2017年11月28日（火）
第12回AIツール入門講座

ロボット用ソフトウェア開発のための ミドルウェア「RTミドルウェア」入門 （基礎編） 第１部 RTミドルウエア: OpenRTM-aist概要

国立研究開発法人産業技術総合研究所
ロボットイノベーション研究センター
ロボット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
- RTコンポーネントの開発
  - RTCプログラミングについて
- RTMコミュニティー活動
  - サマーキャンプ、コンテスト、etc.
- まとめ

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…

Joystick

Joystick
software

Robot Arm
Control software

Robot Arm

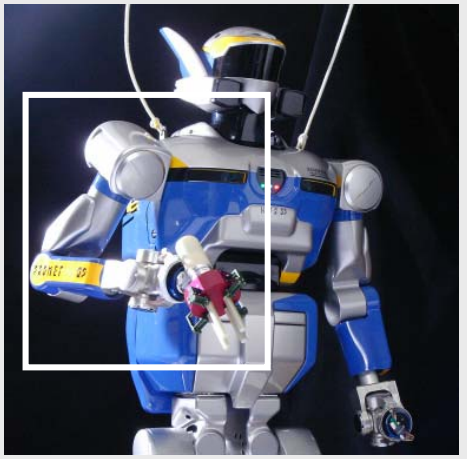
互換性のあるインターフェース同士は接続可能

従来のシステムでは…

Joystick

Joystick
software

Humanoid's Arm
Control software



Humanoid's Arm

Robot Arm
Control software

Robot Arm

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

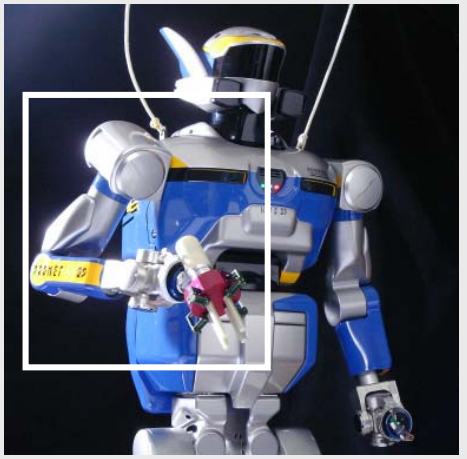
Joystick

Joystick
software

compatible
arm interfaces

Arm A
Control software

Arm B
Control software



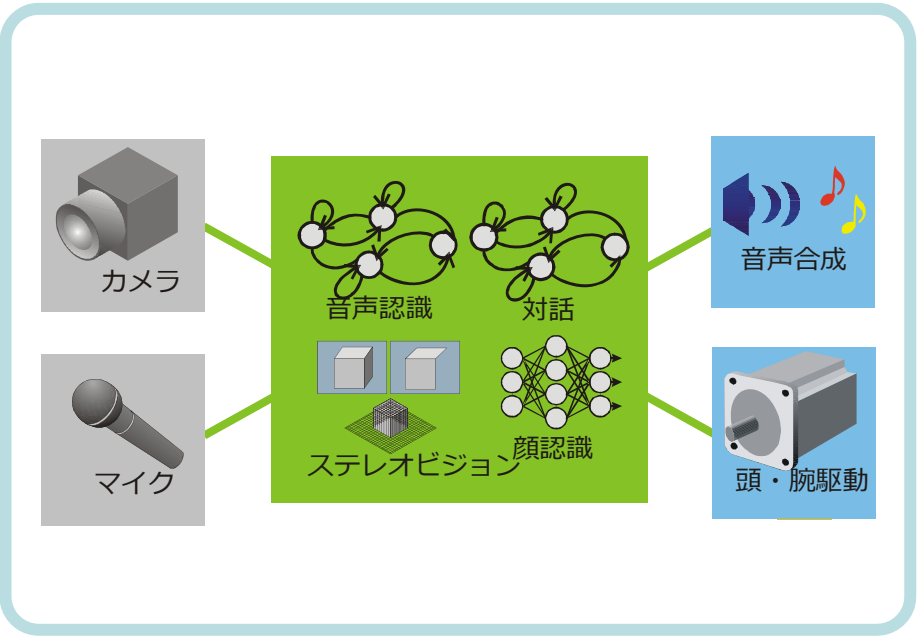
Humanoid's Arm

Robot Arm

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

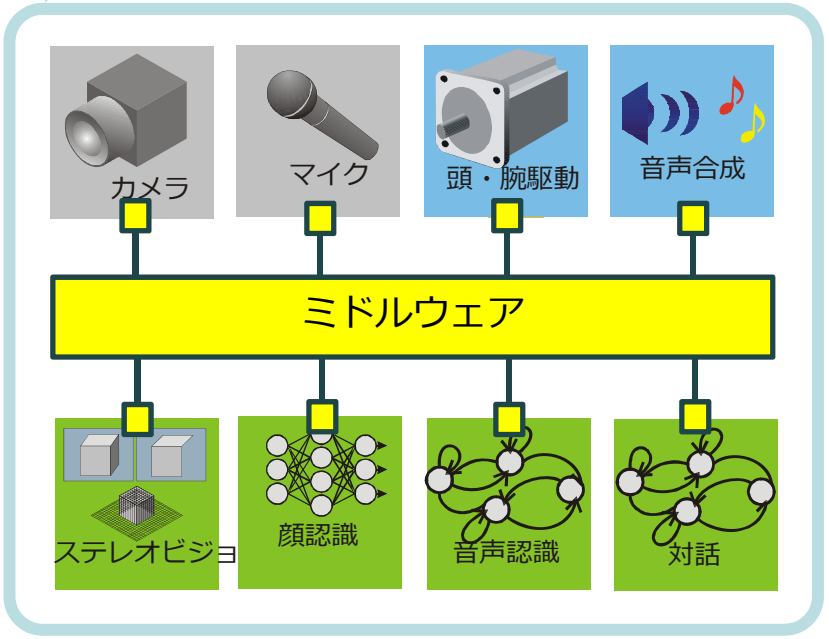
ロボットソフトウェア開発の方向

従来型開発



- ✓ 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

コンポーネント指向開発



- ✓ 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

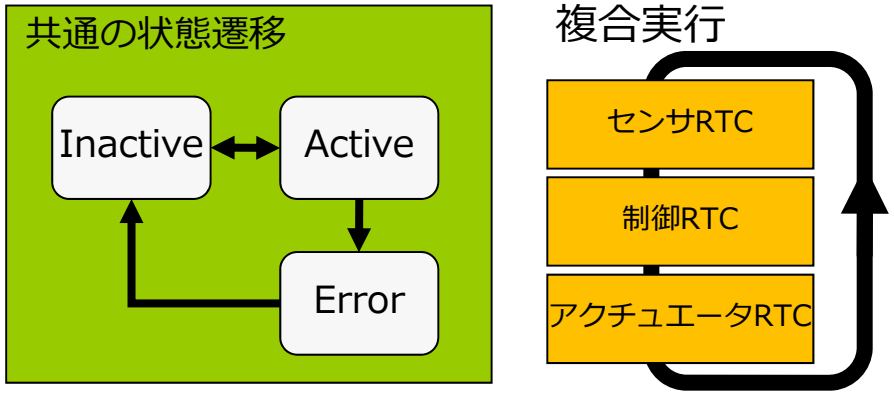
RTコンポーネント化のメリット

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

アクティビティ・実行コンテキスト

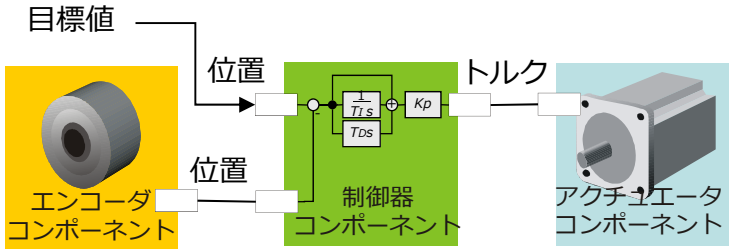


ライフサイクルの管理・コアロジックの実行

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

サーボの例

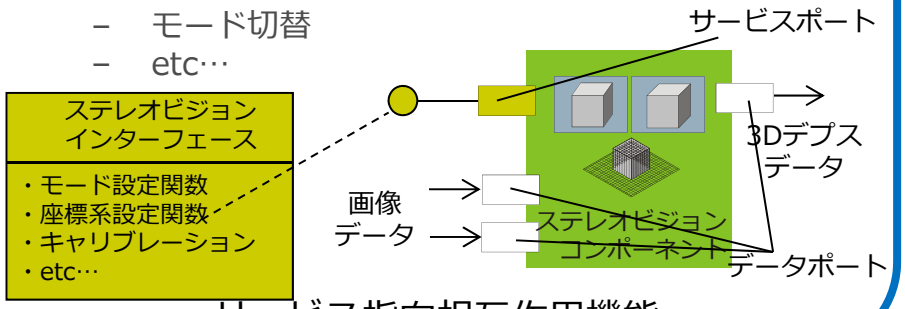


データ指向通信機能

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…

ステレオビジョンの例

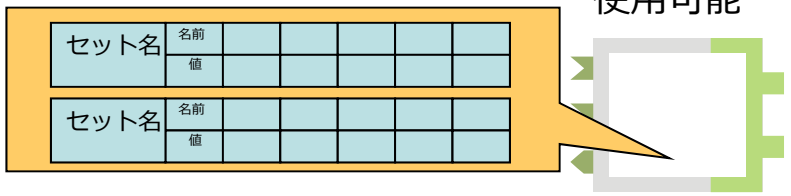


サービス指向相互作用機能

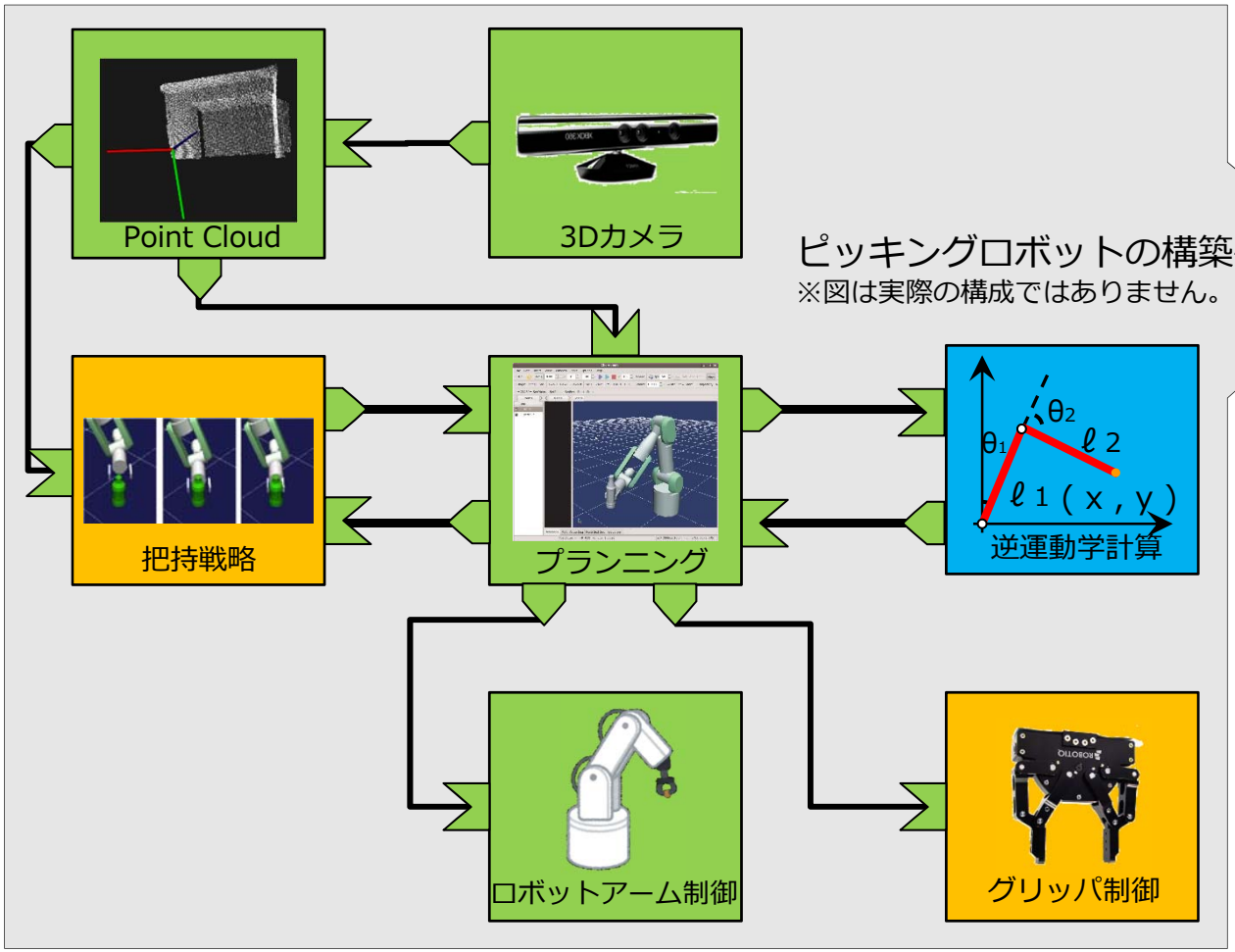
コンフィギュレーション

- 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

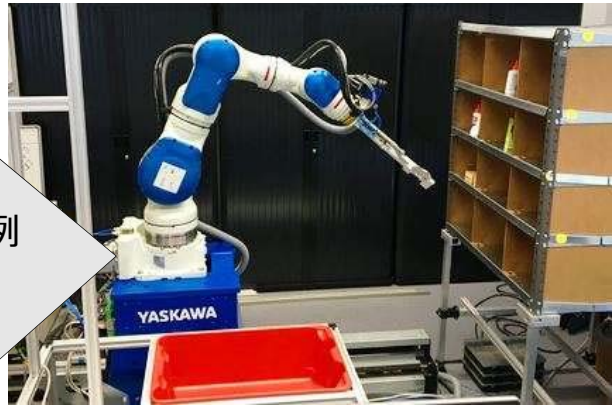
複数のセットを
動作時に
切り替えて
使用可能



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。

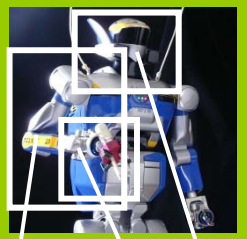
ロボットA

RTC (C++) | RTC (C++)
RTM
VxWorks

ロボットB

RTC (C++) | RTC (C++)
RTM
QNX

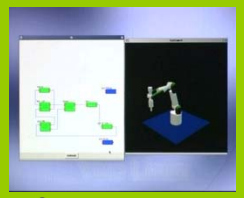
ロボットC



RTC (C++) | RTC (C++) | RTC (Python)
RTM
Real-time Linux

ネットワーク

Linux
RTM
RTC (Java) | RTC (Python)



アプリケーション

Windows
RTM
RTC (C++) | RTC (Java)

操作デバイス

uITRON
RTM
RTC (C++) | RTC (C++)

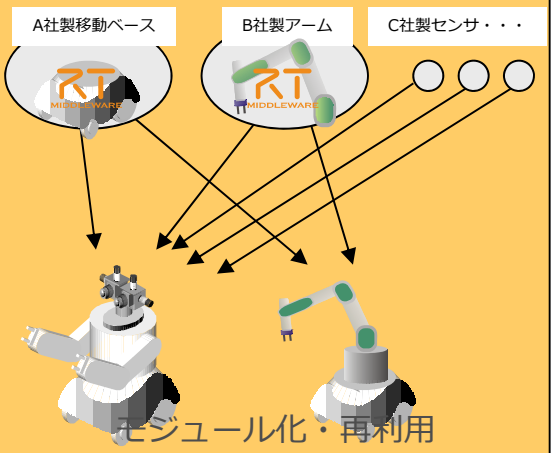
センサ

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ミドルウエアの目的

モジュール化による問題解決

コストの問題

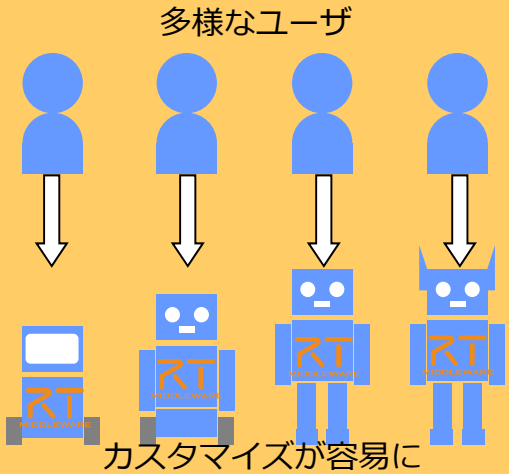


ロボットの低コスト化

技術の問題

最新技術を利用可能

ニーズの問題



多様なニーズに対応

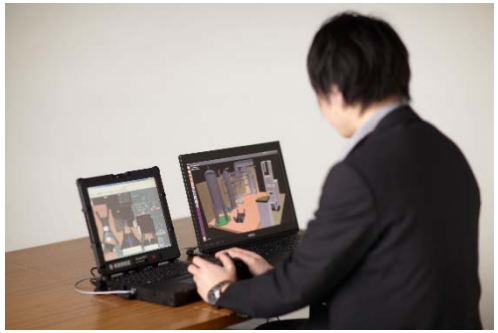
ロボットシステムインテグレーションによるイノベーション

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



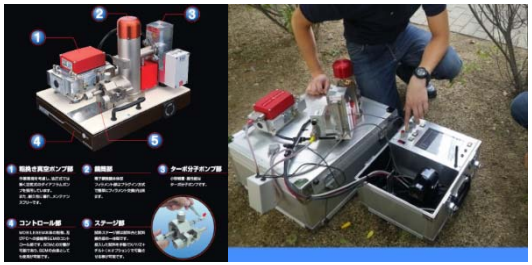
災害対応ロボット操縦シミュレータ：
NEDO/千葉工大

HIRO, NEXTAGE open: Kawada Robotics

RAPUDA : Life Robotics

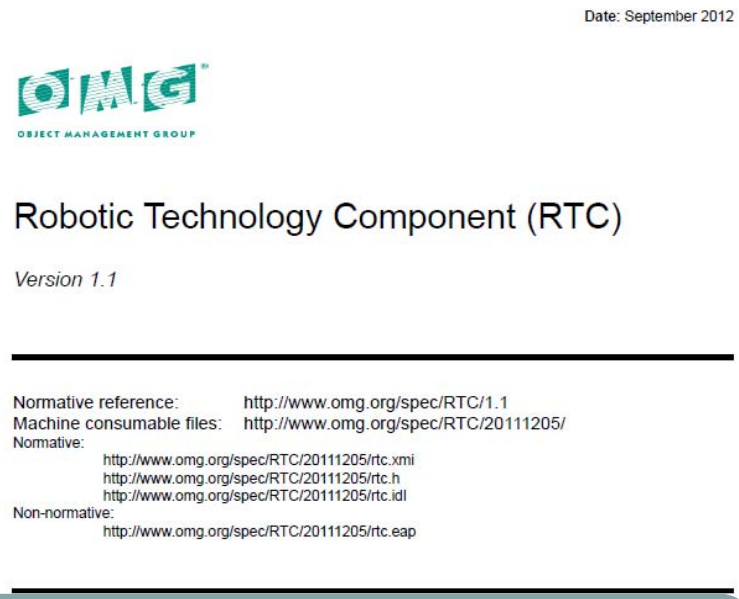
ビュートローバーRTC/RTC-BT(VSTONE)

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

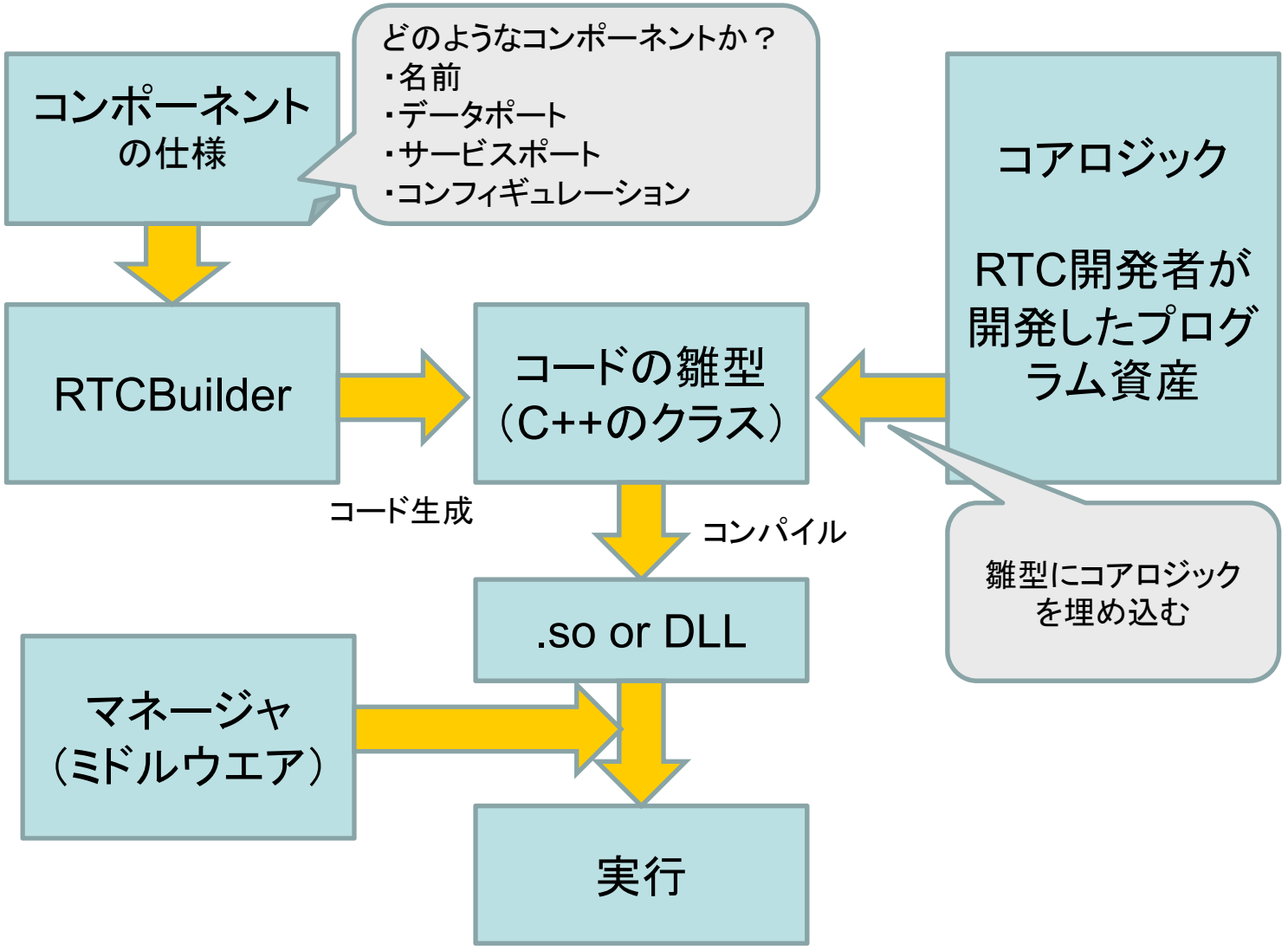
RTミドルウエア互換実装は10種類以上

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	×

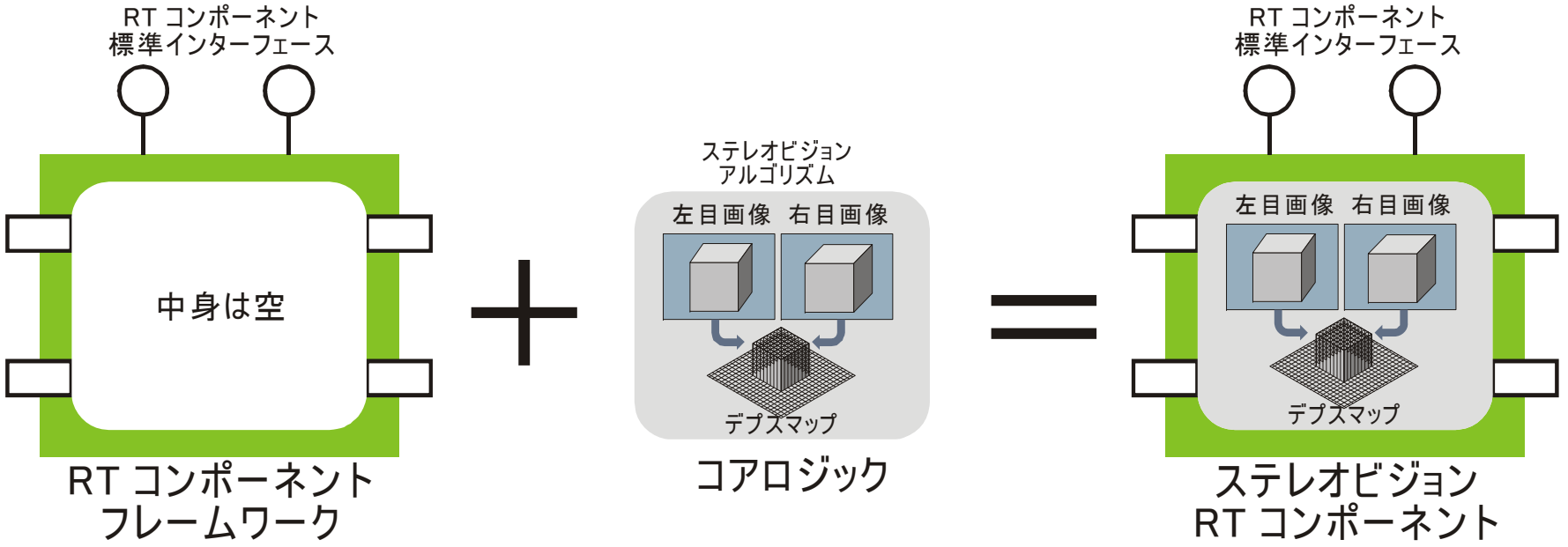
特定のベンダが撤退しても
ユーザは使い続けることが可能

RTコンポーネントの開発

OpenRTMを使った開発の流れ

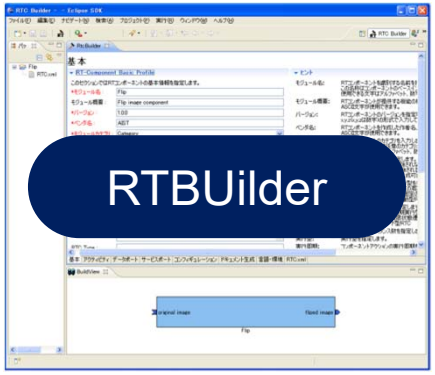


フレームワークとコアロジック

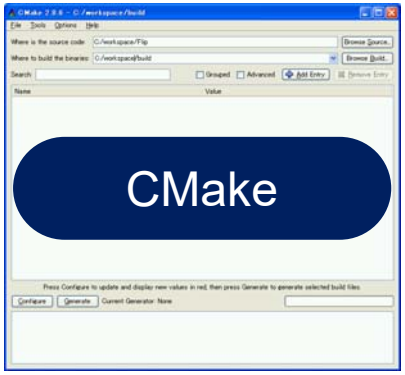


RTCフレームワーク＋コアロジック＝RTコンポーネント

コンポーネントの作成
（Windowsの場合）

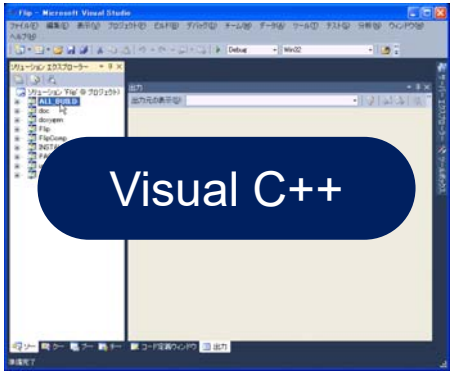


コンポーネントの
仕様の入力



VCのプロジェクト
ファイルの生成

テンプレートコード
の生成



実装および
VCでコンパイル
実行ファイルの生成

コード例

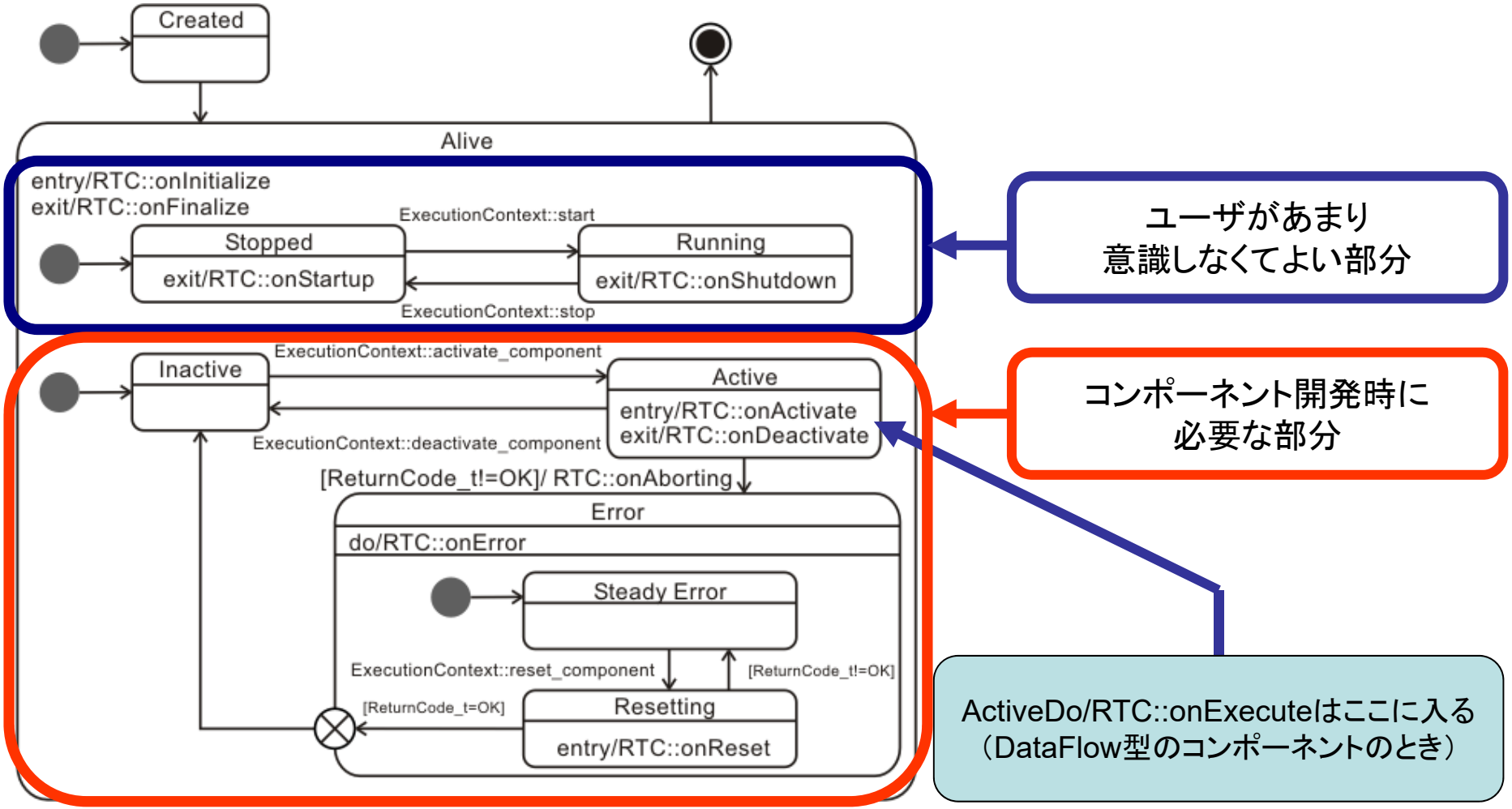
- 生成されたクラスのメンバー関数に必要な処理を記述
- 主要な関数
  - onExecute (周期実行)
- 処理
  - InPortから読む
  - OutPortへ書く
  - サービスを呼ぶ
  - コンフィギュレーションを読む

```
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
  //   :
};
```

コンポーネント内の状態遷移



コールバック関数

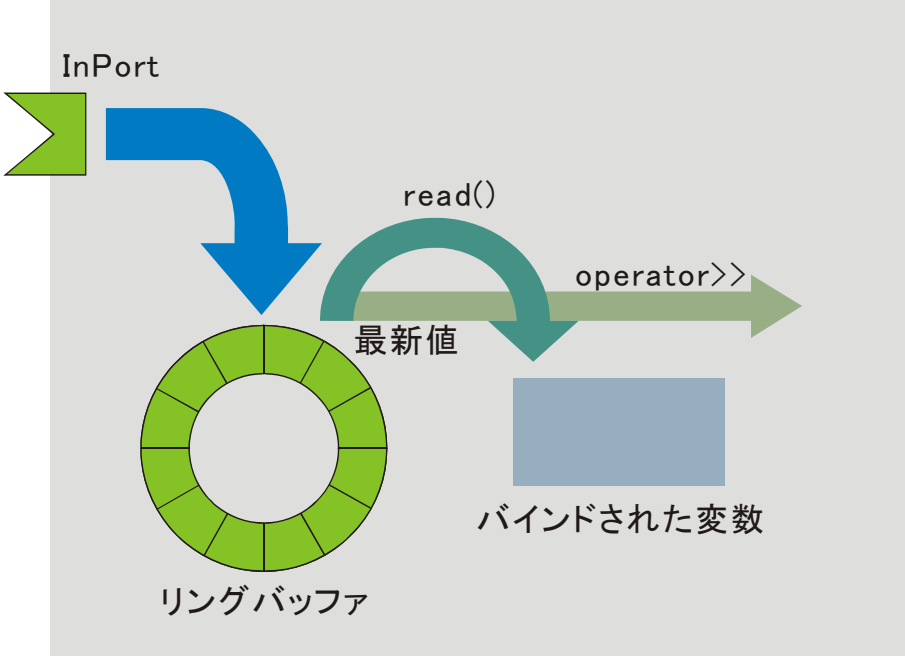
RTCの作成＝コールバック関数に処理を埋め込む

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

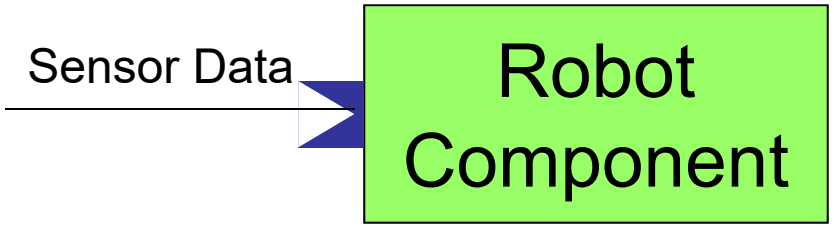
- InPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- InPortのメソッド
  - read(): InPort バッファからバインドされた変数へ最新値を読み込む
  - >> : ある変数へ最新値を読み込む



基本的にOutPortと対になる

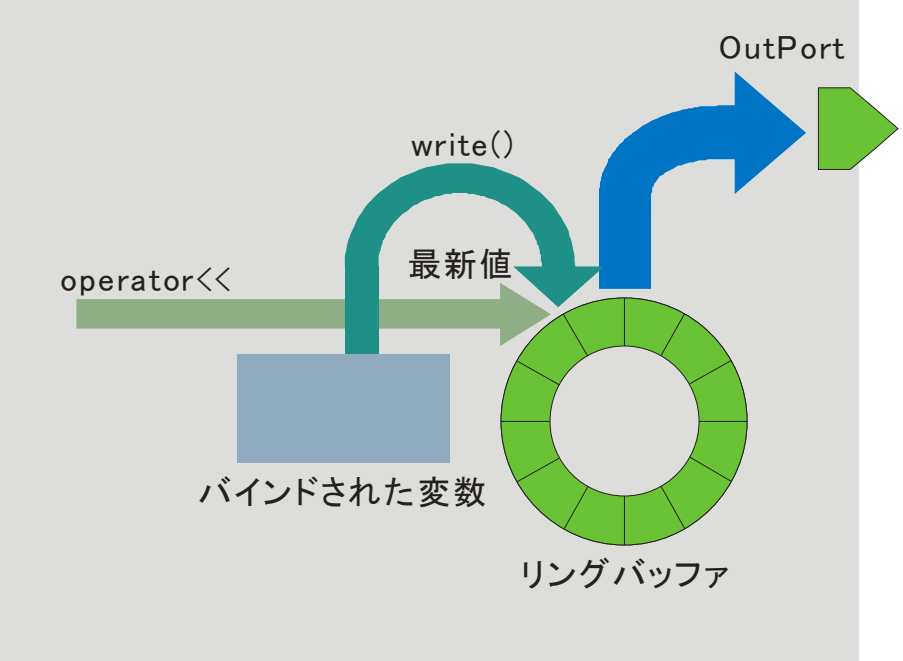
データポートの型を
同じにする必要あり

例



OutPort

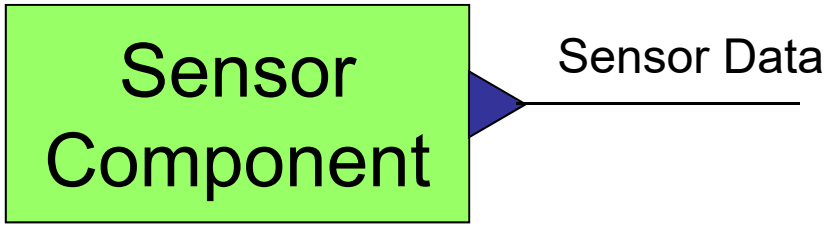
- OutPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- OutPortのメソッド
  - write(): OutPort バッファへバインドされた変数の最新値として書き込む
  - >> : ある変数の内容を最新値としてリングバッファに書き込む



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



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

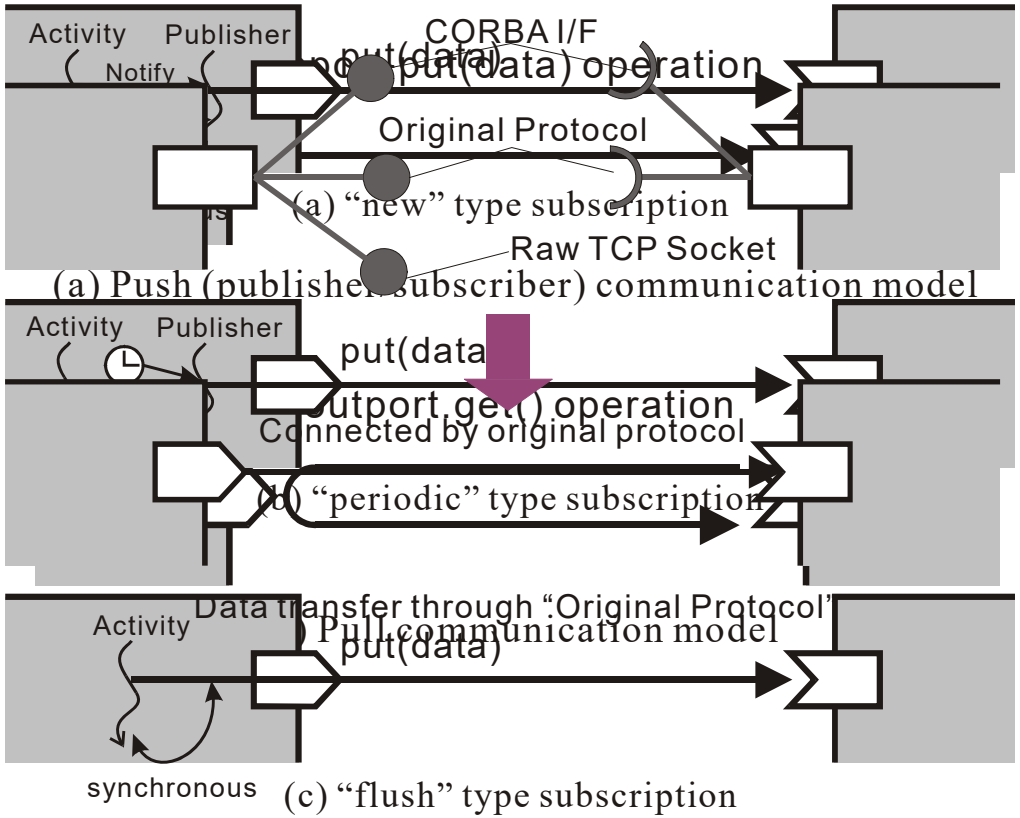
```
struct TimedShortSeq
 {
      Time tm;
      sequence<short> data;
 };
```

- 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

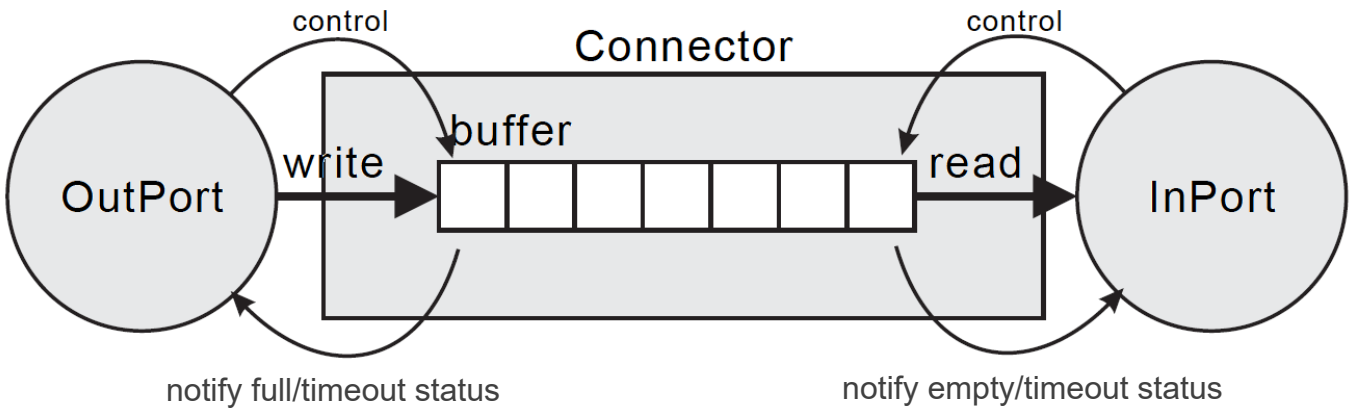
データポート

- データ指向(Data Centric)なストリームポート
  - 型：long, double×6, etc…
    - ユーザが任意に定義可能
  - 出力：OutPort
  - 入力：InPort
- 接続制御（接続時に選択可能）
  - Interface type
    - CORBA,TCP socket, other protocol, etc…
  - Data flow type
    - push/pull
  - Subscription type
    - Flush, New, Periodic



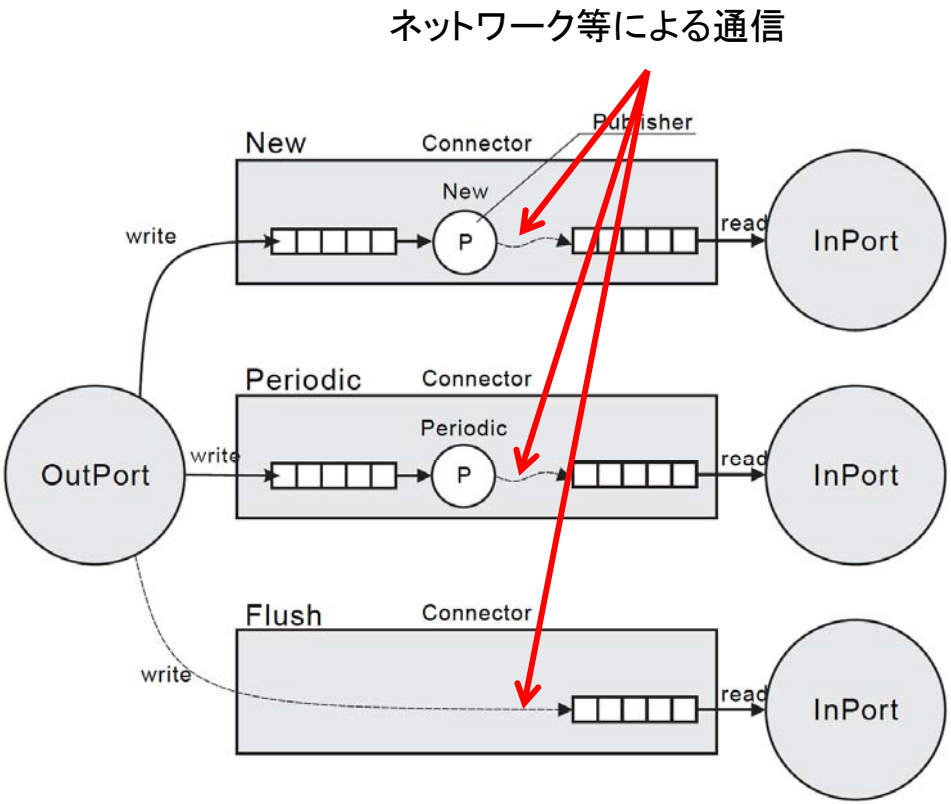
データポートモデル

- 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



バッファインターフェース

- バッファ操作のためのインターフェースを定義
  - より詳細な操作を提供
  - バッファフル/エンプティを検出可能に
  - タイムアウトを検出可能に

BufferBase <T>
advanceRptr(long n) : ReturnCode advanceWptr(long n) : ReturnCode empty() : bool full() : bool get(T& data) : ReturnCode put(T& data) : ReturnCode read(T& data, int sec, int nsec) : ReturnCode write(T& data, int sec, int nsec) : ReturnCode

リターンコード

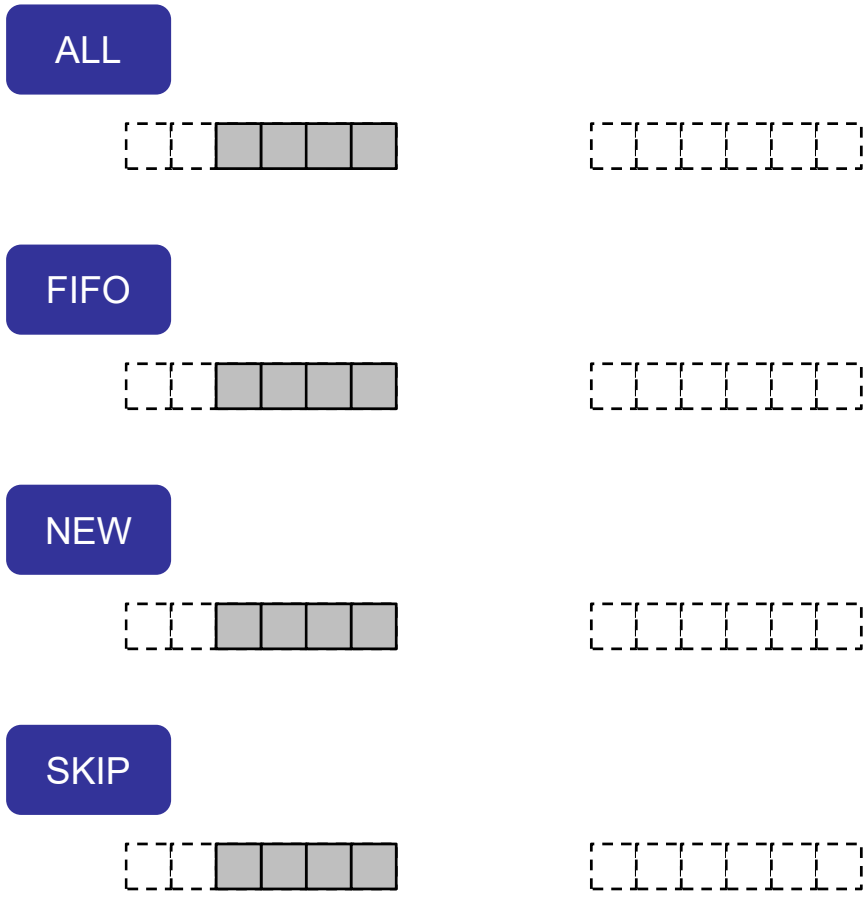
- BUFFER OK 正常終了
- BUFFER ERROR エラー終了
- BUFFER FULL バッファフル状態
- BUFFER EMPTY バッファ空状態
- NOT SUPPORTED サポート外の要求
- TIMEOUT タイムアウト
- PRECONDITION NOT MET 事前状態を満たさない

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。

rtc.confについて

RT Component起動時の登録先NamingServiceや、登録情報などについて記述するファイル

記述例：

corba.nameservers: localhost:9876

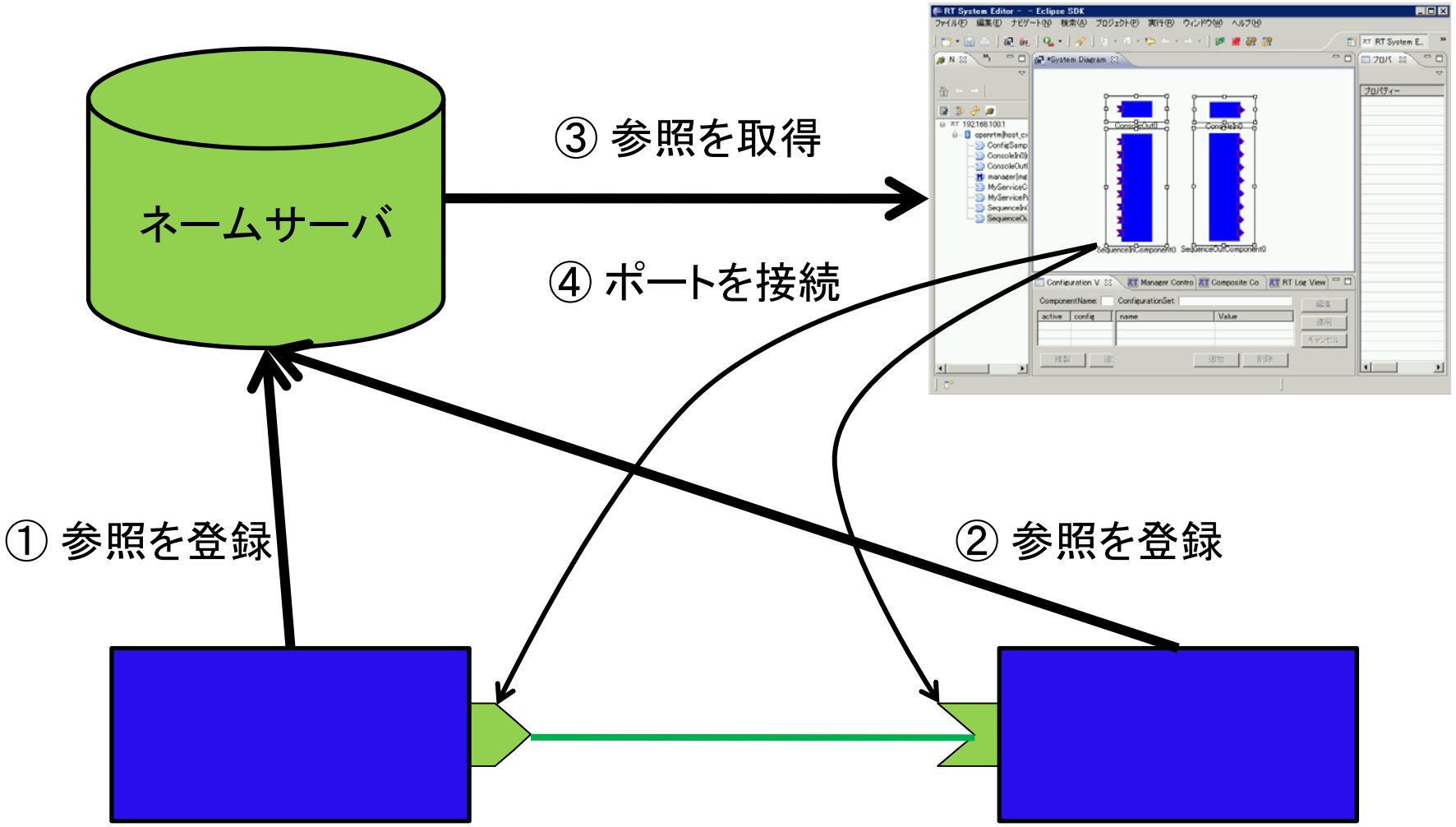
naming.formats: SimpleComponent/%n.rtc

（詳細な記述方法は etc/rtc.conf.sample を参照）

以下のようにすると、コンポーネント起動時に読み込まれる

./ConsoleInComp –f rtc.conf

動作シーケンス



ネーミングサービス設定

corba.nameservers	host_name:port_numberで指定、デフォルトポートは2809(omniORBのデフォルト)、複数指定可能
naming.formats	%h.host_cxt/%n.rtc →host.host_cxt/MyComp.rtc 複数指定可能、0.2.0互換にしたければ、 %h.host_cxt/%M.mgr_cxt/%c.cat_cxt/%m.mod_cxt/%n.rtc
naming.update.enable	“YES” or “NO”: ネーミングサービスへの登録の自動アップデート。コンポーネント起動後にネームサービスが起動したときに、再度名前を登録する。
naming.update.interval	アップデートの周期[s]。デフォルトは10秒。
timer.enable	“YES” or “NO”: マネージャタイマ有効・無効。 naming.updateを使用するには有効でなければならない
timer.tick	タイマの分解能[s]。デフォルトは100ms。

必須の項目　　必須でないOption設定

ログ設定

logger.enable	“YES” or “NO”: ログ出力を有効・無効
logger.file_name	ログファイル名。 %h：ホスト名、%M:マネージャ名,%p：プロセスID 使用可
logger.date_format	日付フォーマット。strftime(3)の表記法に準拠。 デフォルト：%b %d %H:%M:%S → Apr 24 01:02:04
logger.log_level	ログレベル： SILENT, ERROR, WARN, NORMAL, INFO, DEBUG, TRACE, VERBOSE, PARANOID SILENT：何も出力しない PARANOID：全て出力する ※以前はRTC内で使えましたが、現在はまだ使えません。

必須の項目　　必須でないOption設定

その他

corba.endpoints	IP_Addr:Port で指定：NICが複数あるとき、ORBをどちらでlistenさせるかを指定。Portを指定しない場合でも":"が必要。 例 "corba.endpoints: 192.168.0.12:" NICが2つある場合必ず指定。 (指定しなくても偶然正常に動作することもあるが念のため。)
corba.args	CORBAに対する引数。詳細はomniORBのマニュアル参照。
[カテゴリ名]. [コンポーネント名]. config_file または [カテゴリ名]. [インスタンス名]. config_file	コンポーネントの設定ファイル •カテゴリ名：manipulator, •コンポーネント名：myarm, •インスタンス名myarm0,1,2,… の場合 manipulator.myarm.config_file: arm.conf manipulator.myarm0.config.file: arm0.conf のように指定可能

使いたいNICに割り当てられているIPアドレス

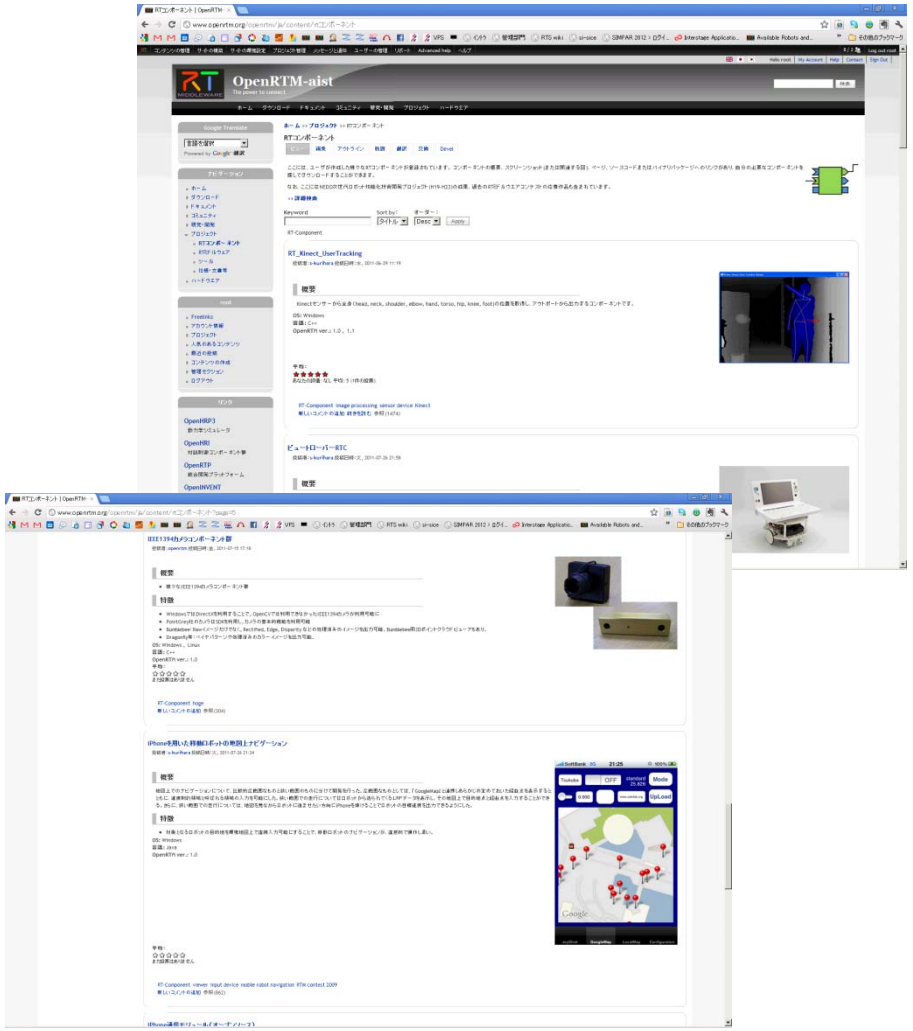
必須の項目　　必須でないOption設定

RTミドルウェアとコミュニティー

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	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



サマーキャンプ

- 毎年夏に1週間開催
- 今年：7月31日～8月4日
- 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
  - 各種奨励賞・審査基準開示:5月頃
  - エントリー〆切：8月21日(SI2017締切)
  - 講演原稿〆切：9月25日
  - 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
- 2016年度実績
  - 応募数：13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）：8件
  - 奨励賞（個人協賛）：5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

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
- RTコンポーネントの開発
  - RTCプログラミングについて
- RTMコミュニティー活動
  - サマーキャンプ、コンテスト、etc.