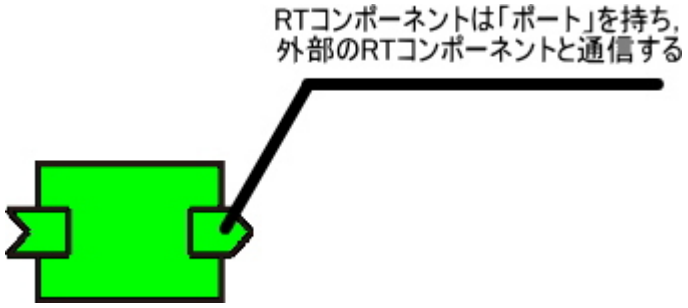
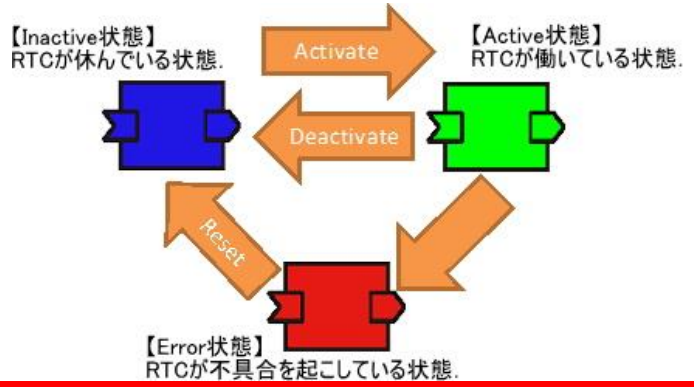
RTミドルウエア リファレンスハードウエア OROCHI

ysuga.net 菅　佑樹

RTコンポーネントとは何か

- RT要素　＝　「RTコンポーネント　(Robotics Technology Component, RTC)」
  - RTCは「**ポート**」を持ち，他のRTCと接続・通信可能
  - RTCは「**状態**」を持ち，それらの遷移によってシステム管理
    - CREATED (初期化前の状態), INACTIVE (非実行状態)，ACTIVE (実行状態)，ERROR (エラー)
  - RTCは特定のタイミングで呼ばれるイベントハンドラを持つ
    - onActivated ・・・ ACTIVE化の際に一度
    - onDeactivated ・・・ INACTIVE化の際に一度
    - onExecute ・・・ ACTIVE状態の場合に周期的に呼ばれる
  - RTCのイベントハンドラは**実行コンテキスト**によって呼ばれる
    - リアルタイム周期実行可能な実行コンテキスト
    - シミュレータに合わせて実行周期を同期する実行コンテキスト
- RTCの組合せによってロボットシステムを開発する





RTミドルウエアを使ったシステム開発イメージ
～モジュールを使うソフトウエアをインストール～

まずはコンピュータに差そう(笑)

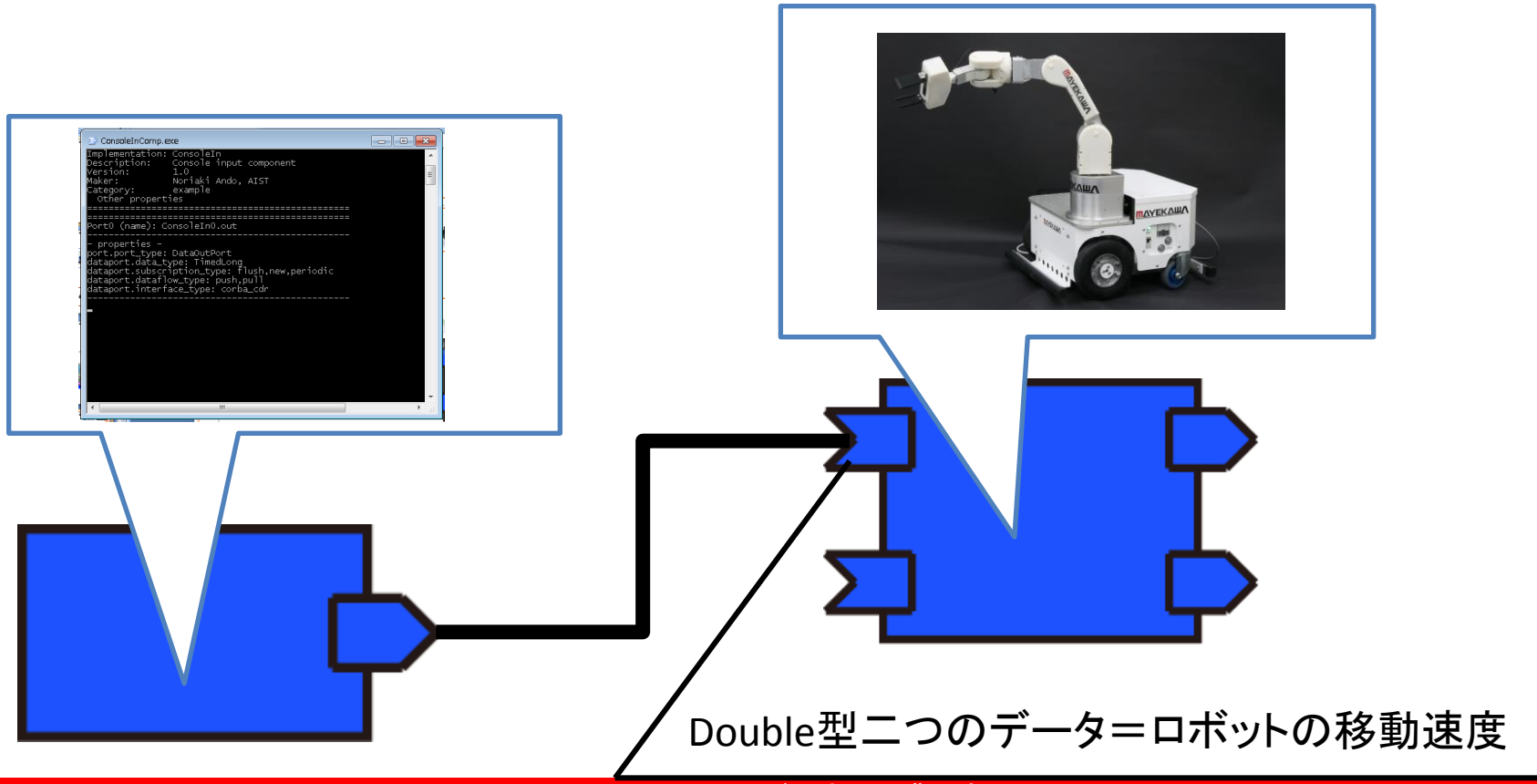
DVD-ROMなど

ドライバと一緒に提供された
RTコンポーネントをインストールすれば
すぐに使うことが出来る

RTシステムインテグレータ

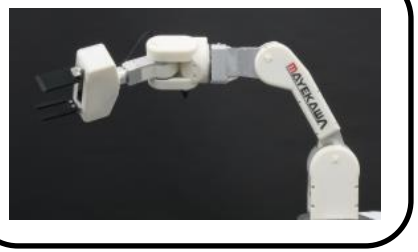
RTミドルウエアを使ったシステム開発
～単体テストが容易～

- 各RT要素のRTCの仕様に応じて，簡単に動かしてみることが出来る



RTミドルウエアを使ったシステム開発
～モジュールの接続～

●コントローラの入力に追従処理
●センサによる障害物回避
●アーム・台車の制御

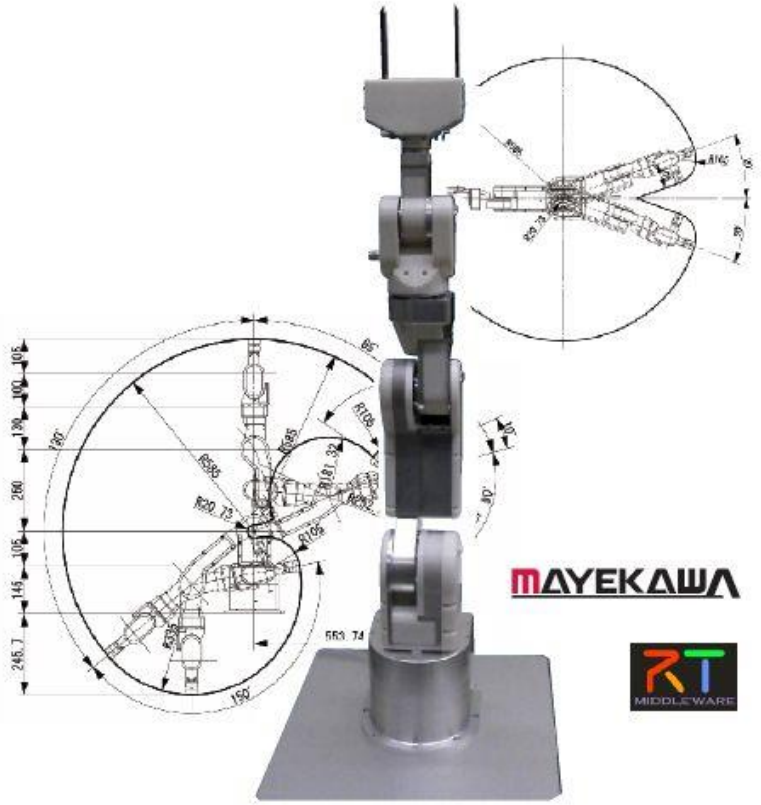


リファレンスハードウエア

- NEDO次世代ロボット知能化プロジェクト(2007～2012)
- 各プロジェクトチームからのRTコンポーネントの動作検証および統合用プラットフォームロボット

リファレンスロボットアーム「OROCHI」

- 垂直多関節型ロボットアーム
  - 全長 835[mm]
  - 重量 11.2 [kg]
  - 可搬重量 2 [kg]
  - アーム6自由度
  - グリッパー1自由度
  - 電源 24[VDC], 10[A]
  - 通信方式 CAN



専用API付属

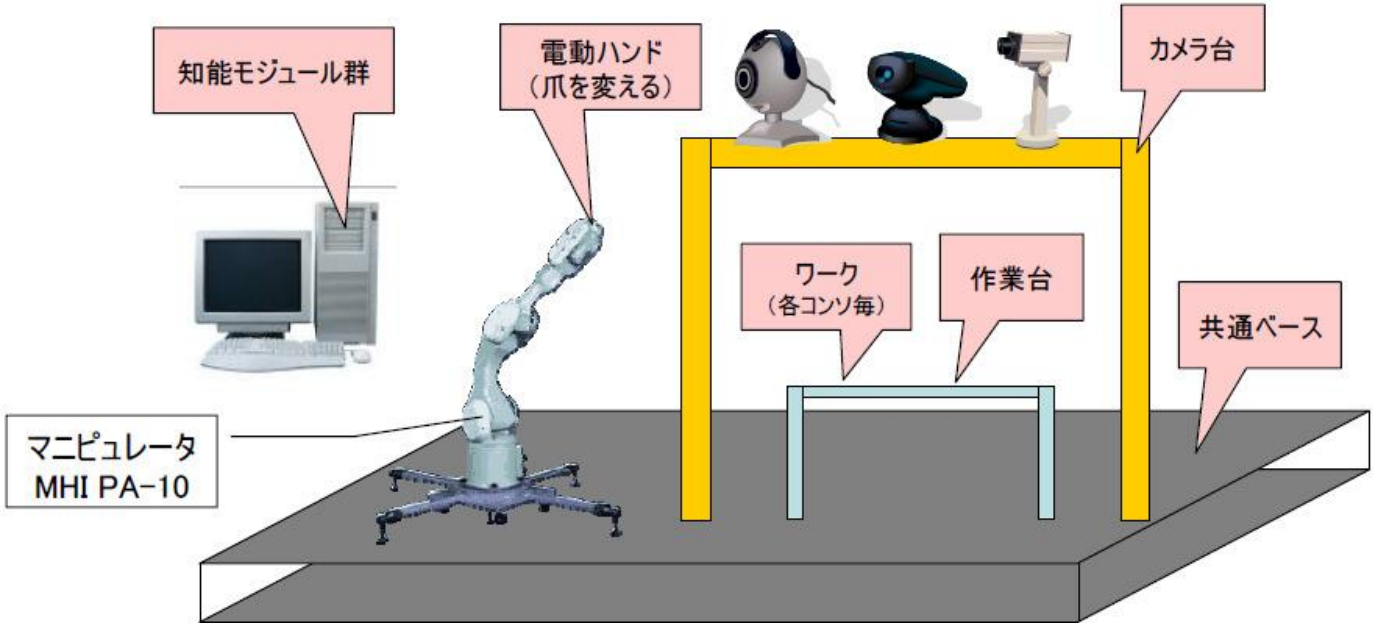
- liborochi (仮称)
  - C++, Java対応
  - Windows, Linux対応
  - 順逆運動学ライブラリ
  - 各関節制御
    - 位置速度制御
    - 速度制御
    - 電流制御
  - サンプルプログラム添付

対応RTコンポーネント付属

- 共通インターフェース仕様書
  - “コンポーネントの相互接続性や相互運用性を確保”
  - http://openrtm.org/openrtm/ja/project/Recommendation_CommonIF
  - 移動ロボット
  - コミュニケーション機能
  - ロボットアーム
  - 双腕ロボット
  - 作業系画像認識
  - カメラ機能

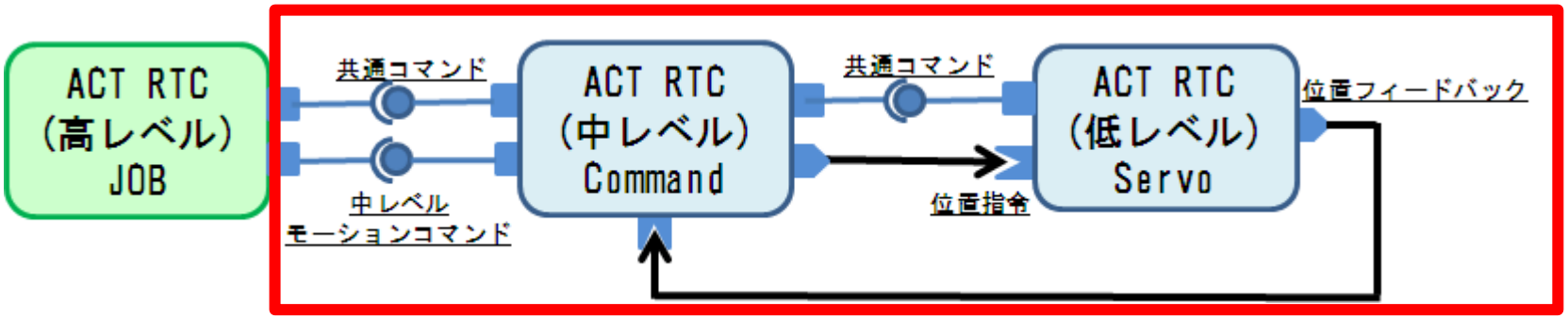
ロボットアーム制御機能 共通インターフェース

- ロボットアーム制御機能共通インターフェース仕様書(第1.0版)
  - “本仕様書では、６自由度あるいは7自由度を有するマニピュレータ及びその先端にエンドエフェクタとして１軸グリッパを取り付けたロボットアームを制御するための共通インタフェース(ACT インタフェース)を規定”
  - 座標系，データ型，単位，インターフェース（関数名）を規定



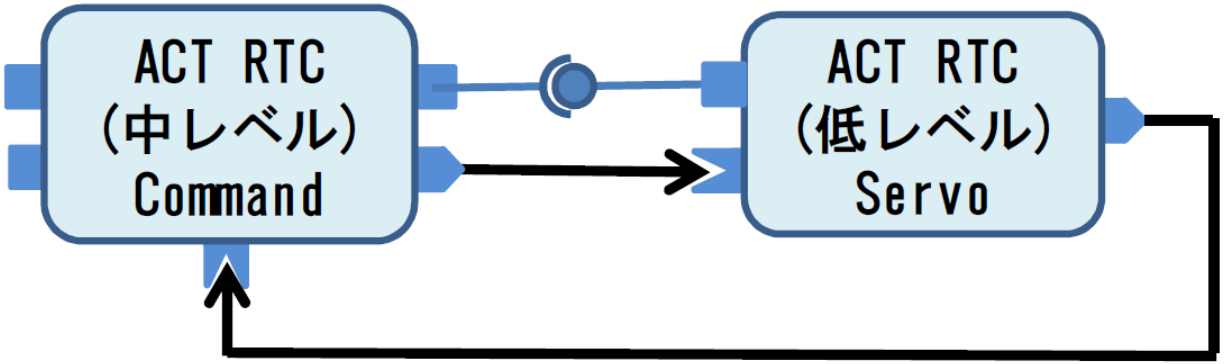
インターフェースのレベル

- 低レベル
  - 関節単位の位置を直接指令
- 中レベル
  - 関節座標空間における直線補完 (PTP制御)
  - 直交座標空間における直線補完 (CP制御)
- 高レベル
  - JOB実行を行うインターフェース



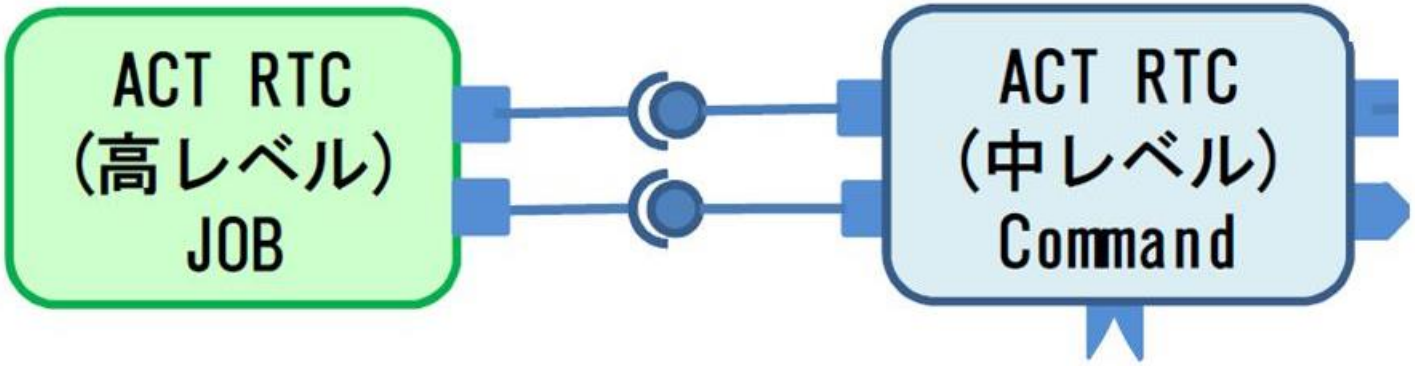
低レベルインターフェース

- 共通インターフェース
  - 「ManiplatorCommonInterface_Common」
    - 各関節位置フィードバック情報
    - 関節ユニットの状態取得
    - サーボON/OFF
- 位置指令
- 位置フィードバック



中レベルインターフェース

- 共通インターフェース
  - 「ManiplatorCommonInterface_Common」
- 中レベル共通インターフェース
  - 「ManiplatorCommonIterface_Middle」
    - グリッパ開閉
    - 手先位置フィードバック（順運動学）
    - 直交座標系における直線起動での手先移動（CP制御）
    - 関節座標系における直線起動での手先移動（PTP制御）
    - 動作時の速度設定



中レベルインターフェースを使う

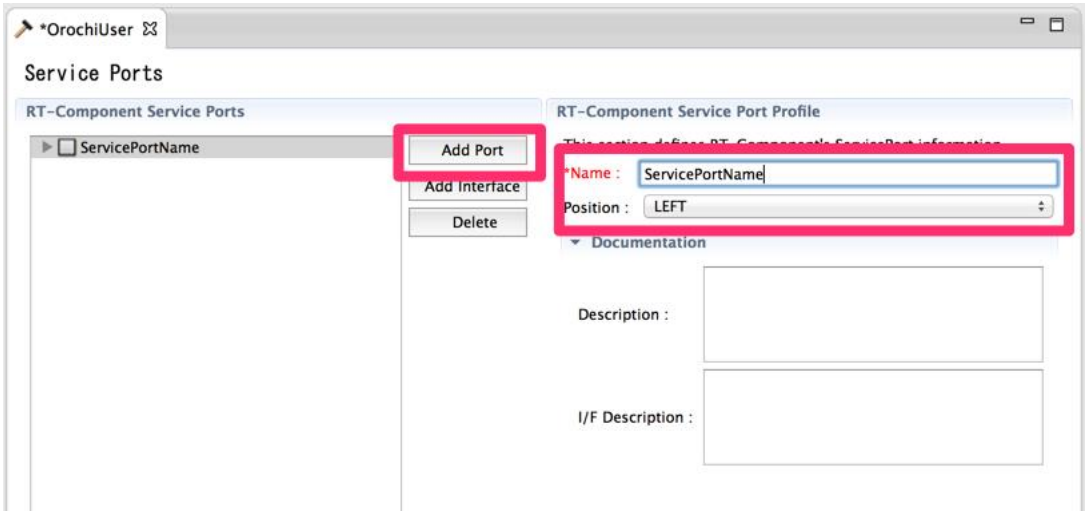
- サービスポートを使わなくてはならない
  - サービスポートは，データの交換だけでなく，関数を呼び出すのと同じ
  - 関数の宣言はIDLファイルで行う

ManipulatorCommonInterface_MiddleLevel.idl （抜粋）

```
interface ManipulatorCommonInterface_Middle  {

   RTC::RETURN_ID closeGripper();

   RTC::RETURN_ID getFeedbackPosCartesian(out RTC::CarPosWithElbow pos);

   RTC::RETURN_ID moveGripper(in RTC::ULONG angleRatio);

   RTC::RETURN_ID moveLinearCartesianAbs(in RTC::CarPosWithElbow carPoint);

   RTC::RETURN_ID movePTPJointAbs(in RTC::JointPos jointPoints);

   RTC::RETURN_ID openGripper();

};
```

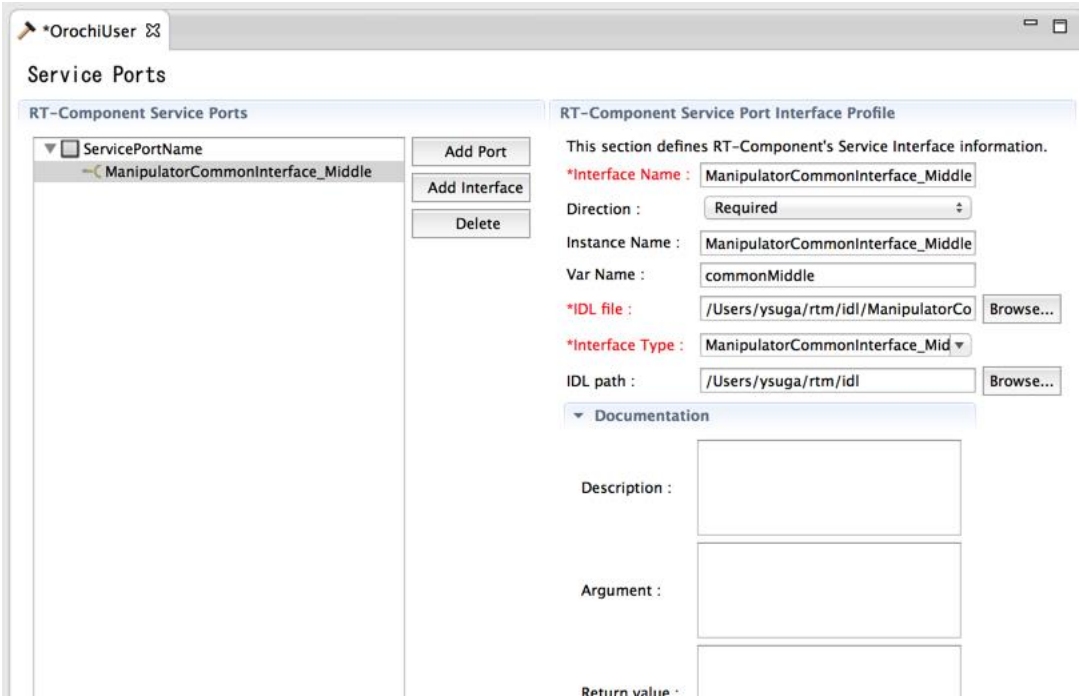
RTC Builder + ServicePort

- Add Port – (ポートを作る)
  - ポートはインターフェースの出口



RTC Builder + ServicePort

- Add Interface – (インターフェースを作る)
  - インターフェースがクラス的なもの



インターフェース名はすべて，下のインターフェースタイプと同じにしておくと便利．

使う側は「Required」

Var Name はプログラム内の変数名

IDLファイルを指定する

ManipulatorCommonInterface_MiddleLevel.idl （抜粋）

```
interface ManipulatorCommonInterface_Middle  {

   RTC::RETURN_ID closeGripper();

   RTC::RETURN_ID getFeedbackPosCartesian(out RTC::CarPosWithElbow pos);

   RTC::RETURN_ID moveGripper(in RTC::ULONG angleRatio);

   RTC::RETURN_ID moveLinearCartesianAbs(in RTC::CarPosWithElbow carPoint);

   RTC::RETURN_ID movePTPJointAbs(in RTC::JointPos jointPoints);

   RTC::RETURN_ID openGripper();

};
```

ManipulatorCommonInterface_Middle Level.idl （抜粋）

```
interface ManipulatorCommonInterface_Middle  {
  RTC::RETURN_ID closeGripper();
  RTC::RETURN_ID getFeedbackPosCartesian(out RTC::CarPosWithElbow pos);
  RTC::RETURN_ID moveGripper(in RTC::ULONG angleRatio);
  RTC::RETURN_ID moveLinearCartesianAbs(in RTC::CarPosWithElbow carPoint);
  RTC::RETURN_ID movePTPJointAbs(in RTC::JointPos jointPoints);
  RTC::RETURN_ID openGripper();
};
```

これらは呼べばいいだけ

ManipulatorCommonInterface_Middle Level.idl （抜粋）

```
interface ManipulatorCommonInterface_Middle  {

   RTC::RETURN_ID closeGripper();

   RTC::RETURN_ID getFeedbackPosCartesian(out RTC::CarPosWithElbow pos);

   RTC::RETURN_ID moveGripper(in RTC::ULONG angleRatio);

   RTC::RETURN_ID moveLinearCartesianAbs(in RTC::CarPosWithElbow carPoint);

   RTC::RETURN_ID movePTPJointAbs(in RTC::JointPos jointPoints);

   RTC::RETURN_ID openGripper();

};
```

JointPos型の引数が必要になる（言語依存）

ManipulatorCommonInterface_DataType.idl （抜粋）

```
typedef sequence<double> JointPos;
```

JointPos型はdoubleの配列

ManipulatorCommonInterface_MiddleLevel.idl （抜粋）

```
interface ManipulatorCommonInterface_Middle  {

   RTC::RETURN_ID closeGripper();

   RTC::RETURN_ID getFeedbackPosCartesian(out RTC::CarPosWithElbow pos);

   RTC::RETURN_ID moveGripper(in RTC::ULONG angleRatio);

   RTC::RETURN_ID moveLinearCartesianAbs(in RTC::CarPosWithElbow carPoint);

   RTC::RETURN_ID movePTPJointAbs(in RTC::JointPos jointPoints);

   RTC::RETURN_ID openGripper();

};
```

CarPosWithElbowは，変換行列の3x4部分

ManipulatorCommonInterface_MiddleLevel.idl （抜粋）

```
typedef double HgMatrix [3][4];


 struct CarPosWithElbow {
   HgMatrix carPos;
   double   elbow;
   ULONG    structFlag;
 };
```

HgMatrixが3x4行列の本体

```
RTC::ReturnCode_t ModuleName::onExecute(RTC::UniqueId ec_id)
{

  m_commonMiddle->openGripper();
  coil::usleep(5000*1000);

  m_commonMiddle->closeGripper();
  coil::usleep(5000*1000);

  JointPos pos;
  pos.length(6);
  pos[0] = 0;    pos[1] = 0.5;
  pos[2] = 0;    pos[3] = 0;
  pos[4] = 0;    pos[5] = 0;
  m_commonMiddle->movePTPJointAbs(pos);

  CarPosWithElbow carPos;
  m_commonMiddle->getFeedbackPosCartesian(carPos);
  std::cout << "X=" << carPos.carPos[0][3] << std::endl;
  std::cout << "Y=" << carPos.carPos[1][3] << std::endl;
  std::cout << "Z=" << carPos.carPos[2][3] << std::endl;

  carPos.carPos[2][3] += 50; //[mm]
  m_commonMiddle->moveLinearCartesianAbs(carPos);


  return RTC::RTC_OK;
}
```

幸いにも

- 動きをシミュレーション (Choreonoid)で確認できる
  - Choreonoid.org
  - 専用のプラグインが必要になる