2020年5月27日
ROBOMECH2020チュートリアル

OpenRTM-aistおよび RTコンポーネントプログラミングの概要

国立研究開発法人産業技術総合研究所
インダストリアルCPS研究センター
ソフトウェアプラットフォーム研究チーム長
安藤　慶昭

はじめに

- RTミドルウエアの概要
  - 基本概念
- ロボットソフトウェアの動向
- モジュール化のメリット
- RTコンポーネントの基本機能
- 標準化
- コミュニティ

RTミドルウェアとは？

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

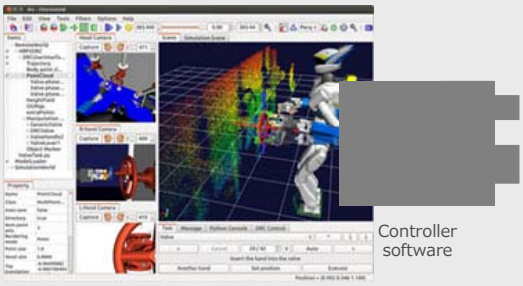
OpenRTM-aist

- 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

ロボットミドルウエアについて

- 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

従来のシステムでは…



Controller

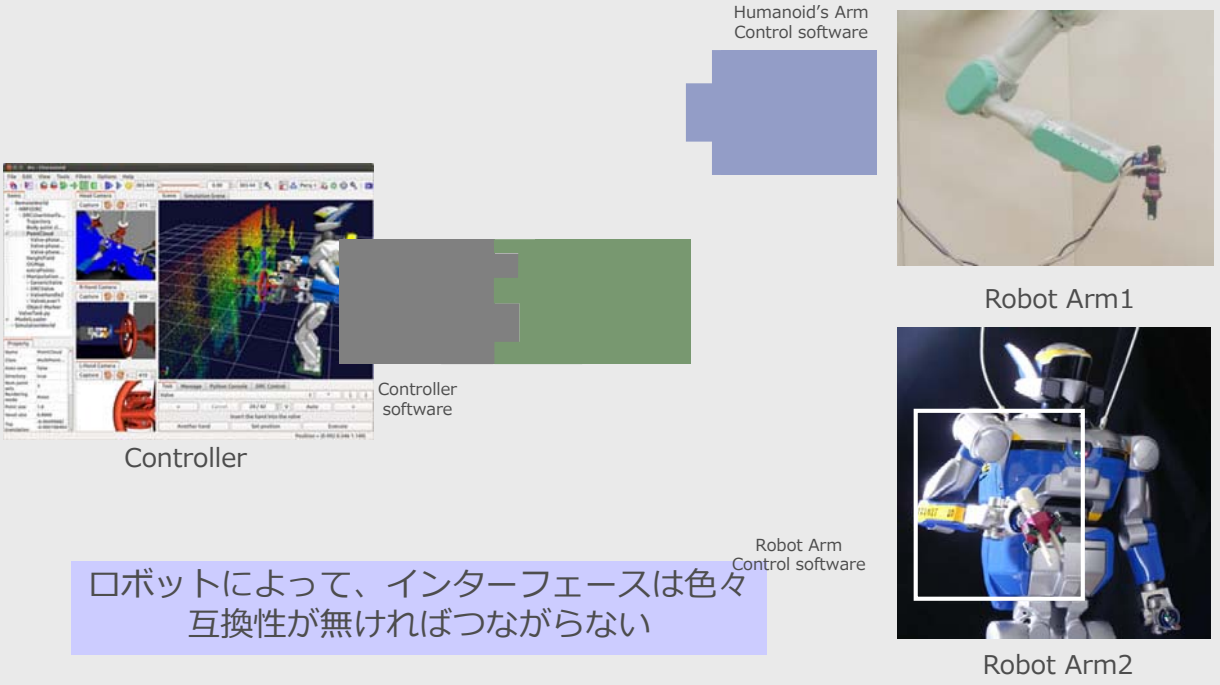
Controller software

Robot Arm Control software

Robot Arm2

互換性のあるインターフェース同士は接続可能

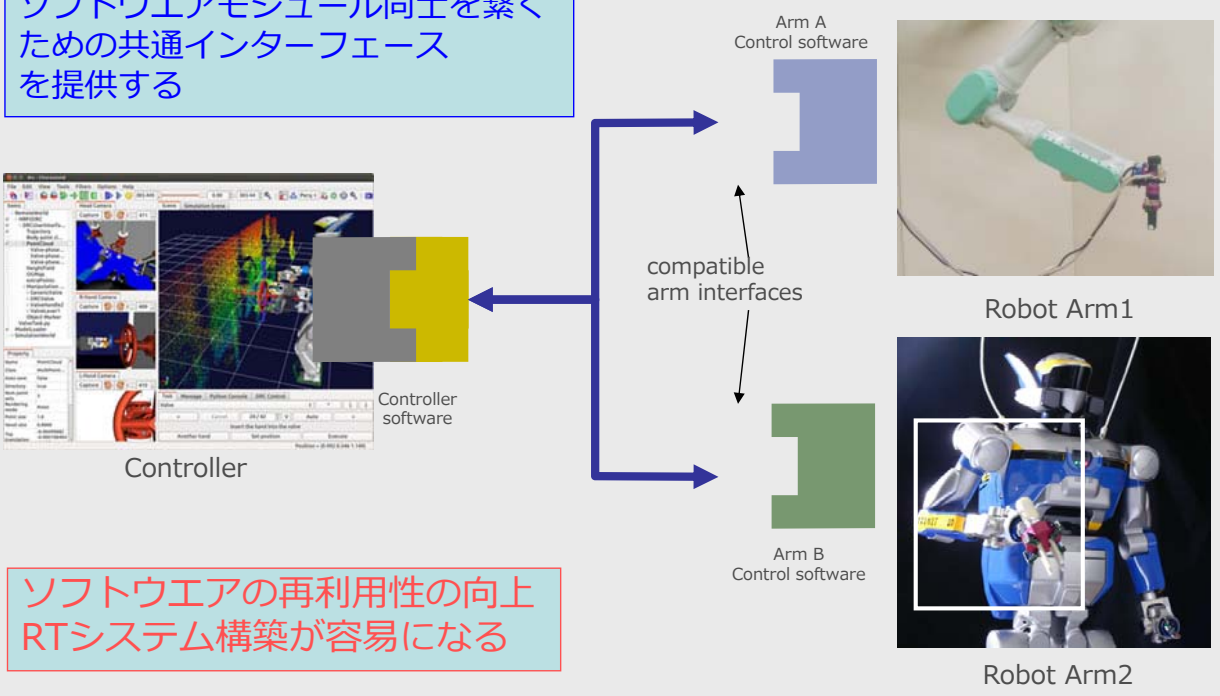
従来のシステムでは…



ロボットによって、インターフェースは色々
互換性が無ければつながらない

RTミドルウエアでは…

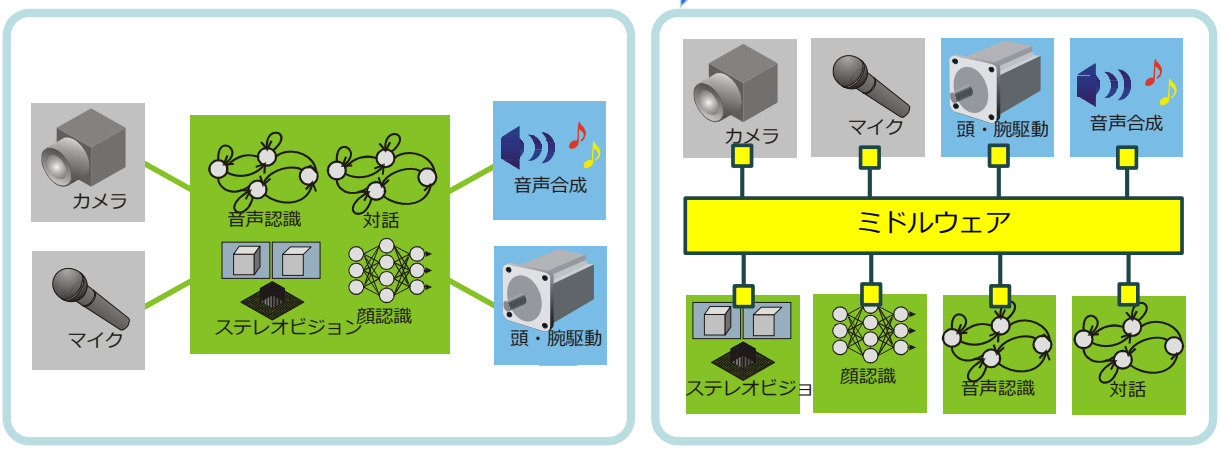
RTミドルウエアは別々に作られた
ソフトウエアモジュール同士を繋ぐ
ための共通インターフェース
を提供する



ソフトウエアの再利用性の向上
RTシステム構築が容易になる

ロボットソフトウェア開発の方向

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
- ✓ 実行時の効率は高いが、柔軟性に欠ける
- ✓ システムが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

モジュール化のメリット

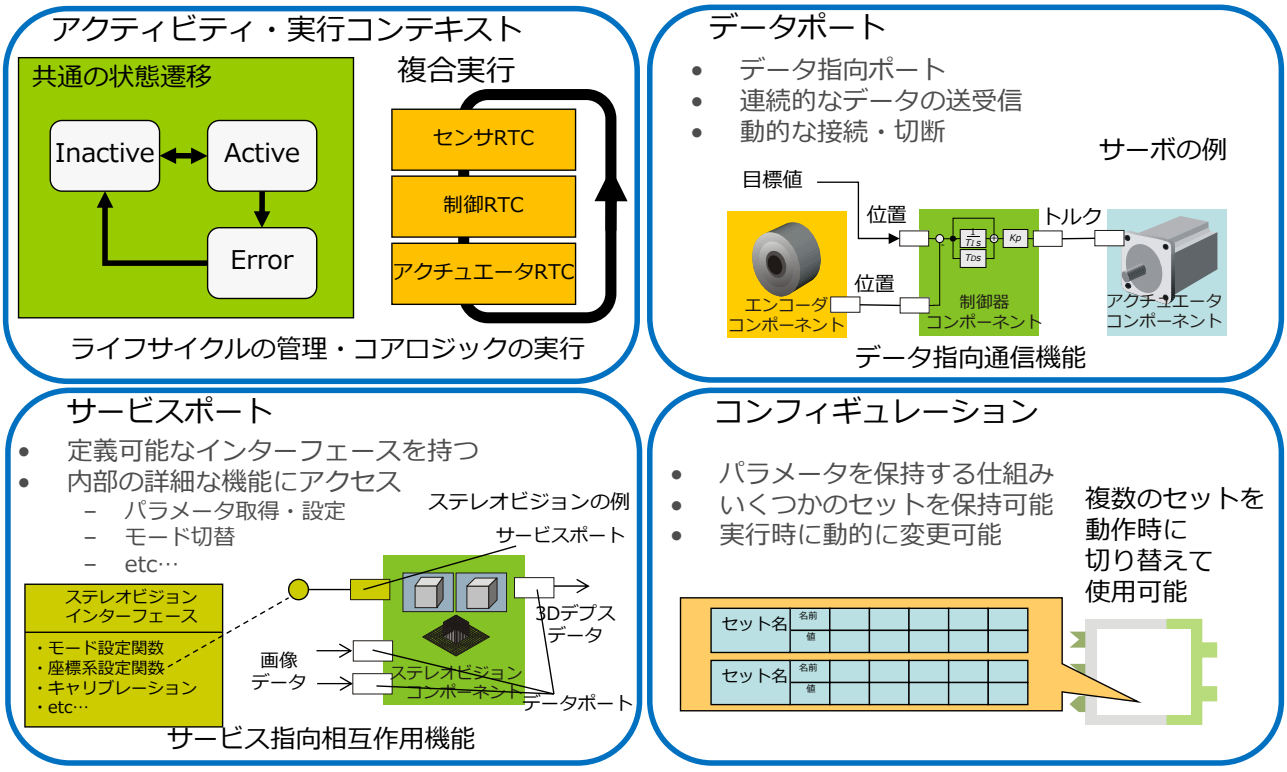
- 再利用性の向上
  - 同じコンポーネントをいろいろなシステムに使いまわせる
- 選択肢の多様化
  - 同じ機能を持つ複数のモジュールを試すことができる
- 柔軟性の向上
  - モジュール接続構成かえるだけで様々なシステムを構築できる
- 信頼性の向上
  - モジュール単位でテスト可能なため信頼性が向上する
- 堅牢性の向上
  - システムがモジュールで分割されているので、一つの問題が全体に波及しにくい

RTコンポーネント化のメリット

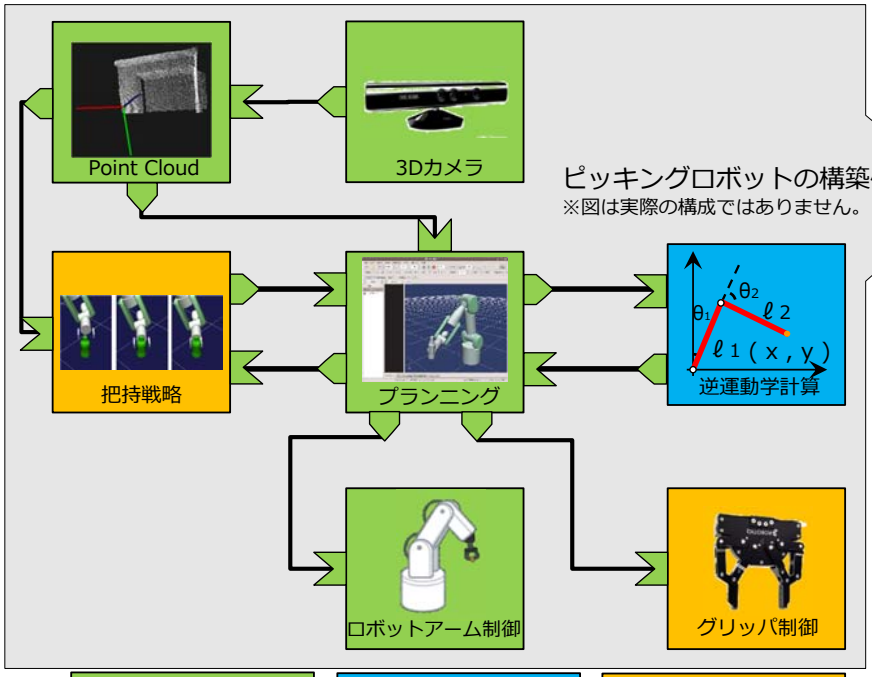
モジュール化のメリットに加えて

- 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

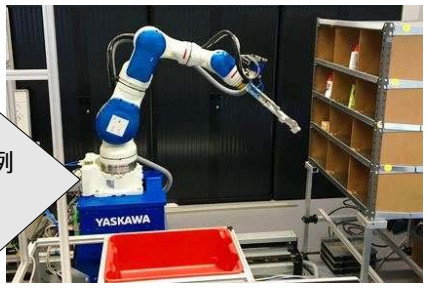
RTコンポーネントの主な機能



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。

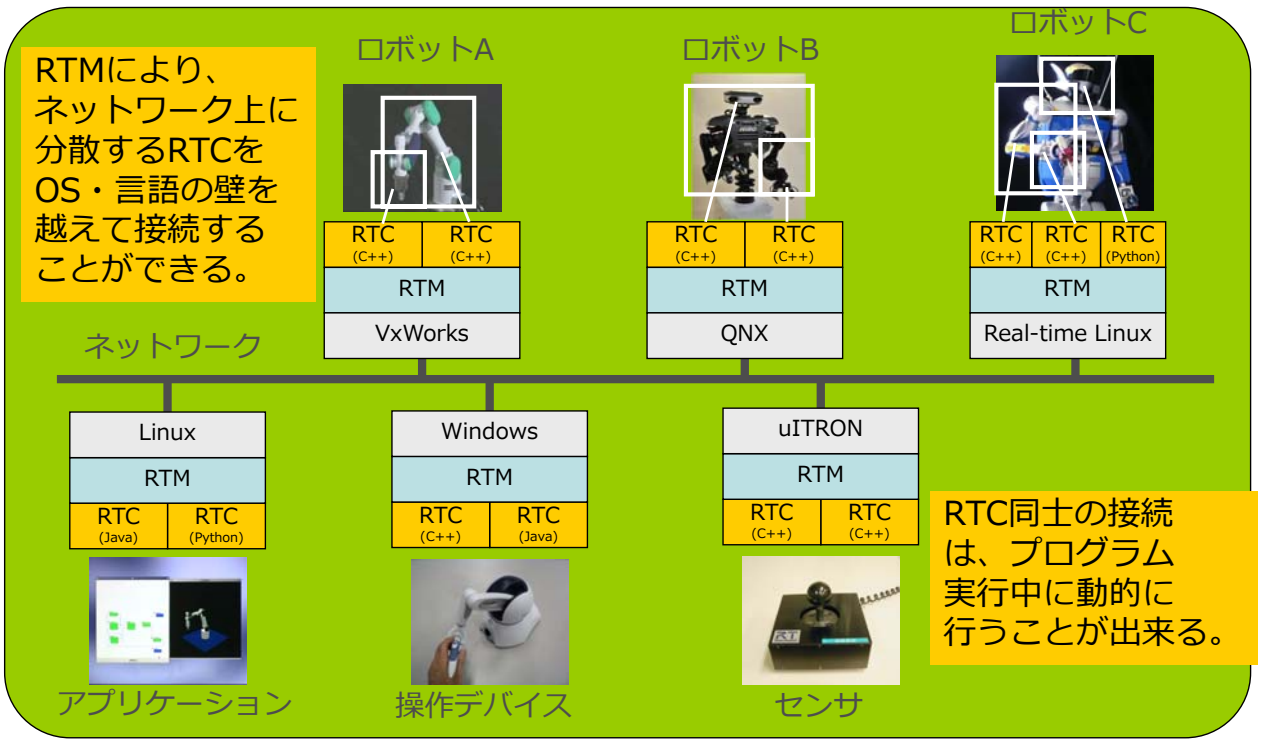


ミドルウェアを利用すると、既存のモジュールが利用できる

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

RTミドルウエアによる分散システム

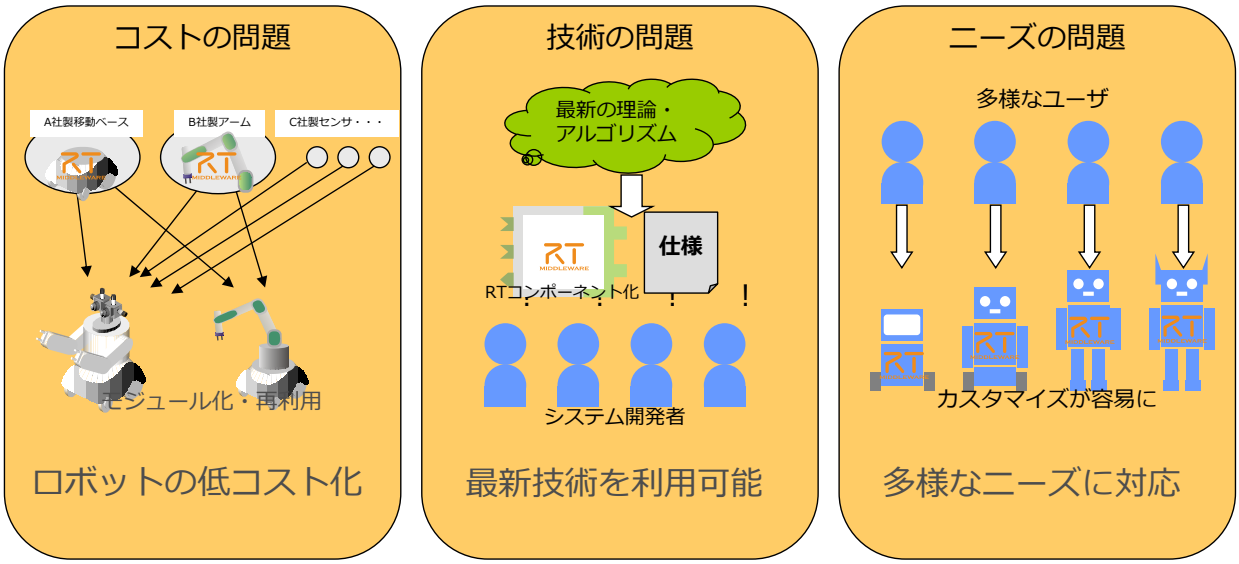
RTMにより、ネットワーク上に分散するRTCをOS・言語の壁を越えて接続することができる。



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ミドルウエアの目的

モジュール化による問題解決



ロボットシステムインテグレーションによるイノベーション

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

RTミドルウェアは国際標準


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

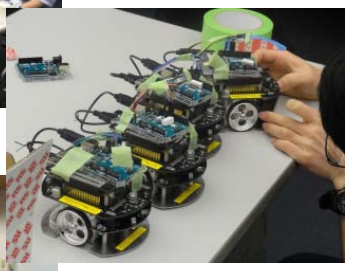
プロジェクトページ

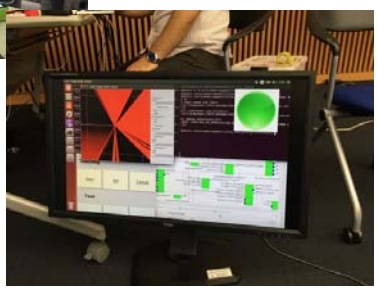
- ユーザが自分の作品を登録
- 他のユーザの作ったRTCを探すことができる

タイプ	登録数
RTコンポーネント群	403
RTミドルウエア	14
ツール	27
仕様・文書	6
ハードウエア	28

サマーキャンプ

- 毎年夏に1週間開催
- 今年：開催検討中
- 募集人数：20名
- 場所：産総研つくばセンター
- 座学と実習を1週間行い、最後にそれぞれが成果を発表
- 産総研内のさくら館に宿泊しながら夜通し？コーディングを行う！

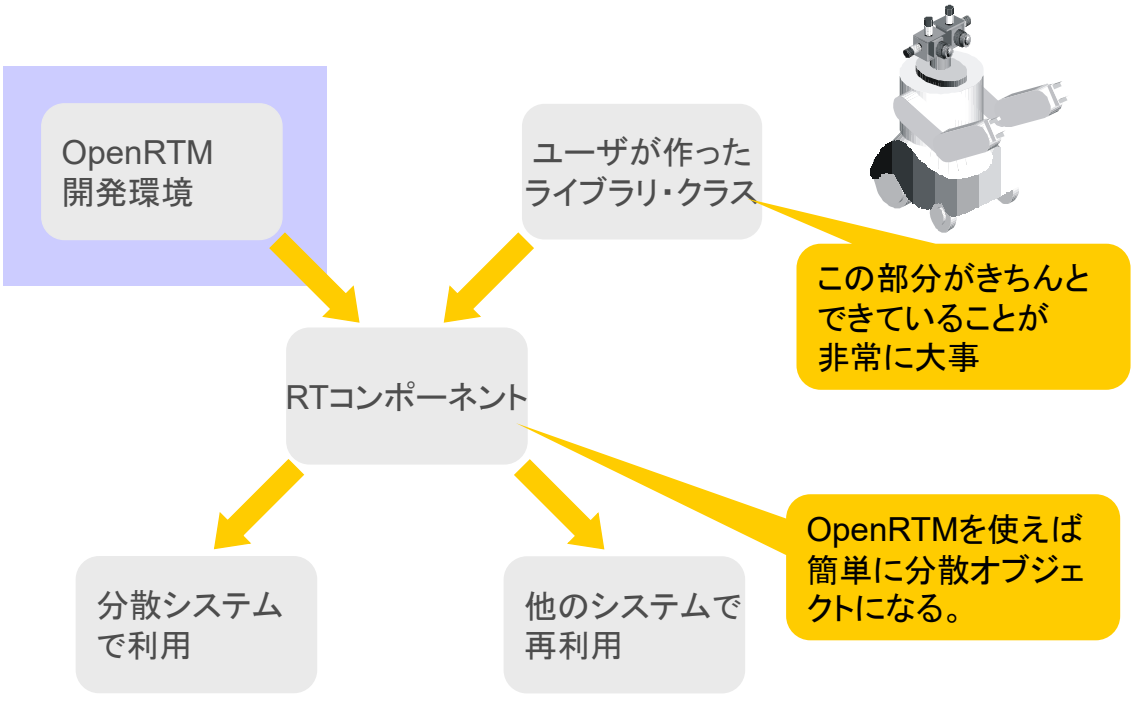



RTミドルウエアコンテスト

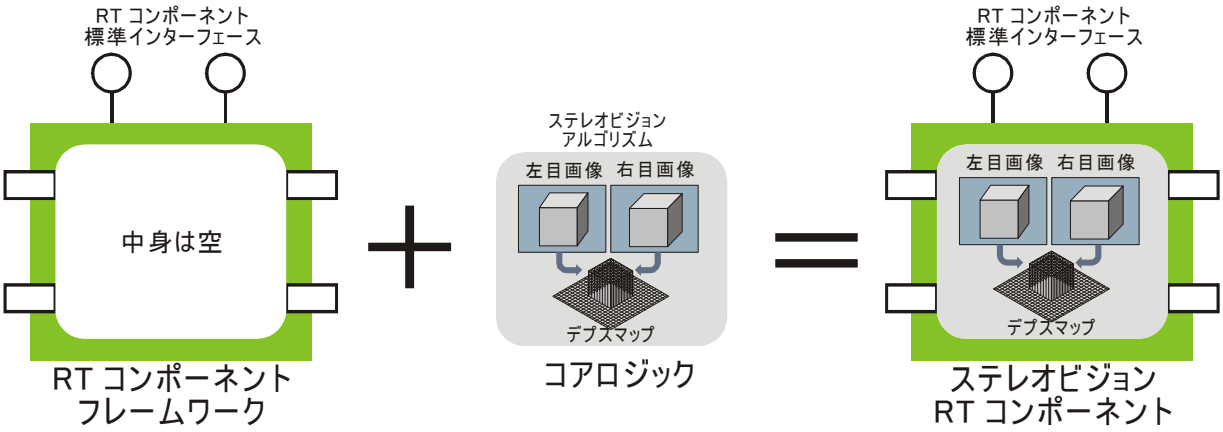
- SICE SI （計測自動制御学会 システムインテグレーション部門講演会 ）のセッションとして開催
  開催の可否を検討中
  - 各種奨励賞・審査基準開示:5月頃
  - エントリー〆切：SI2020締切
  - 講演原稿〆切：9月ごろ
  - ソフトウエア登録：10月ごろ
  - オンライン審査：11月下旬～
  - 発表・授賞式：12月ごろ
- 2019年度実績
  - 応募数：11件
  - 計測自動制御学会学会RTミドルウエア賞（副賞10万円）
  - 奨励賞（賞品協賛）：2件
  - 奨励賞（団体協賛）：9件
  - 奨励賞（個人協賛）：10件
- 詳細はWebページ： openrtm.org
  - コミュニティー→イベント をご覧ください


RTC開発の実際

OpenRTMを使った開発の流れ

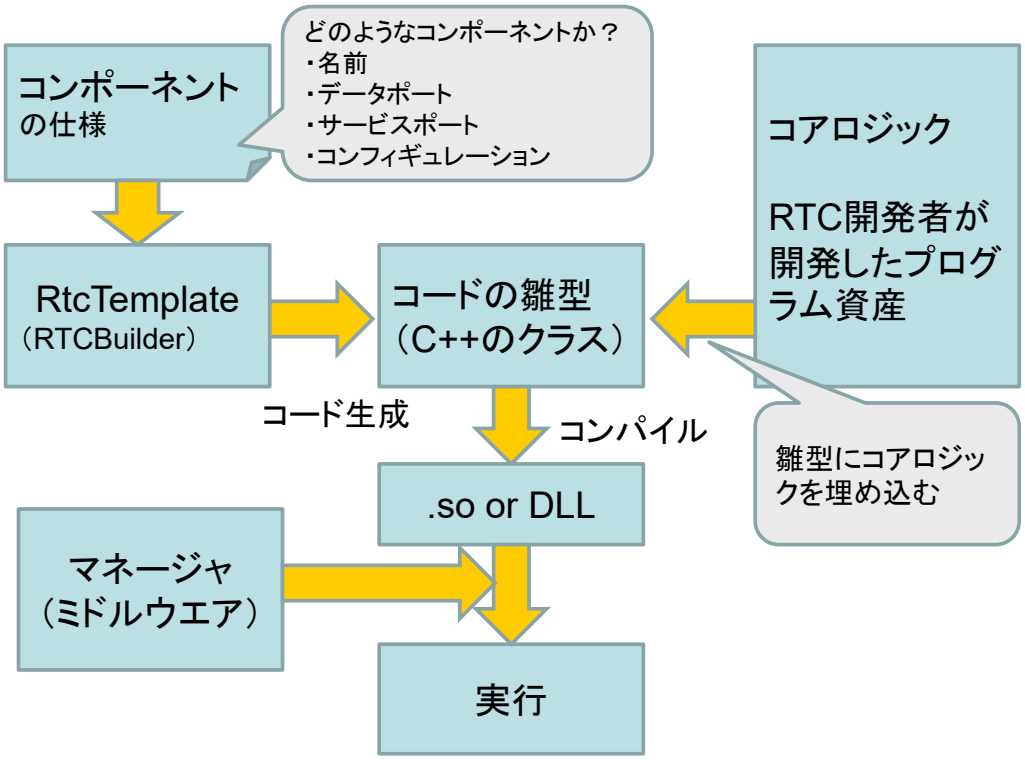


フレームワークとコアロジック

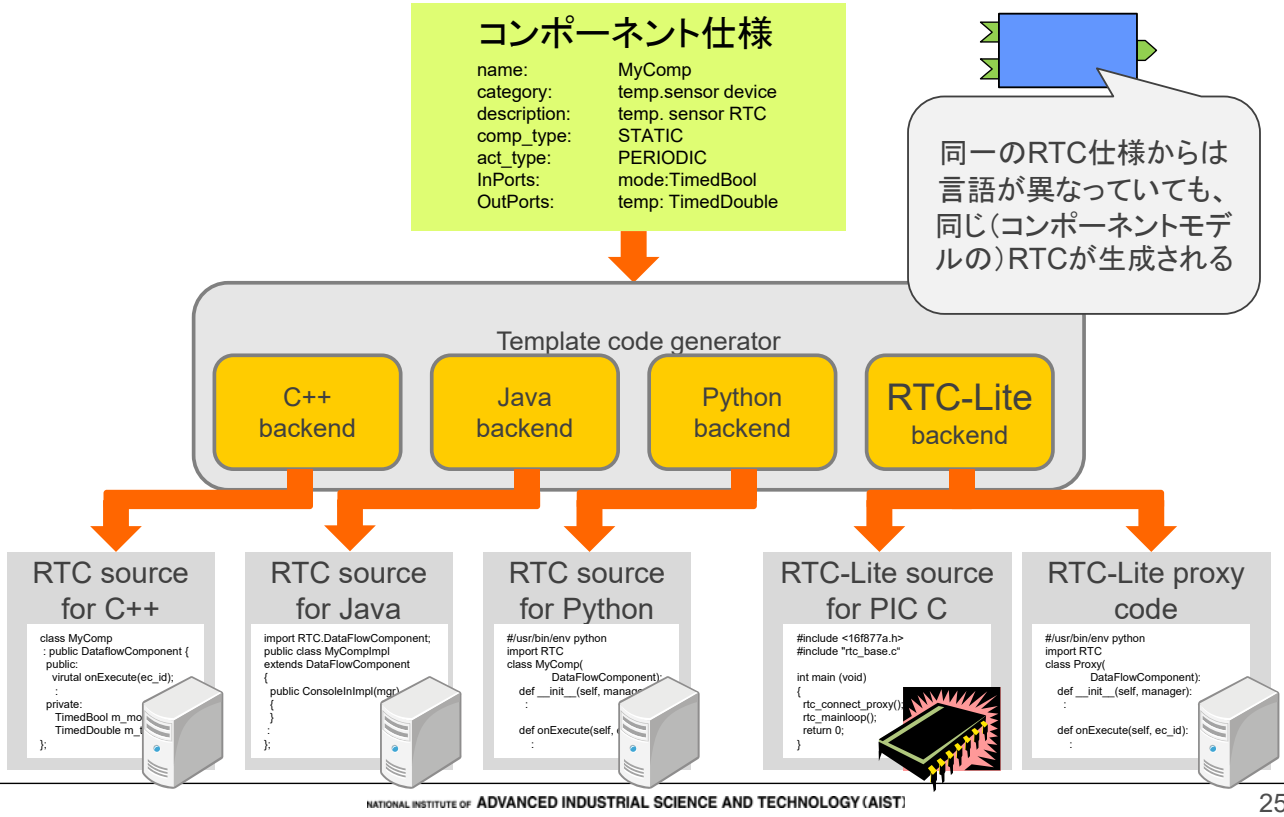


RTCフレームワーク＋コアロジック＝RTコンポーネント

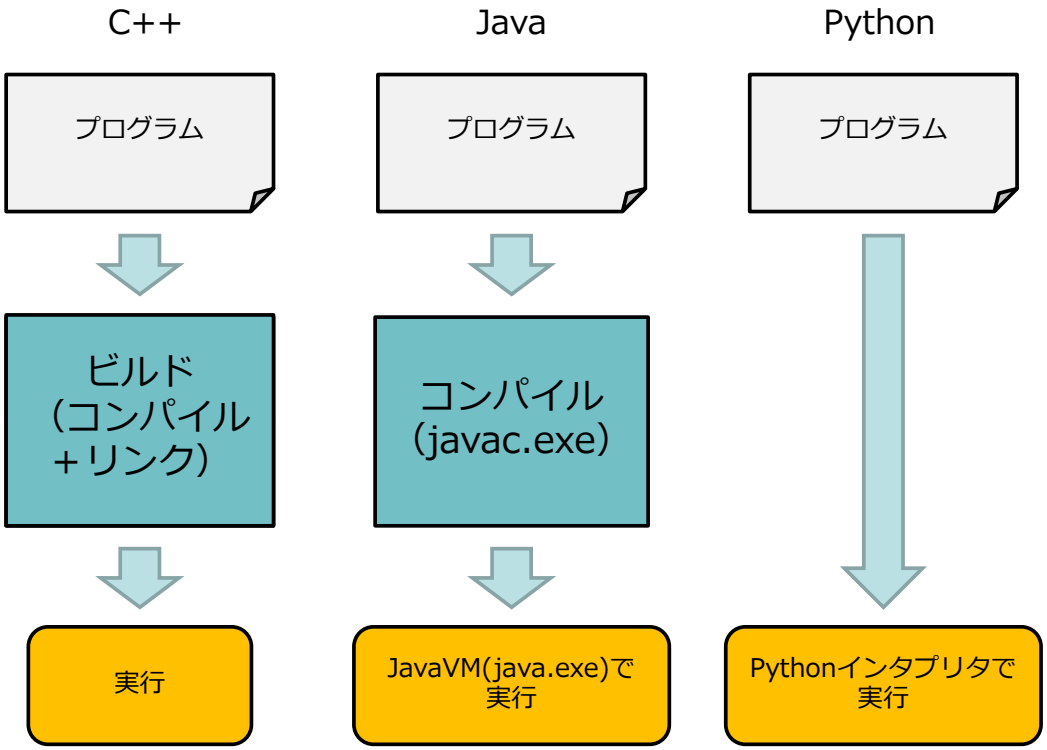
OpenRTMを使った開発の流れ



モデルに基づくコード生成



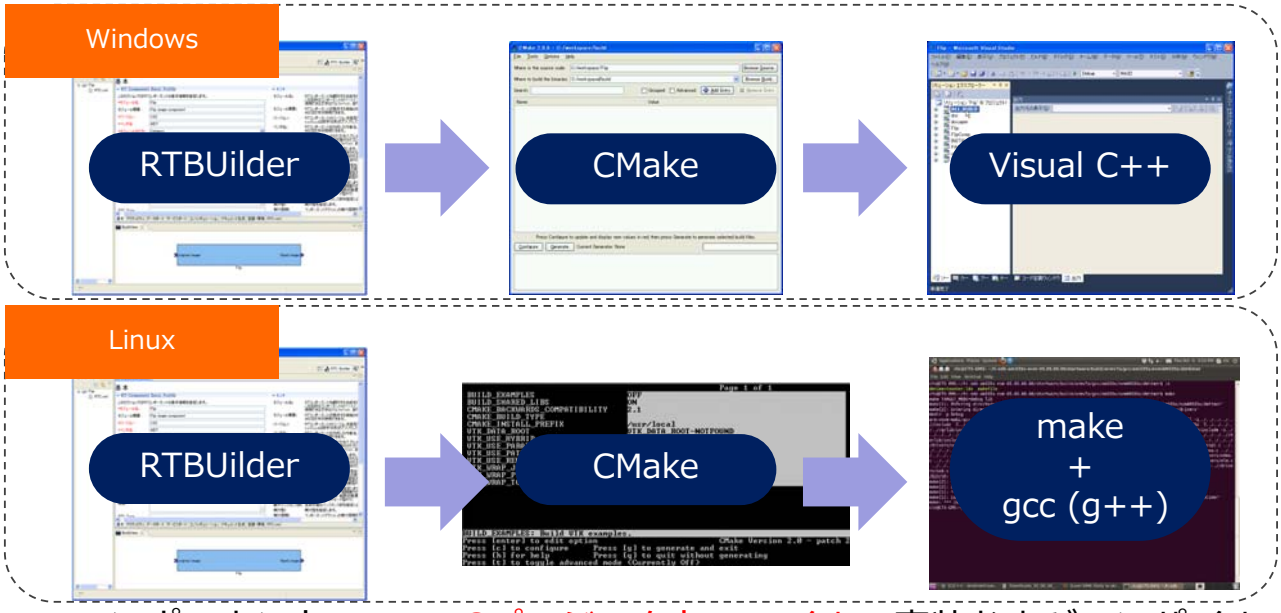
プログラミングの流れ



CMake

- コンパイラに依存しないビルド自動化のためのフリーソフトウェア
- 様々なOS上の様々な開発環境用ビルドファイルを生成することができる
  - Linux では Makefileを生成
  - Windows ではVC(Visual C++)のプロジェクトファイルを生成
- 最近のオープンソースソフトウェアではCMakeでビルドするようになっているものが多数。

コンポーネント作成の流れ



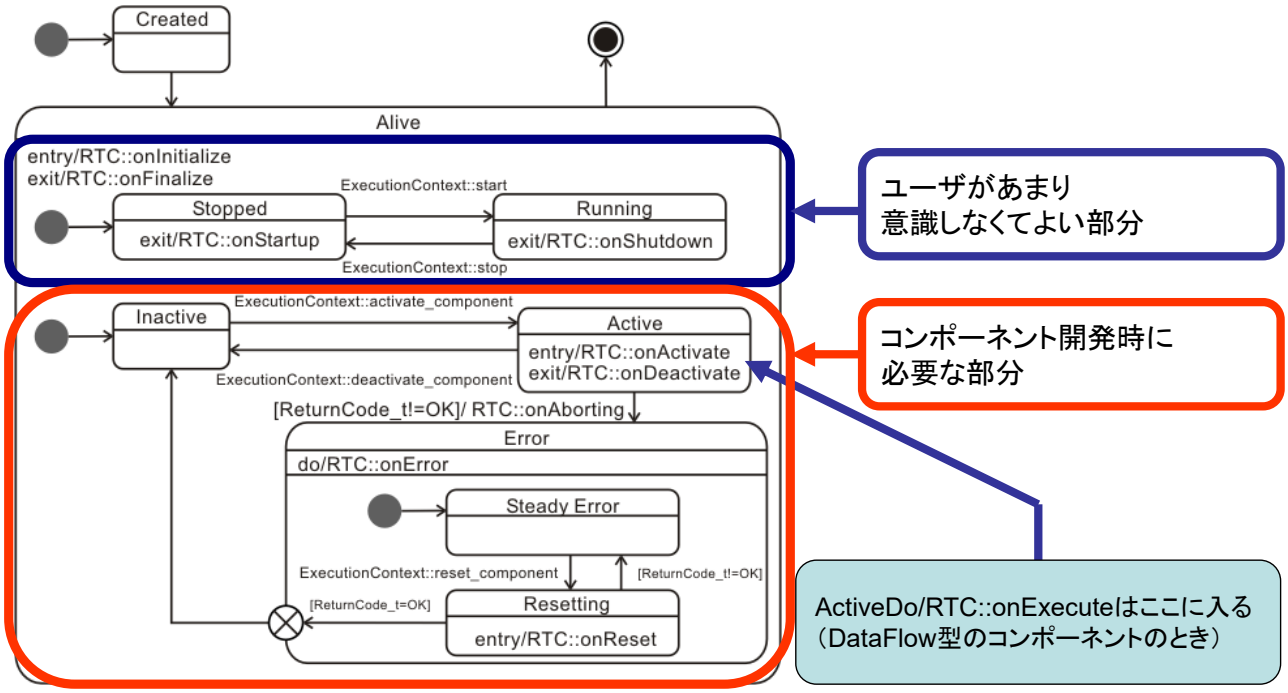
コンポーネントの仕様の入力

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

コンポーネント内の状態遷移

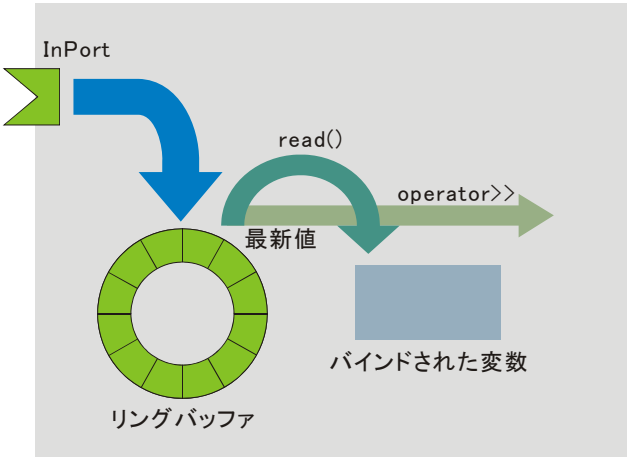


アクティビティ

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

InPort

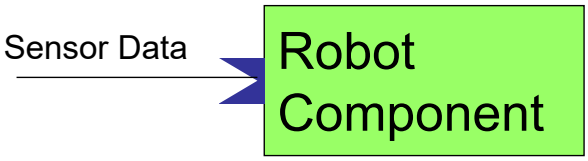
- InPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- InPortのメソッド
  - read(): InPort バッファからバインドされた変数へ最新値を読み込む
  - >> : ある変数へ最新値を読み込む



基本的にOutPortと対になる

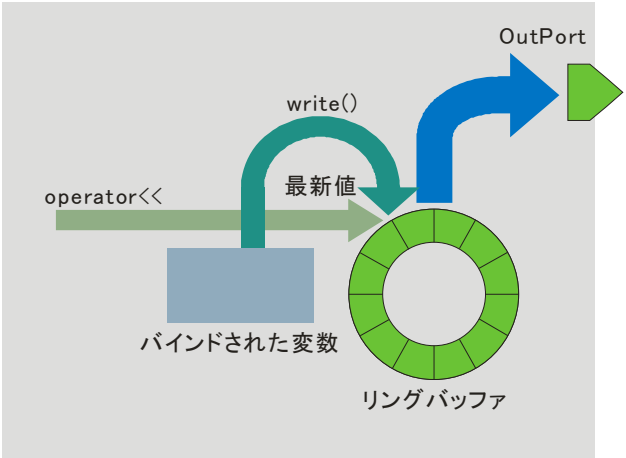
データポートの型を
同じにする必要あり

例



OutPort

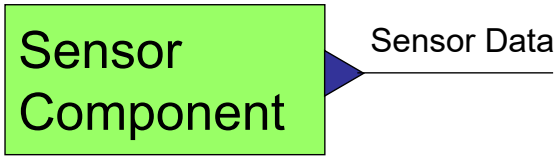
- OutPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- OutPortのメソッド
  - write(): OutPort バッファへバインドされた変数の最新値として書き込む
  - >> : ある変数の内容を最新値としてリングバッファに書き込む



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



データ変数

```
struct TimedShort
 {
      Time tm;
      short data;
 };
```

- 基本型
  - tm：時刻
  - data: データそのもの

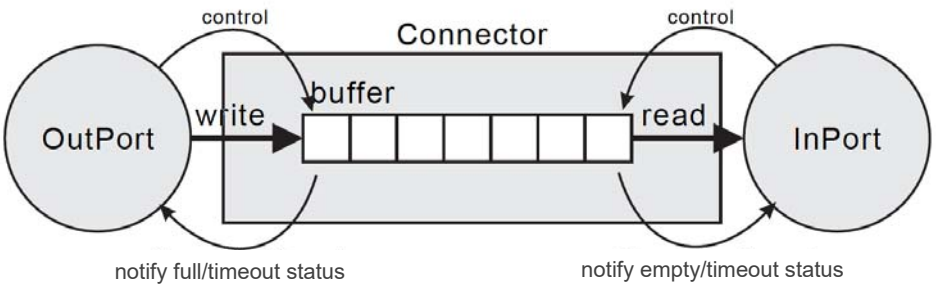
0.2.0では自動で現在時刻をセットしていたが、0.4.0では必要に応じて、手動でセットする必要あり

```
struct TimedShortSeq
 {
      Time tm;
      sequence<short> data;
 };
```

- シーケンス型
  - data[i]: 添え字によるアクセス
  - data.length(i): 長さiを確保
  - data.length(): 長さを取得
- データを入れるときにはあらかじめ長さをセットしなければならない。
- CORBAのシーケンス型そのもの
- 今後変更される可能性あり

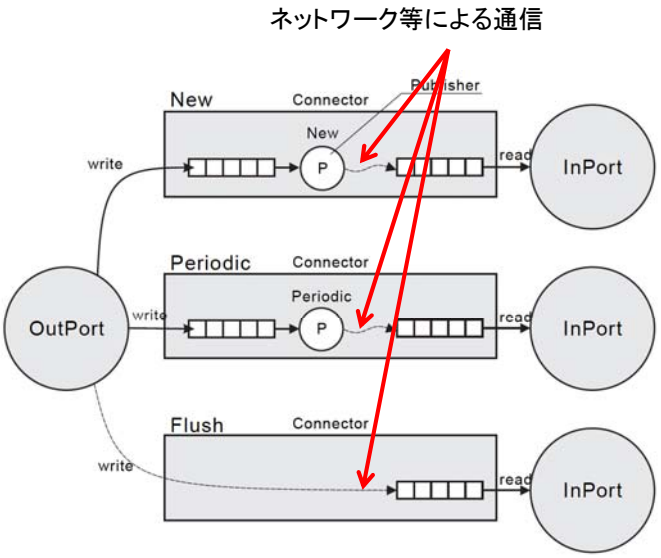
データポートモデル

- Connector:
  - バッファと通信路を抽象化したオブジェクト。OutPortからデータを受け取りバッファに書き込む。InPortからの要求に従いバッファからデータを取り出す。
  - OutPortに対してバッファフル・タイムアウト等のステータスを伝える。
  - InPortに対してバッファエンプティ・タイムアウト等のステータスを伝える。
- OutPort:
  - アクティビティからの要求によってデータをコネクタに書き込むオブジェクト
- InPort:
  - アクティビティからの要求によってデータをコネクタから読み出すオブジェクト



Push型データポートモデル

- Connector
  - 実際には間に通信が入る可能性がある
- 3つの送信モデル
  - “new”,“periodic”,“flush”
  - パブリッシャによる実現
- バッファ、パブリッシャ、通信インターフェースの3つをConnectorに内包

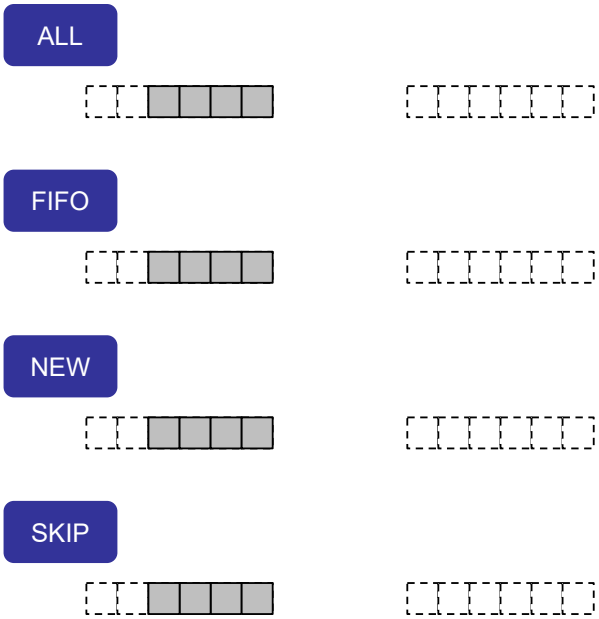


Pushポリシー

- バッファ残留データ
  - 送り方のポリシ

ポリシ	送り方
ALL	全部送信
FIFO	先入れ後だしで1個ずつ送信
NEW	最新値のみ送信
SKIP	n個おきに間引いて送信

  - データ生成・消費速度を考慮して設定する必要がある。



予稿にはNEWの説明にLIFOと記述していましたが正確にはLIFOではなく最新値のみの送信です。LIFO形式のポリシを導入するかどうかは検討中です。ご意見ください。

サービスポート

用途：他のコンポーネントの提供するサービスを必要なときだけ利用したい時
例：カメラからの画像の取得

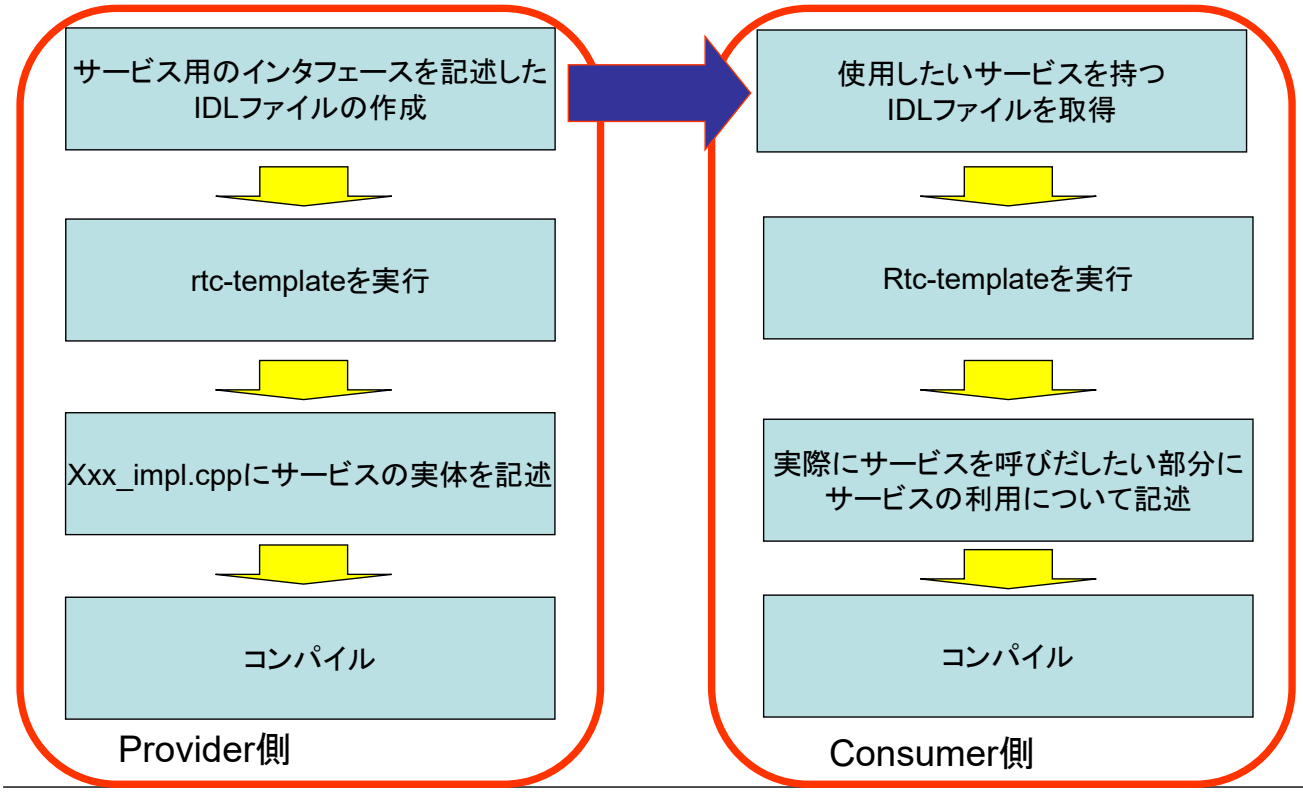
サービス用のインターフェース(IDL)を定義する

プロバイダ
- インターフェースの実装
- コンポーネントに組み込む（宣言）
- Portへのバインド

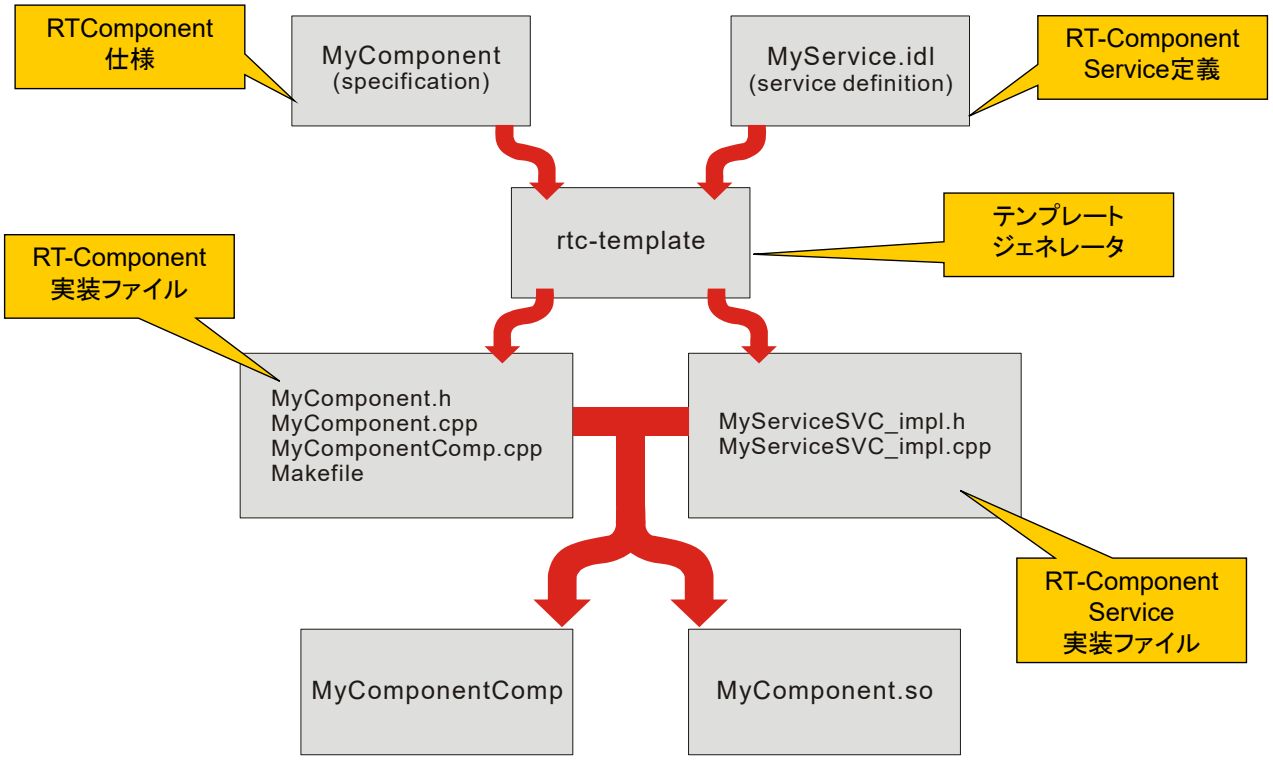
コンシューマ
- スタブを組み込む
- コンシューマを宣言
- Portへのバインド

rtc-templateでコンポーネントを作れば上記のことをほとんど気にしないで作れる

サービスポートを使うためのステップ



生成されるファイル



IDLと実装

IDL(CORBA)定義

```
interface MyRobot
{
  // ゲインをセットする
  void setPosCtrlGain(in short axis, in double gain);
  // ゲインを取得する
  double getPosCtrlGain(in int axis);
};
```

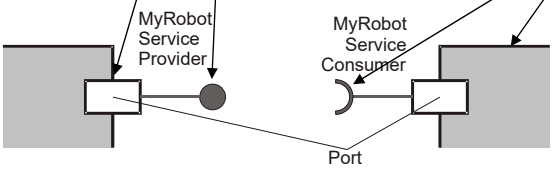
サービス実装(雛形はrtc-templateによる自動生成)

```
class MyRobot_impl
{
  /* この例ではm_robo はロボットを実際に制御する
   * クラスのインスタンスであると仮定する.
   */
  void setPosCtrlGain(const int axis, const double gain)
  {
    // 位置制御ゲインを設定
   m_robo.set_pos_ctrl_gain(axis, gain);
  }
  /* 中略*/
};
```

サービスプロバイダ・コンシューマ

```
class MyRoboComponent
{
private:
  // MyRobot サービスのポートを宣言
  RTC::CorbaPort m_port;
  // MyRobot サービスのインスタンスを宣言
  MyRobot_impl m_robot;
public:
  ManipulatorComponent(Manager manager)
{
  // ポートにサービスを登録
  m_port.registerProvider("Robo0", "MyRobot",
                          m_robot);
  // ポートをコンポーネントに登録
  registerPort(m_port);
}
```

```
class MyRobotUser
{
private:
  // マニピュレータサービスのポートを宣言
  RTC::CorbaPort m_port;
  // サービスコンシューマのインスタンスを宣言
  RTC::CorbaConsumer<MyRobot> m_robot;
public:
  any_functions()
  {// サービスの利用例
   // ゲインをセット
   m_robot->setPosCtrlGain(0, 1.0);
   // ゲインを表示
   std::cout << m_robot->get_pos_ctrl_gain(i) <<
      std::endl;
  }
}
```



より詳細な実装については、
サンプルのSimpleServiceを参照

Configuration

- 複数のパラメータを持つ可能性のあるコンポーネントで、動的にパラメータ変更を行うことができる機能

Rtc-template 引数 (ーーconfig=[名前]:[型]:[デフォルト値]) で指定
例
--config=int_param0:int:0 --config=double_param0:double:1.1
※これらは"default"という名前のConfigurationSetとしてソースに埋め込まれる

- rtc-link上でConfiguration Setを変更可能
- 詳しくはConfigSampleを参照

Configurationの実装例

ヘッダ
変数宣言

```
int m_int_param0;
double m_double_param0;
```

実装ファイル
先頭部分：spec定義にて

```
static const char* configsample_spec[] = {
    :中略
   "conf.default.int_param0", “0",
   "conf.default.double_param0", “1.1",
    :中略};
```

onInitialize()にて

```
bindParameter(“int_param0”, m_int_param0, “0”);
bindParameter(“double_param0”, m_double_param0, “1.1”);
```

istream operator>>が定義されている型であればどんな型でも可能

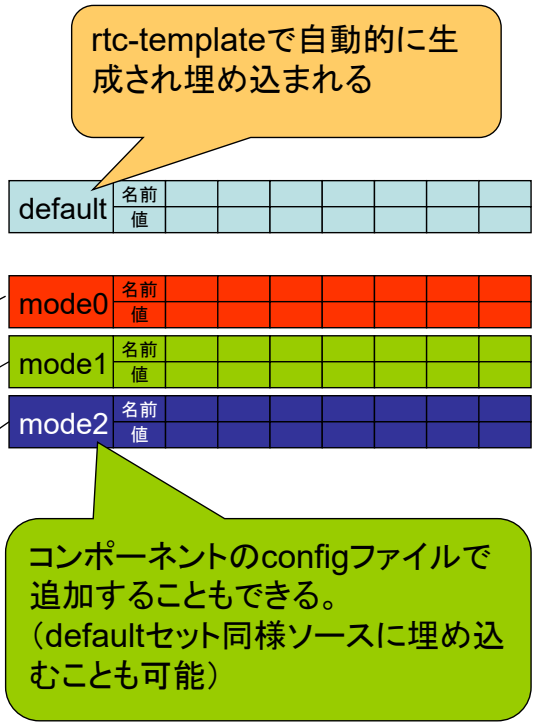
Configuration

rtc.confにて

```
: 略
category.component.config_file: comp.conf
:略
```

comp.confにて

```
conf.mode0.int_param0: 2
conf.mode0.double_param0: 3.14
conf.mode1.int_param0:3
conf.mode1.double_param0: 6.28
conf.mode2.int_param0:4
conf.mode2.double_param0: 12.56
```



提言

- 自前主義はやめよう！！
  - 書きたてのコードより、いろいろな人に何万回も実行されたコードのほうが動くコードである！！
  - 自分にとって本質的でない部分は任せて、本当にやりたい部分・やるべき部分のコードを書こう！！
  - 誰かがリリースしたプログラムは一度は動いたことがあるプログラムである！！
  - 人のコードを読むのが面倒だからと捨ててしまうのはもったいない！！
- オープンソースにコミットしよう！！
  - 臆せずMLやフォーラムで質問しよう！！
  - どんなに初歩的な質問でも他の人にとっては価値ある情報である。
  - 要望を積極的にあげよう！！
  - できればデバッグしてパッチを送ろう！

まとめ

- RTミドルウエアの概要
  - 基本概念
  - モジュール化
  - 標準化
  - RTMコミュニティー