Vol.2021-EMB-57 No.5
2021/6/28

ElixirのROS 2クライアントライブラリRclexの開発促進に向けたテスト自動化環境の構築

武田 大輝[1,a)]　高瀬 英希[2,3]

概要：テスト自動化環境を整備することはバグを減らし，開発の促進につながる．本稿では，我々が開発中の Rclex のためのテスト自動化環境を構築する．Rclex には他言語で実装された ROS ノードとの出版購読通信を実現する機能要件があり，これは Elixir で提供される標準のテストフレームワークでは検証できない．これに対応するため，Rclcpp で実装されたノードとの通信を確認するためのテストプログラムおよびスクリプトの仕組みを作成した．また，異なる環境でのテストを効率化するために，複数のバージョンの組合せでビルドされた Docker イメージを用意した．加えて，提案手法は GitHub Actions のジョブとして登録することで，Rclex の開発管理における CI を実現した．

キーワード：テスト自動化環境，Rclex，Docker

A Test Automation Environments for Development Promotion of Rclex

1. 背景と目的

開発において製品にバグがないことを確認できることは重要であり，動作性の保証や開発容易性に繋がる．本稿では，我々が研究開発を進めている Rclex[1] の開発効率の促進を目的として，テスト自動化環境を構築する．Rclex は，関数型言語 Elixir[2] による ROS 2 のクライアントライブラリであり，他の言語で実装された ROS ノードとの出版購読通信を実現する機能を提供する．

まず，Rclex へテスト自動化環境を導入するにあたって課題を整理する．1 つは他の ROS 2 クライアントライブラリとの通信テストである．Elixir にはテスト機構が備わっているが，他言語の ROS 2 クライアントライブラリを制御するのは困難である．また，Elixir は Erlang VM 上で動作するが Elixir のテスト機構を用いて複数のノードを作成した場合，同一の OS プロセス上にノードが作成されるため，OS のプロセス間通信のテストを行うことができない．2 つめの問題は ROS も Elixir も様々なバージョンがあり，Rclex を動かす環境づくりやバージョンの変更が容易でないことである．これは開発の困難さやユーザーが使用する際のハードルの高さにも繋がる．

本研究で提案するテスト自動化環境では，様々な ROS 2 クライアントライブラリを操作し，テストを実行することを試みる．これにより ROS 2 において重要な通信のテストを実行できる環境を整備する．また，Docker の導入を行うことで Rclex をテストを実行できる環境を容易に作成できるようにする．加えて，ローカルでのテスト自動化環境の作成を行うと同時に，CI 環境を GitHub 上に構築することを行う．これにより，リポジトリに対し加えられた変更にバグが含まれていないことの確認を容易にし，開発の促進や製品の動作性の保証が見込める．

2. テスト自動化環境

2.1 全体像

Rclex のテストは，2 種類存在する．通信を伴うものと伴わないものの 2 種類がある．通信を伴うテストは

[1] 京都大学
Kyoto University
[2] 東京大学
The University of Tokyo
[3] JST さきがけ
JST PRESTO
[a)] takeda.hiroki.68x@st.kyoto-u.ac.jp

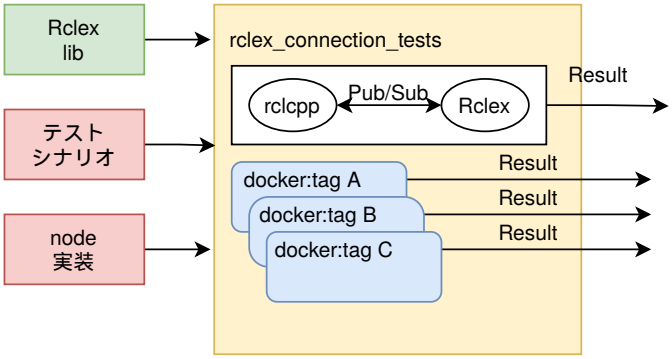

図 1 通信テスト環境構成

rclex_connection_tests[*1] リポジトリとして分離させている．通信テスト環境の全体像を**図 1** に示す．通信を伴うテストを実行する際は rclex_connection_tests ディレクトリでテストコマンドを実行することで Docker が起動し，ローカル環境にある Rclex を参照して実行する．なお，ローカル環境で Rclex や Rclcpp の実行環境がある場合は Docker を使わずにテストを行うことも可能である．通信を伴わないテストは Rclex ディレクトリ内で mix test コマンドを実行することで Elixir のテスト機構で実行される．

通信テストのリポジトリ rclex_connection_tests は Rclex，Rclcpp のノード実装，各種テストシナリオ，テスト制御用スクリプトから構成されている．テスト制御用スクリプトを実行すると，まずテストシナリオディレクトリ中の各テストシナリオを順に実行する．各テストシナリオは対象となるテストアプリケーションを起動し，通信テストを実行する．通信テストでは出版者と購読者のノードの組として，(Rclex,Rclex)，(Rclex,Rclcpp)，(Rclcpp,Rclex) の 3 種類の通信をテストしている．出版者ノードはランダムメッセージを生成した後，メッセージを出版し，同時に外部ファイルに送信したメッセージを書き込む．購読者ノードは一定時間メッセージを待機し，受信に成功した場合，メッセージを外部ファイルに書き込む．テストシナリオは両ノードのメッセージを比較し，一致した場合通信成功とみなし正常終了する．全てのテストシナリオが正常終了した場合，全体のテストが通ったことになる．

2.2 テスト用 Docker 環境

ROS 2 クライアントライブラリの動作が想定される環境は様々であり，各種環境での動作確認を行うことは重要である．そのため対応する環境を即座に用意できる必要がある．本研究では，Docker を活用して複数の異なる環境を事前に用意した．

今回用意した環境では ROS 2 Dashing と Elixir v1.9.1 以降を使用した．Docker は rclex_connection_tests で実行され，ホスト環境の rclex と rclex_connection_tests を参照し，Docker コンテナ内にマウントする．その後，テスト制御用スクリプトを実行することで Rclex の通信テストを行う．これにより異なる環境においてテストを実行することが可能になっている．

[*1] https://github.com/rclex/rclex_connection_tests

2.3 CI 環境の整備

本研究における提案手法を用いて GitHub のリポジトリにおいて CI を実行するように設定した．GitHub では様々な CI サービスを使用することが可能だが，外部アカウントを管理する必要がないこと，GitHub 上で完結することから今回は GitHubActions を使用した．

rclex リポジトリに変更が push される，または pull request が作成されると CI がトリガーされる．2.2 節で作成した Docker コンテナ上で CI は実行される．まず，変更が加えられた rclex，rclex_connection_tests リポジトリがクローンされ，ビルドされる．その後，mix test コマンドが実行され通信を伴わないテストが実行される．その後，rclex_connection_tests に含まれるテスト制御用スクリプトが実行され，通信テストが実行される．これらのテストが全て正常に終了すると事後処理を行い，CI が正常に終了する．CI の結果はタグに反映され，正しく動作しているか確認できる．

3. 導入結果と今後の展望

Docker を用いることで異なる 5 つの Elixir のバージョンでテストを容易に実行することが可能になった．また，テスト自動化環境を導入することで複数の不具合を発見することに成功した．1 つはノードの実行回数を指定してもエラーを出力した後，再起動し動き続けてしまう不具合である．もう 1 つはノードを終了する際，サブプロセスが実行中でも確認せずに終了してしまう不具合である．これらはテストを書く中でテスト用アプリケーションが想定の挙動をしないことを詳しく調査することで発見された．

今回のテスト自動化環境導入は開発をより容易にしている．現在 Rclex では基本的な出版購読通信のみをサポートしている．本研究の成果を活用し，機能を拡充していく．また，ROS 2 の複数バージョンにも対応しやすくなった．現在は Dashing のみの対応だがより新しいバージョンである Foxy に対応していくことを予定している．

謝辞 本研究の一部は，JST さきがけ JPMJPR18M8 の支援を受けたものである．

参考文献

[1] 今西洋偉，高瀬英希：関数型言語 Elixir による ROS 2 のスケーラビリティを向上させるクライアントライブラリ，情報処理学会研究報告，2020-EMB-53(48) (2020).
[2] The Elixir programming language, https://elixir-lang.org/ (アクセス日：2021 年 6 月 2 日)