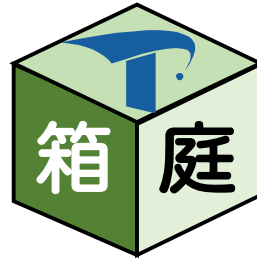


最新の箱庭コア技術紹介



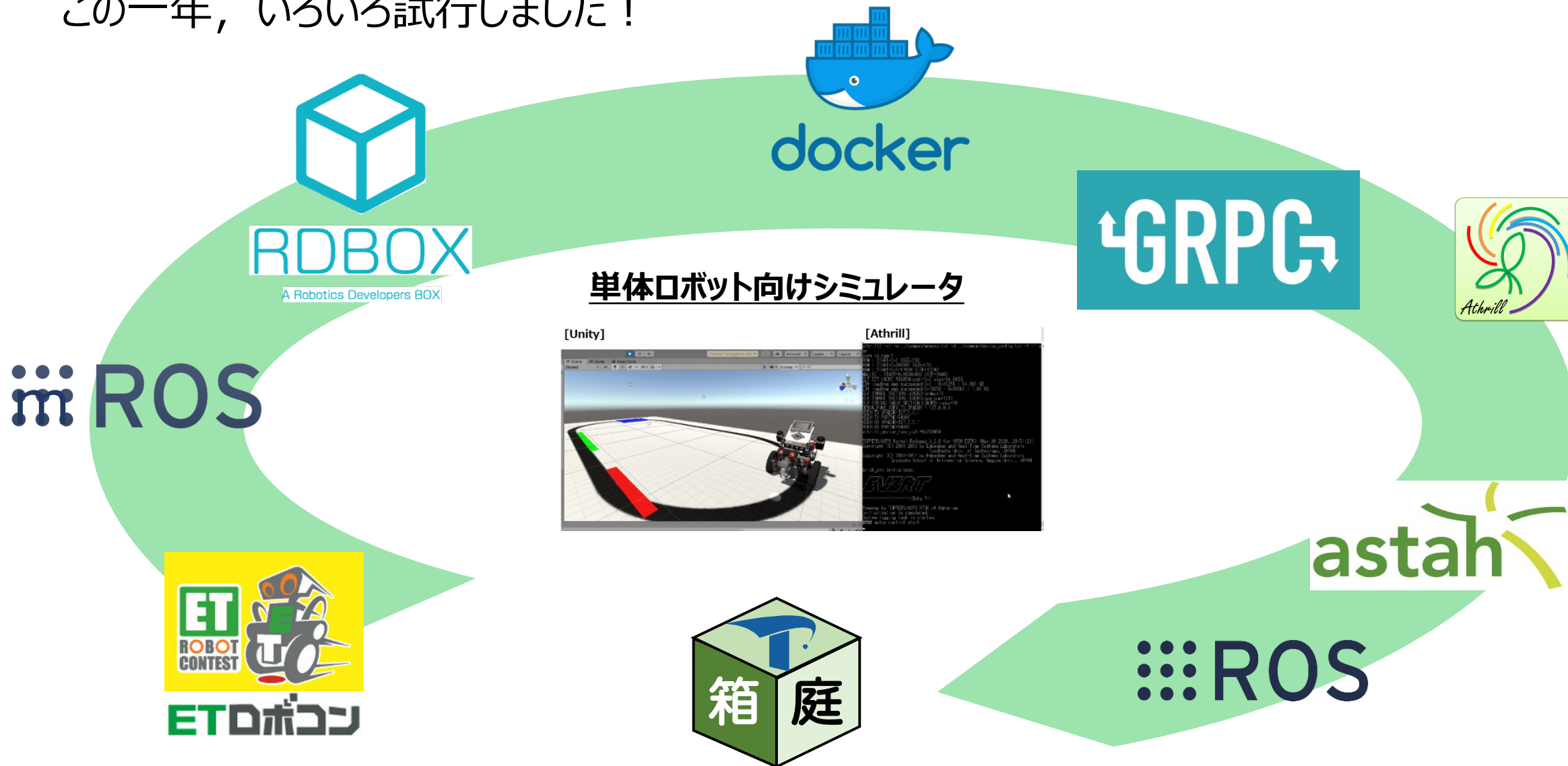
株式会社 永和システムマネジメント 森 崇

アジェンダ

1. 所感
2. 見えてきた課題
3. 取り組み内容
4. 今後

所感

この一年，いろいろ試行しました！



見えてきた課題

箱庭普及に伴い、
利用者と開発者双方にとって乗り越えなければならない課題が見えてきた

箱庭利用者



便利機能の不足



インストールの手間



マシンスペック不足



箱庭開発者



サポート負荷高



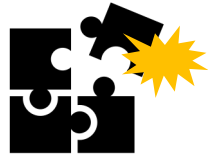
個別カスタマイズ増



プロト開発遅延



便利機能の不足



・シミュレーション実行時の機能

要望	欲しい機能
シミュレーション失敗時のやり直しの手間を削減したい	シミュレーション・停止機能
	シミュレーション・リセット機能
	シミュレーション・再実行機能
シミュレーション成功・失敗を外部ツール側で判定したい	シミュレーション・内部情報の可視化
シミュレーション用パラメータを外部ツール側から設定したい	シミュレーション・コンフィグ機能
シミュレーションを連続実行したい	シミュレーション・遠隔実行機能

インストールの手間

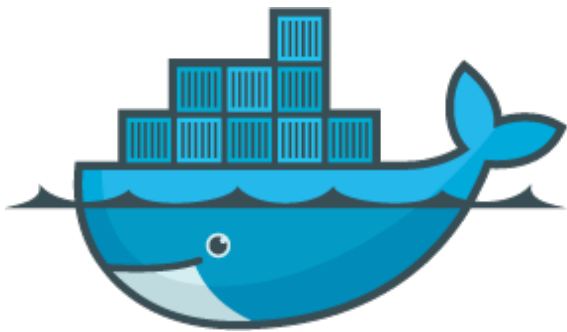


改善ポイント

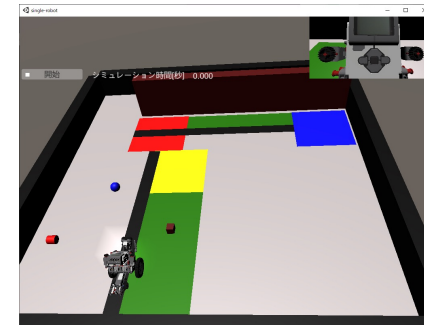
インストールすべきものが多岐にわたる(git, gcc, ruby, Athrill, Unity, ...)

手作業でのインストールのため、操作ミスが起きる(個別問い合わせ大変)

箱庭で利用するツール(Ruby等)がインストール済みの場合の動作不良問題(バージョン差異)



docker



Unityアプリのバイナリ化！

マシンスペック不足



構成要素	負荷要因
Unity	グラフィック表示用GPU負荷
Athrill	CPU負荷



解決案を模索中...

Web / クラウドに強い
 エンジニア / 協力企業・
 募集中です！

取り組み内容

1. 単体ロボット向けシミュレータのdocker化
2. Unity×Athrill×mROS×RDBOX連携
3. Athrillの外部デバイス化
4. 箱庭コア機能の実現
5. 箱庭上で動作するアセットのシステム構成をAstahで記述
6. 箱庭アセット通信データの可視化

単体ロボット向けシミュレータのdocker化(1/2)

2ステップでインストール

インストール

https://github.com/toppers/hakoniwa-single_robot

シミュレータの導入手順

Dockerイメージの展開

シミュレータの実行環境は、ビルド済みのDocker imageをDocker Hubにて公開しています。

https://hub.docker.com/r/toppersjp/hakoniwa-single_robot

現在の最新版は v1.1.0 です。「バージョン情報・更新履歴」も参照してください（バージョン番号はGit/GitHubのtag/releaseおよびDocker Hubのtag番号に対応しています）

次のようにDockerを立ち上げて、imageをpullして展開してください。

```
$ sudo service docker start
$ sudo docker pull topersjp/hakoniwa-single_robot:v1.1.0
```

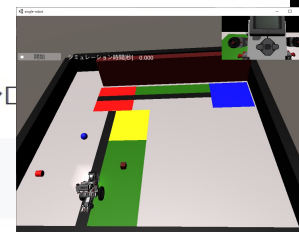
[開発者向け情報] 手元でDocker imageを作成する場合は、下記のように実行してください。

```
$ bash docker/create-image.bash
```

Unityバイナリのダウンロード

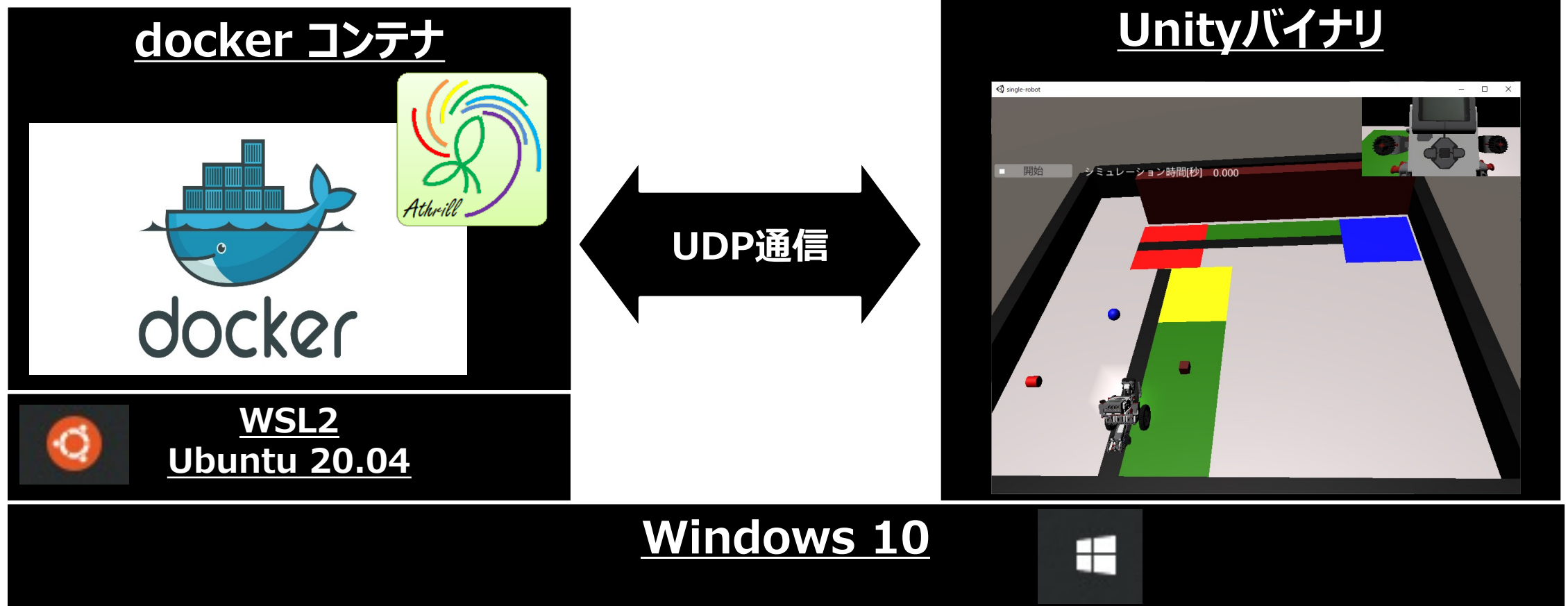
以下のコマンドを実行して、single-robotのUnityシミュレータ(Unityバイナリ)をダウンロード

```
$ bash unity/download.bash single-robot WindowsBinary.zip
```



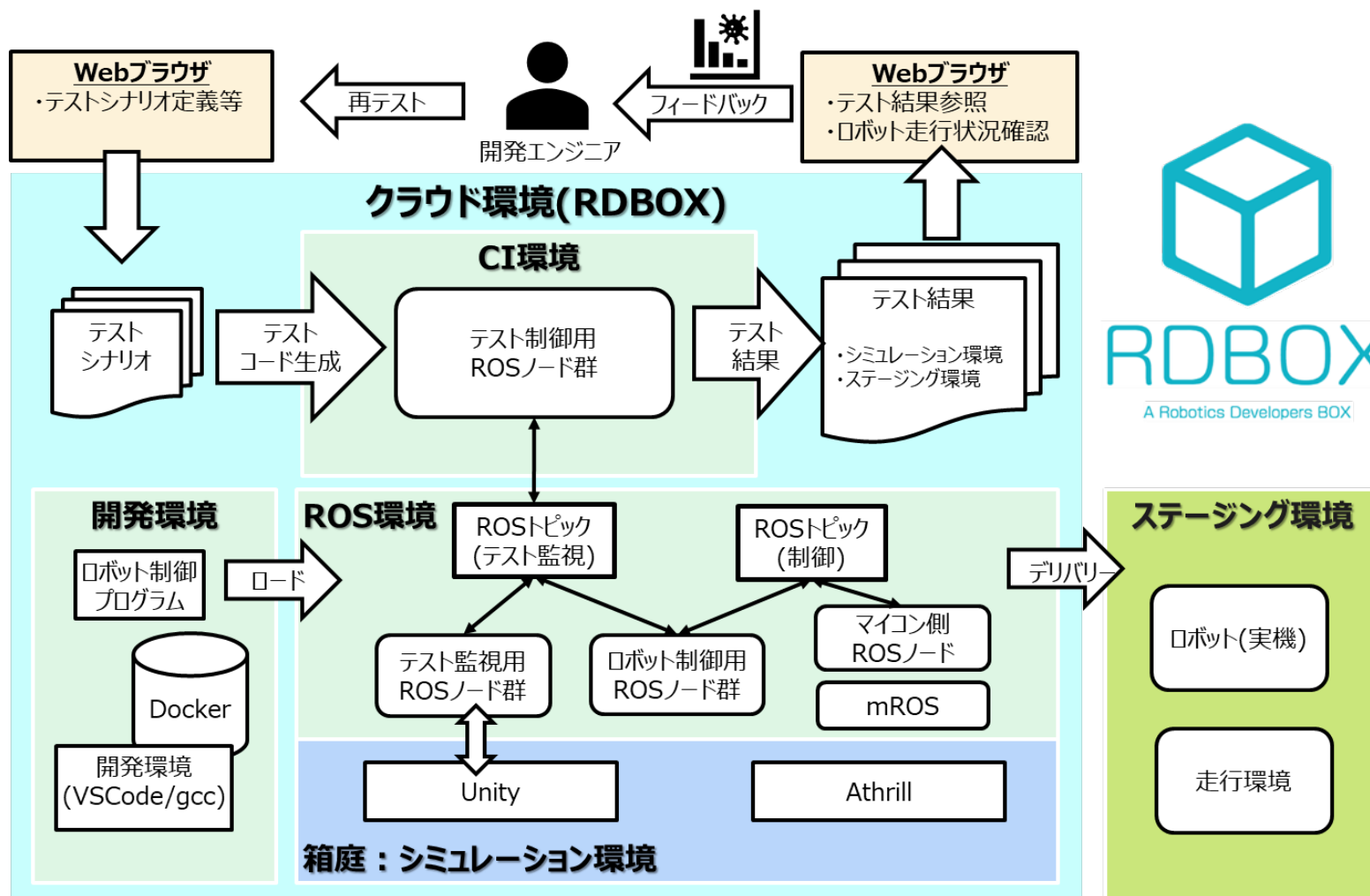
単体ロボット向けシミュレータのdocker化(2/2)

- システム構成



Unity×Athrill×mROS×RDBOX連携

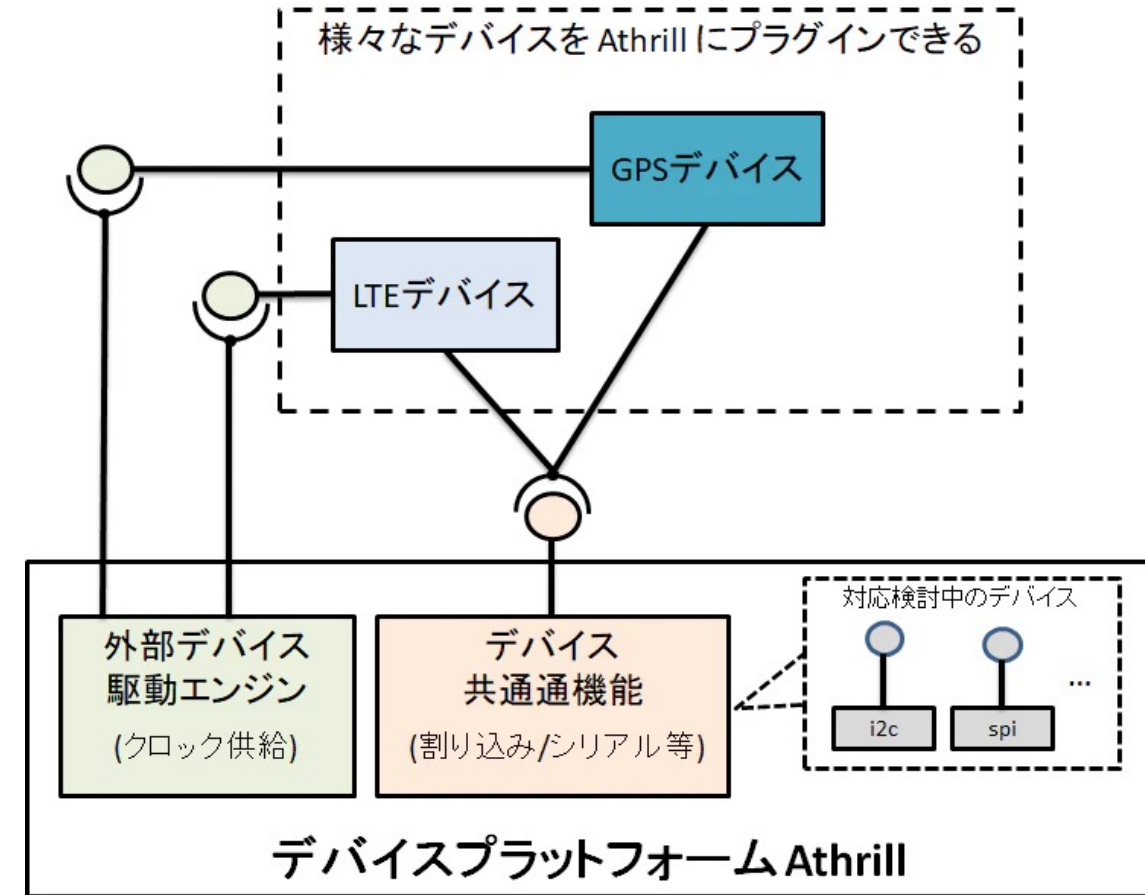
- 単体ロボット向けシミュレータをmROSでROS化！
- ROSシステムテストをRDBOXでCI/CD化！（インテック 福田さん）



【補足：以下は将来課題】
 ・テストコード生成
 ・ステージング環境/デリバリー

Athrillの外部デバイス化

- 狙い：
 - 箱庭アセットであるAthrillの拡張性を向上
 - 周辺デバイス開発者を増やしたい
- 仕組み：
 - 周辺デバイスを共有ライブラリ化
 - Athrillのコンフィグで動的にロード
- 周辺デバイスの開発言語：
 - gRPC連携により多言語対応
 - (C, C++, C#, Ruby, Elixir, ...)
- 公開リポジトリ
 - <https://github.com/toppers/athrill-device>

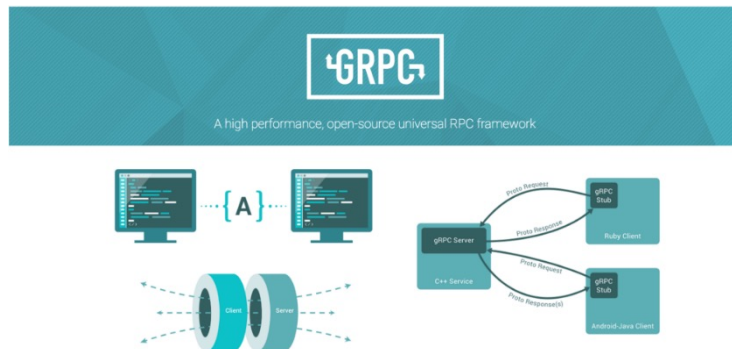


箱庭コア機能の実現

モチベーション

- 単体ロボット向けシミュレータの機能不足を効率よく開発したい
- 箱庭開発者の課題を解決したい
- 箱庭共通機能をコア機能化したい
- gRPC/プロトコルバッファとの出会い
 - DSLによるI/F定義
 - 多言語対応
 - RPCコードの自動生成

⋮



箱庭利用者



便利機能の不足



箱庭開発者



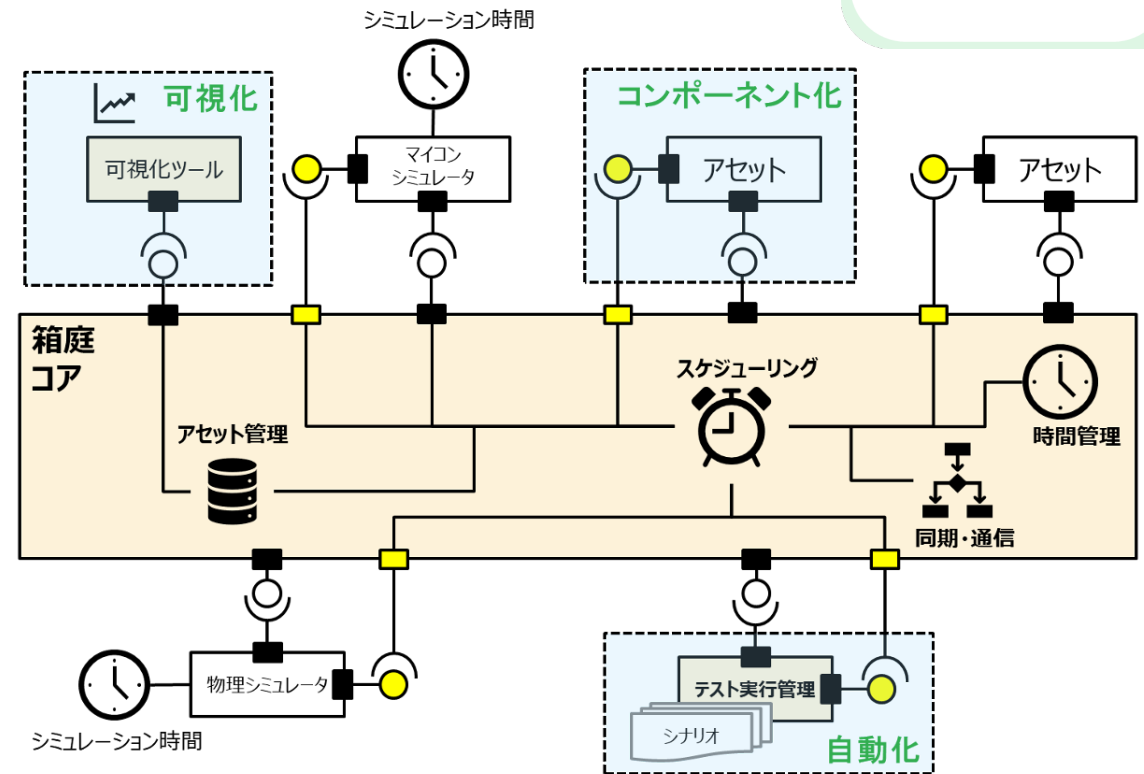
サポート負荷高



個別カスタマイズ増



プロト開発遅延



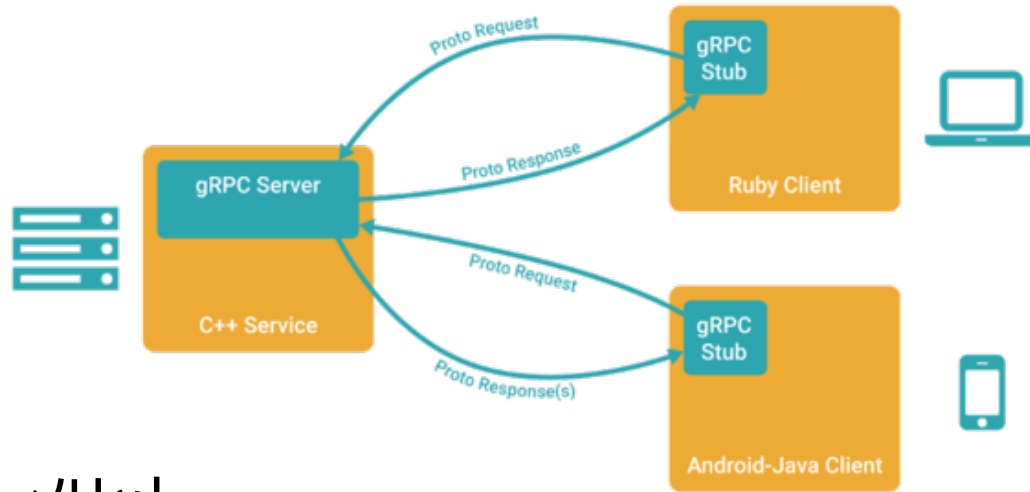
箱庭コア機能のユースケース

- 単体ロボット向けシミュレータの機能要望をユースケースとして整理
- 補足：青塗部分を対応済み（7件）

箱庭コア機能	ユースケース	要望 有無	想定アクター	補足
スケジューリング	シミュレーションを開始する	○	テスト実行管理	シミュレーション実行回数を指定できる
	シミュレーションを終了する	○	テスト実行管理	シミュレーション連続実行中の終了
	シミュレーションを実行開始状態に戻す	○	テスト実行管理	箱庭アセットの状態(配置場所等)を初期状態に戻す
	シミュレーション実行状況を取得する	○	テスト実行管理	実行中かどうか
	シミュレーション成立条件を設定する	○	テスト実行管理	コースアウト判定とか、ゴール判定とか
	シミュレーション実行結果を取得する	○	テスト実行管理	成功したかどうか
アセット管理	箱庭アセットを登録する	×		-
	箱庭アセットを登録解除する	×		-
	箱庭アセットのリストを取得する	×		-
	箱庭アセットの内部状態を参照する	○	箱庭コア機能(スケジューリング)	状態, 位置, 速度, 角度等
	箱庭アセットの内部状態を初期化する	○	箱庭コア機能(スケジューリング)	初期位置に戻す
	箱庭アセットの内部状態をバックアップする	○	箱庭コア機能(スケジューリング)	-
	箱庭アセット・パラメータを登録する	○	テスト実行管理	シミュレーションのPID制御用パラメータ設定など
	箱庭アセット・パラメータを登録解除する	×		-
	箱庭アセットのパラメータ・リストを取得する	×		-
	箱庭アセット・パラメータ情報を参照する	×		-

gRPC

- Google がオープンソースとして公開



メリット

- 箱庭コア機能と箱庭アセットと疎結合にできる
 - サーバ/クライアントモデル(RPC通信)
 - サーバー：箱庭コア機能
 - クライアント：箱庭アセット
- RPC通信はgRPC機能を利用可能
- 開発言語非依存
 - 箱庭コア/アセットを様々な言語で開発できる

箱庭コア機能のRPCサービス定義

```

service CoreService {
    //箱庭アセットを登録する
    rpc Register (AssetInfo) returns (NormalReply) {}
    //箱庭アセットを登録解除する
    rpc Unregister (AssetInfo) returns (NormalReply) {}

    //箱庭アセットのリストを取得する
    rpc GetAssetList (google.protobuf.Empty) returns (AssetInfoList) {}

    //シミュレーションを開始する
    rpc StartSimulation (google.protobuf.Empty) returns (NormalReply) {}

    //シミュレーションを終了する
    rpc StopSimulation (google.protobuf.Empty) returns (NormalReply) {}

    //シミュレーション実行状況を取得する
    rpc GetSimStatus (google.protobuf.Empty) returns (SimStatReply) {}

    //シミュレーションを実行開始状態に戻す
    rpc ResetSimulation(google.protobuf.Empty) returns (NormalReply) {}

    //シミュレーション時間同期度合いを取得する
    rpc FlushSimulationTimeSyncInfo (SimulationTimeSyncOutputFile) returns (NormalReply) {}

    //箱庭アセット非同期通知
    rpc AssetNotificationStart (AssetInfo) returns (stream AssetNotification) {}
    rpc AssetNotificationFeedback (AssetNotificationReply) returns (NormalReply) {}
}
  
```

参照：https://github.com/toppers/hakoniwa-core/blob/main/spec/hakoniwa_core.proto

箱庭コア機能のアーキテクチャ

- 様々な課題に対応すべく、箱庭コア機能のアーキテクチャを設計

箱庭シミュレータ

(プロトタイプモデル A, B, C等)

箱庭コア機能

特徴 1.

IoT/クラウド時代に求められるシミュレーションのユースケースに対応(予定)

<https://github.com/toppers/hakoniwa-core>

gRPC

特徴 2.

箱庭コアと箱庭アセット間の通信はgRPCで行うことにより、疎結合が可能

箱庭アセット群

特徴 3.

様々な言語で開発可能

特徴 4.

箱庭プロキシを利用することで、既存シミュレータ (athrill等)を改修せずに箱庭コアと結合できる

箱庭プロキシ

箱庭コマンド群

特徴 5.

コマンドベースで
シミュレーションを自動化

箱庭上で動作するアセットのシステム構成をAstahで記述

- GUIベースで直感的にアセット/コア間のつながりを記述し、コンフィグファイルを自動生成できる

箱庭向けのコンフィグ情報群(テキストデータ)

アセット定義
(Athrill等)

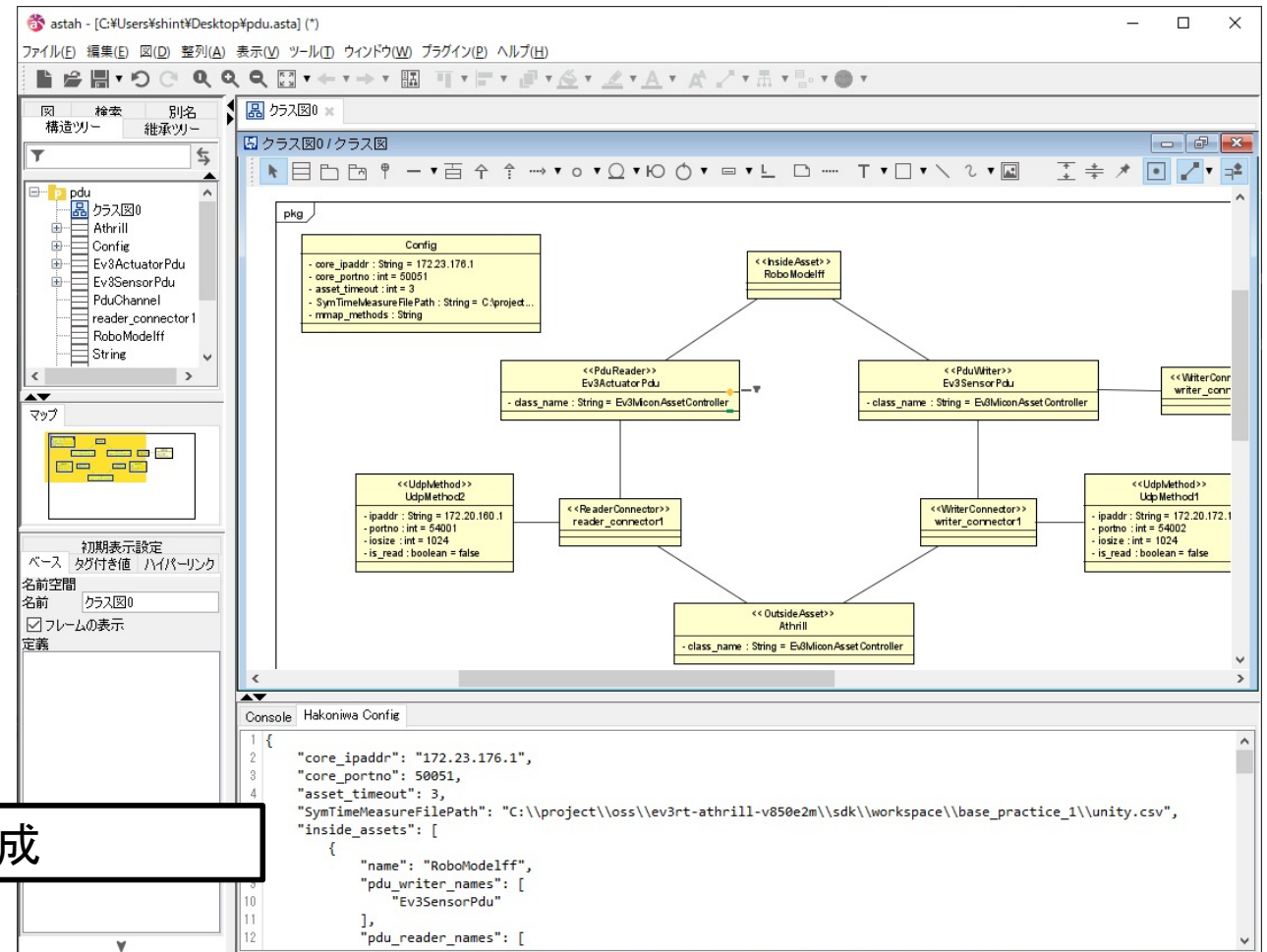
通信方式定義
(UDP/MMAP)

UDPポート
定義

通信データ定義
(PDU)

箱庭コア機能
コンフィグファイル
(json)

自動生成



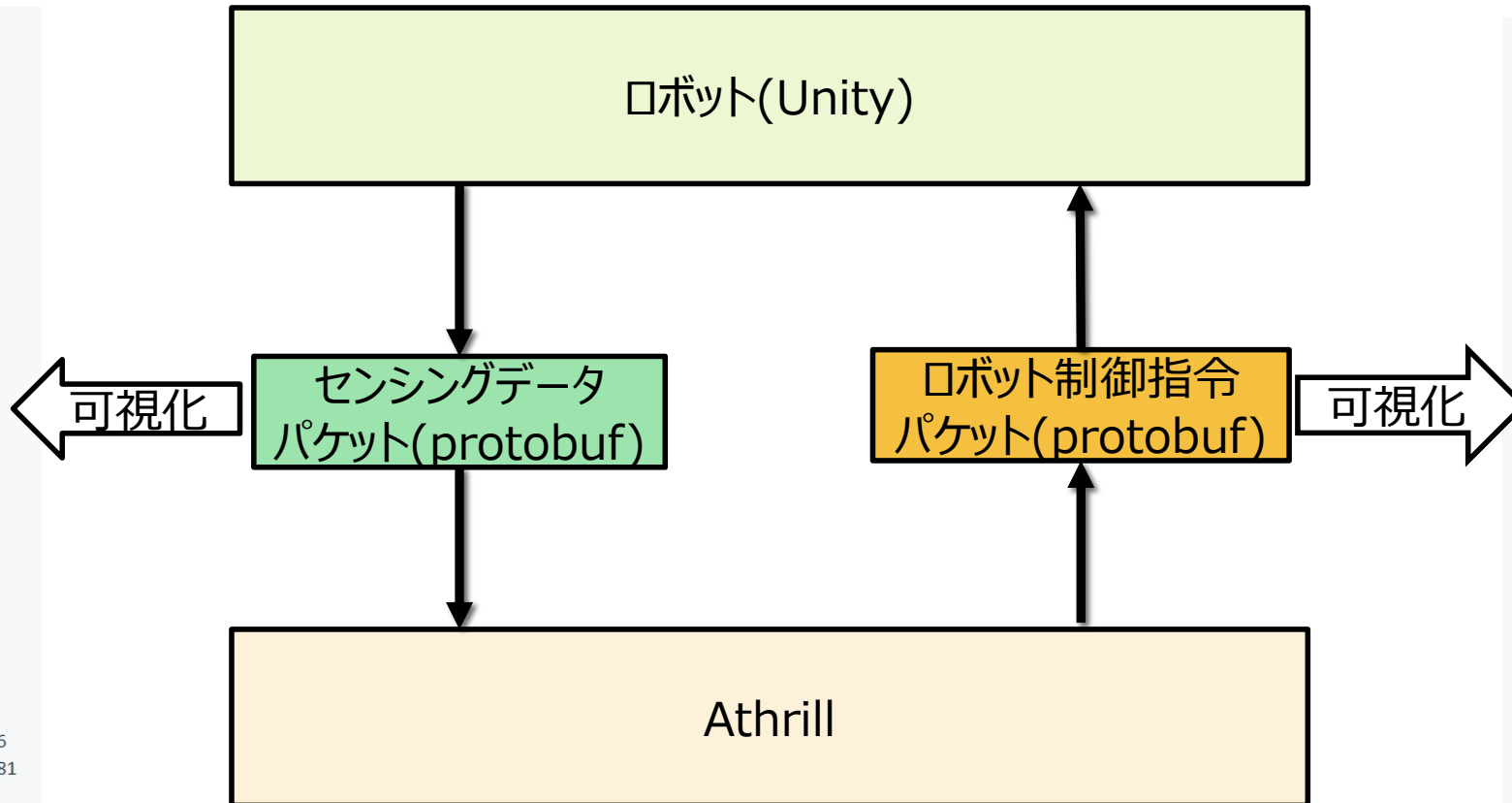
- <https://github.com/s-hosoai/hakoniwa-config-astahplugin>

箱庭アセット通信データの可視化

- 箱庭アセット間の通信データをプロトコルバッファで定義可能にした
- 箱庭コンフィグで通信データをデバッグ用にUDP配信可能にした
 - リアルタイムに通信データの可視化(プロトコルバッファ用コマンドprotocで実現)

センシングデータ

```
header {
  name: "ETRX"
  version: 1
  hakoniwa_time: 151614838
  ext_off: 512
  ext_size: 512
}
body {
  buttons: "\000"
  color_sensors {
    color: 2
    reflect: 38
    rgb_r: 77
    rgb_g: 77
    rgb_b: 255
  }
  color_sensors {
  }
  touch_sensors {
  }
  touch_sensors {
  }
  gyro_degree: 93
  sensor_ultrasonic: 478
  motor_angle_a: 1441
  motor_angle_b: 1274
  gps_lat: 46.629032135009766
  gps_lon: -29.996162414550781
}
```



ロボット制御指令値

```
header {
  name: "ETTX"
  version: 1
  asset_time: 12630200
  ext_off: 512
  ext_size: 512
}
body {
  leds: "\003"
  motors {
    power: 5
    stop: 1
  }
  motors {
    power: 5
    stop: 1
  }
  motors {
    power: -5
  }
}
```

デモ

```

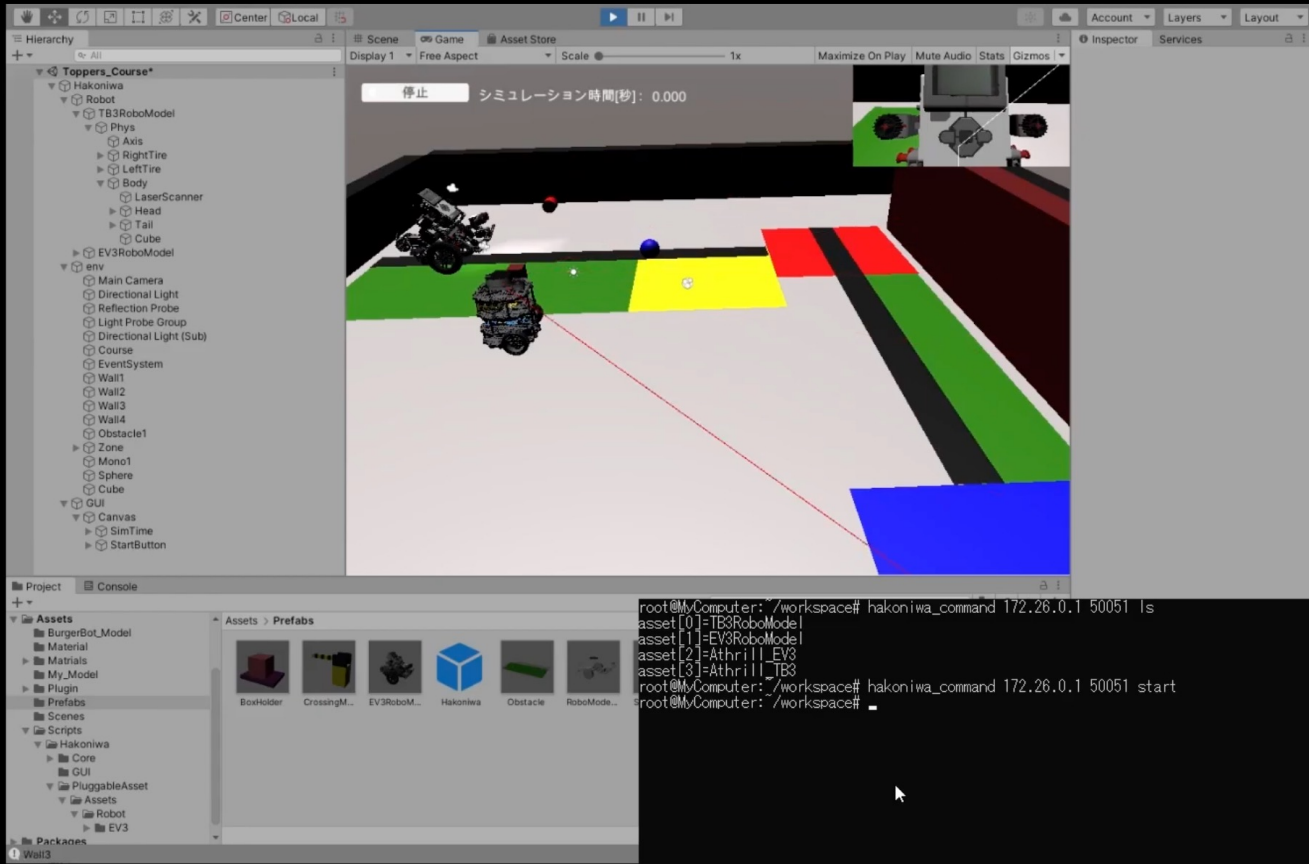
root@MyComputer:~/workspace/proxy# bash hakoniwa_proxy_2.bash
##### START CONNECT #####
Client Register reply received:
INFO: Register Asset Athrill_TB3 success
INFO: Notification Setting success
#####THREAD START:
add_option:./start_atrill.bash
add_option:bt_remote
pid:18753
monitring:
exec_arg count=2
the_arg[0]=./start_atrill.bash
the_arg[1]=bt_remote
ATHRILL_PID=18754
core_id num=1
ROM : START=0x0 SIZE=2048
RAM : START=0x200000 SIZE=2048
RAM : START=0x3ff7000 SIZE=1024
RAM : START=0x5ff7000 SIZE=10240
RAM : START=0x7ff7000 SIZE=10240
DEV : START=0x90f0000 SIZE=1048576
ELF SET CACHE REGION:addr=0x0 size=84 [KB]
Elf loading was succeeded:0x0 - 0x152dc : 84.732 KB
Elf loading was succeeded:0x152dc - 0x157ac : 1.208 KB
ELF SYMBOL SECTION LOADED:index=20
ELF SYMBOL SECTION LOADED:sym_num=1127
ELF STRING TABLE SECTION LOADED:index=21

Elf loading was succeeded:0x0 - 0x15404 : 85.4 KB
Elf loading was succeeded:0x15404 - 0x158d4 : 1.208 KB
ELF SYMBOL SECTION LOADED:index=20
ELF SYMBOL SECTION LOADED:sym_num=1127
ELF STRING TABLE SECTION LOADED:index=21
DEBUG_FUNC_EV3COM_TX_IPADDR = 172.26.0.1
VDEV:TX IPADDR=172.26.0.1
VDEV:TX PORTNO=54001
DEBUG_FUNC_EV3COM_RX_IPADDR = 172.26.12.34
VDEV:RX IPADDR=172.26.12.34
VDEV:RX PORTNO=54002
DEVICE_CONFIG_COMPLEMENTAL_TX_SENDING=1
DEVICE_CONFIG_RESET_AREA_OFFSET=68
DEVICE_CONFIG_RESET_AREA_SIZE=20
DEBUG_FUNC_FT_LOG_SIZE=1024
atrill device_func_call=0x2004cc
brick_dri initialized.

EVERT
=====>Beta 7<=

Powered by TOPPERS/ASP3 RTOS of Hakoniwa
Initialization is completed..
##### motor control start
value=0
Ops!!

```



```

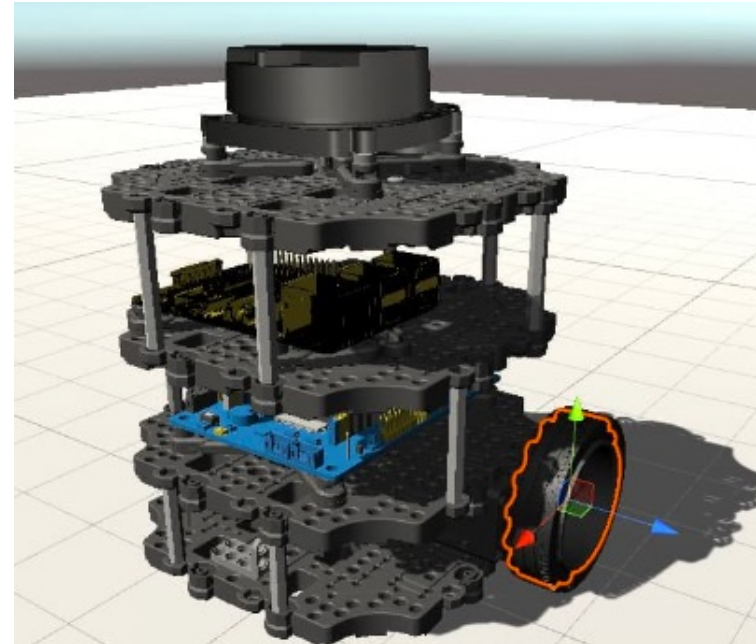
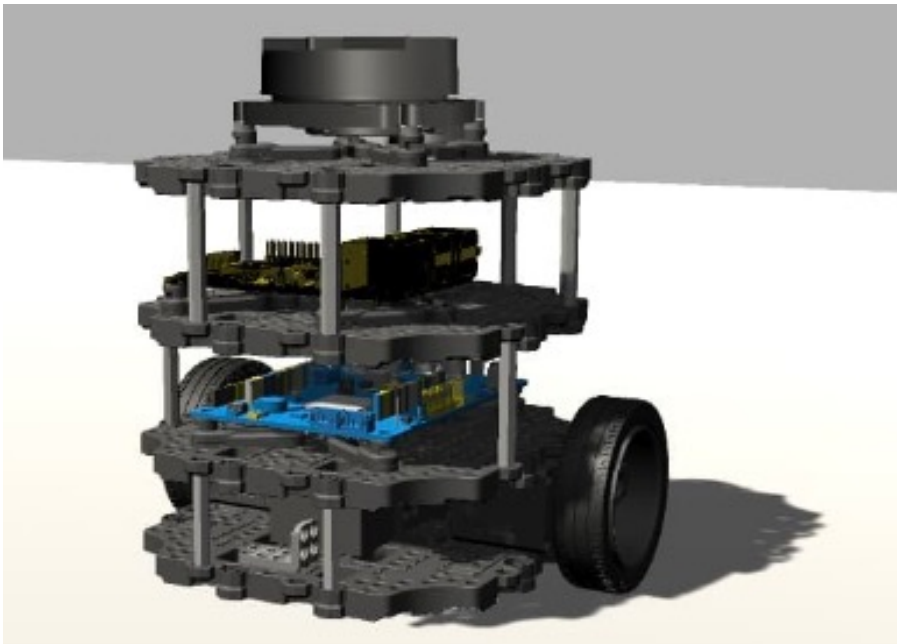
root@MyComputer:~/workspace# hakoniwa_command 172.26.0.1 50051 ls
asset[0]=TB3RoboModel
asset[1]=EV3RoboModel
asset[2]=Athrill_EV3
asset[3]=Athrill_TB3
root@MyComputer:~/workspace# hakoniwa_command 172.26.0.1 50051 start
root@MyComputer:~/workspace#

```

シミュレーション開始!

謝辞・特記事項

- TurtleBot3 の Unity パッケージの設計と作成にあたっては、宝塚大学 東京メディア芸術学部 **吉岡章夫准教授**および学部生の**杉崎涼志さん**、**木村明美さん**にご協力いただきました。
- TurtleBot3 のUnity アセットは、**株式会社ロボティズ様**より提供いただいたデータを基に作成しています。ご協力いただき深く感謝いたします。



今後について

- 箱庭コア機能の適用領域拡充
- RDBOX/クラウド連携強化
- 箱庭アセットを増やす！
 - 4年後，108個をめざそう！

目指せ
大阪万博

