RTミドルウエア「OpenRTM-aist」入門

ysuga.net 菅　佑樹

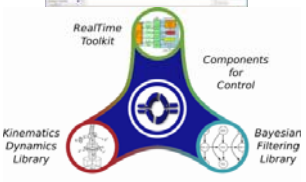
インストール

- OpenRTM-aist バージョン1.1 C++言語版
  - http://www.openrtm.org/openrtm/ja/content/110-release
    - JDK7・・・jdk-7u4-windows-i586.exe
    - Python 2.6・・・python-2.6.6.msi
    - PyYAML・・・PyYAML-3.10.win32-py2.6.exe
    - OpenRTM-aist・・・OpenRTM-aist-1.1.0-RELEASE_vc10.msi
    - GUIツール入りeclipse・・・eclipse342_rtmtools110-rc2_win32_ja.zip
- CMake 2.8
  - http://www.cmake.org/cmake/resources/software.html
    - CMake 2.8 ・・・cmake-2.8.8-win32-x86.exe
- Visual C++
  - OpenRTMが**_vc10ならばVisual C++ 2010
    - http://www.microsoft.com/ja-jp/dev/express/default.aspx
  - OpenRTMが**_vc9ならばVisual C++ 2008
    - 上記ウェブサイト下部の「過去のバージョン」のリンクから

インストールしながら・・・

- RTミドルウエアの基礎知識
  - RTミドルウエアとは
  - OpenRTM-aistとは

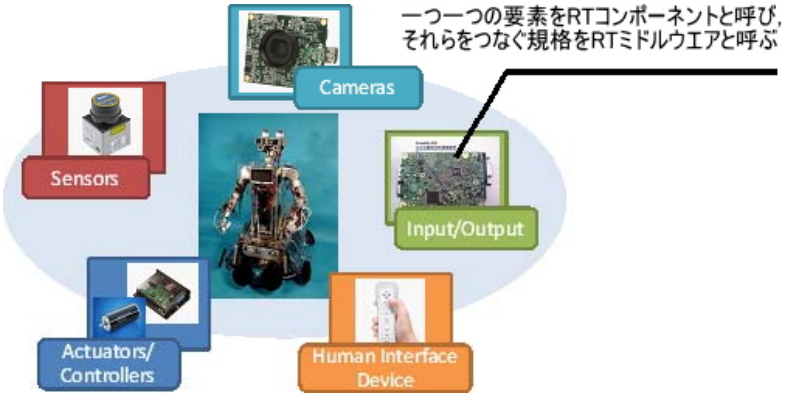
ロボット要素の規格化の流れ



- Microsoft Robotics Developer Studio
  - 物理演算が可能なシミュレータ
  - モジュール化された並列分散なシステム
- OROCOS
  - リアルタイム制御可能なカーネル
- ROS
  - 並列分散なアーキテクチャ
  - ROSコミュニティによる配布のサポート
- ORiN
  - 工場における装置の制御や監視が目的
- RTミドルウエア

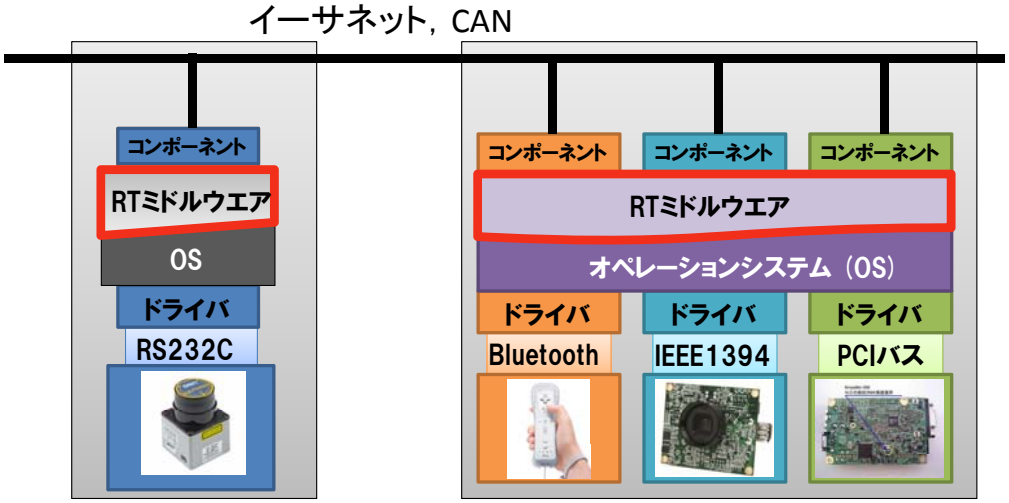
RTミドルウエアとは何か

- ロボット技術(RT)要素のソフトウエアをモジュール化するための規格
  - RT要素＝アクチュエータ，センサ，インターフェース，ソフトウエア
  - RTミドルウエアは，CORBAやUMLの規格化を行うOMG(Object Management Group)によって採択された国際標準規格



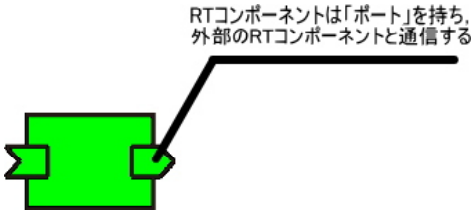
RTミドルウエアによる効果

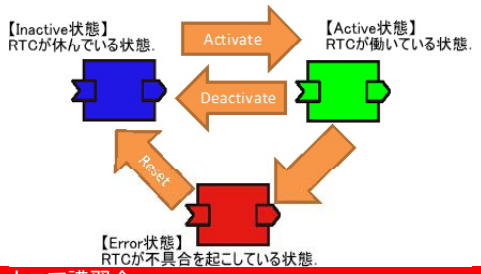
- 規格が統一化され，OSやプログラミング言語に対する依存度が低くなる
- 分散システムによる信頼性の向上
- モジュール化され，他のロボットへの再利用が容易



RTコンポーネントとは何か

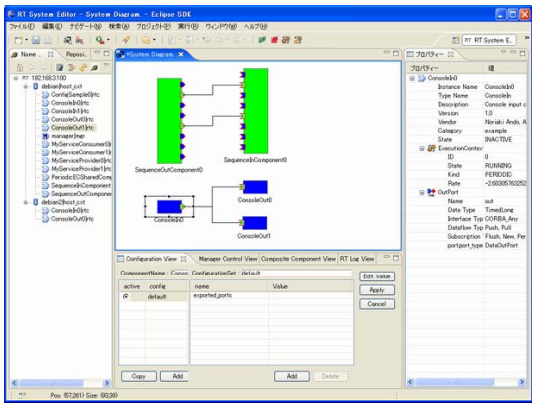
- RT要素　＝「RTコンポーネント　(Robotics Technology Component, RTC)」
  - RTCは「**ポート**」を持ち，他のRTCと接続・通信可能
  - RTCは「**状態**」を持ち，それらの遷移によってシステム管理
    - CREATED (初期化前の状態), INACTIVE (非実行状態)，ACTIVE (実行状態)，ERROR (エラー)
  - RTCは特定のタイミングで呼ばれるイベントハンドラを持つ
    - onActivated ・・・ ACTIVE化の際に一度
    - onDeactivated ・・・ INACTIVE化の際に一度
    - 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



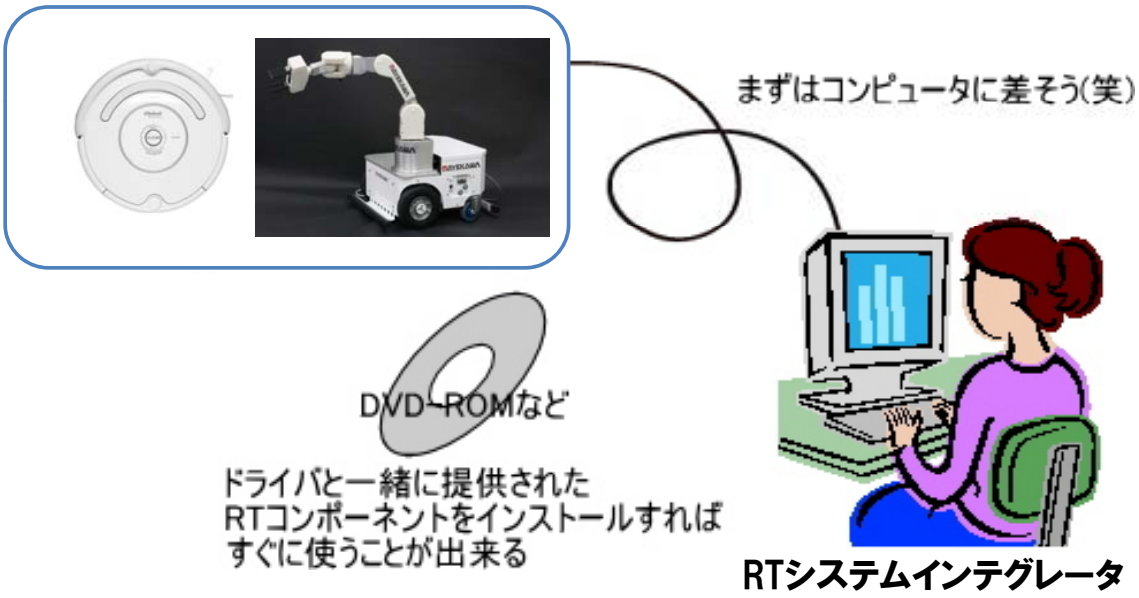


RTミドルウエアはあくまでも「規格」

- OpenRTM-aist　–産総研が開発するRTミドルウエアの実装
  - 分散オブジェクト技術としてCORBAを使用
  - 複数言語(C++,Java,Python) の使用・相互運用が可能
  - Win, Mac, Unix系対応 (Android対応)
  - グラフィカルなツールが充実
  - フリーかつオープン
  - LGPLライセンスで商用利用可能
- OpenRTM.NET
  - 株式会社セックが開発
  - .NET Frameworkを利用
- RTC-Lite (miniRTC, microRTC)
  - 組み込み用RTミドルウエア
- RTC-Safety
  - 安全認証が取られたRTミドルウエア

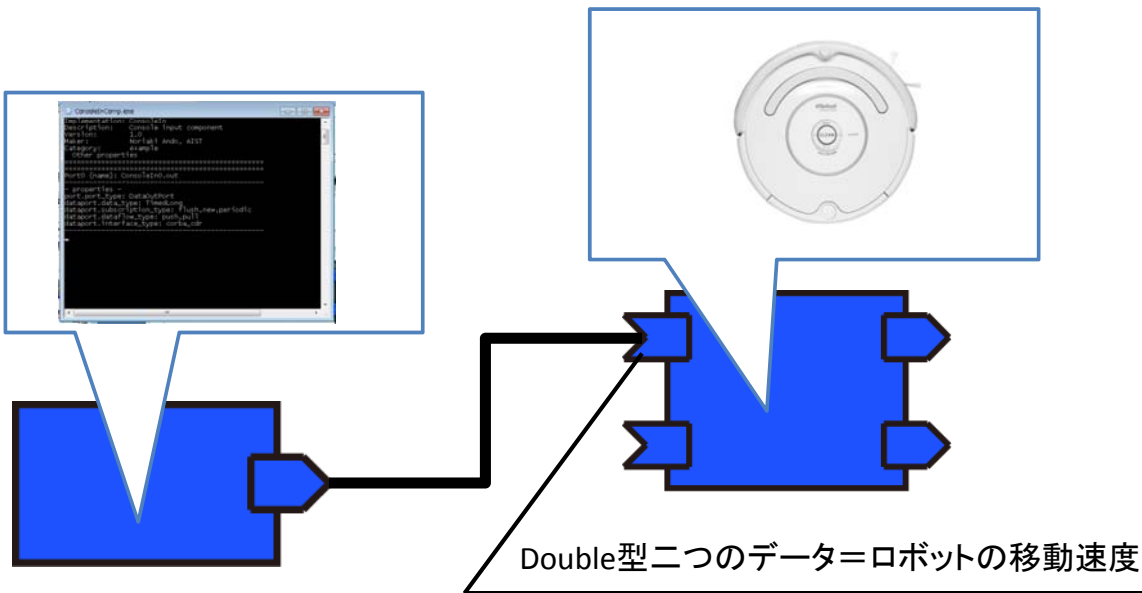


RTミドルウエアを使ったシステム開発イメージ
～モジュールを使うソフトウエアをインストール～

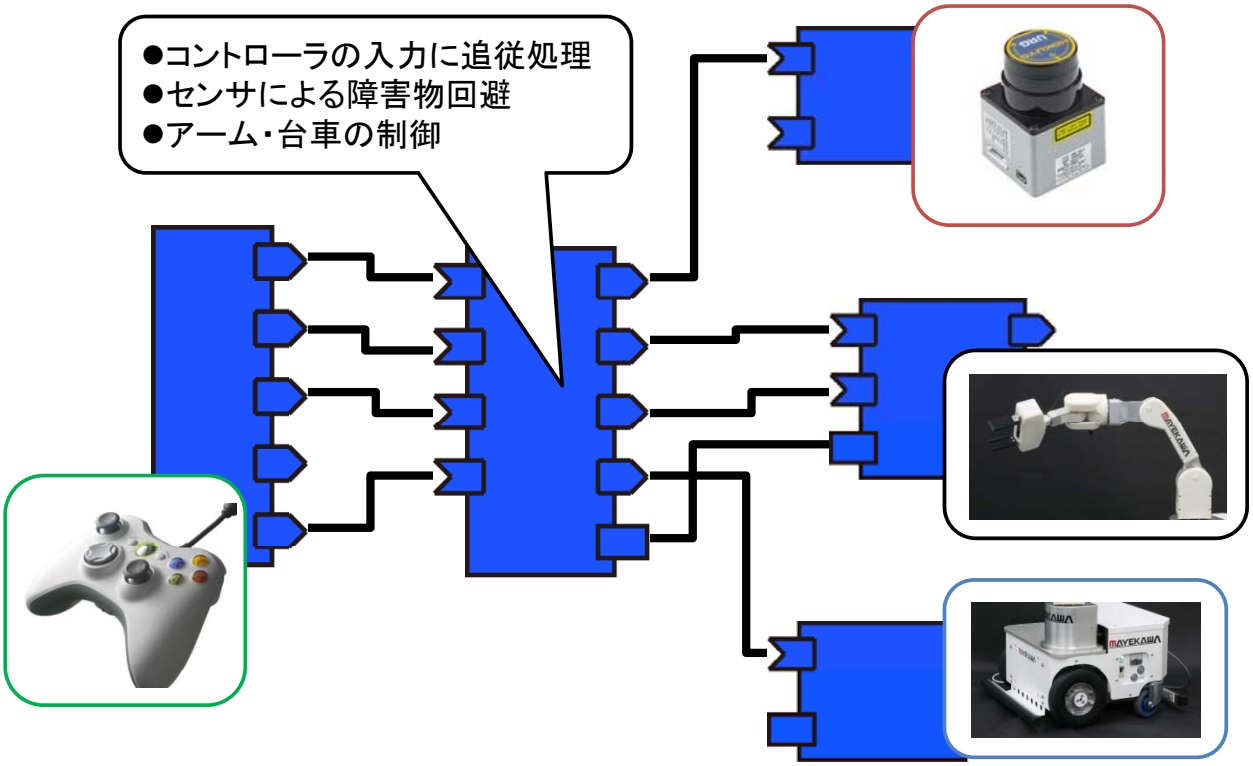


RTミドルウエアを使ったシステム開発
～単体テストが容易～

- 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～



RTミドルウエアを使ったロボット

HRP-4: Kawada/AIST

OROCHI: mayekawa

TAIZOU: General Robotics Inc.

HIRO: Kawada/GRX

HRP-4C: Kawada/AIST

利用者のメリット・デメリット

- すぐに試せて，試したRTCをそのまま再利用が可能
- 基本的にデータのやり取りなので，ポートの仕様（型，データの意味や単位）さえ分かれば簡単に使える
- フリーかつオープンソースであるため，RTミドルウエア自体をカスタマイズすることも可能（ライセンスに注意）
- ネットワークを隠ぺいするので，分散システムが容易に開発できる

- ソフトウエアのオーバーヘッドは存在する
- 初期導入時の時間的コスト
- システムのチューニング作業は不可避

開発者のメリット・デメリット

- ユーザ向けのソフトウエア・インターフェースが決定できる
- RTミドルウエア利用者には簡単に試してもらえる
- ソフトウエアのドキュメントを簡潔にできる
- 新規参入しやすい

- 初期導入時のコスト
- RTC開発自体のコスト
  - ドライバ，ライブラリに加えてRTCの開発も必要だが，Exampleにも出来るので…
- 置き換えも容易

OpenRTM-aist入門

インストール

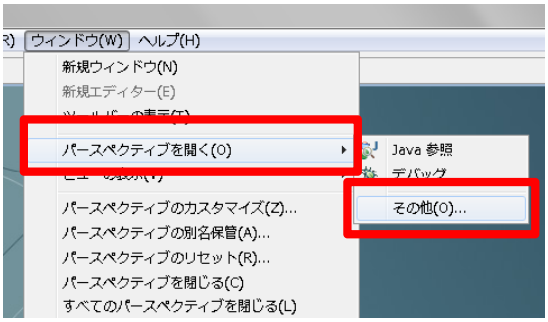
- OpenRTM-aist バージョン1.1 C++言語版
  - http://www.openrtm.org/openrtm/ja/content/110-release
  - OpenRTM-aist・・・OpenRTM-aist-1.1.0-RELEASE_vc10.msi
  - JDK7・・・jdk-7u4-windows-i586.exe
  - Python 2.6・・・python-2.6.6.msi
  - PyYAML・・・PyYAML-3.10.win32-py2.6.exe
  - GUIツール入りeclipse・・・eclipse342_rtmtools110-rc2_win32_ja.zip
- CMake 2.8
  - http://www.cmake.org/cmake/resources/software.html
  - CMake 2.8 ・・・cmake-2.8.8-win32-x86.exe
- Visual C++
  - OpenRTMが**_vc10ならばVisual C++ 2010
    - http://www.microsoft.com/ja-jp/dev/express/default.aspx
  - OpenRTMが**_vc9ならばVisual C++ 2008
    - 上記ウェブサイト下部の「過去のバージョン」のリンクから

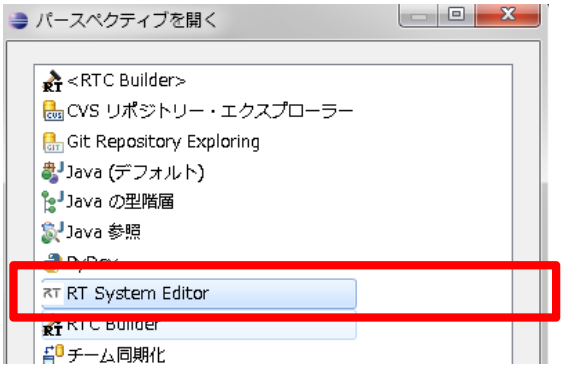
動作確認

- ネームサービスの起動
  - 「スタートメニュー」>「OpenRTM-aist 1.1」>「C++」>「tools」>「Start Naming Service」
- RTCの起動（ConsoleIn, ConsoleOut）
  - 「スタートメニュー」>「OpenRTM-aist 1.1」>「C++」>「components」>「examples」>「ConsleInComp.exe」もしくは「ConsoleOutComp.exe」
- GUIツールの起動
  - eclipse***.zipを展開
  - Eclipseフォルダ内のeclipse.exeを起動
  - ワークスペースはデフォルトでOK

RT System Editorの使い方

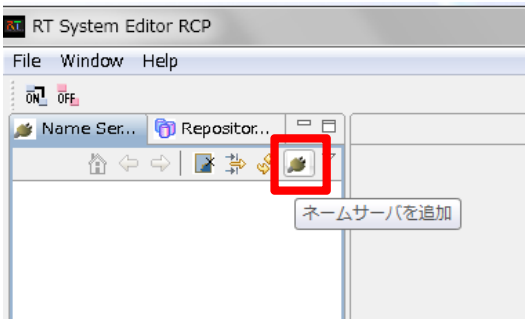
- パースペクティブを「RT System Editor」に変更
  - メニュー>「ウィンドウ」>「パースペクティブを開く」>「その他」
  - 「RT System Editor」を選択



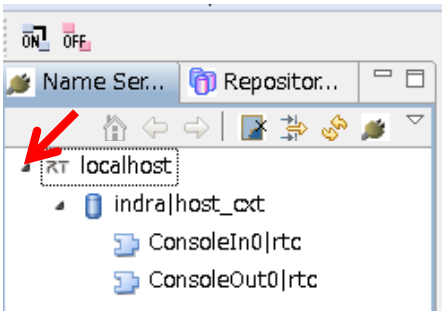


RT System Editorの使い方

- 所望のネームサーバが見つからない場合は追加処理
  - localhost (自分自身)

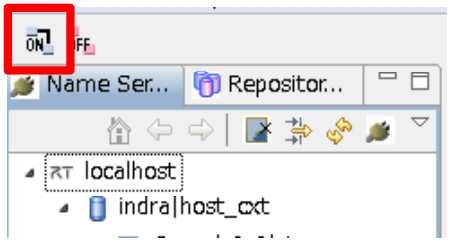


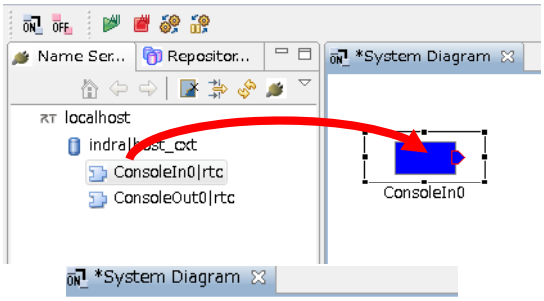
- ネームサーバに起動したRTCが登録されていれば成功

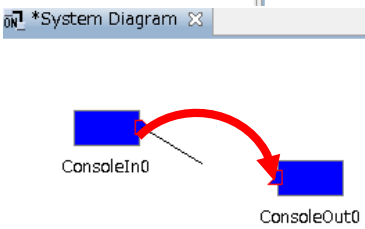


RT System Editorの使い方

- System Editorを開いてRTシステムを編集する
  - メインビューに空の「System Diagram」が表示される
- ネームサービスビューからドラッグ &ドロップ
  - 利用するRTCをすべて表示
- ポートとポートをドラッグ &ドロップで接続
  - 表示されるダイアログはOK

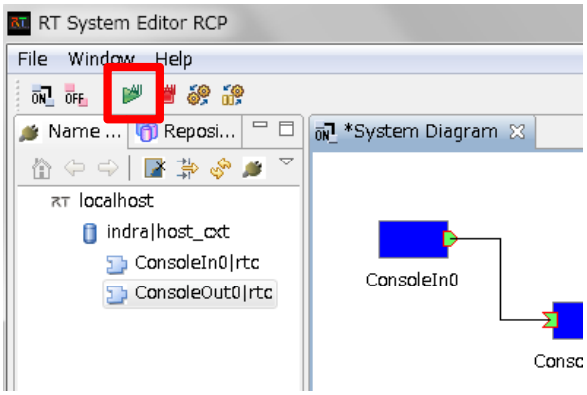


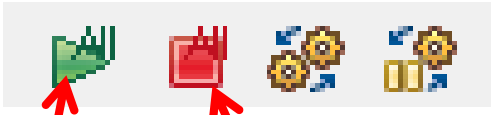




RT System Editorの使い方

- すべてのRTCをACTIVATE
- ConsoleInのウィンドウに「Please Input number」と表示
  - 適当な数字を入れるとConsoleOut側に表示
  - RTC間の通信が行われたことが分かる
- DEACTIVATEする場合は，指令を送ってからConsoleInに数字を送る必要がある
  - （scanf入力待ちになっている）





ACTIVATE　　DEACTIVATE

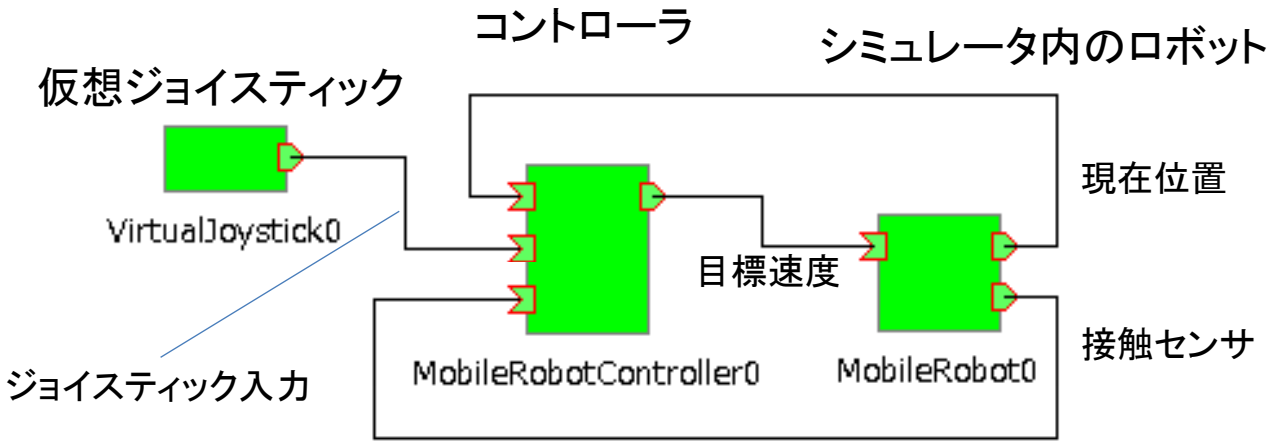
OpenRTM-aist学習用台車シミュレータ

http://ysuga.net/robot/rtm/rtc/mobilerobotsim

- Loader.batを実行
  - シミュレータ
  - 仮想ジョイスティック
  - コントローラ
- すべて接続してACTIVATE
- ジョイスティックで操作
- コントローラGUIで位置や接触センサを確認

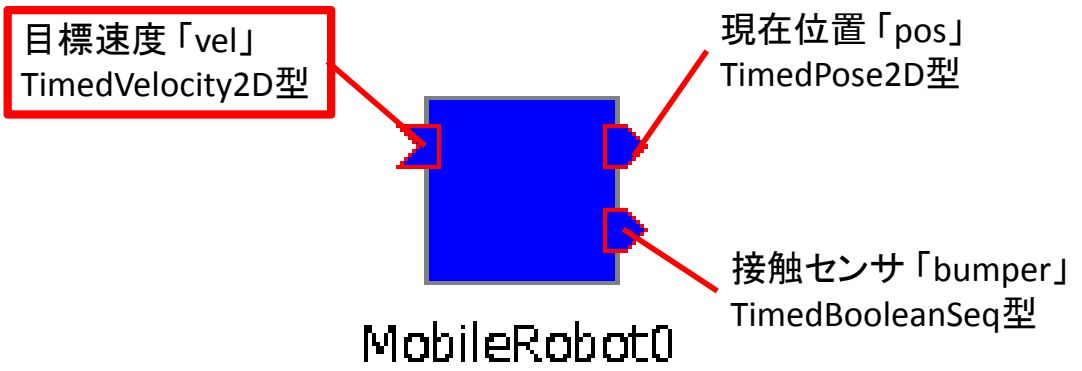
台車シミュレータの中身

- RT System Editorで表示



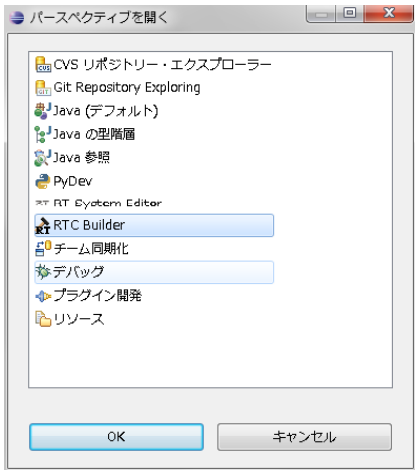
実習1. 台車に指令を送ってみよう

- 台車に直進と回転の指令を同時に送り，その場でぐるぐる回る動作をさせてみよう

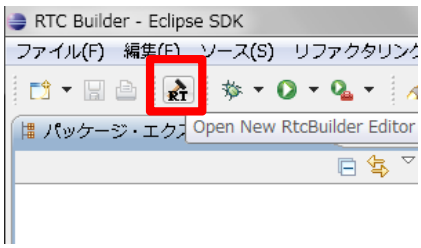


RTC Builder

- パースペクティブをRTC Builderに変更
  - Builder Editorで，RTCの骨格（スケルトン）コードを作成できる

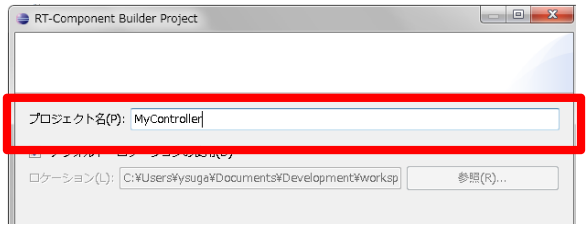


- Builder Editorを開く
  - RTCが持つべき「イベントハンドラ」が実装されているので，それを編集して自分のコードを差し込む



RTC Builder

- プロジェクト名の入力
  - MyController



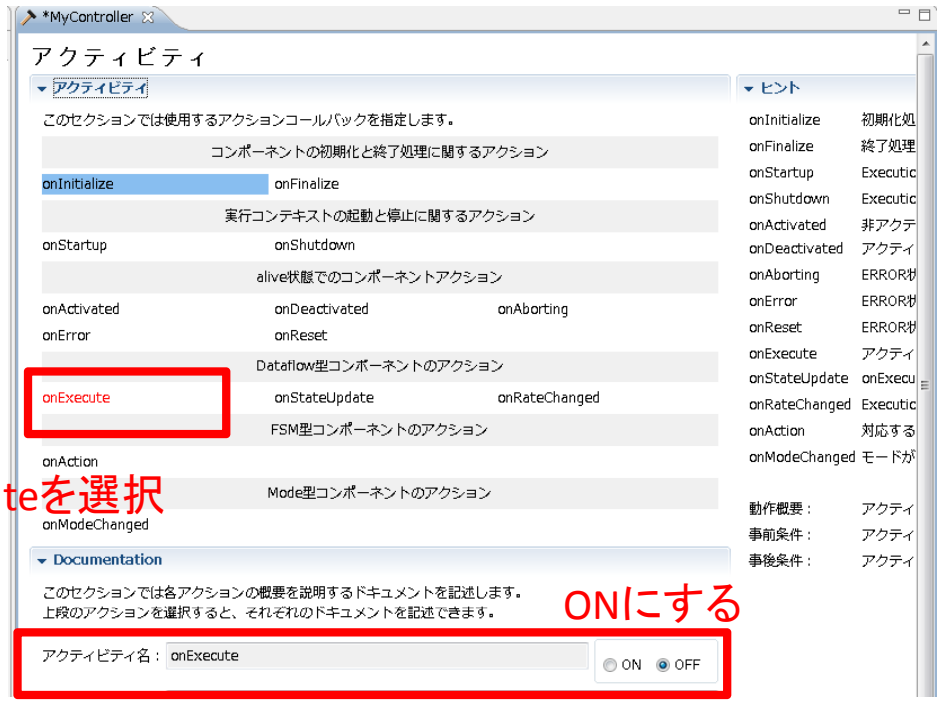
- 画面下部のタブを切り替えながら必要な情報を入力

RTC Builder

- 基本タブ

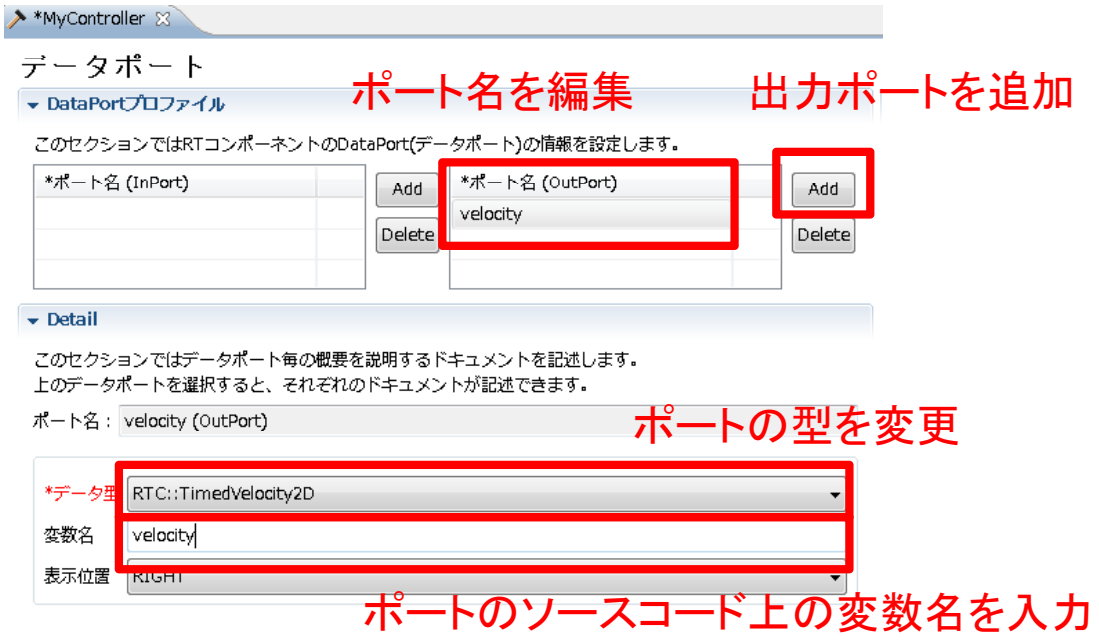
RTC Builder

- アクティビティタブ



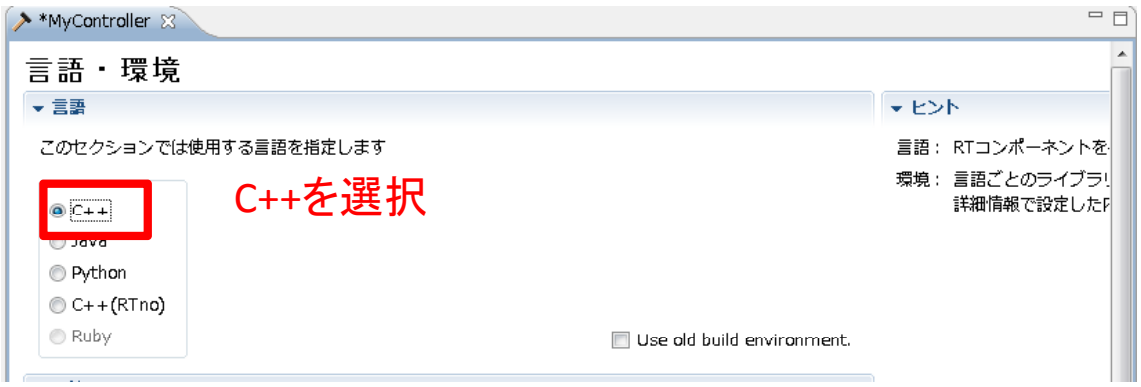
RTC Builder

- データポートタブ



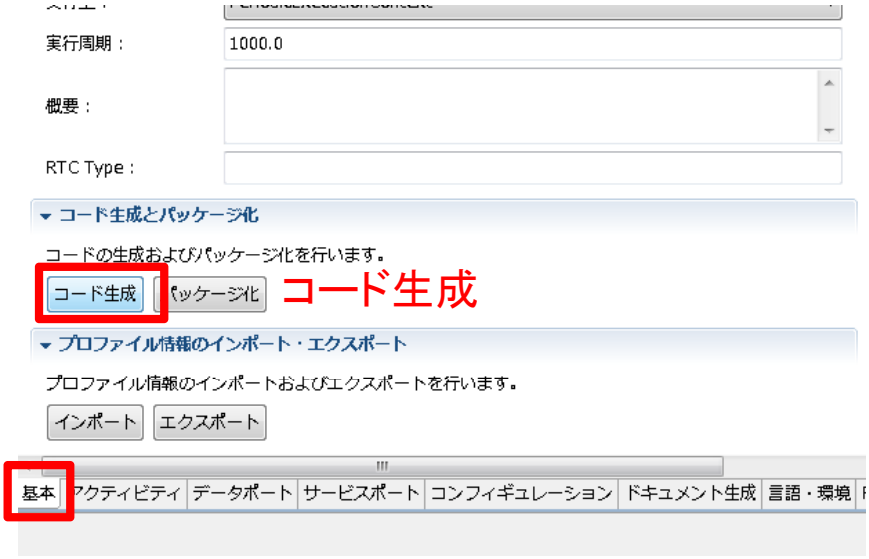
RTC Builder

- 言語タブ



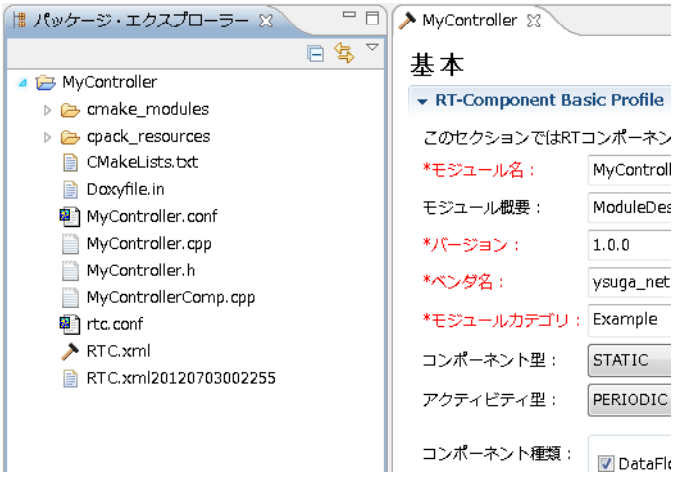
RTC Builder

- 基本タブに戻る



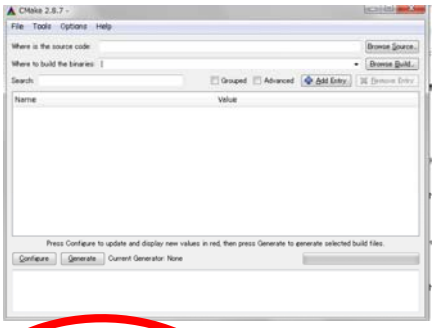
RTC Builder

- MyControllerプロジェクトにソースコードが生成される
  - Builder Editor（真ん中のウィンドウ）は閉じてよい

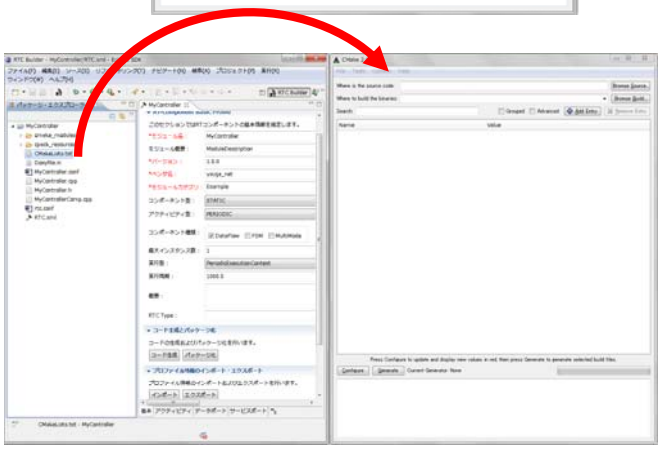


CMakeによるプロジェクト生成

- CMake2.8を起動

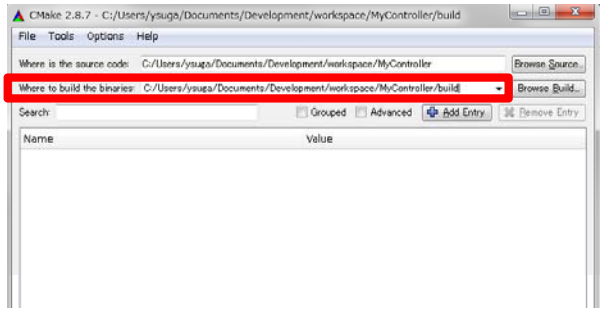


- CMakeLists.txtをCMakeにドラッグ &ドロップ

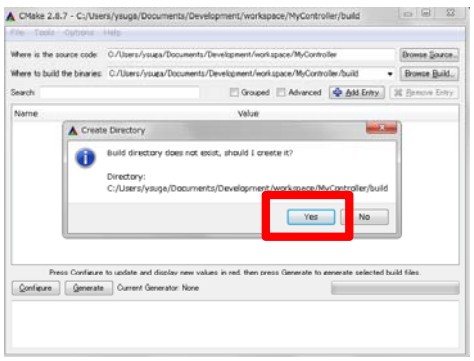


CMake2.8

- 出力フォルダパスに/buildを追加

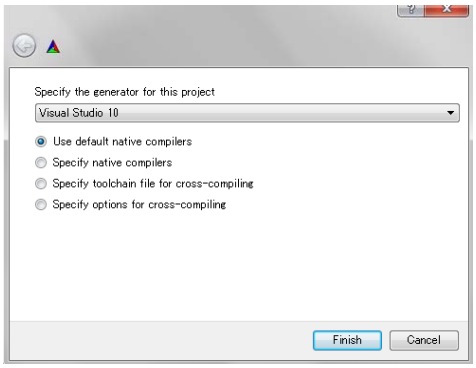


- Configureを押すとフォルダ作成確認のダイアログが出るのでOK



CMake2.8

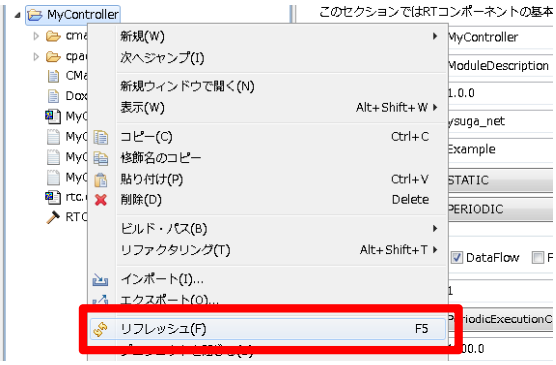
- ビルド環境を選択
  - Visual C++ 2008・・・Visual Studio 9
  - Visual C++ 2010・・・Visual Studio 10
  - Use default native compilersを選択



- 赤い表示が出るが恐れず「Generate」
  - Visual Studio用プロジェクトファイルが生成される

RTCBuilder

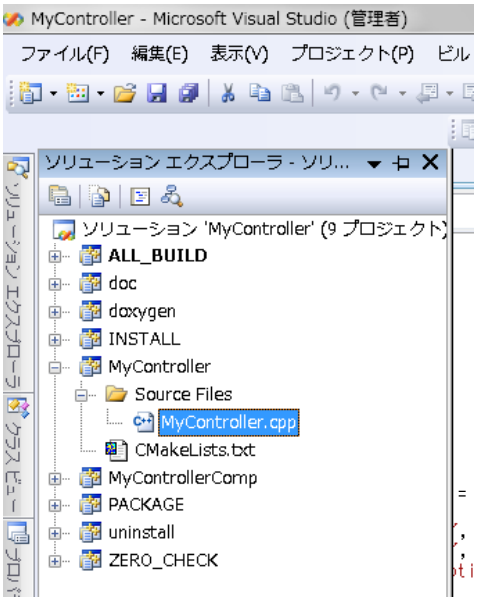
- MyControllerプロジェクトを右クリックして「リフレッシュ」
  - ファイルが加わっているのがわかる



- MyController.slnファイルをダブルクリック
  - Visual Studioが起動する

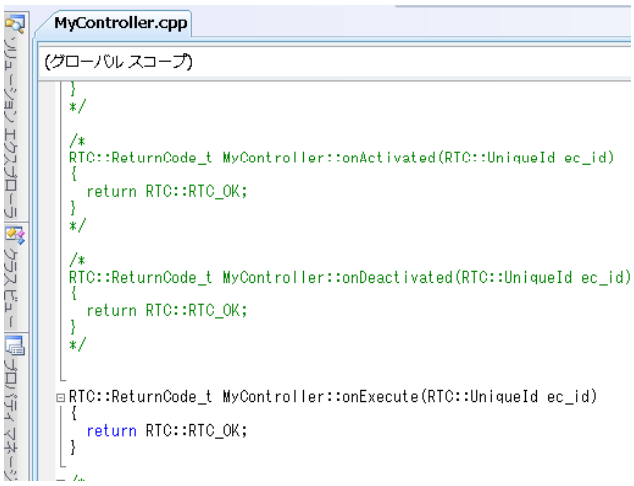
Visual Studio

- 複数のプロジェクトが存在
  - MyController‥RTC本体
  - MyControllerComp･･･RTCを単体のアプリケーションとして実行するためのプロジェクト
- MyControllerプロジェクト
  - MyController.cppを開く



Visual Studio

- MyController::onExecute関数
  - RTCがACTIVE状態の場合に周期的に呼ばれる
    - 実行周期は後で設定
  - ここに周期的に呼ばれるべき制御アルゴリズム等を記述



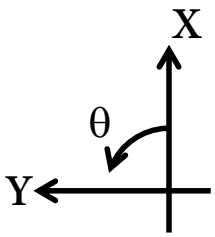
Visual Studio

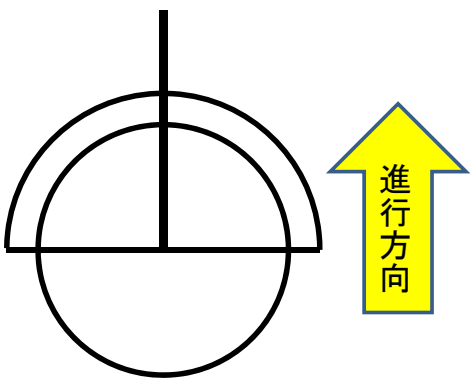
- onExecuteの記述

```
RTC::ReturnCode_t
MyController::onExecute(RTC::UniqueId ec_id)
{
        m_velocity.data.vx = 0.05; // バッファに書込む
        m_velocity.data.vy = 0;
        m_velocity.data.va = 1.0;
        m_velocityOut.write();  // データを送信
        return RTC::RTC_OK;
}
```


実習1. 台車に指令

- TimedVelocity2D・・・2次元平面での速度
  - data.vx・・・X軸方向速度 [m/s]
  - data.vy・・・Y軸方向速度 [m/s]
  - data.va・・・Z軸方向速度 [rad/s]





データポート関連変数

- 出力ポートの場合
  - 「変数名」=「example」
    - m_example
      - データポートのデータを入れるバッファ
    - m_exampleOut
      - データポート本体
- 利用方法

1. m_example　にデータを入力
2. m_exampleOut.write()

- 入力ポートの場合
  - 「変数名」=「example」
    - m_example
      - データポートのデータを入れるバッファ
    - m_exampleIn
      - データポート本体
- 利用方法

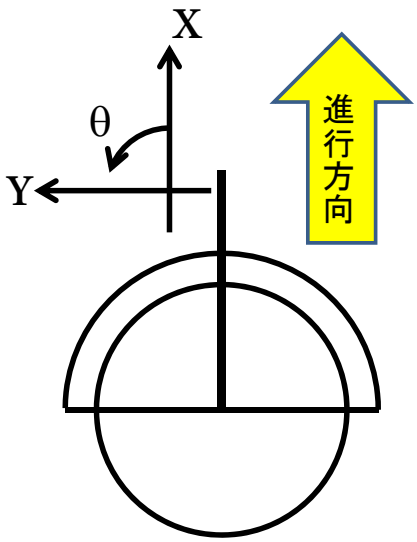
1. m_exampleIn.isNew()で受信確認
2. m_exampleIn.read()でデータ取得
3. m_exampleのデータを読み取る

実行

- ネームサービスの起動確認
- RTCの実行
  - MobileRobotSim.batを実行するとシミュレータのみ起動
- RT System Editorでの接続，ACTIVE化

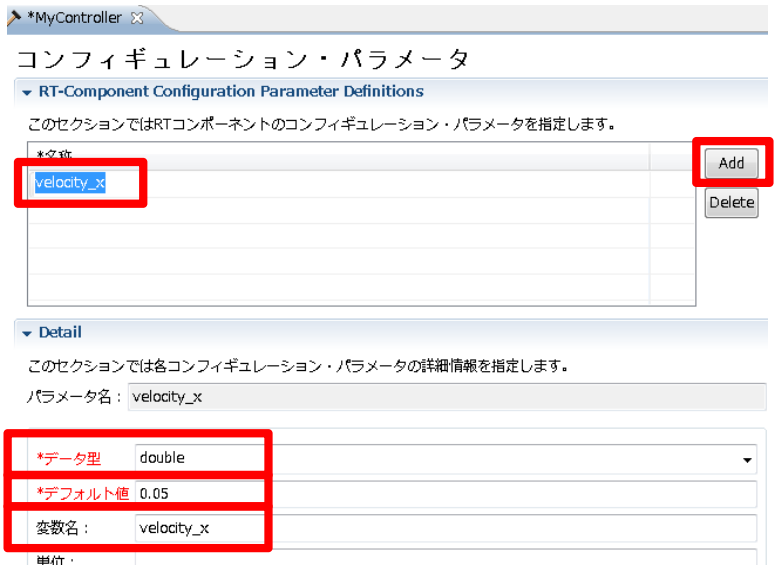
実習2. 台車の動きを調整

- コンフィグレーション機能を試す
  - コンフィグレーション機能を加えて実行中にMyControllerの機能を調整する



RTC Builder

- コンフィグレーションタブ
  - velocity_x というコンフィグレーションを追加

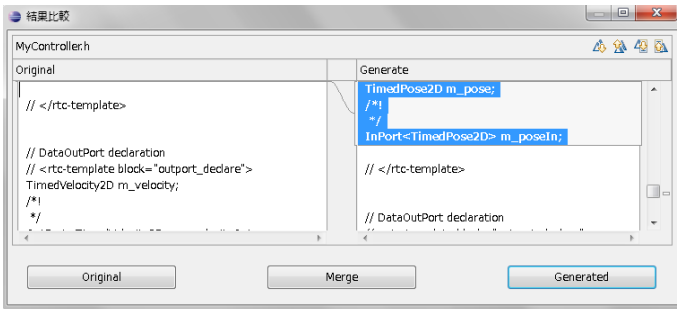


追加したコンフィグレーション

- velocity_x : double型
  - 変数名: velocity_x
  - デフォルト値: 0.05
- velocity_theta : double型
  - 変数名: velocity_theta
  - デフォルト値: 1.0

RTC Builder

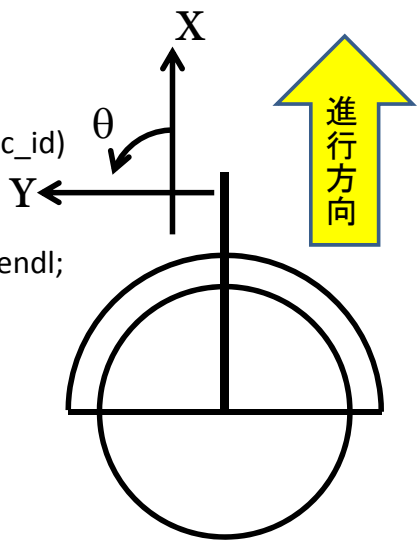
- 比較ダイアログ
  - ここでは「Generated」を選択
  - 基本的に「Merge」を選択
    - 新しいコードの変更点のみ反映
  - Generatedは新しいコード側で上書きされるので，自分の記入したコードが消える



実習2. 台車の動きを調整

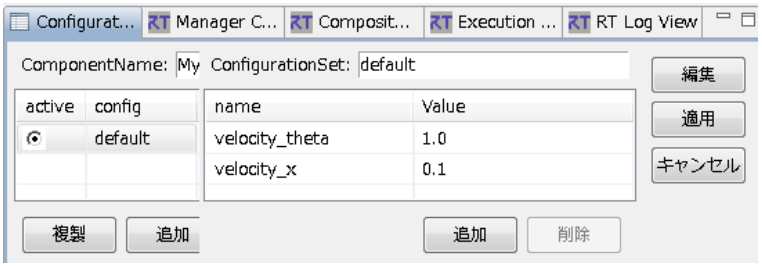
- コンフィグレーション機能を試す
  - コンフィグレーション機能を加えて実行中にMyControllerの機能を調整する

```
RTC::ReturnCode_t MyController::onExecute(RTC::UniqueId ec_id)
{
        std::cout << "Vx=" << m_velocity_x  << std::endl;
        std::cout << "Vtheta=" << m_velocity_theta << std::endl;
        m_velocity.data.vx = m_velocity_x;
        m_velocity.data.vy = 0;
        m_velocity.data.va = m_velocity_theta;
        m_velocityOut.write();
 return RTC::RTC_OK;
}
```



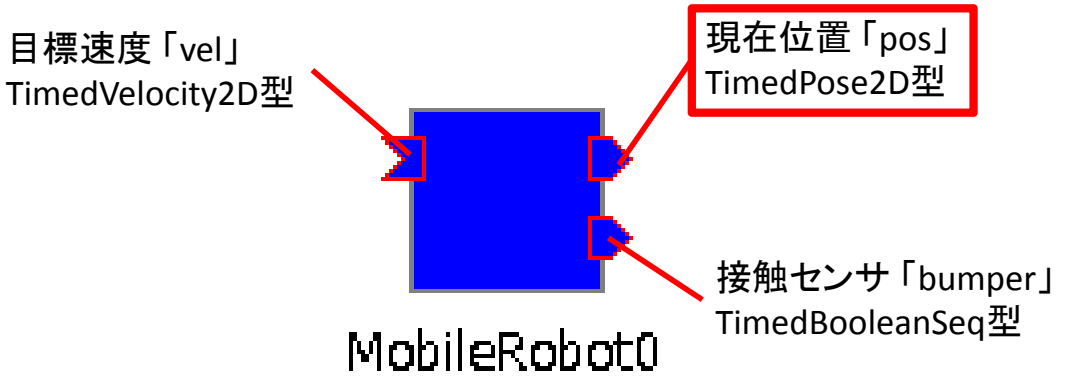
実行

- ネームサービスの起動確認
- RTCの実行
  - MobileRobotSim.batを実行するとシミュレータのみ起動
- RT System Editorでの接続，ACTIVE化
- RT System Editor下部にConfiguration Viewに選択中のRTCのコンフィグが表示されるので変更して「適用」



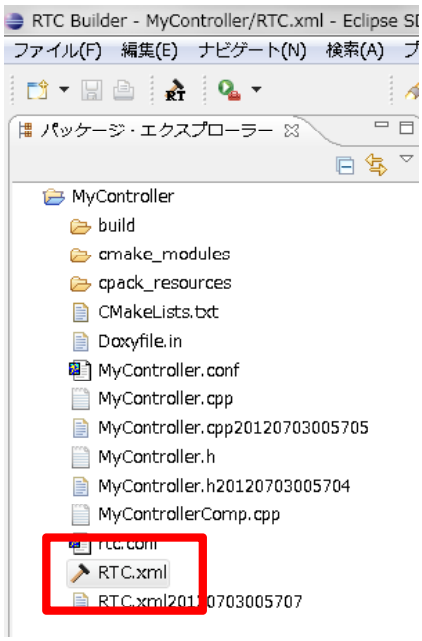
実習3. 位置の取得

- 現在位置をデータポートから受け取ってコンソールに表示する



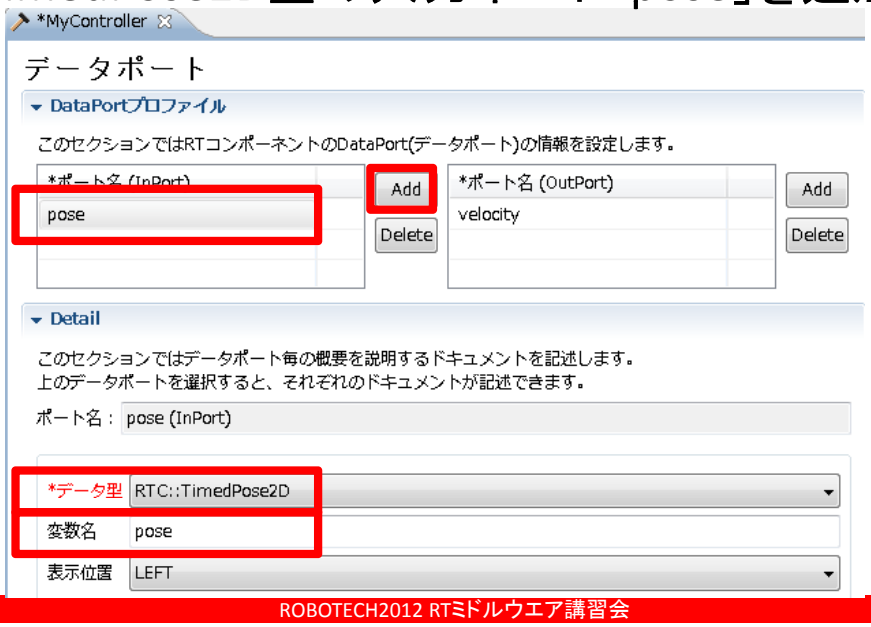
RTC Builder

- EclipseのパースペクティブをRTC Builderに再変更
- パッケージエクスプローラのRTC.xmlファイルをダブルクリックする
  - 先ほど作成した情報が記録されている



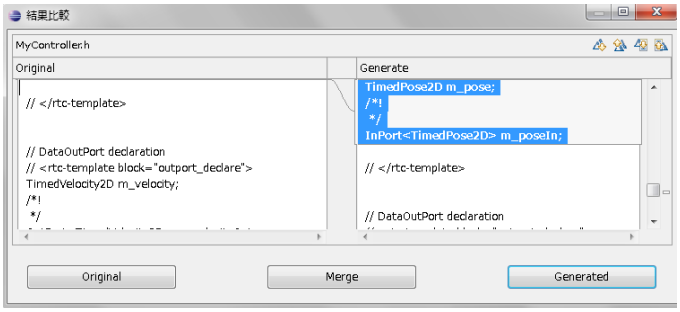
RTC Builder

- データポートタブ
  - TimedPose2D型の入力ポート「pose」を追加



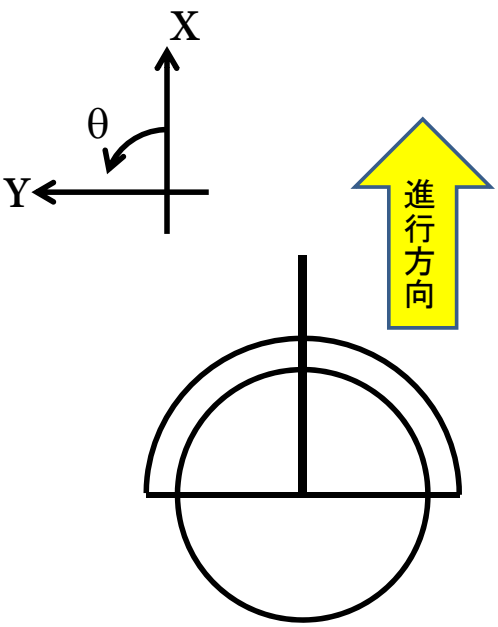
RTC Builder

- 比較ダイアログ
  - 基本的に「Merge」を選択
    - 新しいコードの変更点のみ反映
  - Generatedは新しいコード側で上書きされるので，自分の記入したコードが消えることもある



実習3. 台車の位置を表示

- TimedPose2D・・・2次元平面での位置および姿勢
  - data.position.x・・・X軸方向変位
  - data.position. y・・・Y軸方向変位
  - data.heading・・・Z軸方向回転



Visual Studio

- 再度，onExecute関数を編集
  - 入力ポートはデータが来ているか確認する処理が入る

```
RTC::ReturnCode_t MyController::onExecute(RTC::UniqueId ec_id)
{
        m_velocity.data.vx = m_velocity_x;
        m_velocity.data.vy = 0;
        m_velocity.data.va = m_velocity_theta;
        m_velocityOut.write();

        if(m_poseIn.isNew()) {          // 入力ポートに入力があるか確認
                m_poseIn.read();    // 入力があるならば読み込む
                std::cout << "X = " << m_pose.data.position.x << std::endl;
                std::cout << "Y = " << m_pose.data.position.y << std::endl;
                std::cout << "Z = " << m_pose.data.heading << std::endl;
        }
        return RTC::RTC_OK;
}
```

実行

- ネームサービスの起動確認
- RTCの実行
  - MobileRobotSim.batを実行するとシミュレータのみ起動
- RT System Editorでの接続，ACTIVE化
- 実行時の周期が速すぎる？

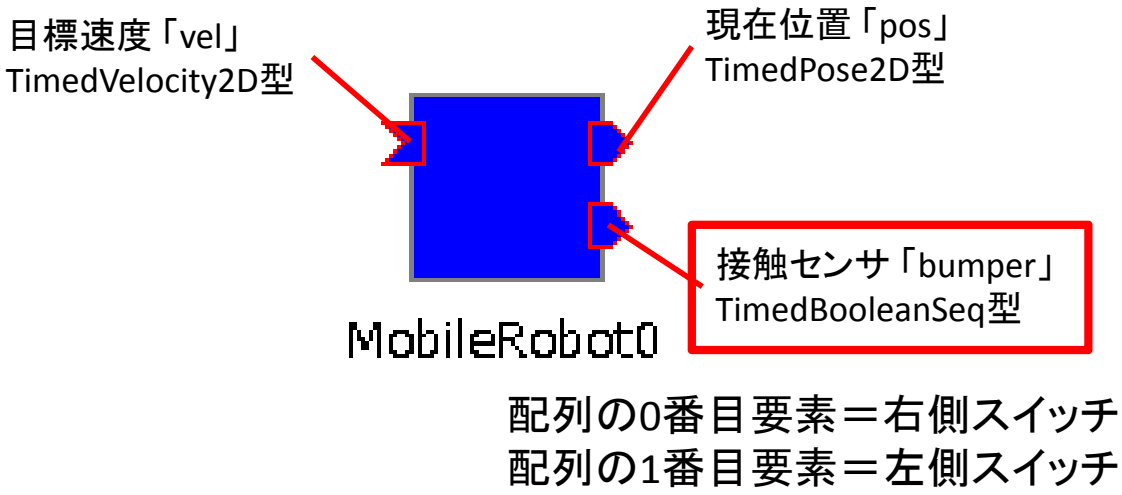
rtc.conf

- RTCの実行時の設定ファイル
  - ネームサーバのIPアドレス，ポート番号
  - 実行周期，実行コンテキストの種類
  - RTCの名前付け規則
  - ログの有無，ログレベル
  - etc…
- rtc.confで実行周期を変更（単位Hz）
  - 右クリック→アプリケーション→テキストエディタ

```
exec_cxt.periodic.rate: 1.0
```

実習4. 台車の接触スイッチ

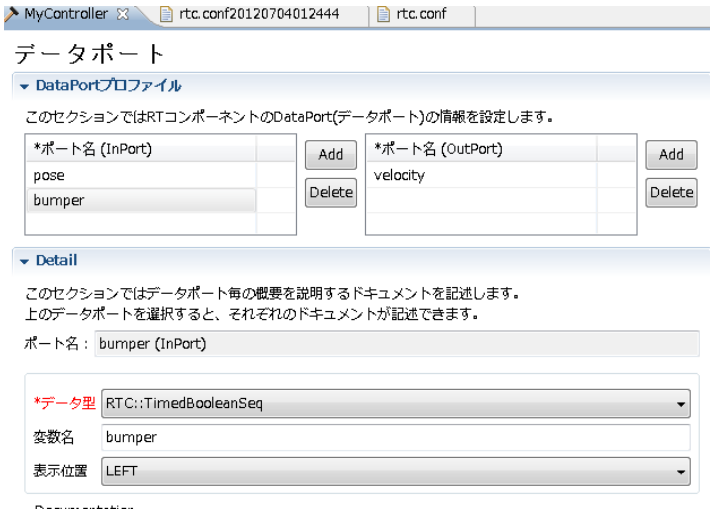
- TimedBooleanSeq　真偽型 (True/False)の配列



配列の0番目要素＝右側スイッチ
配列の1番目要素＝左側スイッチ

RTC Builderによるポートの追加

- InPort : TimedBooleanSeq型
  - ポート名: bumper
  - 変数名 : bumper



Visual Studio

- **Seq型はdataメンバを配列のように使える

```
RTC::ReturnCode_t MyController::onExecute(RTC::UniqueId ec_id)
{
        ・・・省略・・・


        if(m_bumperIn.isNew()) {
                m_bumperIn.read();
                if(m_bumper.data[0] == true) {
                        std::cout << "Right Bumper Hit!!" << std::endl;
                }else if(m_bumper.data[1]  == true) {
                        std::cout << "Left  Bumper Hit!!" << std::endl;
                }
        }
        return RTC::RTC_OK;
}
```

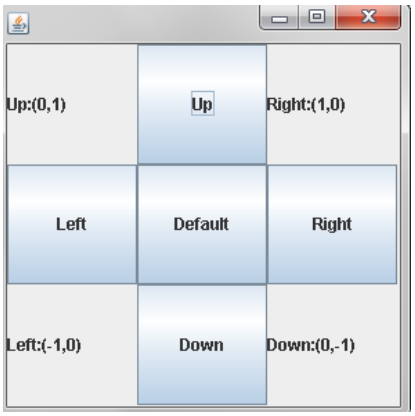

まとめ

- ツールの使い方
  - RT System Editor (RTCの接続，Activate/Deactivate)
  - RTC Builder (RTCのスケルトンコード生成)
  - CMake (Visual C++用プロジェクト生成)
- コーディング方法
  - データポート入出力
    - TimedVelocity2D, TimedPose2D, TimedBooleanSeq
  - コンフィグレーション
  - rtc.confの設定

みなさんへの課題

- 接触スイッチの反応でロボットの動作を変える
  - 左右どちらかに旋回する？
- 位置の値を使って図形を描く
  - 正方形，星型などなど
- 仮想ジョイスティックを使った操作

各ボタンの出力値はGUIに記載されている
例：Up(0,1) → 0番目要素が0，1番目要素が1



- お使いのロボット関連製品をRTC化する
  - onExecute以外の使えるイベントハンドラ例(RTC Builderで使えるように設定)
    - onActivated・・・Activate時に一回呼ばれる（初期化用）
    - onDeactivated・・・Deactivate時もしくはエラー状態遷移時に呼ばれる（終了処理）

情報源

- 公式ウェブサイト
  - http://www.openrtm.org
  - メーリングリストへの登録を勧めます
- Facebook
  - http://www.facebook.org/openrtm
- Twitter
  - ハッシュタグ #openrtm
- 拙著ウェブサイト
  - http://www.ysuga.net/robot/rtm/

関連書籍

- はじめてのコンポーネント指向ロボットアプリケーション開発 ~RTミドルウェア超入門~
- 長瀬 雅之、中本 啓之、池添 明宏 著

- UMLとRTミドルウェアによるモデルベースロボットシステム開発
- 水川 真, 大原 賢一, 坂本 武志 著

- ロボット情報学ハンドブック
  - 第3章:ソフトウエア技術
    - 3.1 概論(安藤慶昭)
    - 3.2 並列処理(山崎信行)
    - 3.3 実時間処理(加賀美聡)
    - 3.4 プログラミング言語(松井俊浩)
    - 3.5 分散処理技術(成田雅彦)
    - 3.6 ロボット用ミドルウェア(安藤慶昭)
    - 3.7 ロボット開発プラットフォーム(金広文男)
    - 3.8 標準化(水川真)

RTミドルウエアサマーキャンプ

- RTミドルウェアサマーキャンプ2012
  - 日時:2012年7月30日～8月3日
  - 場所:産業技術総合研究所 つくばセンター中央第二 ネットワーク会議室
  - 参加費:無料(ただし, 宿泊費や食事代は参加者負担. 産総研の宿泊施設を安価で提供できる予定です)
  - 学部4年生, 大学院生や企業の若手研究者などに対して, 実習形式の講習会を集中的に行い, RTミドルウェアを用いたロボット開発の機会を提供する。
  - http://openrtm.org/openrtm/ja/node/5048

RTミドルウエアコンテスト

- RTミドルウェアを利用した技術・コンポーネントに関するコンテスト
  - 日時：2012年12月18日(予定)
  - 場所：福岡国際会議場
    - 第13回 計測自動制御学会システムインテグレーション部門講演会(SI2012)の併催行事として開催予定
  - 表彰(2011年度)
    - 最優秀賞(副賞10万円)
    - 団体協賛(副賞2万円)×11件
    - 個人協賛(副賞1万円)×7件
  - 応募点数(2011年度)：１４件

お疲れ様でした

菅　佑樹　フリーランス

ysuga@ysuga.net

http://www.ysuga.net/

補足資料

OMG RTC ファミリ

Name	Vendor	Feature
OpenRTM-aist	AIST	C++, Python, Java
OpenRTM.NET	SEC	.NET(C#,VB,C++/CLI, F#, etc..)
miniRTC, microRTC	SEC	CAN・ZigBee等を利用した組込用RTC実装
Dependable RTM	SEC/AIST	機能安全認証 (IEC61508) capableなRTM実装
RTC CANOpen	SIT, CiA	CANOpenのためのCiA (Can in automation) におけるRTC標準
PALRO	富士ソフト	小型ヒューマノイドのためのC++ PSM 実装
OPRoS	ETRI	韓国国家プロジェクトでの実装
GostaiRTC	GOSTAI, THALES	ロボット言語上で動作するC++ PSM実装

同一標準仕様に基づく多様な実装により
- 実装(製品)の継続性を保証
- 実装間での相互利用がより容易に

RTミドルウェアの広がり

ダウンロード数

2012年2月現在

	2008年	2009年	2010年	2011年	2012年	合計
C++	4978	9136	12049	1851	253	28267
Python	728	1686	2387	566	55	5422
Java	643	1130	685	384	46	2888
Tool	3993	6306	3491	967	39	14796
All	10342	18258	18612	3768	393	51373

ユーザ数

タイプ	登録数
Webページユーザ	365 人
Webページアクセス	約 300 visit/day 約 1000 view/day
メーリングリスト	447 人
講習会	のべ 572 人
利用組織（Google Map）	46 組織

プロジェクト登録数

タイプ	登録数
RTコンポーネント群	287
RTミドルウエア	14
ツール	19
仕様・文書	4
ハードウエア	28

OMG RTC規格実装（11種類）

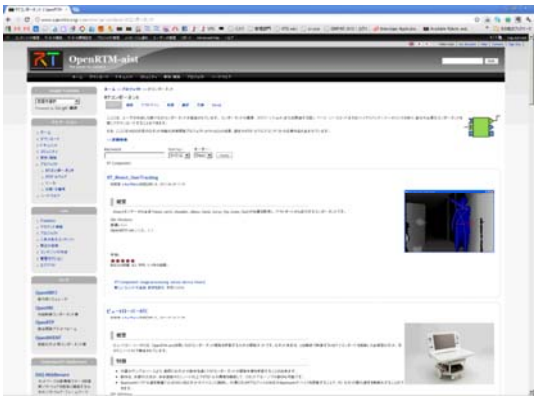
Name	Vendor	Feature
OpenRTM-aist	AIST	C++, Python, Java
OpenRTM.NET	SEC	.NET(C#,VB,C++/CLI, F#, etc..)
miniRTC, microRTC	SEC	CAN・ZigBee等を利用した組込用RTC実装
Dependable RTM	SEC/AIST	機能安全認証 (IEC61508) capableなRTM実装
RTC CANOpen	SIT, CiA	CANOpenのためのCiA (Can in automation) におけるRTC標準
PALRO	富士ソフト	小型ヒューマノイドのためのC++ PSM 実装
OPRoS	ETRI	韓国国家プロジェクトでの実装
GostaiRTC	GOSTAI, THALES	ロボット言語上で動作するC++ PSM実装

既存コンポーネントの再利用

- プロジェクトとは
  - ユーザが作成した様々なコンポーネントやツールの公開場所
  - ユーザ登録すれば、誰でも自分の成果物の紹介ページを作成可能
  - 他のユーザに自分のコンポーネント等を紹介することができる

- プロジェクトのカテゴリ
  - RTコンポーネント: 1つのコンポーネントまたは複数のコンポーネント群などが登録されています。
  - RTミドルウエア：OpenRTM-aistや他のミドルウエア、ミドルウエア拡張モジュール等が登録されています。
  - ツール：各種ツール(RTSystemEditorやrtshellを含む)ツールはこのカテゴリになります。
  - 関連ドキュメント：関連ドキュメントとは、各種インターフェースの仕様書やマニュアル等を含みます。

プロジェクトページ



タイプ	登録数
RTコンポーネント群	287
RTミドルウエア	14
ツール	19
仕様・文書	4
ハードウエア	28

既存コンポーネントの再利用

- プロジェクトから対象コンポーネントを取得
  - 「顔検出コンポーネント」

    http://www.openrtm.org/openrtm/ja/project/facedetect

    対象コンポーネントをダウンロード