TOPPERS活用アイデア・アプリケーション開発

コンテスト

部門　　：アプリケーション開発部門

作品のタイトル　　：C++版箱庭コア機能

作成者　　：森 崇 ((株)永和システムマネジメント)

共同作業者　　：高田 光隆 (名古屋大学NCES)

対象者　　：TOPPERS/箱庭開発者

TOPPERS/AUTOSAR 開発成果物利用者

使用する開発成果物　：

- TOPPERS/RH850版 Athrill
- TOPPERS/AUTOSAR 開発成果物
    - TOPPERS/ATK2
    - TOPPERS/A-COMSTACK
    - TOPPERS/A-RTEGEN

目的・狙い：

目的：箱庭開発成果物として、「C++版箱庭コア機能」を提供する

狙い：

C#版箱庭コア機能は、「Linux上のネイティブプロセスから直接関数コールできない」という課題がある。そのため、箱庭コア機能を利用する上位アプリ(AthrillデバイスやROS2ノード等)は、オーバーヘッドが高く&&手間のかかる方法(UDP/gRPC通信等)でしかコア機能を利用できない。本提案では、C/C++言語で作成されたLinuxネイティブプロセスから直接関数コールして箱庭コア機能を利用できる「C++版箱庭コア機能」を提案したい。

アイデア/アプリケーションの概要：

「C++版箱庭コア機能」の提供に加えて、本コア機能のメリットを享受できるデモ環境を構築する。具体的には、仮想環境上でTOPPERS/AUTOSAR 開発成果物を手軽にお試しできるようにする環境である。また、本環境を利用するとCAN通信状況をビジュアライズする機能を合わせて用意する。ECU間のCAN通信データのビジュアライゼーションはROS2トピックで行う。本デモ環境では、2ECU構成で、箱庭コア機能によってスケジューリングされたRH850版 AthrillがTOPPERS/AUTOSARスタックを実行し、ECU間のCAN通信データをROS2トピックで観測できるようにする。

提案内容

C++版箱庭コア機能提供にあたり、以下の機能開発及び改良を行う(詳細は後述)。

- C++版箱庭コア機能、C++版箱庭コマンド、C++版箱庭プロキシ、箱庭時間同期用Athrillデバイス、ECU間通信を実現するCAN通信デバイス、箱庭CANモニタ・プロキシ

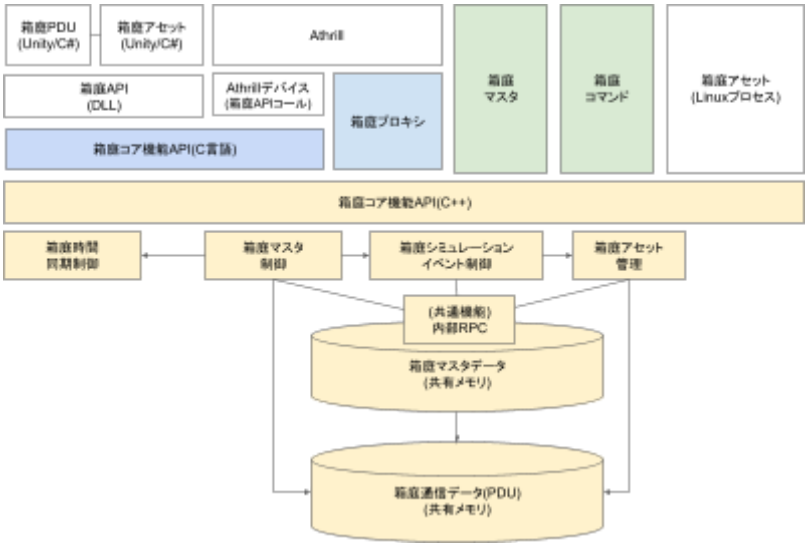
また、C++版箱庭コア機能の最初のデモとして、AUTOSARサンプルプログラムを手軽にお試しできる環境を構築する。具体的には、以下の2つの環境を構築する。

- AUTOSARお試し環境、AUTOSAR仮想ランタイム環境

C++版箱庭コア機能

C++版箱庭コア機能概要

C++版箱庭コア機能のスタック構成を下図に示す。



オレンジ部分が箱庭コア部分であり、青色・緑色部分が箱庭コアを利用しやすくするためのコマンドやプロキシなどである。それぞれ、以下で公開されている。

C++版箱庭コア機能	https://github.com/toppers/hakoniwa-core-cpp.git 補足：緑色部分は参照実装として、本リポジトリ側に配置している
C++版箱庭コマンド等	https://github.com/toppers/hakoniwa-core-cpp-client.git

C++版箱庭コア機能の詳細を解説するには紙面が不足するため、ここではその特徴的なポイントに焦点を当てて概要説明する。

本機能の基本的なコンポーネントは以下の3機能に集約される。

- 箱庭マスタ制御

        - 箱庭シミュレーション全体を制御するAPI機能群
            - 補足1:箱庭時間同期制御はここで実現されている
            - 補足2:箱庭プロセス間通信は共有メモリベース(箱庭マスタデータ)
- 箱庭シミュレーションイベント制御
    - 箱庭シミュレーションのライフサイクルを管理するAPI機能群
        - 補足1:箱庭マスタ及びアセット間の同期・通信はセマフォベース
- 箱庭アセット管理
    - 箱庭アセットが箱庭コア機能を利用する際のAPI機能群
        - 補足1:箱庭アセット間のデータ伝搬は共有メモリベース(PDU)

C++版箱庭コマンド概要

C++版箱庭コマンドのサンプルプログラムは以下で公開されている。

- hako_cmd.cpp

コマンド引数仕様は以下のとおりである。

- `hako-cmd {start|stop|reset|status|ls}`
    - `start`
        - 箱庭シミュレーションを開始する
    - `stop`
        - 箱庭シミュレーションを停止する
    - `reset`
        - 箱庭シミュレーションをリセットする(停止後に呼び出す必要あり)
    - `status`
        - 箱庭シミュレーション状態を表示する
    - `ls`
        - 登録されている箱庭アセット一覧を表示する

C++版箱庭マスタ概要

C++版箱庭マスタのサンプルプログラムは以下で公開されている。

- hako_master.cpp

コマンド引数仕様は以下のとおりである。

- `hako-master <delta_msec> <max_delay_msec>`
    - `delta_msec`
        - 箱庭時間の最小ステップ時間[単位:msec]
    - `max_delay_msec`
        - 箱庭アセット時間の最大遅延許容可能時間[単位:msec]

C++版箱庭プロキシ概要

C++版箱庭プロキシのプログラムは以下で公開されている。

- hako_proxy.cpp

コマンド引数仕様は以下のとおりである。

- `hako-proxy <param_file>`
    - `param_file`
        - `json`形式のパラメータ定義ファイル(設定例:`Athrill`の箱庭プロキシパラメタ)
        - `asset_name`
            - 箱庭アセット名
        - `target_exec_dir`

        - 箱庭プロキシの実行ディレクトリ
    - target_bin_path
        - 箱庭プロキシが実行するバイナリのパス
    - target_options
        - 箱庭プロキシが実行するバイナリに渡す引数
    - suppress_start_feedback
        - 箱庭プロキシのシミュレーション開始応答を抑制する
        - 起動プロセス(Athrill等)側の初期化処理完了を待つ必要がある場合は本フラグをtrueにする。
        - 補足：シミュレーション開始応答は起動プロセス側で行う必要がある。

箱庭時間同期用Athrillデバイス概要

AthrillがC++版箱庭コア機能と時間同期するための専用デバイスを用意した。

- hakotime

本デバイスは、Athrillのメモリコンフィグファイルで指定することで、Athrill起動時に自動的にローディングされる(以下、設定例)。

```
DEV, 0x<アドレス(16進数)>, <共有ライブラリ配置ディレクトリの絶対パス>/libhakotime.so
```

※補足：Athrillデバイスの解説記事

ECU間通信を実現するCAN通信デバイス

RH850版AthrillにはCAN通信デバイスが存在している。本デバイスの設計では、CAN通信の実現方式を変更可能にするためのインタフェースが定義されている。そのインタフェース実装として、C++版箱庭コア機能のPDU(Protocol Data Unit)というアセット間通信APIを利用したCANデータのI/Oを実現するためのプラグインを以下に用意した。

- can_bus_impl_hako.c

本プラグインは、Athrillのデバイスコンフィグファイルで ***DEBUG_FUNC_HAKO_ASSET_NAME*** というパラメータで箱庭アセット名を設定すると有効化される。また、CAN送信するCANデータ数を ***DEBUG_FUNC_HAKO_PUB_NUM*** で定義し、各CANデータを ***DEBUG_FUNC_HAKO_PUB_<0からの連番>*** で定義できる。

設定例(送信側):

```
DEBUG_FUNC_HAKO_ASSET_NAME      athrill_test_node1
DEBUG_FUNC_HAKO_PUB_NUM   1
DEBUG_FUNC_HAKO_PUB_0 channel0/CAN_IDE0_RTR0_DLC4_0x002/0
```

同様に、CAN受信するCANデータ数および各CANデータを定義できる。

設定例(受信側):

```
DEBUG_FUNC_HAKO_ASSET_NAME      athrill_test_node2
DEBUG_FUNC_HAKO_SUB_NUM   1
DEBUG_FUNC_HAKO_SUB_0 channel0/CAN_IDE0_RTR0_DLC4_0x002/0
```

※補足:CANデータ定義における可変パラメータ部分

※channel<チャネル***ID***>/CAN_IDE<***IDE***>_RTR<***RTR***>_DLC<***DLC***>_0x<***CANID***>/<箱庭***PDU***チャネル***ID***>

箱庭CANモニタ・プロキシ

上記のCANデバイスが書き込みしたCANデータは、本プロキシがROS2トピックに自動的に変換してくれる。本プログラムは以下で公開されている。

- run.bash

コマンド引数仕様は以下のとおりである。

- `run.bash can_proxy rx`
  - `can_proxy`
    - 本プログラム名
  - `rx`
    - 箱庭上に送信されたCANデータを受信してROS2トピックに変換するモード

本プロキシを起動すると、以下のように`ros2 topic` コマンドでトピックデータを参照できるようになる。

```
# ros2 topic list
/channel0/CAN_IDE0_RTR0_DLC4_0x2
```

AUTOSAR箱庭シミュレーション環境

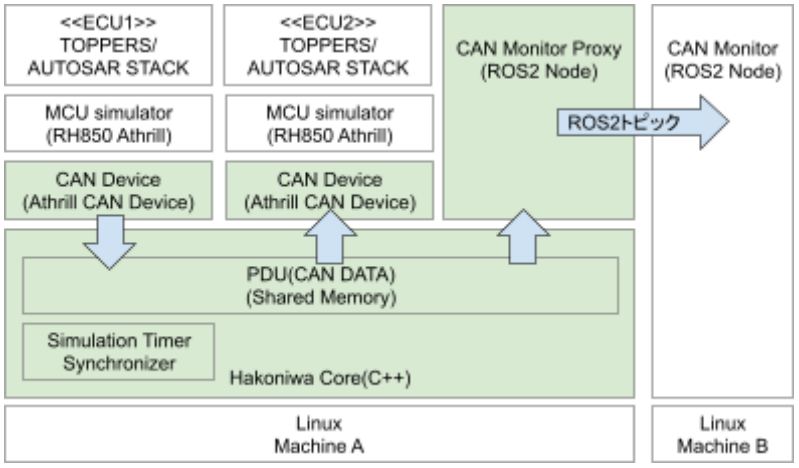
開発環境

AUTOSARサンプルプログラムをビルド実行するための環境一式をDockerコンテナ(ベースイメージ=ros:foxy) として構築する。

- RH850版GCCコンパイラ、RH850版Athrill、Athrillデバイス、C++版箱庭コア機能、C++版箱庭コマンド等、TOPPERS/AUTOSARサンプル・アプリプログラム、TOPPERS/A-RTEGEN、TOPPERS/A-COMSTACK、TOPPERS/ATK2

仮想ランタイム環境

AUTOSARサンプルプログラムを実行する仮想ランタイム環境は以下のとおりである[1]。



サンプルプログラムは、RTE API を利用して、ECU1からECU2にデータを送信する。通信データは A-COMSTACKの CAN デバイスを通して、ECU2に転送される。

コンテナ環境の構築

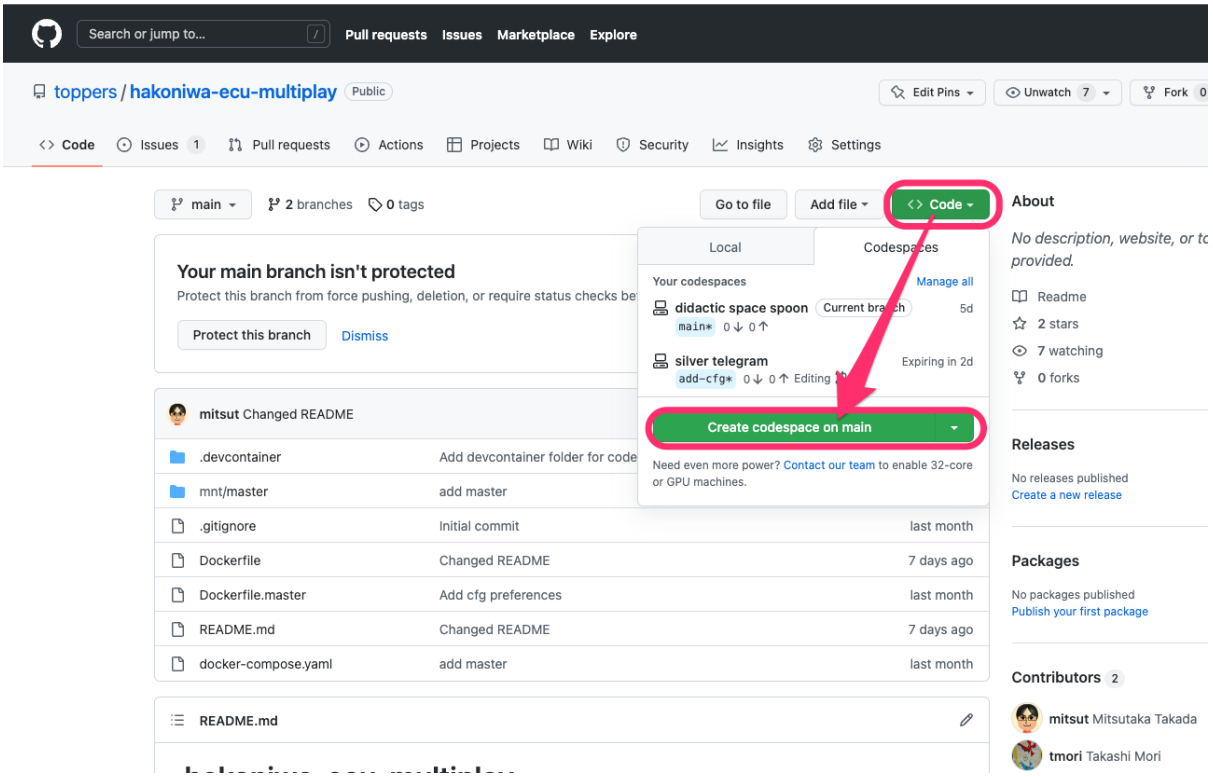
箱庭のDockerコンテナ環境を動作させるためには、使用する端末にDocker Desktopをインストールしておく必要がある。またコンテナ内には上記開発環境のセットアップが必要になる。また

[1] 本提案で作成した範囲を色塗りしている(緑色)

最近ではGitHub codespaces[2]や Gitpod[3]と言ったコンテナを活用したクラウド開発環境も無償での利用が可能である。
　本提案ではGitHub codespacesによるコンテナ環境構築について紹介する。GitHubにコンテナ環境のプロジェクトリポジトリを用意した。

- hakoniwa-ecu-multiplay
  https://github.com/toppers/hakoniwa-ecu-multiplay

リポジトリに移動し、Code を押下し、Create codespace on main　を選択する。
これらのコンテナを最初に準備するのに少し時間がかかる。

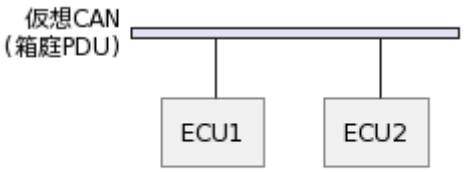


codespacesの起動が完了すると、リポジトリのワークスペースが起動され、次のフォルダが表示されることを確認する
atk2-sc1, a-comstack, a-rtegen

[2] https://github.co.jp/features/codespaces
[3] https://www.gitpod.io/

デモ内容

デモの構成は仮想ランタイム環境にある図のとおりである。
A -RTEGENにあるHelloAutosarWithComのサンプルを使って、ECU1、ECU2間でCAN通信を行う。
箱庭環境ではECU1、ECU2はそれぞれ1つの箱庭アセットとして振る舞う。
実機環境において、ECU1とECU2の間は実CANデータが流れるが、箱庭を使った仮想シミュレーション環境ではCANデータは箱庭PDUにラップされ、アセット間でデータ伝送されている。箱庭PDUへの変換はathrillと箱庭コアを接続するathrillデバイスにおいて変換を行なっているので、各ECU上で動作しているプログラムは箱庭PDUを意識せずCAN通信を行うことができる。



本デモ環境を行うにあたり、A -RTEGENサンプルのターゲット依存部設定(hsbrh850f1k_gcc)を新たに作成した。作成したターゲット依存部はA-COMSTACK、A-RTEGENリポジトリから入手できる。

動作確認方法

1. ターミナルの分割方法
動作確認ではターミナルが4つ(オプションの操作を含めると6つ)必要となる。またターミナルは同時に表示されて見えることが望ましいため、最初にターミナルを分割して表示する方法について説明する。

メニューからターミナル＞ターミナルの分割を指定することでターミナルが分割される。

もしくはターミナルウィンドウの右にある扉のようなアイコン（ゴミ箱アイコンの左）を押下することでターミナルが分割される。

ターミナルを4つに分割した場合は次のようになる.

2. 箱庭マスタの起動

一番左の端末を使って以下のコマンドを発行する。

```
# hako-master 100 200
START
INFO: shmget() key=255 size=12160
```

3. ECU1の箱庭起動準備（箱庭プロキシの起動）

2つ目のターミナルを使って、次のコマンドを発行する

```
# cd $HAKO_WS_ECU1
# hako-proxy ./proxy_config_rte_ecu1.json
add_option:/root/athrill-target-rh850f1x/athrill/bin/linux/athrill2
add_option:-c1
add_option:-t
add_option:-1
add_option:-d
add_option:device_config_with_rte_hako_ecu1.txt
add_option:-m
add_option:memory_with_hako.txt
add_option:atk2-sc1
INFO: PROXY start
```

4. ECU2の箱庭起動準備（箱庭プロキシの起動）

3つ目のターミナルを使って、次のコマンドを発行する

```
# cd $HAKO_WS_ECU2
# hako-proxy ./proxy_config_rte_ecu2.json
add_option:/root/athrill-target-rh850f1x/athrill/bin/linux/athrill2
add_option:-c1
add_option:-t
add_option:-1
add_option:-d
add_option:device_config_with_rte_hako_ecu1.txt
add_option:-m
add_option:memory_with_hako.txt
add_option:atk2-sc1
INFO: PROXY start
```

5. 箱庭のシミュレーション実行
4つ目のターミナルを使って、次のコマンドを実行する。
これでアセットが2つあることを確認できる。これは前に起動準備した箱庭プロキシである.これが見えない場合は箱庭マスタの確認から行う。

```
# hako-cmd ls
athrill_test_node1
athrill_test_node2
```

次のコマンドで箱庭のステータスを確認できる

```
# hako-cmd status
status=stopped
```

箱庭は停止状態であることがわかる
箱庭を実行するためには、次のコマンドを実行する

```
# hako-cmd start
```

これで、ECU1、ECU2のターミナルでathrillの起動待ちになっている状態からシミュレーションの開始が行われ、ECU1-ECU2間でCANデータを送受信していることがわかる

6. 箱庭のシミュレーションの停止と再開
箱庭を停止するときは以下のコマンドを発行すると各ターミナルの出力が停止される

```
# hako-cmd stop
stop
```

再開するときは最初に箱庭のリセットを行う

```
# hako-cmd reset
```

箱庭マスタのターミナルが次のように表示される

```
EVENT: reset
```

箱庭の再開はリセット後に開始コマンドを発行することで再開される。

```
# hako-cmd start
```

動作確認方法（オプション）

ROSツールを用いた箱庭**PDU**のデバッグ表示
仮想ランタイム環境で示しているように、CANモニタProxyの箱庭アセットを追加することで、箱庭PDUをROS2トピックとしてPublishすることができ、既存のROSツールを用いて箱庭PDUの情報をダンプする。

1. CANモニタProxyの起動方法
5つ目のターミナルを使って、次のコマンドを実行する。
本アセットを起動する前に、箱庭は停止状態であることを確認する。

```
# hako-cmd status
status=stopped
```

次に以下のコマンドを実行することで、アセットを実行するフォルダへ移動し、proxyを起動させる。起動するとproxyの時刻がインクリメントされ表示される。

```
# cd $HAKO_WS_ROS
# bash run.bash can_proxy rx
LOADED: PDU DATA
TIME: 100000
TIME: 200000
TIME: 300000
TIME: 400000
TIME: 500000
```

2. ROS2コマンドによるトピック情報の表示
6つ目のターミナルで以下のコマンドを実行する。
ROS2のコマンドを使用するための環境変数を設定する。

```
# cd $HAKO_WS_ROS
# source install/setup.bash
```

続いてCANモニタproxyで使用しているトピック情報を表示する。

```
# ros2 topic list
/channel0/CAN_IDE0_RTR0_DLC4_0x2
/parameter_events
/rosout
```

*/channel0/CAN_IDE0_RTR0_DLC4_0x2*がproxyで使用しているトピック情報となる。
次のコマンドを実行するとトピック情報がダンプされる。
body部のdataがインクリメントされて、ECU1,2で表示されている値と同じデータが表示される。

```
# ros2 topic echo /channel0/CAN_IDE0_RTR0_DLC4_0x

head:
  channel: 0
  ide: 0
```

```
  rtr: 0
  dlc: 4
  canid: 2
body:
  data:
  - 51
  - 0
  - 0
  - 0
  - 0
  - 0
  - 0
  - 0
```

TIPS

箱庭シミュレーションが実行しない場合

箱庭マスタのターミナルで次のような表示がでた場合は、箱庭マスタが起動してから箱庭コマンドでの開始実行が遅くなってしまい、箱庭マスタがタイムアウトしたために箱庭が実行されない場合がある。

```
ERROR: timeout asset_id=
```

この表示が出た場合は、次の手順でやり直す。
1）全てのターミナルで実行しているプロセスをCtrl＋Cによって、プロセスを終了する
2）箱庭マスタを起動
3）箱庭アセットを起動
4）箱庭コマンドで箱庭シミュレーションを開始

ローカルPCのDocker環境でのデモ実行方法

今回はデモの動作確認方法をGitHub Codespacesを使って説明をしたが、ローカルPCにDocker環境が構築されている場合は*hako/dodcker/run.bash*を使用することで、同様のコンテナ環境の構築とデモの実行を行うことができる。

今後の展望

今回は、C++版箱庭コア機能のデモ環境としてマルチECU構成でのAUTOSARシミュレーション環境を構築したが、本コア機能の適用範囲はここだけに止まらない。C++版箱庭コア機能は、C/C++のライブラリ（静的、動的）として提供されるため、ネイティブアプリケーションはライブラリをリンクするだけで利用可能となる。また、本ライブラリはDLL化することも可能であり、C#アプリケーションやUnityスクリプトからも利用できるようになる。さらに、Unityと同種のゲームエンジンであるUnreal EngineはC++でプログラミングするため、C++版箱庭コア機能との親和性は言うに及ばない。

以上より、今回提案した成果物は、TOPPERS/箱庭の適用範囲をさらに広げる土台になると考えている。今後の箱庭WG活動の一助になれば幸いである。