RTミドルウェアによるロボットプログラミング技術

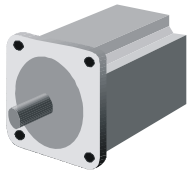
1．コース概要

ロボットを構成する要素

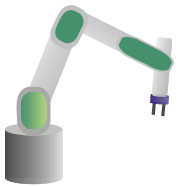
- センサ
  - エンコーダ
  - 力・磁気・加速度など
- アクチュエータ
  - モータなど
- 機構（メカ）

＋

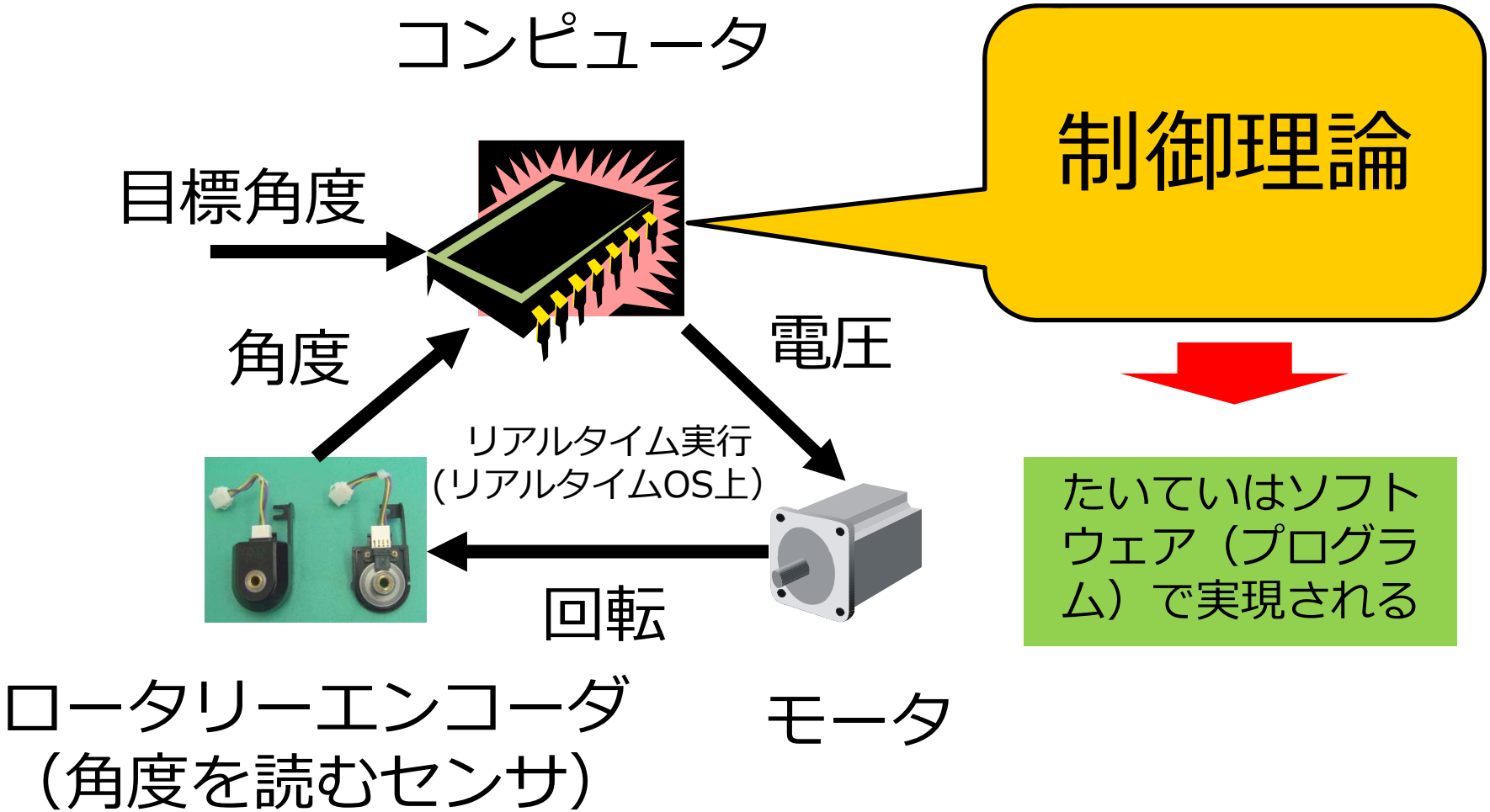
- コンピュータ
  - ソフトウエア

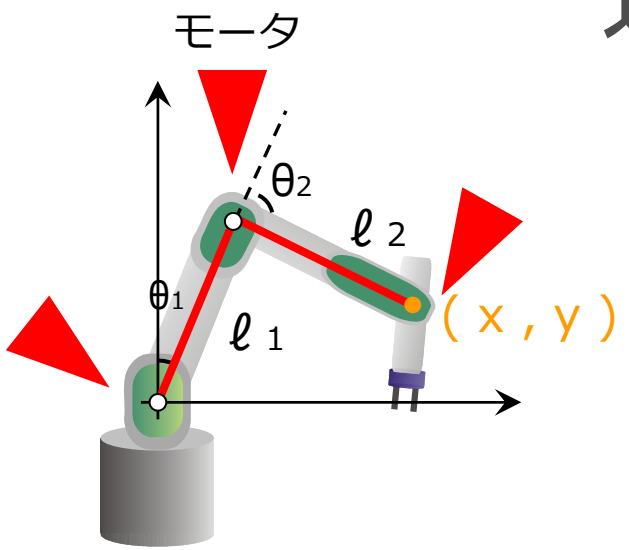



フィードバック制御

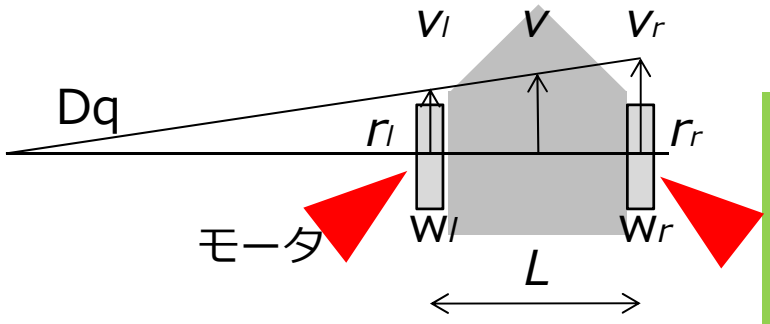

コンピュータ
制御理論
目標角度
角度
電圧
リアルタイム実行
(リアルタイムOS上)
回転
たいていはソフトウェア（プログラム）で実現される
ロータリーエンコーダ
（角度を読むセンサ）
モータ

運動学



ロボットアームの運動学

運動学（kinematics）とは位置の移動とその時間変化を対応させて考える学問であり、数理的な手法であり、物理学の原理を基礎としていない。一般的に物体の状態は、移動と回転の運動学の両方を兼ね合わせて記述する。

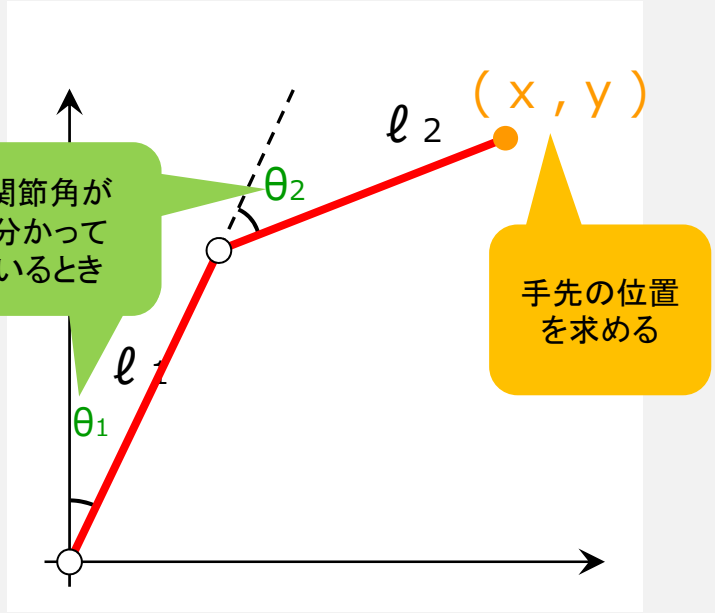


移動ロボットの運動学

ロボットの機構（メカ）の運動を数式で（プログラムとして）記述する

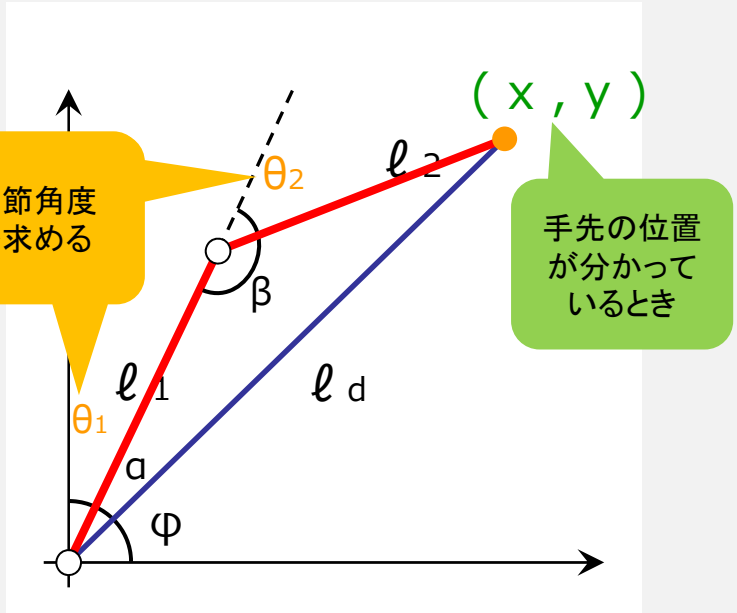
順運動学・逆運動学（2自由度の場合）

- 順運動学



関節の角度がわかっているとき、XY座標を求めることができる

- 逆運動学



XY座標がわかっているとき、関節の角度を求めることができる

注：どの場合も腕の長さははじめから与えられている

移動ロボットの制御

内界センサ：車輪の角速度やジャイロ等ロボットの内部の情報を計測するセンサ
外界センサ：レーザーや音波、カメラ、GPS等ロボットの外部の情報を取得するセンサ

自己位置推定 (Localization, ローカリゼーション、ローカライゼーション)：種々のセンサを利用し、ロボットの現在の位置を推定する技術。移動ロボットを制御するために最も基本的かつ必要とされる技術。

SLAM (スラムと読む、Simultaneous Localization and Mapping)：外界センサを用いて、ロボット周辺のマップを作成しながら同時に自己位置も推定する技術。センサには、レーザー（2次元、3次元）やカメラ、音波などが用いられる。相対位置を比較的安定的に推定できる。

パスプランニング (Path Planning)：与えられたマップ上で、現在位置から目的地までの経路を計画する方法。

ナビゲーション(Navigation): 現在位置を推定しながらロボットを目的地まで移動させること。

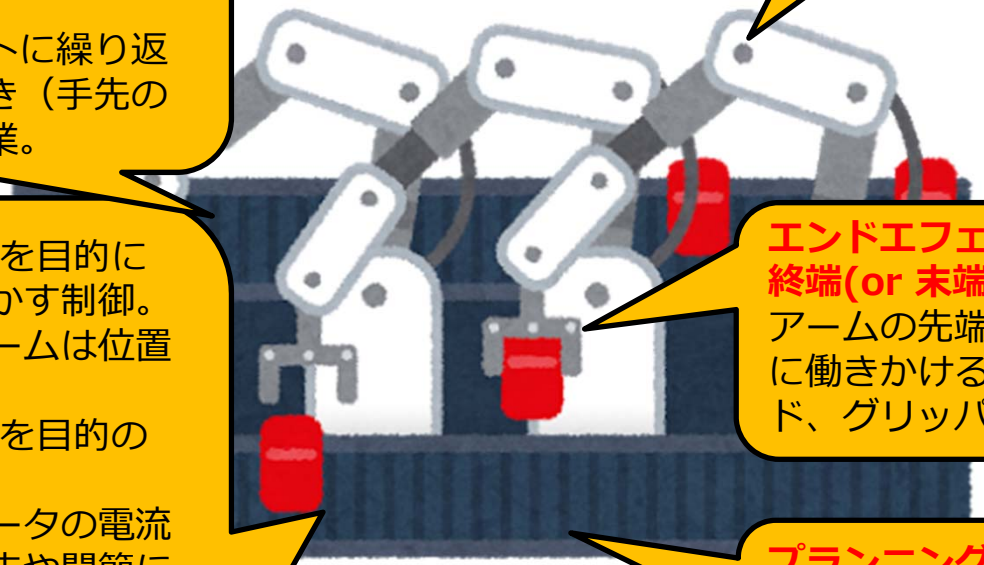
デッドレコニング (Dead Reckoning): 車輪のエンコーダやジャイロ等内界センサのみ利用する自己位置を推定手法。誤差が蓄積するため長時間使用できない。オドメトリ(Odometry) と呼ばれることもある。

ロボットアームの制御

ロボットアーム：物体を把持するなどしてある位置からある位置まで移動させることを目的としたロボット。マニピュレータとも呼ばれる。

ティーチング・プレイバック：ロボットにあらかじめ覚えさせた動作を繰り返しさせること、またそうした利用方法。
ティーチング：ロボットに繰り返し動作させるための動き（手先の軌道）を覚えさせる作業。

位置制御： 手先や関節を目的に位置または角度まで動かす制御。ほとんどのロボットアームは位置制御で利用されている。
速度制御： 手先や関節を目的の速度で制御する方法。
力制御：カセンサやモータの電流を利用することで、手先や関節にかかる力を制御する方法。主に接触、倣い動作（コンプライアンス制御）が必要な作業で利用される。



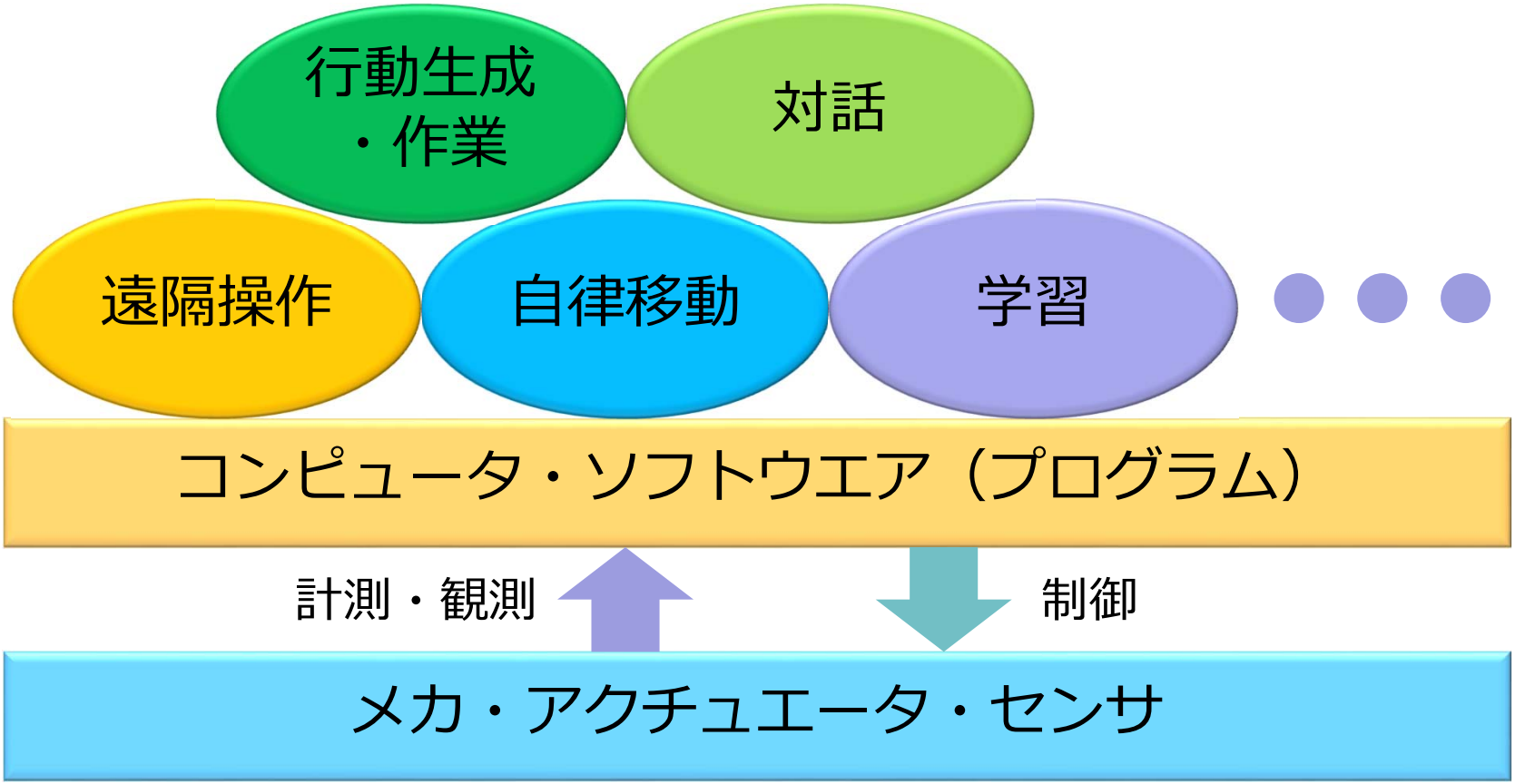
自由度 (Degree of freedom): 制御できる軸の数。3次元空間内で、物体を任意の位置姿勢へ移動させるためには、6自由度以上が必要。

エンドエフェクタ (End-effecter 終端(or 末端、手先）効果器)：アームの先端に取り付けられた外界に働きかける部分。≒ロボットハンド、グリッパなどとも呼ばれる。

プランニング: 現在位置から目的位置まで（主に手先を）どのようにアームを制御し動かすかを計画すること、またはその手法。

ロボットを構成する要素

これらのほとんどは
ソフトウエア（プログラム）として
実現されコンピュータ上で実行される



概要

1. ロボットシステムプログラミングの現状
2. ロボットOS・ミドルウェア
3. RTミドルウェア(RTM)を用いたロボット開発

ロボットシステムプログラミングの現状

コンピュータとソフトウエア

パソコン、ロボット、組込みシステム

プログラム (高級言語)	プログラム (高級言語、C・アセンブリ)	プログラム (C言語、アセンブリ)
OS Windows・Macなど	OS RTOS, 汎用OS (Windows･UNIX)・組込OS(uITRON)・OS無し, etc…	OS 無し、uITRON，etc…
パソコン	コンピュータ	マイコン
PC・周辺機器	ロボット	組込みシステム

ロボットのシステム構成：
組込機器と汎用OS（のヘッドレスシステム）的側面を持つ

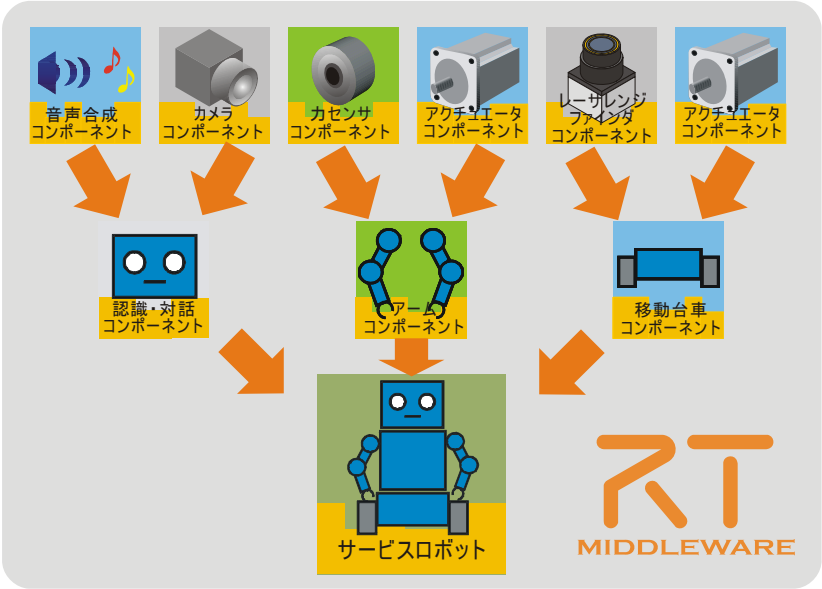
ロボットソフトウェア基盤（ロボットOS）

- 現状 ROS，RT-Middleware, OROCOS, OPRoS, NAOqi, etc. がある

モジュール（コンポーネント）指向でソフトウェアを作成し組み合わせる手法が主流

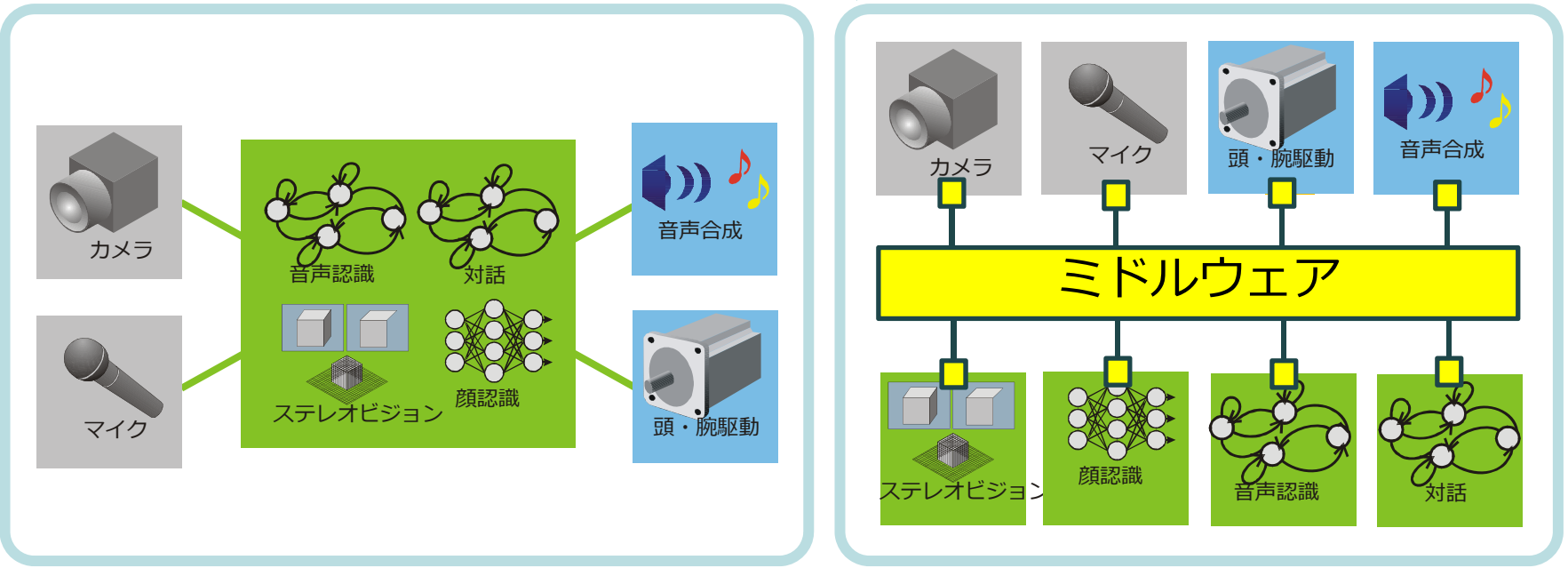
- 再利用性
- 柔軟性
- 開発の容易さ
- 信頼性

等の面から、こうしたミドルウェア・ロボットOSを利用しないことは、世界的潮流から見ても不利である。



ロボットソフトウェア開発の方向性

従来型開発 → コンポーネント指向開発



- ✓ 様々な機能を融合的に設計
- ✓ 実行時効率は高いが、柔軟性に欠ける
- ✓ 複雑化してくると開発が困難に

- ✓ 大規模複雑な機能の分割・統合
- ✓ 開発・保守効率化（機能の再利用等）
- ✓ システムの柔軟性向上

ミドルウエア・コンポーネント等

- ミドルウエア
  - OSとアプリケーション層の中間に位置し、特定の用途に対して利便性、抽象化向上のために種々の機能を提供するソフトウエア
  - 例：RDBMS、ORB等。定義は結構曖昧
- 分散オブジェクト（ミドルウエア）
  - 分散環境において、リモートのオブジェクトに対して透過的アクセスを提供する仕組み
  - 例：DDS、CORBA、Ice、Java RMI、DCOM等
- コンポーネント
  - 再利用可能なソフトウエアの断片（例えばモジュール）であり、内部の詳細機能にアクセスするための（シンタクス・セマンティクスともにきちんと定義された）インターフェースセットをもち、外部に対してはそのインターフェースを介してある種の機能を提供するモジュール。
- CBSD（Component Based Software Development）
  - ソフトウエア・システムを構築する際の基本構成要素をコンポーネントとして構成するソフトウエア開発手法

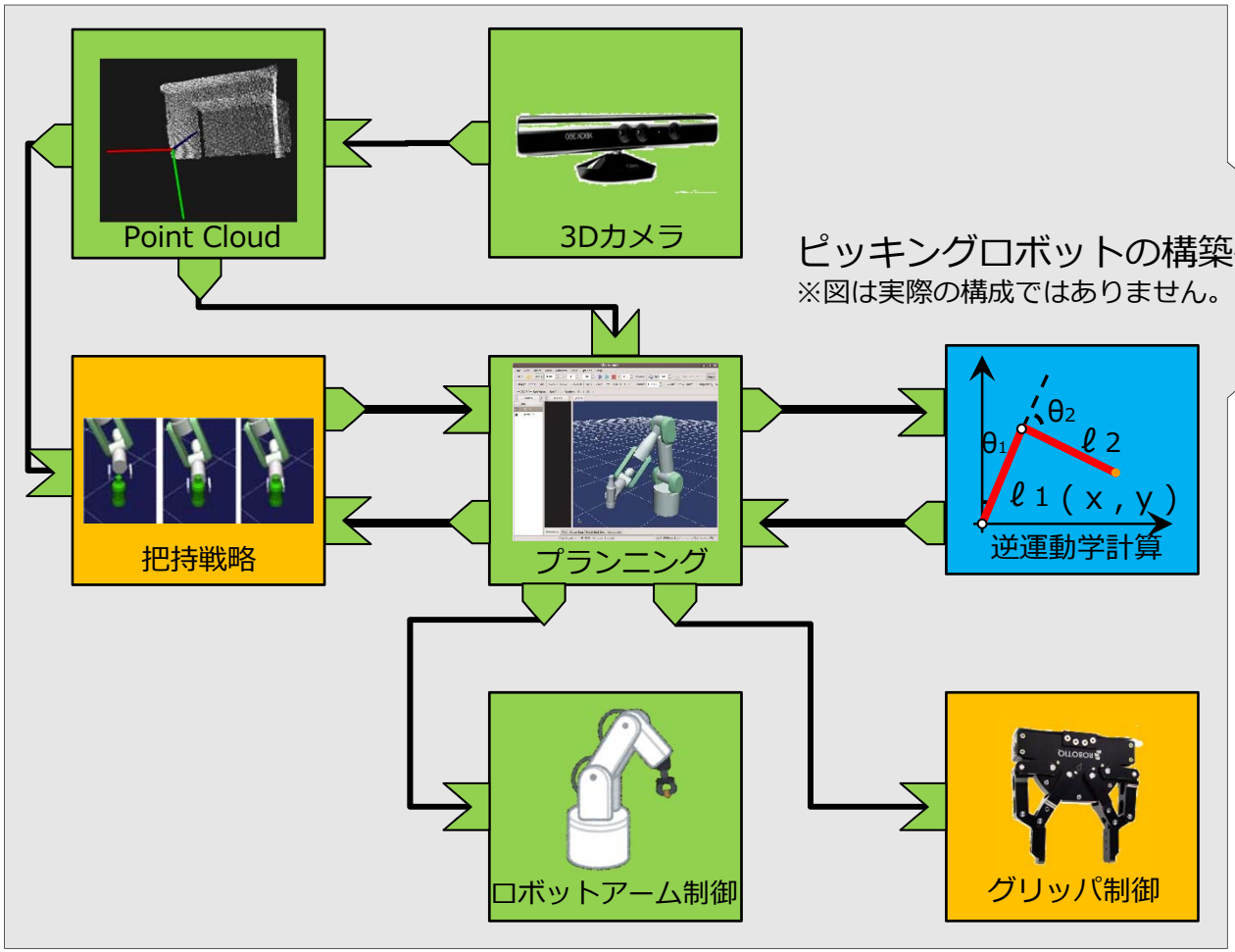
モジュール化のメリット

- 再利用性の向上
  - 同じモジュールを様々なシステムに使いまわせる
- 選択肢の多様化
  - 同じ機能を持つ多様なモジュールを利用可能
- 柔軟性の向上
  - モジュールの接続構成を変えるだけで様々なシステムを構築可能
- 信頼性の向上
  - モジュール単位でテスト可能なため信頼性の向上につながる
- 堅牢性の向上
  - システムがモジュールで分割されているので、一つの問題が全体に波及しにくい

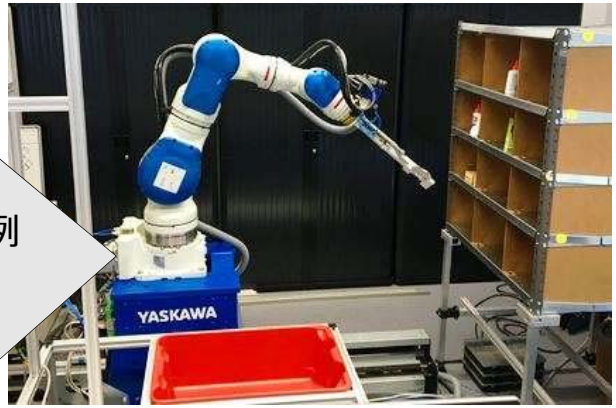
ロボットOS・ミドルウエアとは？

- ロボットシステム構築を効率化するための共通機能を提供する基盤ソフトウエア
  - 「ロボットOS」と呼ばれることもある
  - インターフェース・プロトコルの共通化、標準化
  - 例として
    - モジュール化・コンポーネント化フレームワークを提供
    - モジュール間の通信をサポート
    - パラメータの設定、配置、起動、モジュールの複合化（結合）機能を提供
    - 抽象化により、OSや言語間連携・相互運用を実現
- 2000年ごろから開発が活発化
  - 世界各国で様々なミドルウエアが開発・公開されている

ミドルウェアを利用した開発の利点



ピッキングロボットの構築例
※図は実際の構成ではありません。



ミドルウェアを利用すると、既存のモジュールが利用できる

開発するときに新規に作らなければならない部分は少なくて済む

ロボットOS・ミドルウェア

RTミドルウェア

OpenRTM-aist

RTとは?

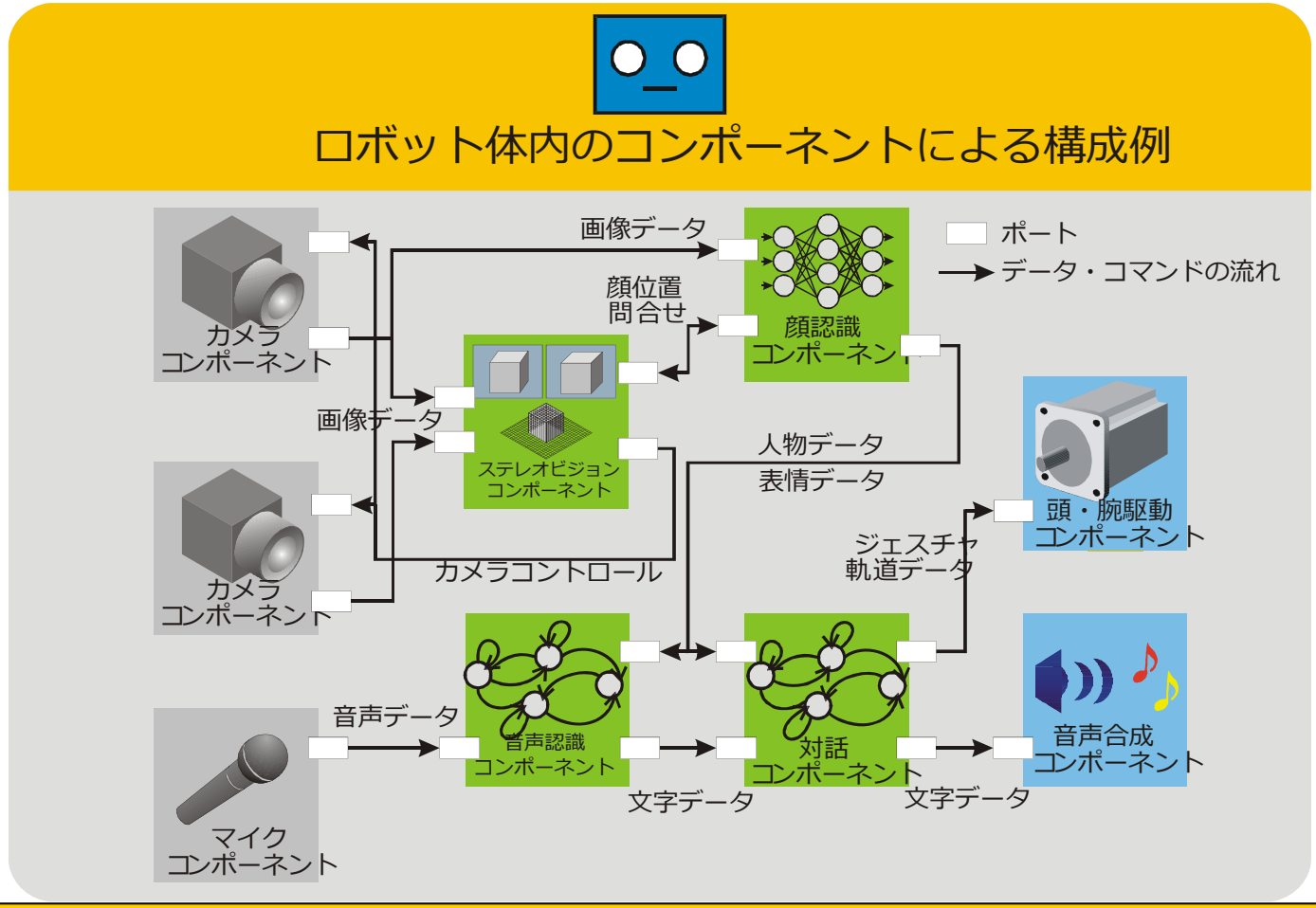
- RT = Robot Technology　　cf. IT
  - ≠Real-time
  - 単体のロボットだけでなく、さまざまなロボット技術に基づく機能要素をも含む (センサ、アクチュエータ, 制御スキーム、アルゴリズム、etc….)

産総研版RTミドルウエア

OpenRTM-aist

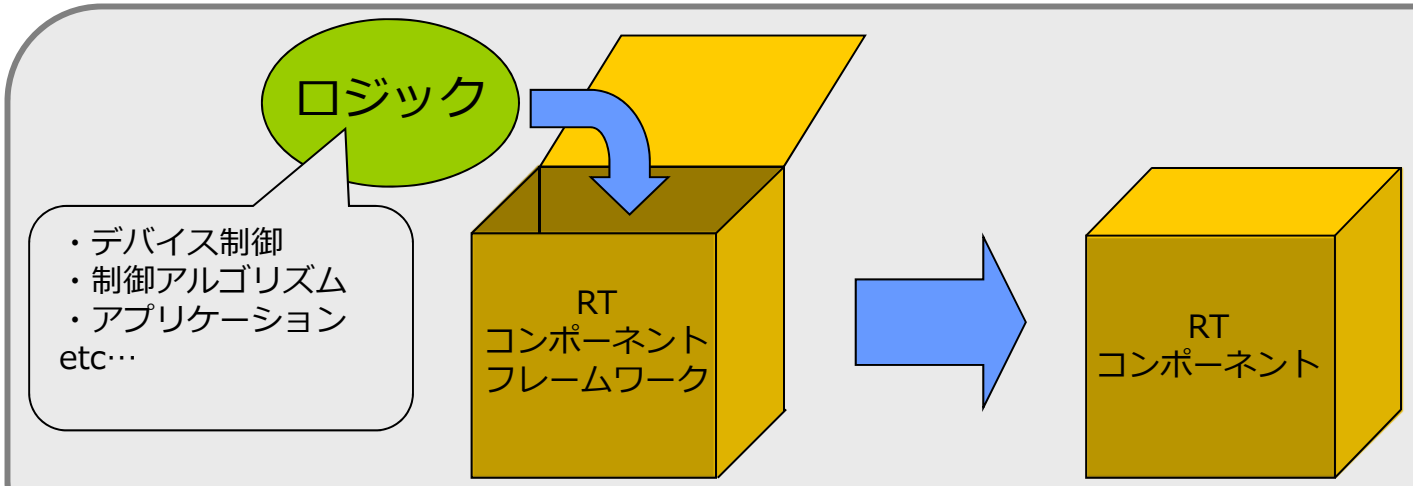
- RT-Middleware （RTM)
  - RT要素のインテグレーションのためのミドルウエア
- RT-Component （RTC)
  - RT-Middlewareにおけるソフトウエアの基本単位

ロボットシステムの構造



多くの機能要素（コンポーネント）から構成される

RTミドルウエアとRTコンポーネント



ロジックを箱（フレームワーク）に入れたもの＝RTコンポーネント（RTC）

RTCの実行環境（OSのようなもの）＝RTミドルウエア（RTM）
※RTCはネットワーク上に分散可能

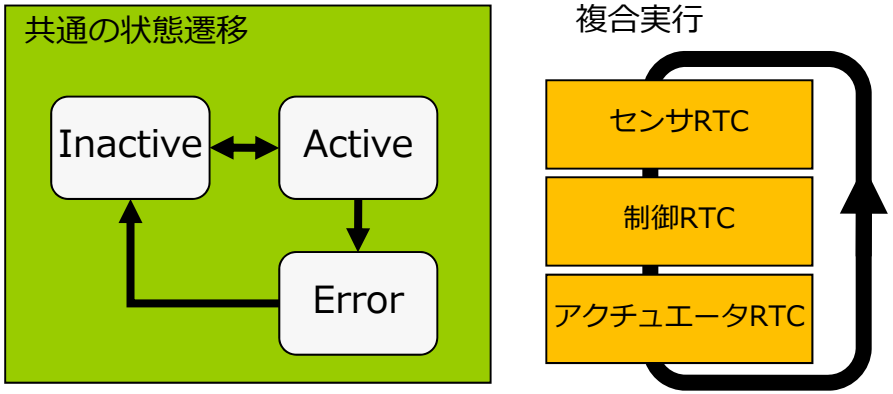
RTコンポーネント化のメリット

モジュール化のメリットに加えて

- ソフトウエアパターンを提供
  - ロボットに特有のソフトウエアパターンを提供することで、体系的なシステム構築が可能
- フレームワークの提供
  - フレームワークが提供されているので、コアのロジックに集中できる
- 分散ミドルウエア
  - ロボット体内LANやネットワークロボットなど、分散システムを容易に構築可能

RTコンポーネントの主な機能

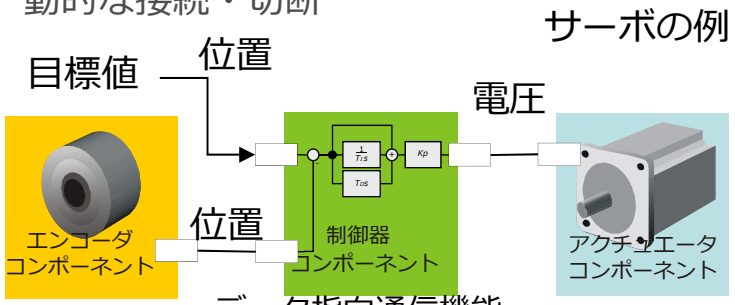
アクティビティ・実行コンテキスト



ライフサイクルの管理・コアロジックの実行

データポート

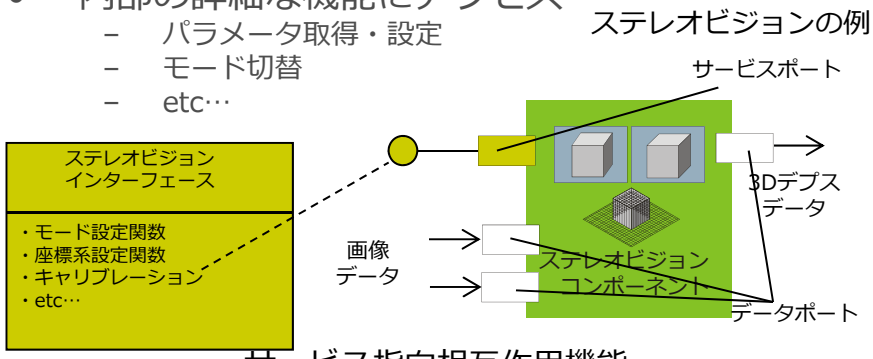
- データ指向ポート
- 連続的なデータの送受信
- 動的な接続・切断



データ指向通信機能

サービスポート

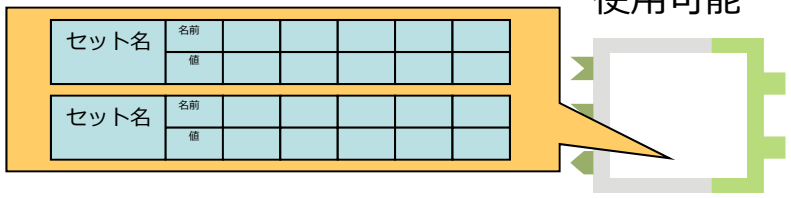
- 定義可能なインターフェースを持つ
- 内部の詳細な機能にアクセス
  - パラメータ取得・設定
  - モード切替
  - etc…



サービス指向相互作用機能

コンフィギュレーション

- パラメータを保持する仕組み
- いくつかのセットを保持可能
- 実行時に動的に変更可能



RTミドルウエアによる分散システム

RTMにより、
ネットワーク上に
分散するRTCを
OS・言語の壁を
越えて接続する
ことができる。

ロボットA

RTC | RTC
RTM
VxWorks

ロボットB

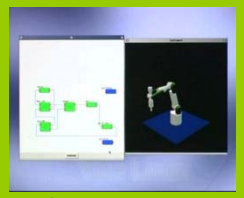
RTC | RTC
RTM
QNX

ロボットC

RTC | RTC | RTC
RTM
Real Time Linux

ネットワーク

Linux
RTM
RTC | RTC



アプリケーション

Windows
RTM
RTC | RTC

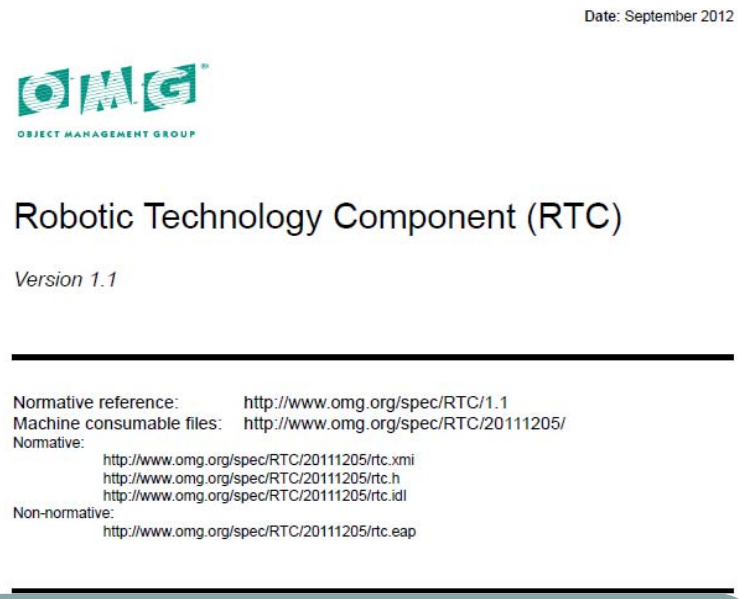
操作デバイス

uITRON
RTM
RTC | RTC

センサ

RTC同士の接続
は、プログラム
実行中に動的に
行うことが出来る。

RTミドルウェアの国際標準化



Date: September 2012

OMG
OBJECT MANAGEMENT GROUP

Robotic Technology Component (RTC)

Version 1.1

Normative reference: http://www.omg.org/spec/RTC/1.1
Machine consumable files: http://www.omg.org/spec/RTC/20111205/
Normative:
http://www.omg.org/spec/RTC/20111205/rtc.xmi
http://www.omg.org/spec/RTC/20111205/rtc.h
http://www.omg.org/spec/RTC/20111205/rtc.idl
Non-normative:
http://www.omg.org/spec/RTC/20111205/rtc.eap

標準化履歴
- 2005年9月
  Request for Proposal 発行(標準化開始)
- 2006年9月
  OMGで承認、事実上の国際標準獲得
- 2008年4月
  OMG RTC標準仕様 ver.1.0公式リリース
- 2012年9月
  ver. 1.1改定
- 2015年9月
  FSM4RTC(FSM型RTCとデータポート標準) 採択

OMG国際標準

標準化組織で手続きに沿って策定
→ 1組織では勝手に改変できない安心感
→ 多くの互換実装ができつつある
→ 競争と相互運用性が促進される

RTミドルウエア互換実装は10種類以上

名称	ベンダ	特徴	互換性
OpenRTM-aist	産総研	NEDO PJで開発。参照実装。	---
HRTM	ホンダ	アシモはHRTMへ移行中	◎
OpenRTM.NET	セック	.NET(C#,VB,C++/CLI, F#, etc..)	◎
RTM on Android	セック	Android版RTミドルウエア	◎
RTC-Lite	産総研	PIC, dsPIC上の実装	○
Mini/MicorRTC	SEC	NEDOオープンイノベーションPJで開発	○
RTMSafety	SEC/AIST	NEDO知能化PJで開発・機能安全認証取得	○
RTC CANOpen	SIT, CiA	CAN業界RTM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	×

特定のベンダが撤退しても
ユーザは使い続けることが可能

実用例・製品化例

HRPシリーズ: 川田工業、AIST

S-ONE : SCHAFT

DAQ-Middleware: KEK/J-PARC
KEK: High Energy Accelerator Research Organization
J-PARC: Japan Proton Accelerator Research Complex

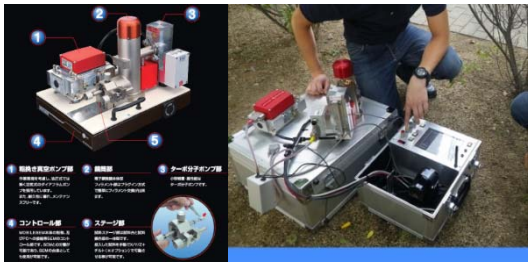
災害対応ロボット操縦シミュレータ：NEDO/千葉工大

HIRO, NEXTAGE open: Kawada Robotics

RAPUDA : Life Robotics

ビュートローバーRTC/RTC-BT(VSTONE)

OROCHI（アールティ）



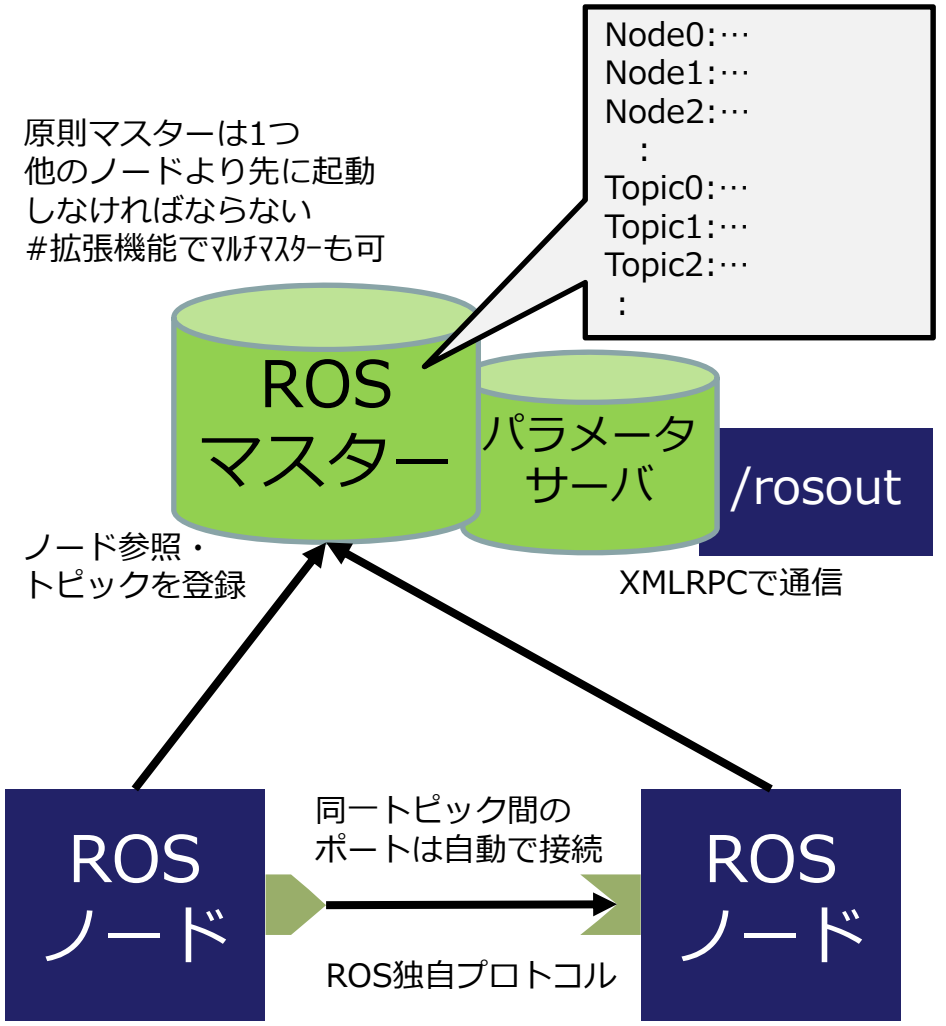
新日本電工他: Mobile SEM

ROS（Robot Operating System）

- 目的：ロボット研究を効率的にする
- OSではない
  - Robot Operating System, Robot Open Source
- UNIX哲学
  - UNIX的なコンパクトなツール、ライブラリ群による効率的なインテグレーション環境を提供する。
  - モジュール性、柔軟性、再利用性、言語・OS非依存、オープンソース
- プラットフォームアーキテクチャ
  - 独自ミドルウエア（独自IDLからコード生成）
  - メッセージベース
  - pub/subモデル、RPCモデル

ROSの仕組み

- **ノード**：ROSのモジュール化単位
  - 1ノード=1プロセス
- **マスター**：ノードの参照やトピックを保持するネームサービス
  - システム全体で原則一つ→SPOF（Single Point of Failure）
  - 他のノードより先に起動しなければならない
- **パラメータサーバ**：ノードのパラメータを保持するデータベース
  - マスター内で動作、ノードからはXMLRPCでアクセス
- **rosout**: ノードに対してstdout, stderrのような役割を果たす



RTMとROSの比較

ROS

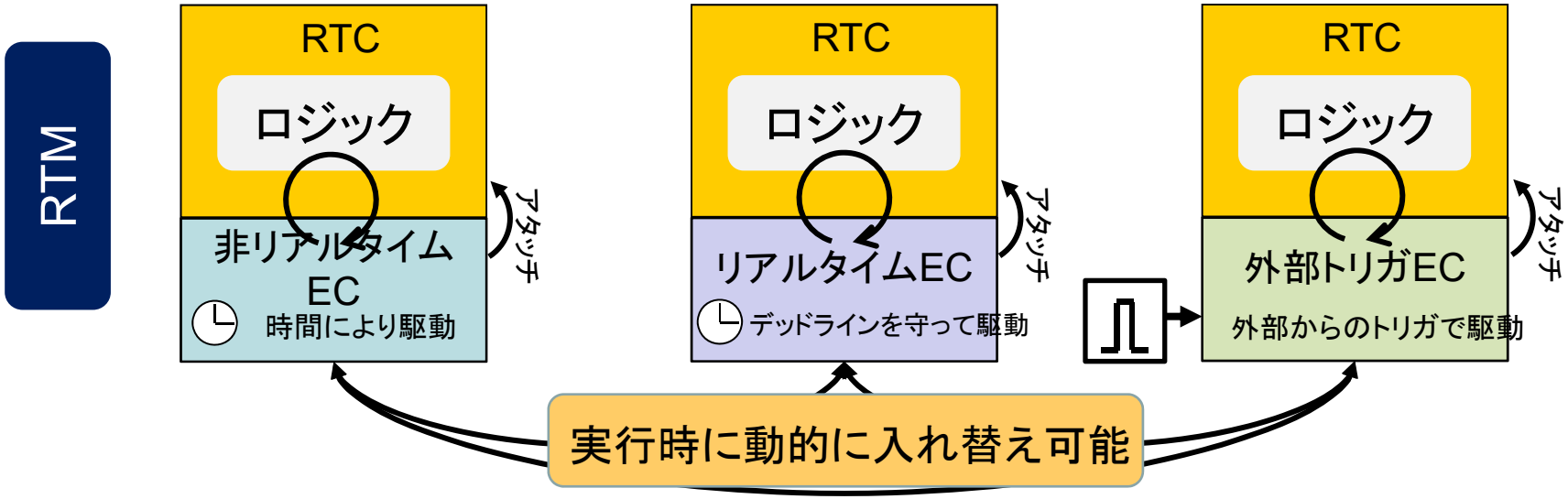
- UNIXユーザに受けるツール（特にCUI）が豊富
  - →基本はLinuxのみサポート
- 独自のパッケージ管理システムを持つ
- 実装方法が比較的自由
  - コア部分の使用が固まっていない
  - コンポーネントモデルがない
- ノード（コンポーネント）の質、量ともに十分
  - WGが直接品質を管理しているノードが多数
- ユーザ数が多い
- メーリングリストなどの議論がオープンで活発
- 英語のドキュメントが豊富

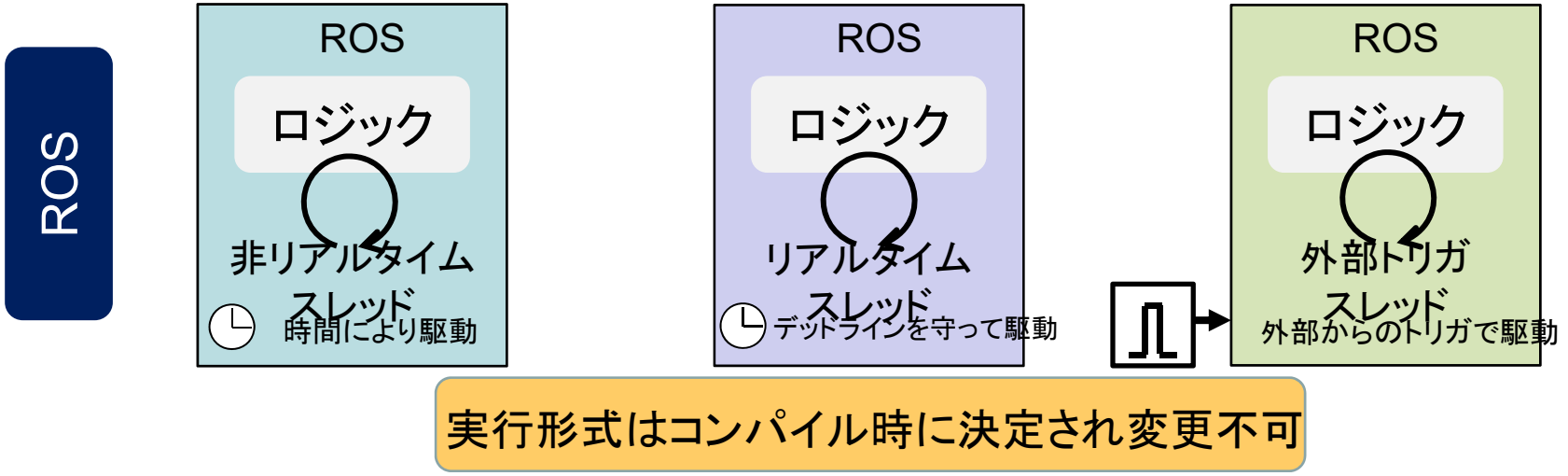
RTM

- 対応OS・言語の種類が多い
  - Windows、UNIX、uITRON、T-Kernel、VxWorks、QNX
  - Windowsでネイティブ動作する
- 日本製
  - 国外のユーザが少ない
  - 英語ドキュメントが少ない
- GUIツールがあるので初心者向き
- 仕様が標準化されている
  - OMGに参加すればだれでも変更可
  - サードパーティー実装が作りやすい
  - すでに10程度の実装あり
- コンポーネントモデルが明確
  - オブジェクト指向、UML・SysMLとの相性が良い
  - モデルベース開発
- IEC61508機能安全認証取得
  - RTMSafety

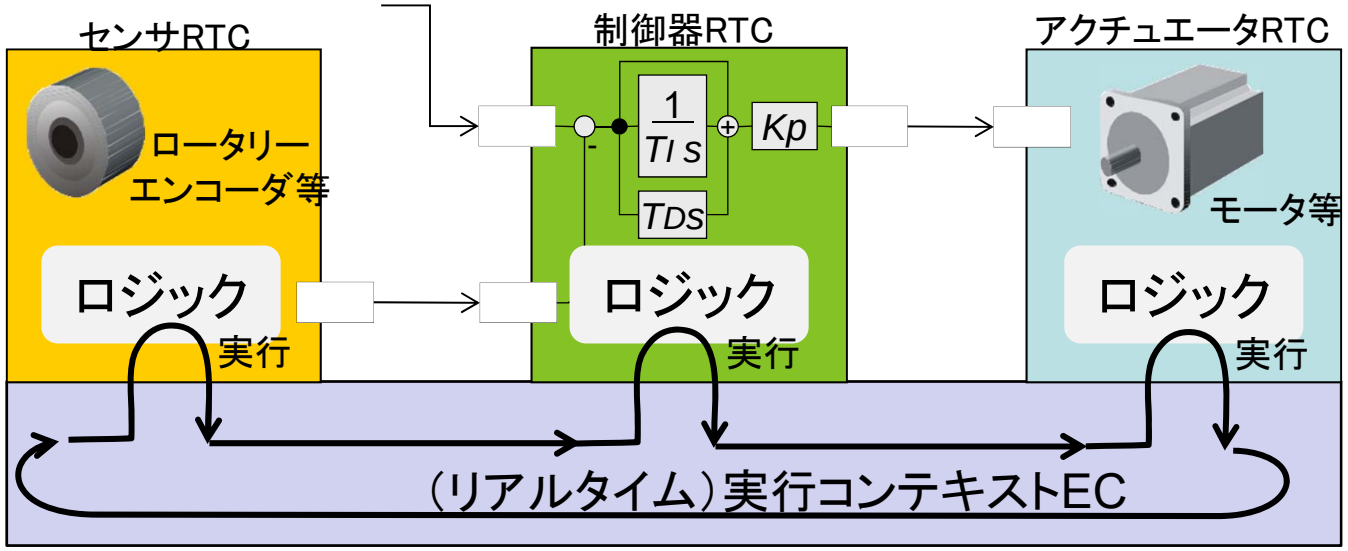
第三者による比較：http://ysuga.net/?p=146

リアルタイムシステムにおける違い





リアルタイム・複合化



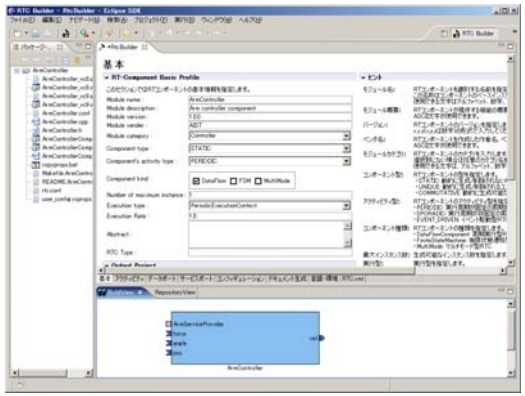
複数のRTCによる複合化、リアルタイム化が可能
EC・RTCにより実行ロジックが分離されており様々な実行形態を実現可能

- ROS: 1ノード1プロセス
  - 組合せて密結合(シーケンシャルにリアルタイム実行)は出来ない
  - roscontrol, realtime-tools といった3rd partyのツールを利用すると同様のことが可能
  - 但し、ノードの作り方は全く異なる

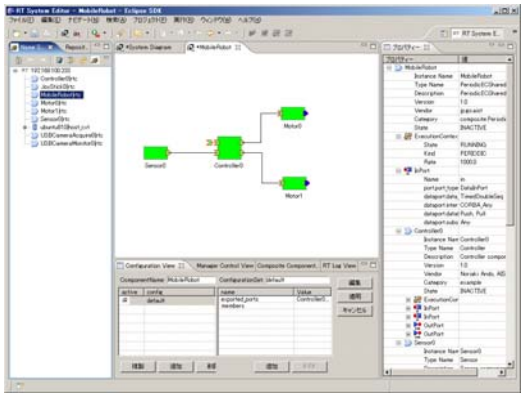
RTミドルウェア(RTM)を用いたロボット開発

システム開発の流れ

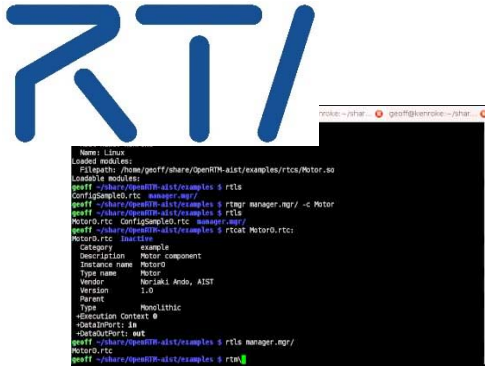
RTCBuilder



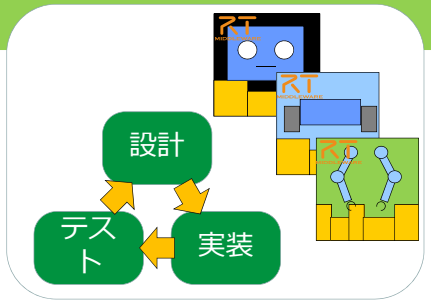
RTSystemEditor



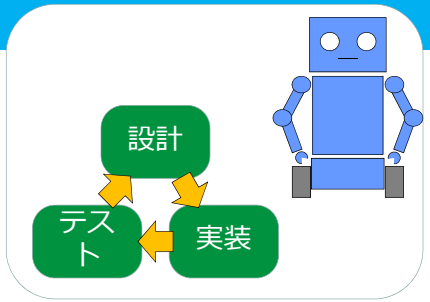
rtshell



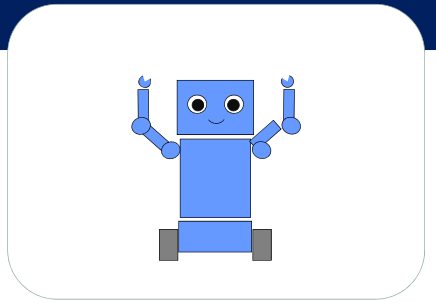
コンポーネント開発



システム開発

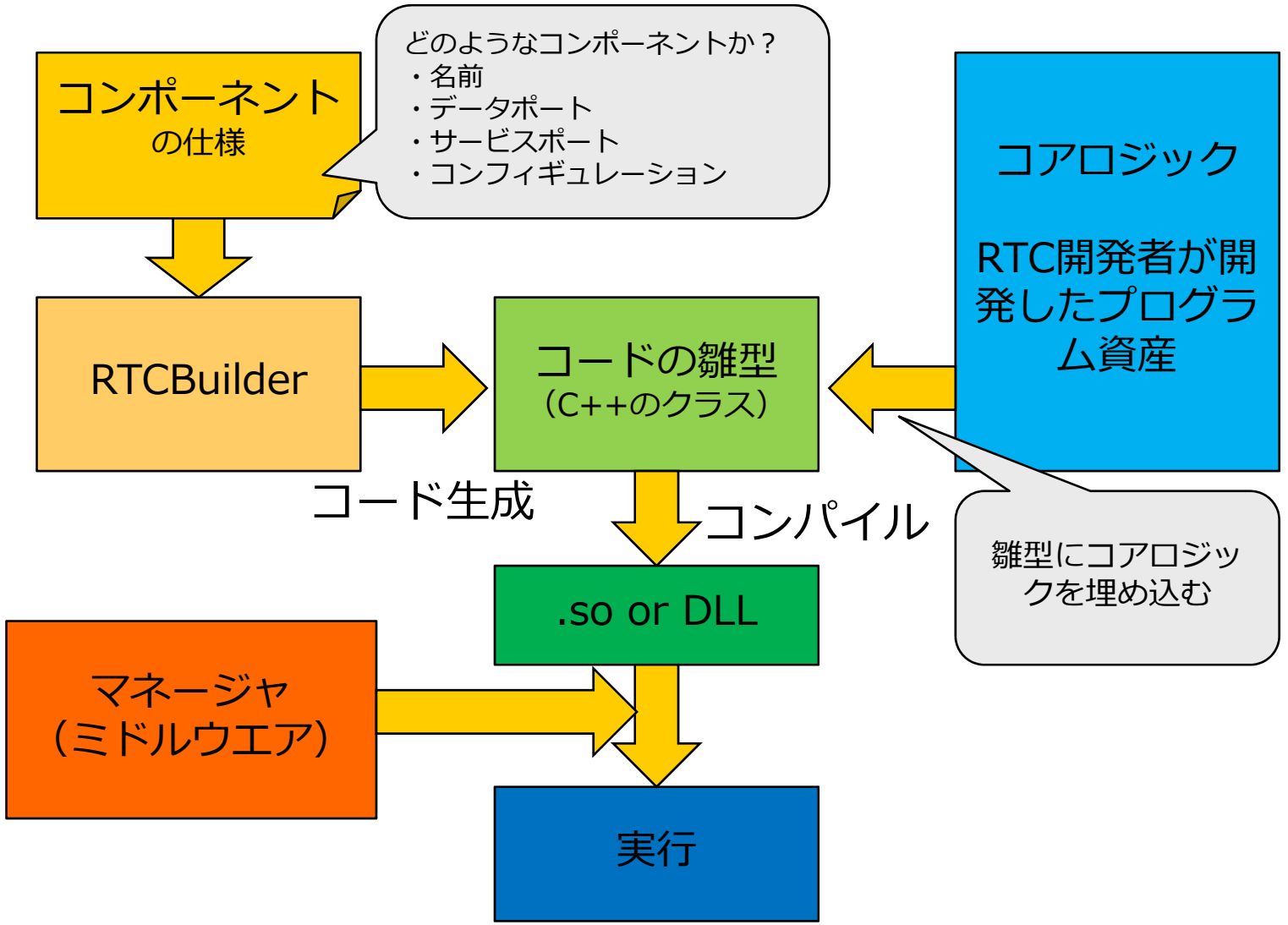


システム運用



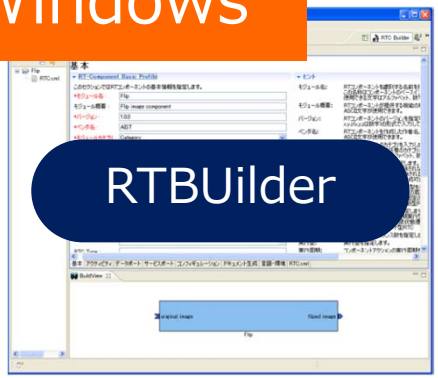
システム開発のフェーズごとに様々なツールを利用

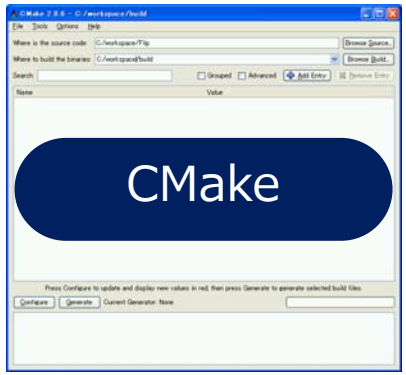
RTCBUilderを用いたRTCの開発

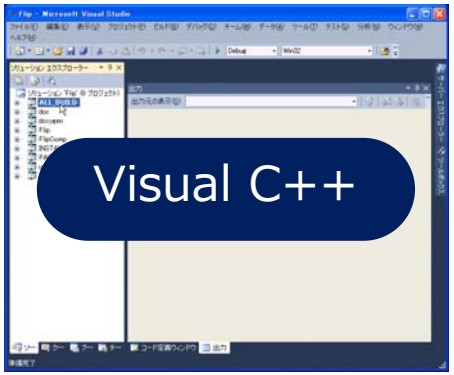


コンポーネント作成の流れ

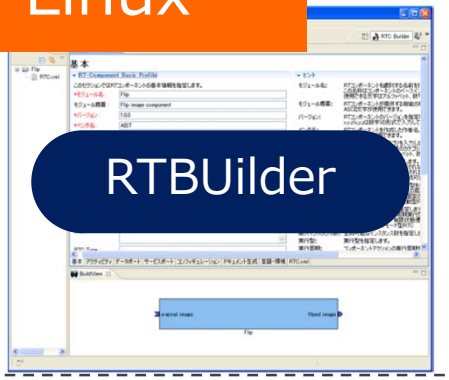
Windows

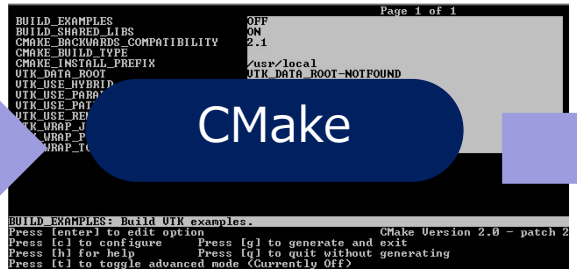


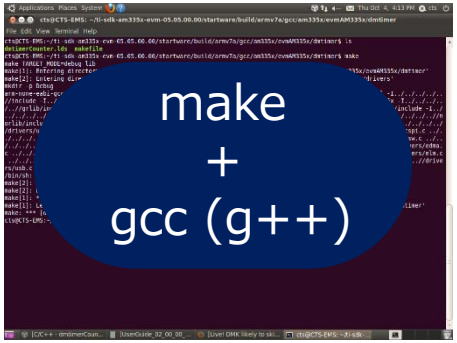




Linux







コンポーネントの
仕様の入力

VCプロジェクトファイル
またはMakefileの生成

実装およびコンパイル
実行ファイルの生成

途中まで流れは同じ、コンパイラが異なる

コード例

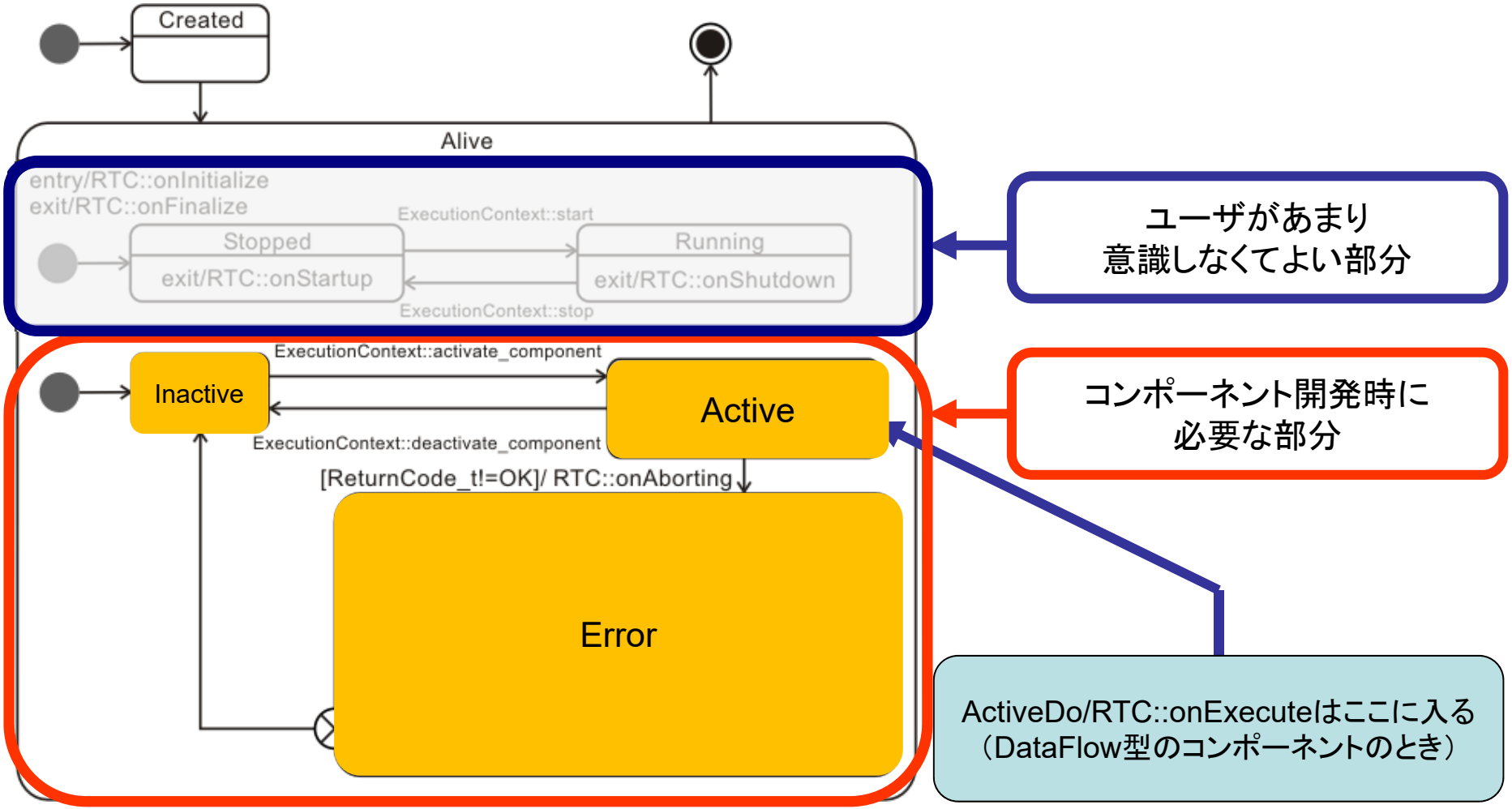
- 生成されたクラスのメンバー関数に必要な処理を記述
- 主要な関数
  - onExecute (周期実行)
- 処理
  - InPortから読む
  - OutPortへ書く
  - サービスを呼ぶ
  - コンフィギュレーションを読む

```
class MyComponent
 : public DataflowComponentBase
{
public:
  // 初期化時に実行したい処理
  virtual ReturnCode_t onInitialize()
  {
    if (mylogic.init())
      return RTC::RTC_OK;
    return RTC::RTC_ERROR;
  }

  // 周期的に実行したい処理
  virtual ReturnCode_t onExecute(RTC::UniqueId ec_id)
  {
    if (mylogic.do_someting())
      return RTC::RTC_OK;
    RTC::RTC_ERROR;
  }

private:
  MyLogic mylogic;
  // ポート等の宣言
  //   :
};
```

開発者が書くのはこの部分だけ

大部分はツール（RTCBuilder）で自動生成される

コンポーネント内の状態遷移



コールバック関数

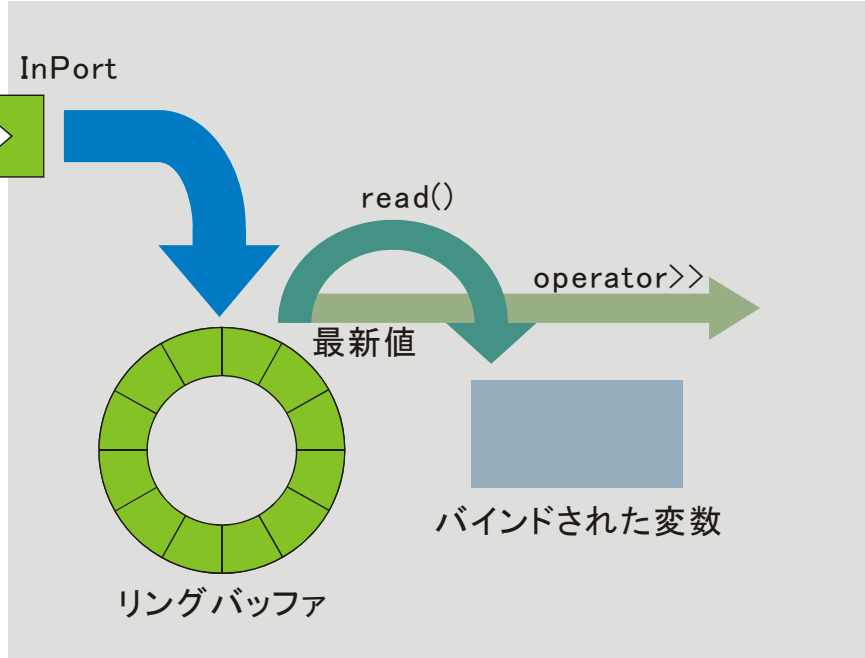
RTCの作成＝コールバック関数に処理を埋め込む

コールバック関数	処理
onInitialize	初期化処理
onActivated	アクティブ化されるとき1度だけ呼ばれる
onExecute	アクティブ状態時に周期的に呼ばれる
onDeactivated	非アクティブ化されるとき1度だけ呼ばれる
onAborting	ERROR状態に入る前に1度だけ呼ばれる
onReset	resetされる時に1度だけ呼ばれる
onError	ERROR状態のときに周期的に呼ばれる
onFinalize	終了時に1度だけ呼ばれる
onStateUpdate	onExecuteの後毎回呼ばれる
onRateChanged	ExecutionContextのrateが変更されたとき1度だけ呼ばれる
onStartup	ExecutionContextが実行を開始するとき1度だけ呼ばれる
onShutdown	ExecutionContextが実行を停止するとき1度だけ呼ばれる

とりあえずは
この5つの関数
を押さえて
おけばOK

InPort

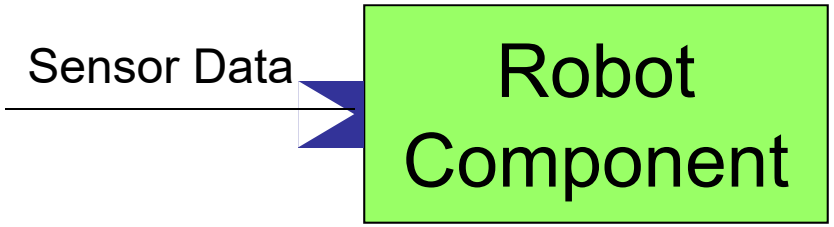
- InPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- InPortのメソッド
  - read(): InPort バッファからバインドされた変数へ最新値を読み込む
  - >> : ある変数へ最新値を読み込む



基本的にOutPortと対になる

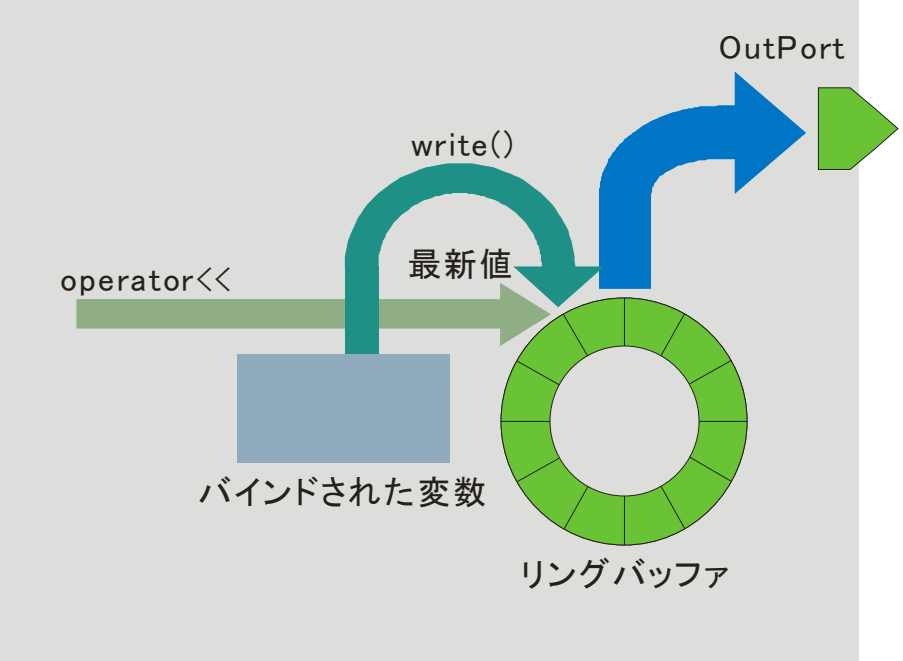
データポートの型を
同じにする必要あり

例



OutPort

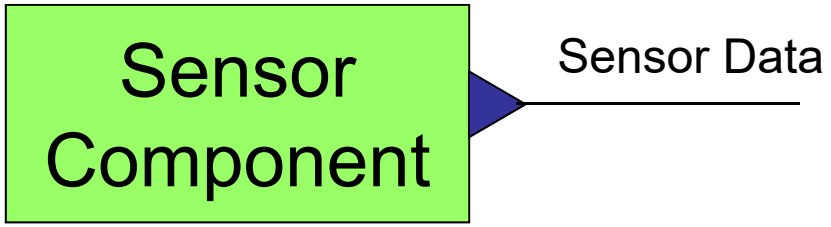
- OutPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- OutPortのメソッド
  - write(): OutPort バッファへバインドされた変数の最新値として書き込む
  - >> : ある変数の内容を最新値としてリングバッファに書き込む



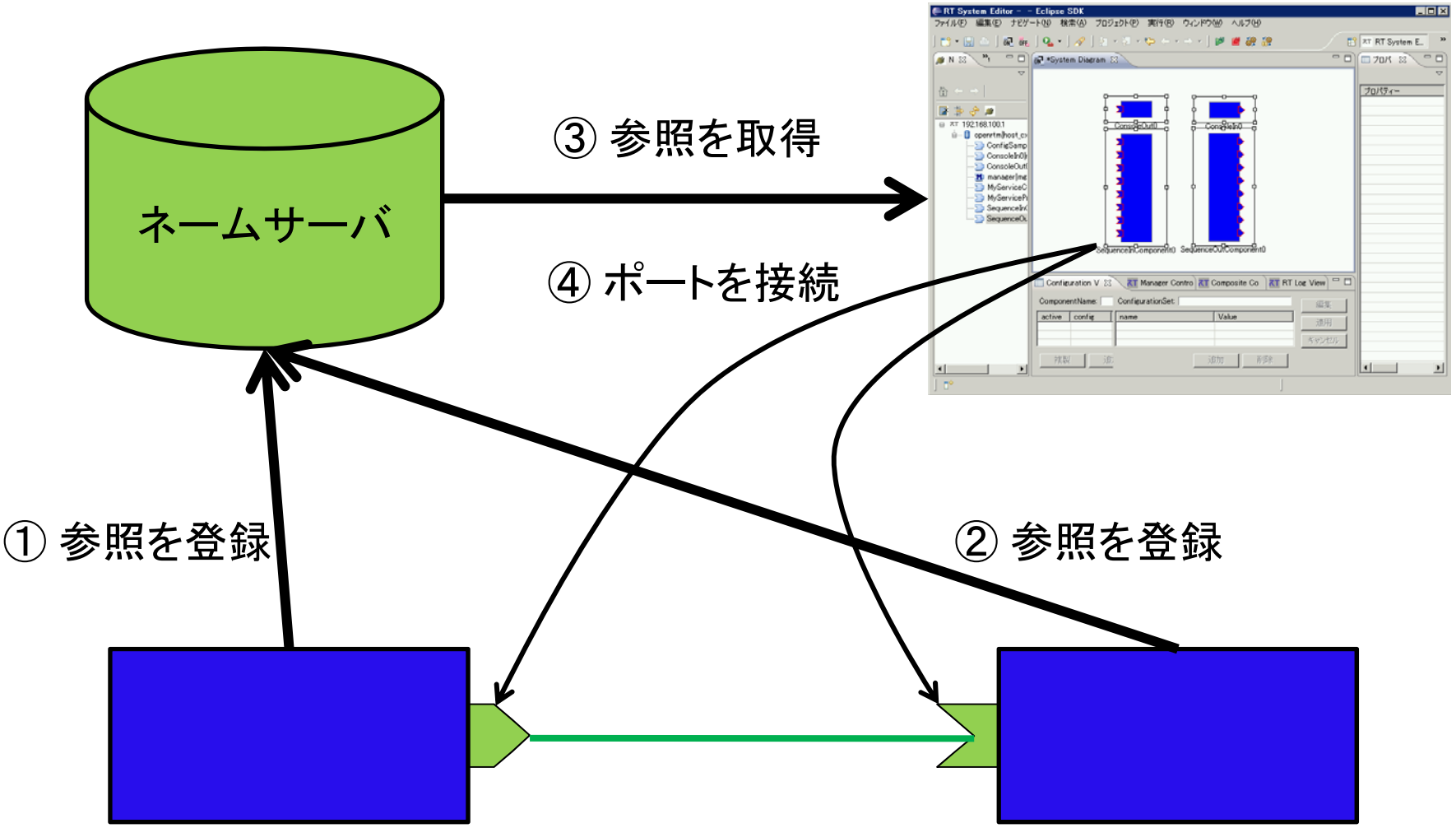
基本的にInPortと対になる

データポートの型を
同じにする必要あり

例



動作シーケンス



まとめ

1. ロボットシステムプログラミングの現状

   → コンポーネント指向、ミドルウエアの利用が主流になりつつある

2. ロボットOS・ミドルウェア

   → RTM、ROS等のミドルウエアがある

   → RTM：国際標準、多様なOS・言語・リアルタイム制御をサポート

3. RTミドルウェア(RTM)を用いたロボット開発

   → RTC作成 → システム構築 → 運用

   → ツールの利用で作業が容易に