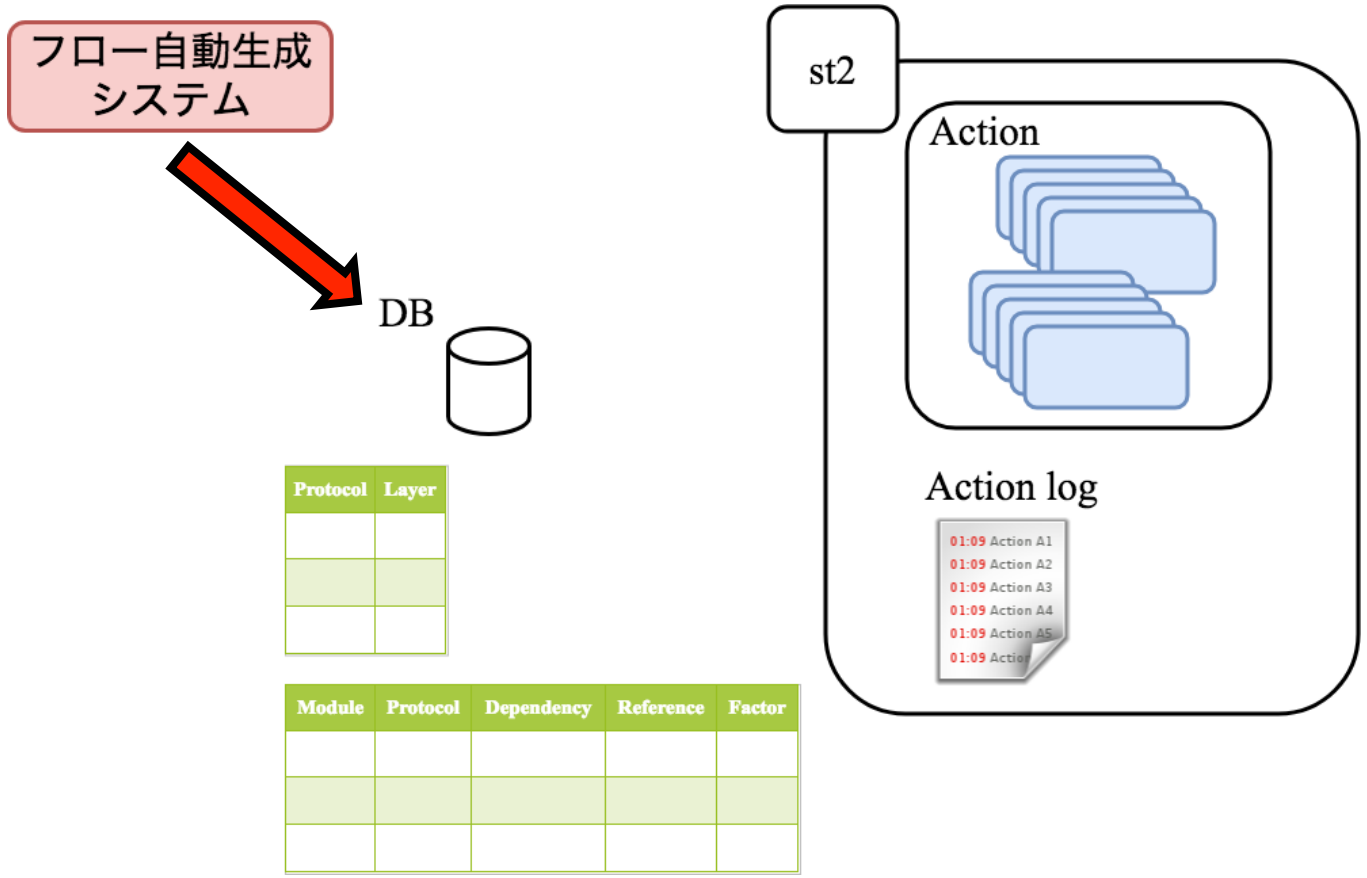
- インシデント情報とともにフロー自動生成システムを呼び出す

{Protocol, Client_IP, Server_IP}



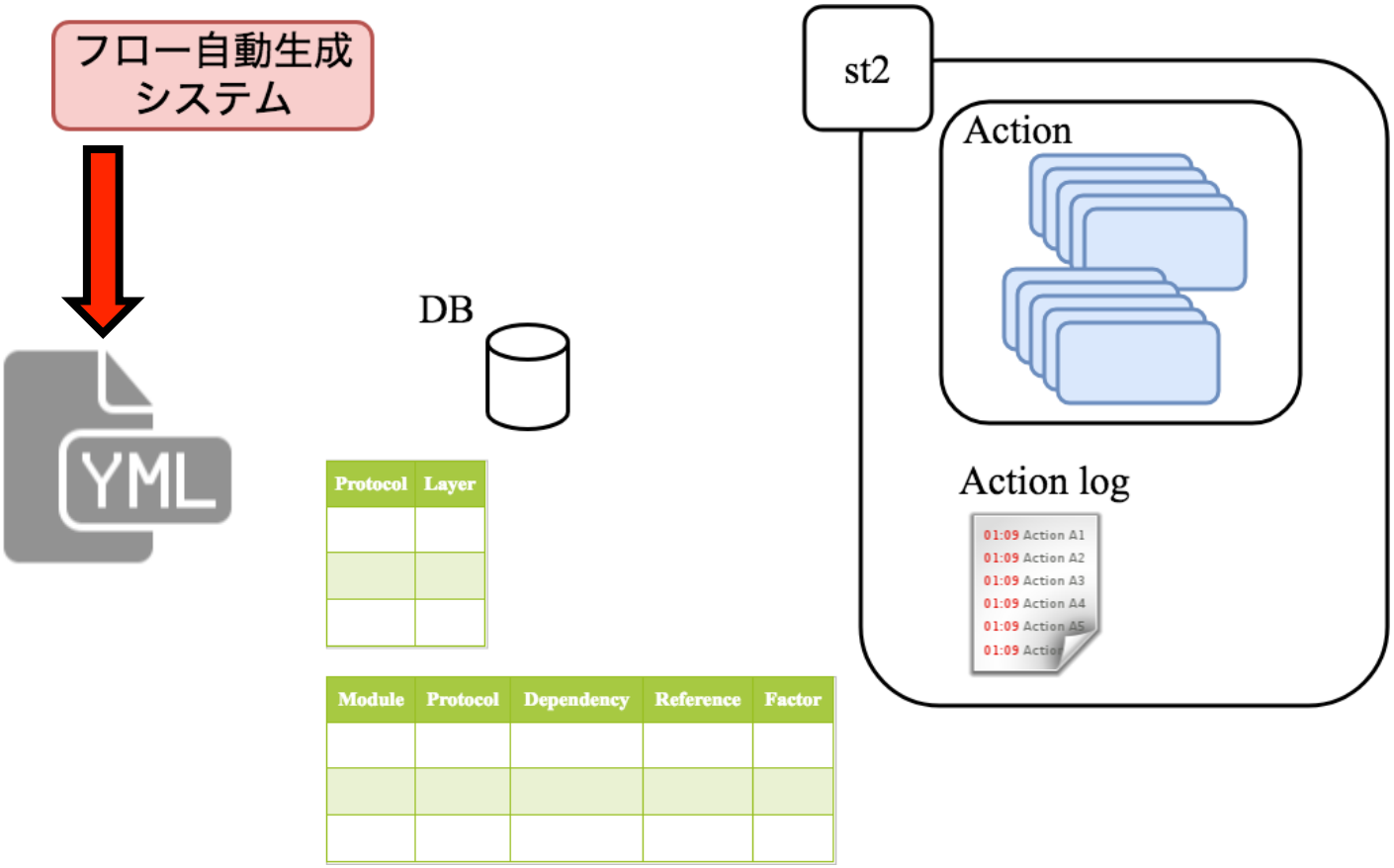
実装

- テーブルの属性値を参照し調査開始レイヤ，調査順序を決定

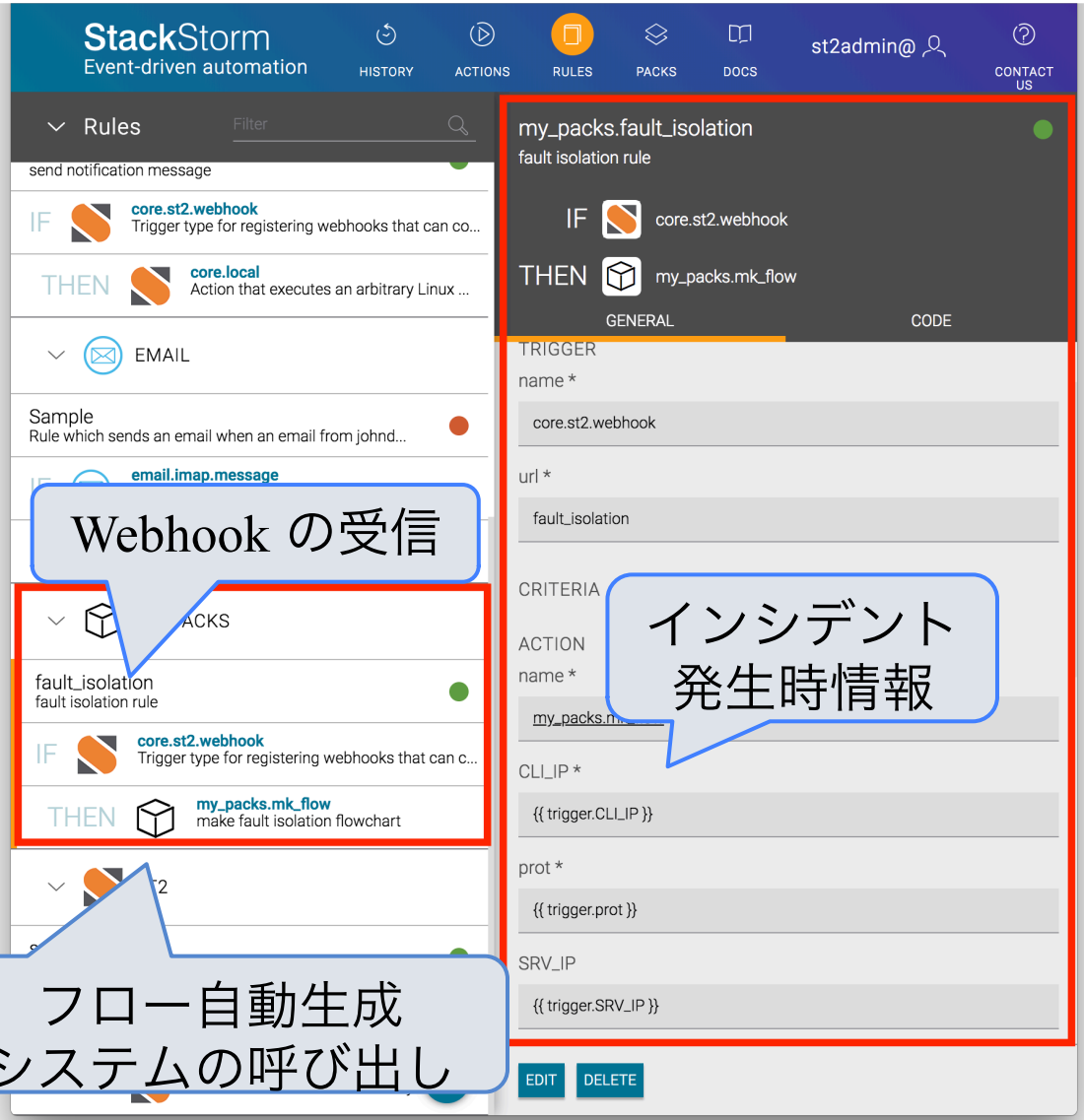


実装

- ActionChain で実行する YML ファイルを生成



結果




```
chain:
  -
    name: "DNS_dig-x-IPDN"
    ref: "my_packs.DNS_dig-x-IPDN"
    parameters:
      CLI_IP: "cli_iip"
      SRV_IP: "srv_ip"
    on-success: "DNS_dig-ns-NSREC"
    on-faulure: "to-failure"
  -
    name: "DNS_dig-ns-NSREC"
    ref: "my_packs.DNS_dig-ns-NSREC"
    parameters:
      CLI_IP: "cli_iip"
      SRV_IP: "srv_ip"
    on-success: "HTTP_curl-status-code"
    on-faulure: "to-failure"
  -
    name: "HTTP_curl-status-code"
    ref: "my_packs.HTTP_curl-status-code"
    parameters:
      CLI_IP: "cl ip"
      SRV_IP: "s  "
    on-success"  ccess"
    on-fa
  -
    name:
    ref:
  -
    name: "to-failure"
    ref: "my_packs.failure"
```

依存元モジュール
から記述

まとめ

- st2 を使った障害切り分け自動化システムを提案
- これまで課題とされてきたインシデントの多様化による自動化のコスト増加に対し，処理の階層化による分類と依存関係の考慮などにより自動で調査手順を作成

今後の課題

- 処理フローのモジュール化，複数処理フロー並列動作時におけるワークフローの最適化
- 障害範囲を活用した障害箇所切り分け

  例）特定の端末 → 端末と外部の通信部
  特定の LAN → LAN と WAN の接続部
  全ての端末 → サーバ