# 開発ツール連携のための ProxyChatBot フレームワーク

高木 豪[1]　小林 隆志[1]

**概要**：本稿では，既存の開発支援ツールを ChatOps に導入するためのモデル及びツールフレームワークを提案する．既存の開発支援ツールは出力が膨大である場合や，開発者の立場に応じて必要な出力情報が異なる場合がある．そのため，そのまま ChatOps に導入することが難しい開発支援ツールが存在する．提案モデルでは，既存の ChatOps のモデルを拡張し，ChatBot とチャットツールの間を ProxyChatBot を経由することで，開発支援ツールの出力を柔軟に加工してチャットツールに通知する仕組みを確立する．モデルに基づいて実装したツールフレームワーク ProxyChatBot を用いて，既存の静的検査ツールの除外手法を再実装する実験を通し，モデルの実現可能性及び有用性を議論する．

**キーワード**：ChatOps, ChatBot, DevOps, 開発支援ツール

## 1. はじめに

生産性と成果物の品質は開発者が常に意識するものであり，ソフトウェア開発では高品質なソフトウェアを効率良く開発するために開発支援ツールが利用される．しかし，ソフトウェアの大規模化や分散開発が進むほど開発に用いるツールの数は増加する．その結果，開発者はツール全てを把握することに時間を割き，生産性の低下につながってしまう [1]．この問題を解決するために，複雑で大規模なツールを用いた開発プロセスの一部を自動化するツールとして Bot が利用されている．一般にルールベースの Bot は繰り返し処理や事前定義されたタスクを自動化するアプリケーションとしてみなされる．ソフトウェア開発の分野では，Bot は開発者が過去に行ってきた煩雑なタスクを自動化する．煩雑なタスクを Bot が開発者に代わって行うことで，開発者は自身が行っている作業を中断せずに集中して行うことができる．

ChatOps は，Bot を用いてチャットツール上で会話主導の開発を行う方法である．チャットツールは本来開発者間のコミュニケーションツールとして利用されている．チャットツール内で Bot はユーザーの一人としてチャットに参加し，ユーザーと対話する．ユーザーとの対話を通して，Bot は開発支援ツールを実行し，結果を通知する．チャットツールは開発チームにとって擬似的な共有コンソールを実現し，開発フローの透過性を高めることができる．

ChatOps はチーム間の情報共有手段としても有用である．ChatOps では Bot が開発支援ツールから得られた情報を集約し，一つのチャット上のタイムラインに通知する．開発者はチャットツールのタイムラインを監視するだけでツールを利用することと同等の情報を取得できる．このように，ChatOps は開発を通して行う様々な情報の出力先をチャットツールのタイムラインに一元化することで，開発者の迅速な意思決定及びチーム間の情報共有を適切に行える点で非常に有用である．

しかし，現実的には ChatOps 化することが難しいツールが存在する．それは開発支援ツールの出力が膨大である場合や，開発者の立場に応じて不必要な情報を含んでいるようなケースである．ChatOps は，開発支援ツールからの情報提示の即時性及び一元化の観点で優れる一方で，多量の情報が提示される場合，開発者の利用意欲の低下や必要な情報の見逃しの原因となりうる．そのため，これらの開発支援ツールを ChatOps に導入するには開発者にとって必要なデータを抽出しなければいけない．しかし，開発支援ツールに対する運用方法は開発者やプロジェクトに強く依存するため，出力を制御する方法は一様ではない．これらのツールを ChatOps 化するためには，ツールの出力の制御をツールをまたいで柔軟に行える仕組みが必要である．

本稿では，既存の開発支援ツールの ChatOps 化を支援するモデルツールを提案する．本モデルは既存の ChatOps の

[1] 東京工業大学 情報理工学院
School of Computing, Tokyo Institute of Technology
{gtakagi@sa.,tkobaya@}cs.titech.ac.jp

モデルを拡張し，ChatBot とチャットツールの間で ProxyChatBot を仲介することで，開発支援ツールの出力にデータ加工を施してチャットツールに通知する仕組みを確立する．更に，ツールとして実際に ProxyChatBot を実装し，ProxyChatBot 上で静的検査ツールの除外手法を再実装することでモデルの実現可能性及び有用性を議論する．

## 2. ChatOps

### 2.1 DevOps と ChatOps

ソフトウェア開発を効率的に行うには，継続的かつ高速な運用改善を行ってプロダクトの成長速度を向上させることが必要である．中でも企画，運用，開発の 3 つのプロセスの垣根を取り払っていくために，ツールや組織の文化を最適化していくことを DevOps と呼ぶ．DevOps については継続的インテグレーション (CI) や，Infrastructure as Code(IaC) など開発・運用環境に関する技術的な話題 [2] が主であったが，近年では学術界においても組込みソフトウェア [3] や，医療機器 [4] の開発への適用の可能性などが議論されている．

DevOps のためには，特に開発から運用に至るプロセスを自動化し，その過程で起こったことを正確に把握することが重要である [5, 6]．そのため，開発者と運用担当者は DevOps ツールを複数同時に利用しながら開発と運用を行う．しかし，ソフトウェアの大規模化や分散化が進むに連れて用いる DevOps ツールの数も増加する．その結果，開発者と運用担当者はツール全てを把握することに時間を割き，生産性の低下につながってしまう．

この問題を解決するために，複雑で大規模な開発プロセスの一部を自動化するツールとして Bot が利用されている．ChatOps は，Bot を用いてチャットツール上で会話主導の開発を行う GitHub が提唱した運用方法である．DevOps ツールで行う冗長なタスクを自動化し，また開発者間の知識量の差や必要なコミュニケーションを埋め合わせるサポートを Bot が行う．冗長なタスクを Bot が開発者に代わって行うことで，開発者は自身が行っている作業を中断せずに集中して行うことができる．また，ここで使用される Bot は特に ChatBot と呼ばれる．

### 2.2 ChatBot に関する関連研究

既存の ChatOps における ChatBot の役割は，チャットを通してユーザーのコマンドを受け，開発支援ツールの出力をチャットに通知することである．

Storey らは，まだ新興の概念である ChatOps がソフトウェア開発に与える影響を明らかにするために，ChatOps がソフトウェア開発の生産性に与える影響を測るための枠組みを提案している [7]．彼女らの枠組みでは，生産性を効率性と有効性の観点で区別した上で，ChatOps は煩雑なタスクの自動化や情報の集約・通知を行うことで開発者が行っている作業のフローの状態を長く維持する点で効率性を向上し，開発チームの意思決定・状況認識を促進させる点で有効性を向上させることを主張している．

Calefato らは，分散開発における開発支援ツールの多様化や情報の断片化の処理に役立つ開発者全体の状況認識を向上させるためのツールの統合モデルを提案している [8]．提案している統合モデルにおいても，ChatOps は開発全体を通して開発と運用に共有責任が生まれてチーム全体の意識を向上させることに役立つと主張されており，対話主導型の開発を行うことを推奨している．

Lin らは，Slack と他のツールと Slack を連携する Bot がソフトウェア開発に及ぼす影響を理解するために，開発チームがどのように Slack を活用するのか予備的な調査を行っている [9]．Slack とはチーム向けのコミュニケーションプラットフォームとしてソフトウェア開発チームが幅広く採用しているチャットツールである．開発者にとって，Slack が個人及びチーム全体で利用法がそれぞれあることを理解し，その中で Bot を用いてコードホスティング，アプリケーションのモニタリング，デプロイステータス，VCS や issue トラッキングを自動化することで，開発とデプロイのサポートに利用していることを確認した．

ChatBot はチャットツールにユーザーの一員として参加し，開発者と対話する．ChatBot を用いることで，開発者を監視し，開発者の状態に応じた支援を行うことが可能となっていることが大きな特徴として言える．

Hannebauer らは，FLOSS プロジェクト (Free Libre Open Source Software) のコードレビューに対する適切なレビュワー (査読者) の推薦を行う 8 種のアルゴリズムに対して実証比較を行っている [10]．推薦に用いる既存のアルゴリズムは，過去に行ったレビューの履歴を解析する手法とプロジェクトのソースコードの変更履歴を VCS(Version Controll System) から取得して解析する手法の二種に大別されている．しかし，プロジェクトにおいて有能な開発者にコードのレビューを任せてしまうと，その作業に時間を取られて有能な開発者の生産力を下げてしまう．そのため，実際には必ずしも有能な人にレビュアーを推薦するのではなく，その人の抱えているタスクなどの状態を考慮したレビューを行う必要があることを議論している．この推薦を行う Bot も ChatBot として導入することで，ユーザーの状態を常時監視しその時点に応じた適切なレビュワーを推薦することが可能となる．

このように，DevOps や ChatOps に向けた ChatBot に関する研究はいくつか存在するが，著者らが知る限り，ChatBot から流れてくるデータを加工する役割を持った ChatBot は存在していない．

既存の ChatOps のモデルは，マイクロサービスアーキテクチャに基づいてサービスとして提供されている個々の支援ツールとのやり取りを Bot が代行して行うことで自動

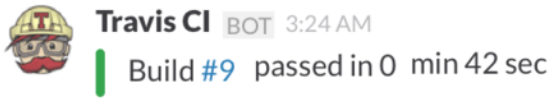


図 1　TravisCI の ChatOps 例

化等の支援を行う．サービスとして開発支援手法や支援手法の基盤となる情報を提供する研究にまつ本らが提唱する Service-Oriented MSR(SO-MSR) [11] がある．

まつ本らは，Mining Software Repositories(MSR) と総称される，ソフトウェアの開発履歴等が蓄えられたリポジトリをマイニングする技術を，ネットワーク上のサービスとして実現させるためのフレームワークとして SO-MSR を提案している．更に MSR の代表的な手法の一つであるソースコードメトリクスの計算に着目し，SO-MSR のフレームワークに従った Web サービス，MetricsWebAPI の開発 [12] や MSR サービスのインタラクション改善を目的として MSR キャッシュ機構を導入している [13]．サービス指向アーキテクチャ (SOA) の考えに従い MSR をネットワーク上のサービスにすることで，既存の MSR を用いた開発支援ツールのサービス化を推進することができる．

このように，開発支援ツールだけでなく，その基盤技術に関するサービス化は提案されてきたが，現時点ではそれらを用いた ChatOps に至る研究は著者らが知る限り存在していない．

### 2.3　開発支援ツールの ChatOps 化の課題

前節で述べた通り, DevOps/ChatOps においては自動化だけでなく自動化された処理の状況をできるだけ把握することが重要である [5]．ChatOps は開発支援ツールからの情報提示の即時性及び一元化の観点で優れる一方で，多くの警告やエラー報告などの処理すべき情報が提示される場合，開発者の利用意欲の低下や必要な情報の見逃しの原因につながる．そのため，出力が膨大である場合や，開発者の立場に応じて不必要な情報が多い開発支援ツールはそのままでは ChatOps に導入することが困難である．

出力が膨大となるツールの例として TravisCI といった CI(Continuous Integration) ツールがあげられる．TravisCI は GitHub と連動し，指定したリポジトリ上にあるコードを自動的に取得してビルドやテストを実行できるサービスである．本来ツールはテストの実行結果を出力するページを用意しているが，公式で配布してる Slack と連携する ChatBot は図 1 のように，テストの可否のみをチャットツールに通知し，詳しい詳細はページの URL をリンクする形となっている．簡易的にチャットのタイムラインを埋め尽くさないように対応しているが，これは ChatOps として不完全である．詳細を確認する際に結局ツールを開く必要があり，ツールからの情報提示の即時性及び一元化の観点で優れている状態となっていない．

開発支援ツールに対する運用方法は開発者やプロジェクトに強く依存し，必ずしも一様なものではない．ChatOps では複数の開発支援ツールの処理結果を，多様なステークホルダが受け取る．この際に，受け取るステークホルダによって必要な情報は異なるため，柔軟な出力の制御が必要となる．プロジェクトによっては複数のツールの出力に応じて出力の制御を変化させるケースも存在する．複数のツールの出力をまとめて，データの加工や出力の制御を行えるシステムが必要である．

既存の ChatBot 上でツールからの出力を加工する方法では，導入の際に ChatBot を理解し改変するといったコストが発生する．また，出力部を加工する方法が異なる複数の ChatBot を構成することは開発支援ツールと連携した ChatBot としての一般性を損なうことになる．このため，既存の ChatBot の構成を変えずに開発者やプロジェクトに合わせてデータの加工を行う手段を用意する必要がある．

## 3.　提案手法

### 3.1　提案モデル

開発者やプロジェクトに依存した開発支援ツールの出力を柔軟に調節するためには，以下の機能が必要である．

- 開発支援ツールの出力から必要な情報を抽出
- プロジェクト，ユーザーの状態に応じた出力の制御
- 複数の開発支援ツールの出力を受け取ってデータ加工

本研究では，これらを実現するための新たな ChatBot を定義する．

従来の ChatOps モデルと提案モデルを図 2 に示す．マイクロサービスアーキテクチャに沿って既存の ChatOps モデルを考えると左の図のようになる．既存の ChatOps では ChatBot がサービスとして提供されている開発支援ツールとチャットツールの双方に対して双方向に通信する．ChatBot はチャットツールから送られたコマンドを受けて開発支援ツールと通信してツールを実行し，ツールから得られた出力をチャットツールに通知する．

提案モデルでは，ChatBot とチャットツールとの通信を仲介する別の ChatBot を導入する．本研究では，この通信を仲介する ChatBot を ProxyChatBot と呼ぶ．ProxyChatBot は開発支援ツールと接続した ChatBot とチャットツールとの通信を仲介しデータ加工を行う．この提案モデルでは，ProxyChatBot 内でデータ加工を行うため，既存の ChatBot の構成を変更する必要がない．既存の ChatOps モデルとの切り替えを容易に行えるため導入コストを低くできる．

開発と運用チームの双方で意思決定・状況認識の効率を向上させるためには，利用者の特性と状況に合った情報の選別が必要となる．ProxyChatBot を開発者やプロジェクトに応じて変化させることで，開発者の部署やプロジェク

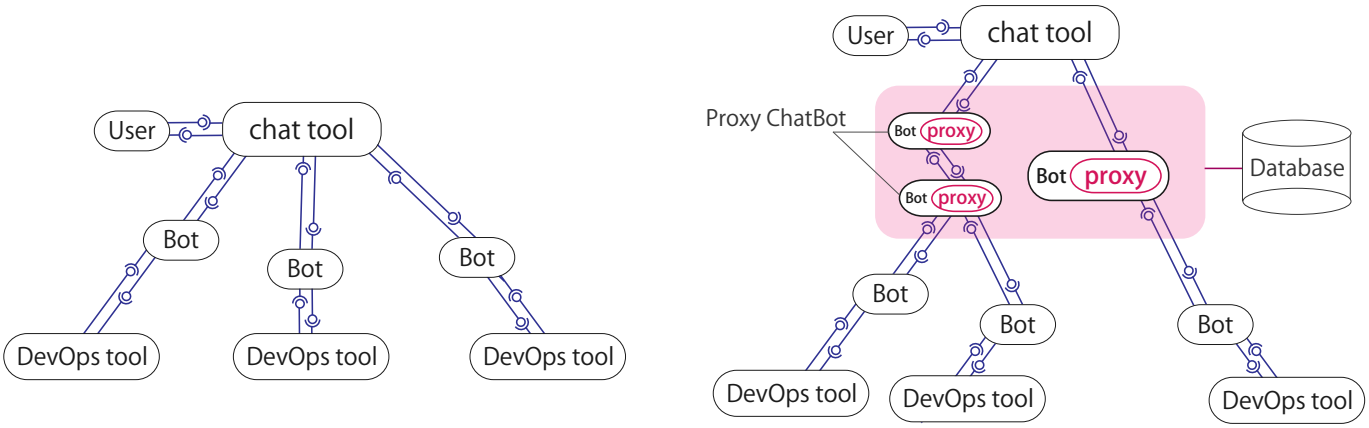


図 2　従来の ChatOps モデルと ProxyChatBot を介した ChatOps モデル

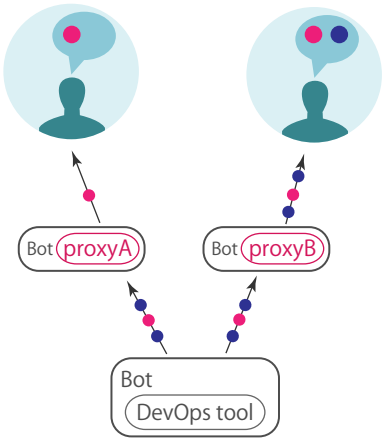


図 3　ユーザーに応じた出力の実現

トに応じた柔軟なデータ抽出を可能とする．例えば，図 3 のように構成することで ProxyChatBot 内で受け取るユーザーに合わせたデータ加工を行うことができる．

ProxyChatBot は ChatBot として振る舞うため，既存の ChatOps と差異なく ProxyChatBot を仲介した ChatOps を利用することができる．さらに図 2 の右図に示すように，ProxyChatBot は ProxyChatBot 同士で通信を行って，複数の ProxyChatBot を仲介してチャットツールに通知することも可能である．シンプルなデータ加工を行う ProxyChatBot を繰り返し仲介することで大きなデータ加工を実現できる．出力が大きいツールと連携した ChatBot も ProxyChatBot を仲介させることで，必要な情報のみを抽出する最適化した出力を行える．

ProxyChatBot は開発者の状態を考慮した通知システムも実現する．既存の ChatBot から得られた出力やチャットツールを通して開発者の状態を ProxyChatBot 内に保持することで，開発者にとって最適なタイミングで通知を行う．開発者はチャットの通知とその対応で阻害されることなく作業を継続できる．

ProxyChatBot を利用することで，今まで ChatOps に導入することができなかった開発支援ツールを導入することができるようになる．これは開発者が常に把握しなければいけないツールの数を減らすことを意味し，開発者が現在行っている作業のフローの状態を長く維持することに貢献する．同様に，ProxyChatBot が開発者にとって最適な時間に通知のタイミングを調整することも，フローの状態を長く維持することに大きく貢献する．加えて ChatOps に導入した開発支援ツールの出力をより最適な形に加工することで，情報が流れるチャットツールのタイムラインを最適化し，開発者の意思決定・状況認識の向上を促進する．ProxyChatBot は Storey ら [7] が主張した，効率性と有効性という 2 点から開発者の生産性を高める ChatOps の効能をより向上させることができる．

### 3.2　Chat as Stream

前述したとおり，提案モデルでは，ProxyChatBot がデータの抽出・利用者の目的や状態に合わせた通知内容とタイミングの制御・複数ツールをまたいだデータ加工といった処理を行う．本研究では，既存の ChatBot やチャットツール間でやり取りしていたデータ (チャット) をストリームとして捉え，ストリームに対する処理としてこれらを実現する．

開発支援ツールから得られる出力はツールの実行中の経過や警告と実行結果を通知するものであり，ツールごとに特定の単位で区切ることができるメッセージの列である．ProxyChatBot では，非同期に届く任意長のメッセージ列に対して処理を行うことが必要となる．そのため，ChatBot やチャットツールから受け取る情報をメッセージを単位としたストリームに変換する．

開発支援ツールの出力を，利用者の特徴やプロジェクトの状況に合わせて通知を柔軟に調節するためには，チャットツールから得られる利用者の状態を受けて通知を制御できる機構がストリーム処理に必要である．ツールによっては大量のデータを通知する場合があり，ProxyChatBot の処理能力を超えた場合にも対応できなければならない．

また，一つのツールから送られる情報をプロジェクトごとに適した形で加工して通知するためにストリームの分岐を行うことや，更に複数のツールの出力をまとめてデータの加工を行うためには，図 2 に示すようそれぞれのツールの出力が流れるストリーム同士を合成することが容易にできる必要もある．

上記の実現のために本研究では，ストリームに対する処理に関数型リアクティブプログラミングの要領でフィルタリングや型変換，集約などの高階関数を適用する方法を採用する．リアクティブプログラミング [14] とは，通知されてくるデータを受け取るたびに関連したプログラムがリアクションして処理するというプログラミングスタイルであり，イベントなどを扱う非同期の処理に適している．この方法を採用することで，ストリーム中のメッセージに対して，単純化した宣言的な処理を段階的に記述することで加工の過程を簡潔に記述できる．

また, 非同期にこれらを処理するために Reactive Streams 仕様に準拠した処理方法を採用する．Reactive Streams[*1] はストリームに対する非同期処理を扱うための共通仕様である．ストリーム処理では受信側の処理能力を超えるデータを送信し続けると，キャパシティを超えオーバーフローが発生してしまう．これを回避するために受信側の処理能力が低い場合は送信側が手加減してデータを送信する必要があるが，送信側が受信側に影響されて処理を行わなければいけないため，システムの疎結合性を下げてしまう．Reactive Streams は Observer パターンを利用することで，受信側が必要なときにだけデータを送信側に要求する．受信側は自律的にストリームに流れる流量を調整してオーバーフローを回避する．Reactive Streams 仕様に従ってストリーム処理を実装することでより可用性の高いストリーム処理を実現できる．

## 4. ProxyChatBot フレームワークの実装

提案モデルの実現を支援するために，本研究では ProxyChatBot フレームワークを実装する．開発者が ChatBot に対してコマンドを送るためのユーザーインターフェースにあたるチャットツールに Slack を使用する．元とする ChatBot には Github が提供している Hubot を用いる．現段階の実装では ChatBot は Hubot，チャットツールは Slack に限定している．しかし本モデルはこれらに限らず既存の開発支援ツールと連携してチャットツールに通知する ChatBot に対して適用可能である．

Hubot は GitHub 社が提供している Node.js 上で動作する ChatBot フレームワークである．ChatBot はユーザーの一員としてチャットツールに常駐して，チャットを通してユーザーからのコマンドを待ち受け，やり取りする

[*1] http://www.reactive-streams.org/

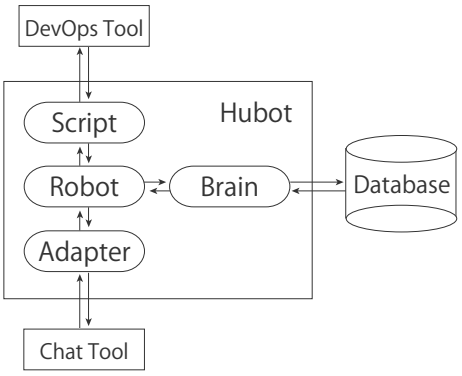


**図 4** 通常の Hubot を用いた ChatOps の実装

機構が必要である．Hubot はそれらの機構を持っており，Script モジュールに通常 CoffeeScript で処理を記述し，簡易に拡張することができる．CoffeeScript はコンパイルすると Javascript コードに変換できるシンタックスシュガーである．チャットツールとの通信を Adapter モジュールに分割することで，本体部分のコードを変更せずにさまざまなチャットツールに対応することができる．

Hubot での処理の流れを以下に示す．

(1) チャットツールからコマンドを Adapter モジュールが受け取る
(2) コアである Robot モジュールが対応するコマンドを Script モジュールから実行する
(3) 実行結果を Adapter モジュールを通してチャットツールに通知する

このように Adapter モジュールと Script モジュールを切り離すことで，ChatBot を作る開発者は接続するチャットツールを意識せずに ChatBot の振る舞いを記述できる．Brain モジュールでは各種データストアと接続してデータの永続化を行う．hubot の標準の generator では redis と接続しており，オンメモリで ChatBot に簡易的な状態を持たせておくことができる．

今回実装した ProxyChatBot フレームワークも Hubot と同じ感覚で記述できるように以下の形で拡張する．

(1) ChatBot からコマンドを Adapter モジュールが受け取る
(2) コアである Robot モジュールが Script モジュールに記述された抽出処理を実行する
(3) 実行結果を Adapter モジュールを通してチャットツールに通知する

ChatBot は Web サーバーと異なり，能動的に発話するため対話対象を指定しなければいけない．図 4 の通り，通常 Hubot の場合は Adapter クラスを継承した各チャットツールの Adapter を実装してチャットツールと接続する．Adapter モジュールの内部は Coffeescript で Adapter クラスで実装されている．Adapter というインターフェースを通してチャットツールに依らない汎用的な応答処理を

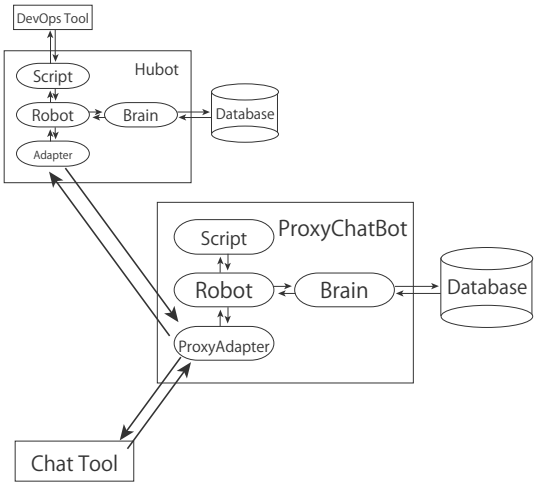


**図 5** Hubot Adapter を拡張した ProxyChatBot フレームワーク

Hubot 上で記述することを可能にしている．提案モデルの実装にあたって，Hubot の Adapter を拡張した hubot-proxy Adapter を実装した[*2]．hubot-proxy Adapter が ChatBot とチャットツール間の通信を仲介することで ProxyChatBot を実現する．この Adapter は Hubot 2.0 以上で動作確認している．

Hubot 上で hubot-proxy Adapter を用いて実装したモデルツールが図 5 である．ProxyChatBot は発話対象をチャットツール，ChatBot と少なくとも 2 つ以上持つ．そのため，ProxyChatBot の Adapter は一般的な通常の ChatBot の Adapter と異なり，複数の発話対象を考慮しなければいけない．

- チャットツールからの出力に対して任意のデータ加工を行い ChatBot へ通知
- ChatBot からの出力に対して任意のデータ加工を行いチャットツールへ通知
- ChatBot からの出力に対して任意のデータ加工を行い ChatBot へ通知

Hubot の Adapter ではチャットツールからの出力を待ち受けるエントリーポイントを定義し，Robot モジュールの Receive 関数に出力を渡すことで Robot が Script モジュールに出力を渡す．したがって，チャットツールからの出力，ChatBot からの出力をそれぞれ待ち受けられる形にするにはこれらを拡張すればよい．しかし，今回は最小限の ChatBot の構成変更に留めるために，Adapter のみを拡張する形としている．その結果，実装では ChatBot からチャットツールへの出力一方向のデータ加工に限定し，Adapter モジュールを通して Script モジュールで開発支援ツールからの出力を受けとり，一定の規則で区切られたメッセージが流れるストリームに変換してデータ加工を行う．Hubot でチャットツールからの出力を受けとり，実行する処理を記述することと同じ形で ProxyChatBot の抽出処理も記述することができる．このように実装することで，既存の ChatOps の形から ProxyChatBot を導入するコストを低くする．

前述したとおり，ChatBot も能動的に発話するため，Adapter を通して対話対象を指定しなければいけない．実装では，ProxyChatBot に対して発話するための Adapter も用意する．

3.2 で議論したように，ProxyChatBot 内での柔軟なデータ加工，複数の ChatBot との接続を行うために，ProxyChatBot 内では入力データをメッセージのストリームに変換し，ストリーム中の個々のメッセージに対して処理を行う．実装では ReactiveX[*3]を使用してストリーム処理を実現した．. ReactiveX は Java や Javascript など様々な言語でリアクティブプログラミングを行うための Reavtive Streams 仕様に準拠したライブラリである．今回は Javascript で実装された RxJS を用いた．ReactiveX は厳密には関数型リアクティブプログラミングではないが，関数型プログラミングの影響を受けており引数に関数型インターフェースを受け取るメソッドが多く用意されている．これを利用することで，開発支援ツールから流れてくるデータに対して関数型インターフェースを受け取るメソッドを組み合わせて加工することができる．また，ReactiveX は複数のストリームに対する処理も柔軟に記述できるため，複数の ChatBot と接続してデータ加工を行うことも可能である．

## 5. 評価

提案モデル及びモデルに基づいたフレームワークの評価を行うために，既存の静的検査ツールの出力を制御する渥美らのツール MAFP [15] を ProxyChatBot 上で再実装した[*4]．

### 5.1 MAFP とその実現方法

MAFP は桑原らによるソースコードの検査における警告の版間追跡手法 [16] を実現したものである．MAFP は静的検査ツールが報告する警告を版間追跡し，過去の検査において開発者が確認した警告と同じ警告を将来の検査において報告しないように制御するツールである．警告内容を確認済みとして記録する際に，版間履歴として Git の blame 機能を用いて，報告された警告内容，該当する行が変更された直近のコミット，その時のファイル名および行番号も合わせて記録する．あるコミットに対する静的検査警告について，警告内容及び報告された行が変更された直近のコミット，ファイル名，行番号の全てが等しい警告が確認済みと記録されていれば除外する．

今回は Java 言語を対象として，静的検査ツールには

[*2] https://github.com/shimastripe/hubot-proxy にて公開

[*3] http://reactivex.io/

[*4] https://github.com/shimastripe/mafp-proxy にて公開

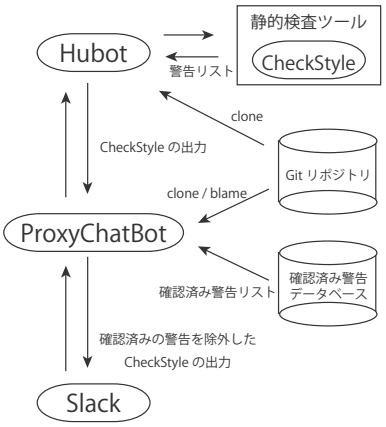


**図 6** 実装構成

Checkstyle を利用する．Checkstyle は Java のソースコードを静的に解析して，指定した規約に沿っているか判定するツールである．評価の際 Checkstyle による検査を行う Chatbot は，Hubot 上で Checkstyle を実行する子プロセスを生成し，その出力を受け取って利用する方法で実装した．Checkstyle の設定にはパッケージに標準で含まれている google_checks.xml を利用している．開発者が確認した警告の記録には mongoDB を利用する．Git のリポジトリ取得には，Node.js 向けの Git リポジトリ操作 API である NodeGit を用いた．

### 5.2 Checkstyle と連携した ChatBot

Checkstyle を ChatOps 化するにあたって，現状 Checkstyle はサービスとして存在しない．そのため，本実装では NodeGit を用いてリポジトリを clone し，その上で clone したリポジトリに対して，hubot-script 上で Node.js の子プロセスとして Checkstyle を実行し，得られた検査結果を発話する．この hubot-script は coffeescript で記述し，72 行で記述することができた．

### 5.3 MAFP を再実装した ProxyChatBot

ProxyChatBot では ChatBot から送られてきた Checkstyle の検査結果を hubot-script 上で ReactiveX を用いてストリームに変換し，データ加工を行う．Hubot にはデータを永続化する機能として Brain というモジュールが存在するが，今回は Brain を用いずに hubot-script 上で直接 mongoDB と接続している．ProxyChatBot 側でも対象のリポジトリを clone し，Checkstyle が警告している行に対して git blame を実行して版間履歴を取得する．これらを用いて，対象の警告が確認済み警告リストに登録済みでない警告のみを集約し，チャットツールへと応答する．この hubot-script は coffeescript で記述し，145 行で記述することができた．

図 6 が実際の実装の構成である．ProxyChatBot 上では，事前にデータベースに確認済みの警告を登録した上で，ユーザーへ Checkstyle の結果を送信するときに該当する警告は除外して通知するように実装した．

実装の結果，図 7 のように確かに確認済みの警告は出現されないことを確認した．ChatBot とチャットツールの間で ProxyChatBot を仲介することで，MAFP を用いて不要な情報をフィルタリングした Bot を利用できるようになった．既存の ChatOps の構成から ChatBot の Adapter のみを変更することで，元の構成をほとんど変更せずに少ない労力で実現できたことをもって，提案手法の実現可能性と有用性を示せたと考える．

## 6. おわりに

本稿では，開発者やプロジェクトに依存した開発支援ツールの出力を柔軟に調節を実現するために支援するモデル及びモデルに基づいたフレームワークを提案した．既存の開発支援ツールの出力は膨大である場合や，開発者の立場に応じて不必要な情報を含んでいるため，現状 ChatOps に導入することができない開発支援ツールが存在する．本モデルは既存の ChatOps のモデルを拡張し，ProxyChatBot を仲介することで既存の開発支援ツールと連携した ChatBot が送る出力にデータ加工をしてチャットツールに通知する仕組みを確立した．更にツールとして Hubot とチャットツール Slack の間で仲介する ProxyChatBot，及び Hubot の接続 Adapter を実装し，実際に ProxyChatBot 上で渥美らのツール MAFP を再実装することでモデルの実現可能性及び有用性を議論した．

今後の課題としては，提案モデルの実装とそれらの詳細な評価が必要である．本研究ではユーザーの状態に合わせて通知のタイミングを調整するような ChatBot の実装を行っていない．提案モデルに対する有用性を評価するためにこれらを実装し評価する必要がある．

また，実装したフレームワークは Slack のみに対応しているため，様々なチャットツールに対応する実装だけでなく，複数の ProxyChatBot を仲介するための ProxyChatBot 同士を接続する Hubot の Adapter も実装しなければいけない．

提案手法は静的検査ツールに限らず様々な開発支援ツールに用いることができる．様々な開発支援ツールを ChatBot 化し，既存の ChatOps ツールなどと合わせて ProxyChatBot を用いた ChatOps 環境を構築することで，提案モデルによる ChatOps 環境の得失を議論する必要がある．

**謝辞** 本研究の一部は JSPS 科研費 JP15H02683 の助成を受けた．

### 参考文献

[1] Storey, M. A., Zagalsky, A., Filho, F., Singer, L. and German, D.: How Social and Communication Channels Shape and Challenge a Participatory Culture in Software

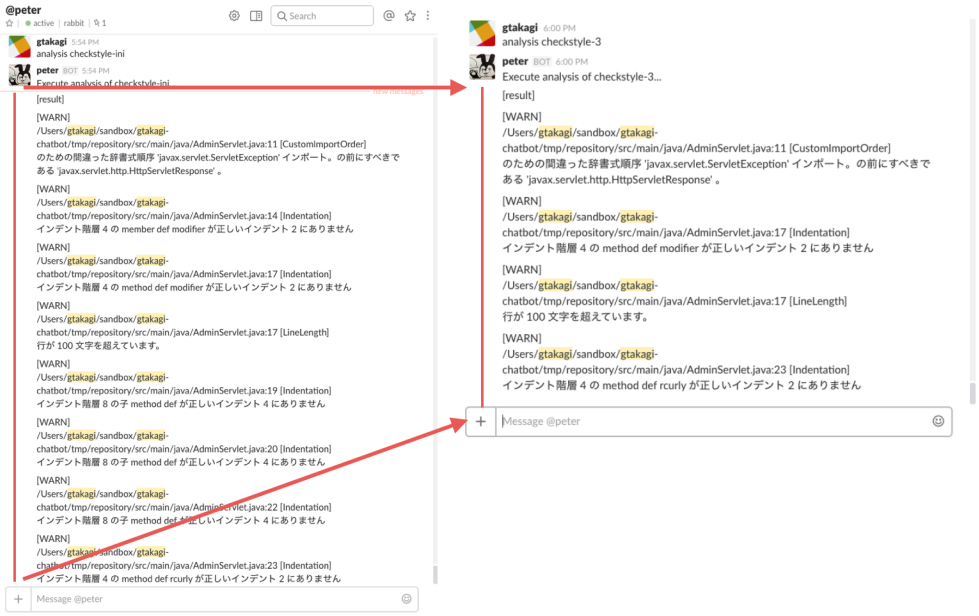


**図 7** ProxyChatBot を用いた MAFP の再実装


Development, *IEEE TSE*, Vol. 43, No. 2, pp. 185–204 (2016).

[2] Artac, M., Borovsak, T. and Nitto, E. D.: DevOps: Introducing Infrastructure-as-Code, *Proc. ICSE 2017 Companion*, pp. 497–498 (2017).

[3] Lwakatare, L. E. et al.: Towards DevOps in the Embedded Systems Domain: Why is It so Hard?, *Proc. HICSS 2016*, pp. 5437–5446 (2016).

[4] Laukkarinen, T., Kuusinen, K. and Mikkonen, T.: DevOps in Regulated Software Development: Case Medical Devices, *Proc. ICSE 2017 NIER*, pp. 15–18 (2017).

[5] Spinellis, D.: Being a DevOps Developer, *IEEE Software*, Vol. 33, No. 3, pp. 4–5 (2016).

[6] Shang, W.: Bridging the Divide between Software Developers and Operat ors Using Logs, *Proc. ICSE 2012 Companion*, pp. 1583–1586 (2012).

[7] Storey, M.-A. and Zagalsky, A.: Disrupting developer productivity one bot at a time, *Proc. FSE 2016*, pp. 928–931 (2016).

[8] Calefato, F. and Lanubile, F.: A Hub-and-Spoke Model for Tool Integration in Distributed Development, *Proc. ICGSE 2016*, pp. 129–133 (2016).

[9] Lin, B., Zagalsky, A., Storey, M. and Serebrenik, A.: Why Developers Are Slacking Off: Understanding How Software Teams Use Slack, *Proc. CSCW 2016 Companion*, pp. 333–336 (2016).

[10] Hannebauer, C., Patalas, M., Stünkel, S. and Gruhn, V.: Automatically Recommending Code Reviewers Based on Their Expertise: An Empirical Comparison, *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, New York, NY, USA, ACM, pp. 99–110 (online), DOI: 10.1145/2970276.2970306 (2016).

[11] まつ本真佑，中村匡秀：リポジトリマイニング技術共有のためのサービス指向フレームワーク，ソフトウェア工学の基礎ワークショップ FOSE2011，pp. 231–236 (2011).

[12] Matsumoto, S. and Nakamura, M.: Service oriented framework for mining software repository, *Proc. IWSM-MENSURA 2011*, pp. 13–19 (2011).

[13] 坂元康好，まつ本真佑，中村匡秀ほか：サービス指向リポジトリマイニングを効率化するキャッシュ機構の実装，研究報告ソフトウェア工学 (SE)， Vol. 2013, No. 12, pp. 1–6 (2013).

[14] Bainomugisha, E., Carreton, A. L., Cutsem, T. v., Mostinckx, S. and Meuter, W. d.: A Survey on Reactive Programming, *ACM Comput. Surv.*, Vol. 45, No. 4, pp. 52:1–52:34 (2013).

[15] 渥美紀寿，桑原寛明：MAFP:ソースコードに対する静的検査における警告の管理ツール，コンピュータソフトウェア，Vol. 33, No. 4, pp. 50–66 (2016).

[16] 桑原寛明，渥美紀寿：ソースコードの静的検査における警告の版間追跡ツール，ソフトウェアエンジニアリングシンポジウム 2015 論文集，pp. 38–47 (2015).