第1部：OpenRTM-aistおよびRTコンポーネントプログラミングの概要

名城大学
理工学部メカトロニクス工学科
大原賢一

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

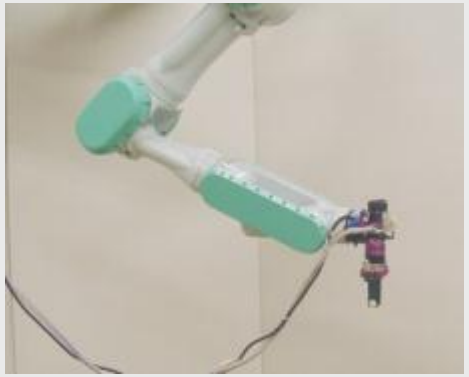
- RT-Middleware（RTM)
  - RT要素のインテグレーションのためのミドルウエア
- RT-Component（RTC)
  - RT-Middlewareにおけるソフトウエアの基本単位

従来のシステムでは…

Joystick

Joystick
software

Robot Arm
Control software



Robot Arm

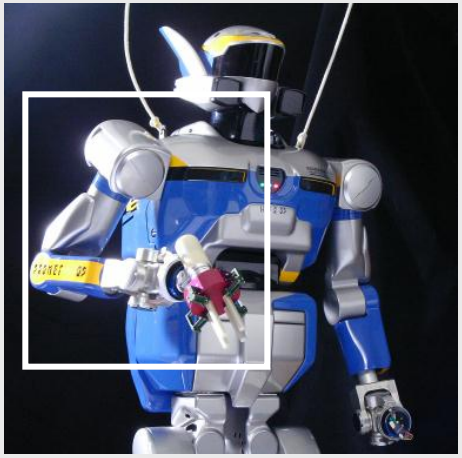
互換性のあるインターフェース同士は接続可能

従来のシステムでは…

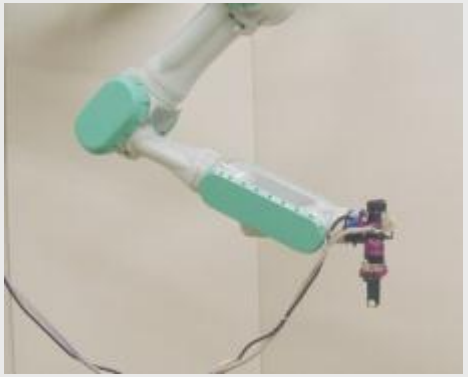
Joystick

Joystick
software

Humanoid's Arm
Control software



Humanoid's Arm



Robot Arm

Robot Arm
Control software

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

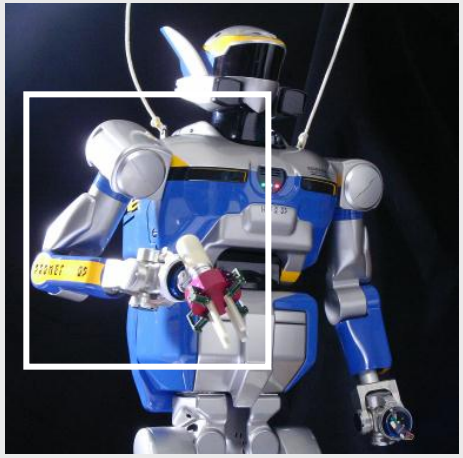
Joystick

Joystick software

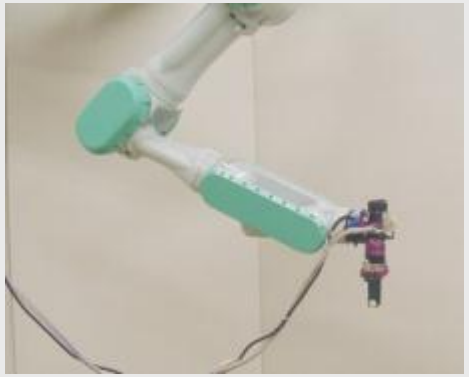
compatible arm interfaces

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

ミドルウエア、コンポーネント、etc…

- 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：CORBA、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

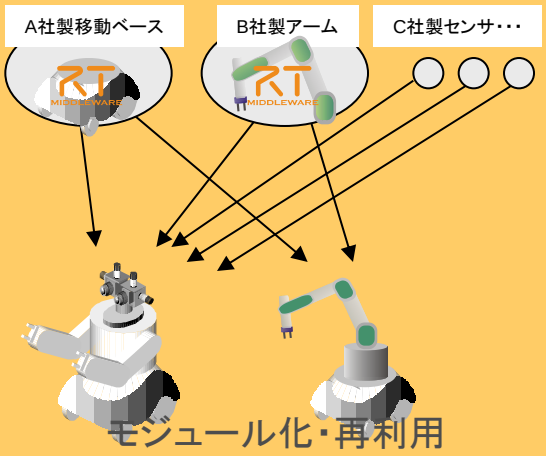
モジュール化のメリットに加えて

- 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

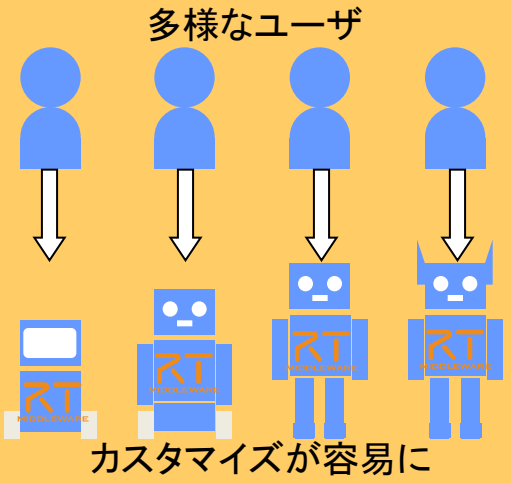


ロボットの低コスト化

技術の問題

最新技術を利用可能

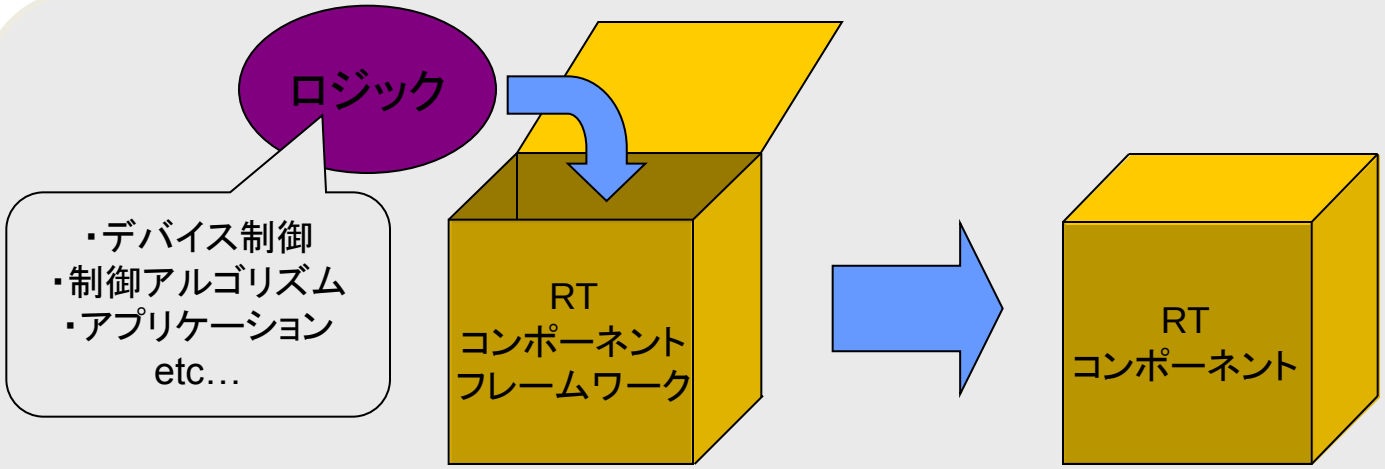
ニーズの問題



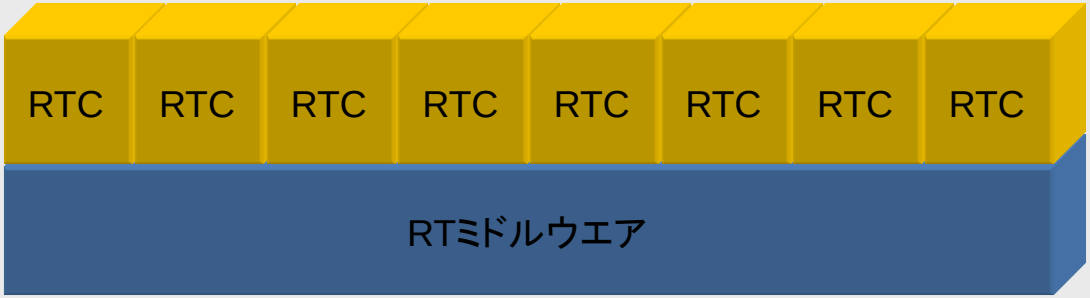
多様なニーズに対応

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

RTミドルウエアとRTコンポーネント



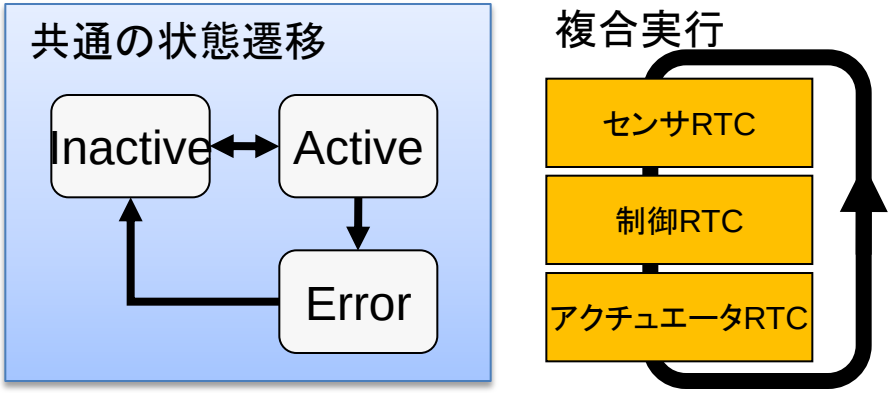
ロジックを箱（フレームワーク）に入れたもの＝RTコンポーネント（RTC）



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

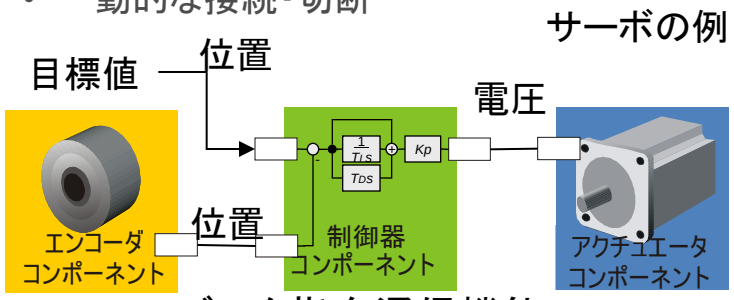
アクティビティ・実行コンテキスト



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

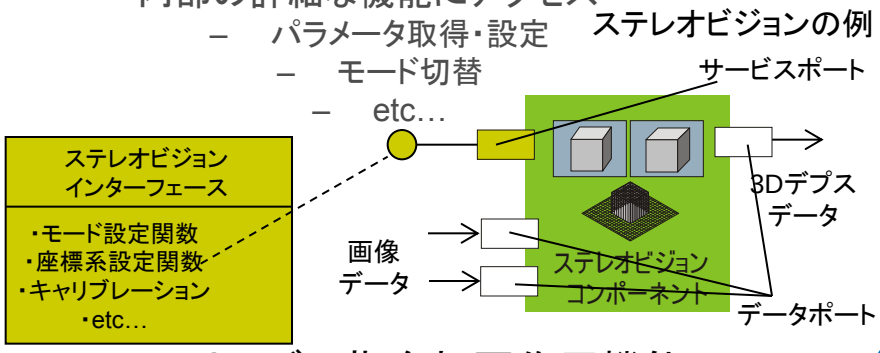
- データ指向ポート
- 連続的なデータの送受信
- 動的な接続・切断



データ指向通信機能

サービスポート

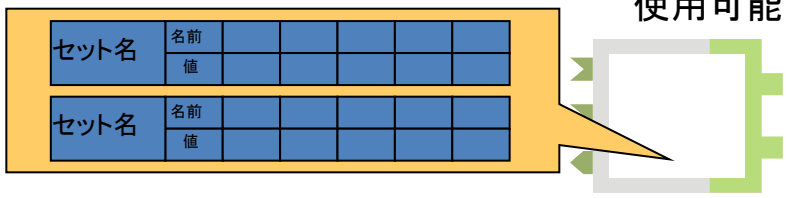
- 定義可能なインターフェースを持つ
- 内部の詳細な機能にアクセス
  - パラメータ取得・設定
  - モード切替
  - etc…



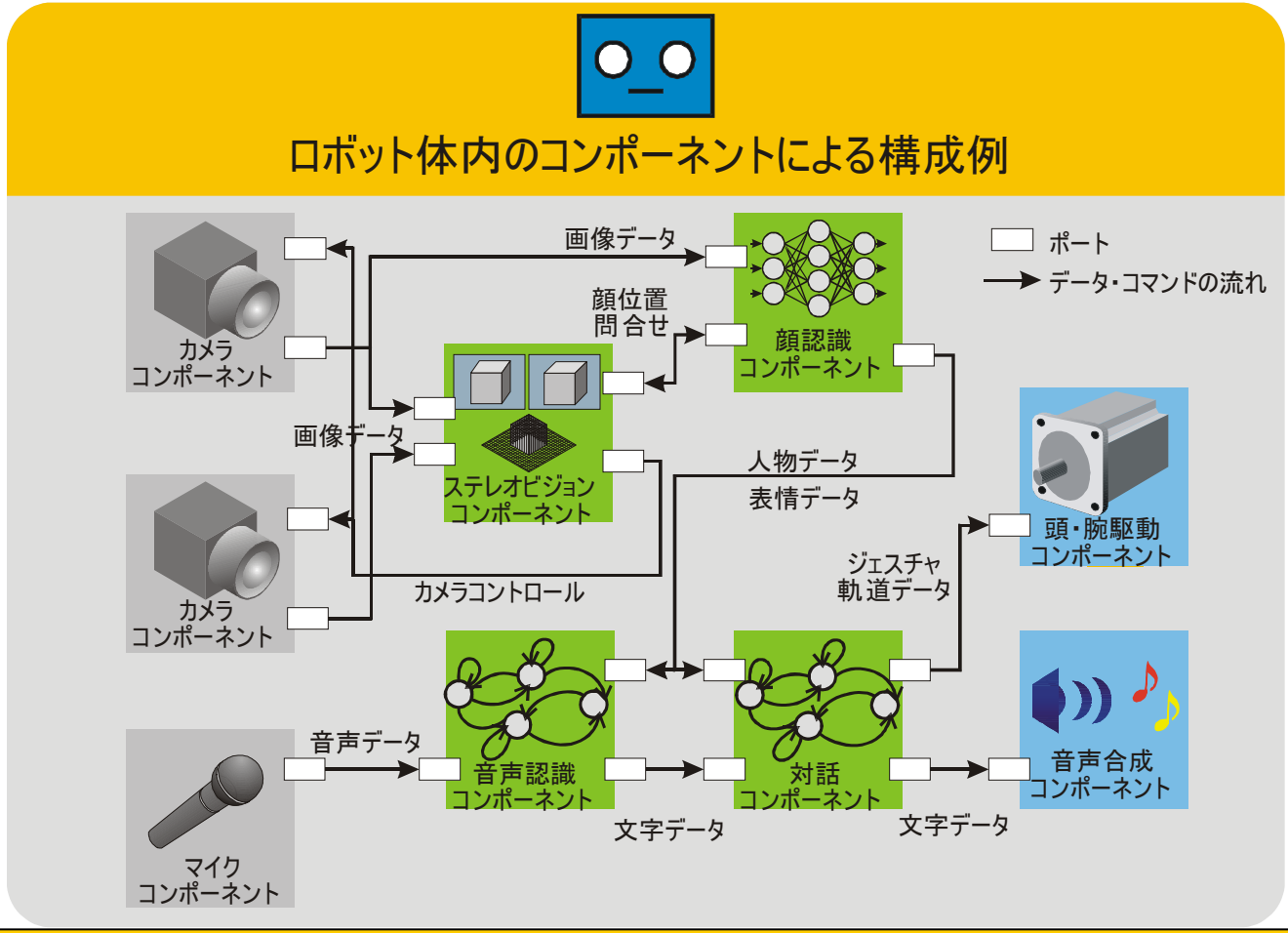
サービス指向相互作用機能

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



RTCの分割と連携



（モジュール）情報の隠蔽と公開のルールが重要

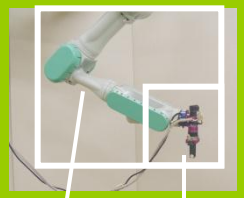
RTミドルウエアによる分散システム

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
Solaris

ロボットB



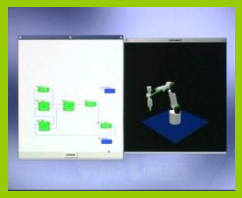
RTC RTC
RTM
FreeBSD

ロボットC

RTC RTC RTC
RTM
ARTLinux

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

TRON
RTM
RTC RTC

センサ

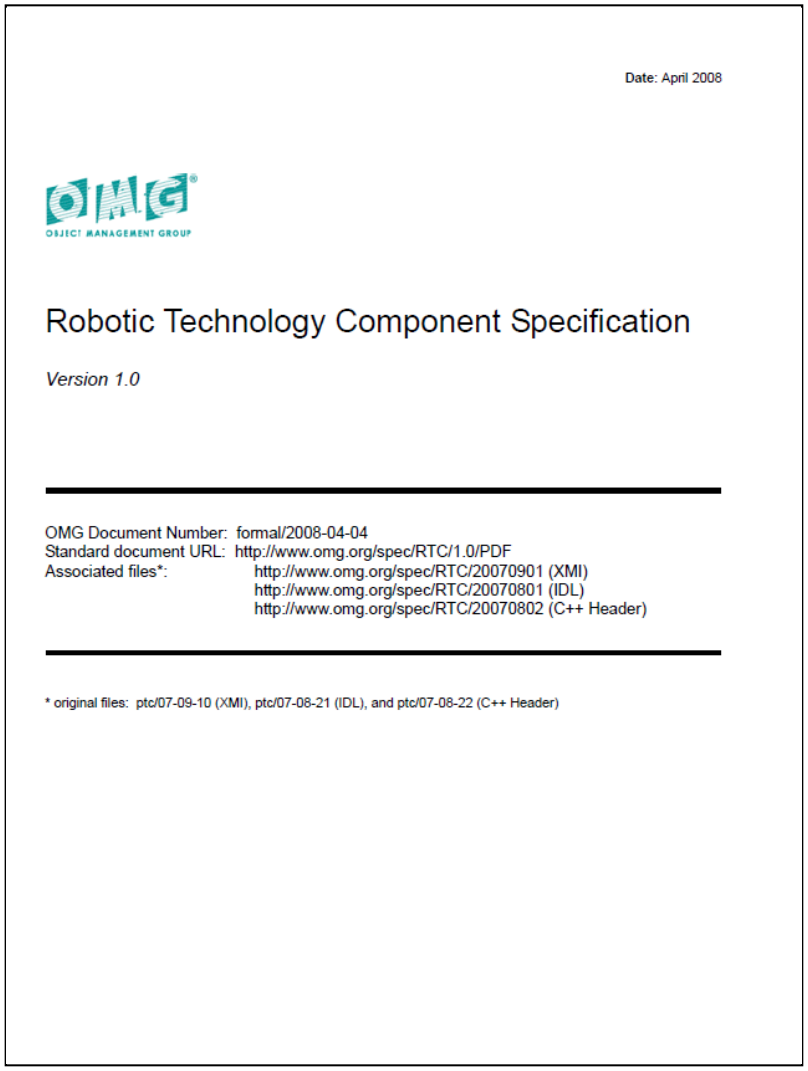
RTC同士の接続は、プログラム実行中に動的に行うことが出来る。

OpenRTM-aist

- コンポーネントフレームワーク + ミドルウエアライブラリ
- コンポーネントインターフェース:
  - OMG Robotic Technology Component Specification ver1.0 準拠
- OS
  - 公式:Linux, FreeBSD, Windows, Mac OS X, QNX
  - 非公式: uITRON, T-Kernel, VxWorks
- 言語:
  - C++ (1.1.0), Python (1.0.0), Java (1.0.0)
  - .NET (implemented by SEC)
- CPU アーキテクチャ (動作実績):
  - i386, ARM, PPC, SH4
  - PIC, dsPIC, SH2, H8 (RTC-Lite)
- ツール (Eclipse プラグイン)
  - テンプレートソースジェネレータ: rtc-template、RTCBuilder
  - システムインテグレーションツール: RTSystemEditor

OMG RTC 標準化

- 2005年9月
  RFP：Robot Technology Components (RTCs) 公開。
- 2006年2月
  Initial Response : PIM and PSM for RTComponent を執筆し提出
  提案者：AIST(日)、RTI(米)
- 2006年4月
  両者の提案を統合した仕様を提案
- 2006年9月
  ABにて承認、事実上の国際標準獲得
  FTFが組織され最終文書化開始
- 2007年8月
  FTFの最後の投票が終了
- 2007年9月
  ABにてFTFの結果を報告、承認
- 2008年4月
  OMG RTC標準仕様公式リリース
- 2010年1月
  OpenRTM-aist-1.0リリース



Date: April 2008

OMG
OBJECT MANAGEMENT GROUP

Robotic Technology Component Specification

Version 1.0

OMG Document Number: formal/2008-04-04
Standard document URL: http://www.omg.org/spec/RTC/1.0/PDF
Associated files*: http://www.omg.org/spec/RTC/20070901 (XMI)
http://www.omg.org/spec/RTC/20070801 (IDL)
http://www.omg.org/spec/RTC/20070802 (C++ Header)

* original files: ptc/07-09-10 (XMI), ptc/07-08-21 (IDL), and ptc/07-08-22 (C++ Header)

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
H-RTM (仮称)	本田R&D	OpenRTM-aist互換、FSM型コンポーネントをサポート

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

PSM(Platform Specific Model): プラットフォーム特化型の仕様

RT-Middleware関連プロジェクト

from 2002～

RTM0.1 リリース

OpenRTM-aist 0.2.0リリース

OMG RTC spec. draft

OMG RTC spec. approved

OpenRTM-aist 0.4.0リリース

OMG RTC 標準仕様リリース

OpenRTM-aist 1.0リリース

FY

2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011

RTミドルウエアプロジェクト

科研費 若手(B)

科振費 分散コンポーネントシミュレータ

NEDO 次世代ロボット共通基盤開発

工業標準部　標準基盤研究

科振費 環境と作業構造のユニバーサルデザイン

産総研イニシアチブ UCROA

NEDO 戦略的先端ロボット要素技術開発

経済産業省・NEDO次世代ロボット知能化技術開発プロジェクト

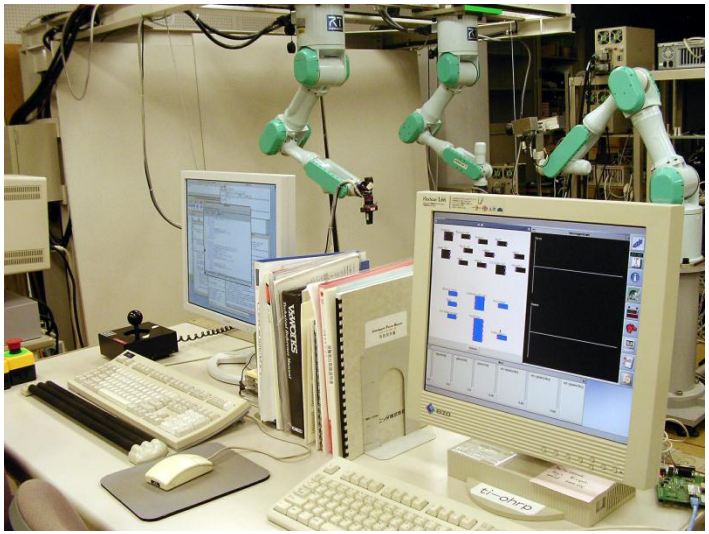
ロボット大賞 2007

さまざまなプロジェクトで
標準ソフトウエアプラットフォーム
として採用されている

RTミドルウエアPJ

FY2002.12-FY2004

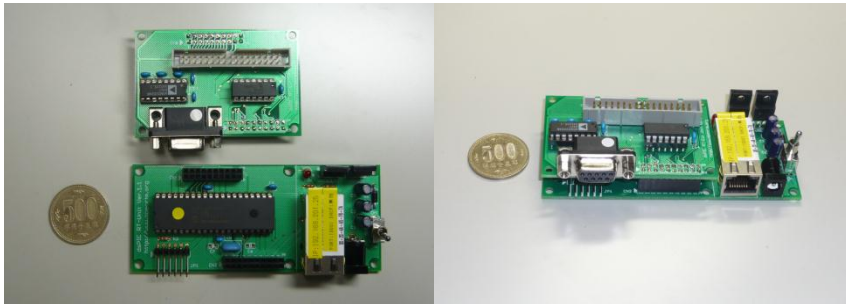
- 名称：NEDO 21世紀ロボットチャレンジプログラム
  - 「ロボット機能発現のために必要な要素技術開発」
- 目的：
  - RT要素の部品化（モジュール化）の研究開発
  - 分散オブジェクト指向開発
  - RT要素の分類・モジュール化に必要な機能・インタフェース仕様の明確化
- 予算規模：
  - 65百万円
  - 全体267.3百万円



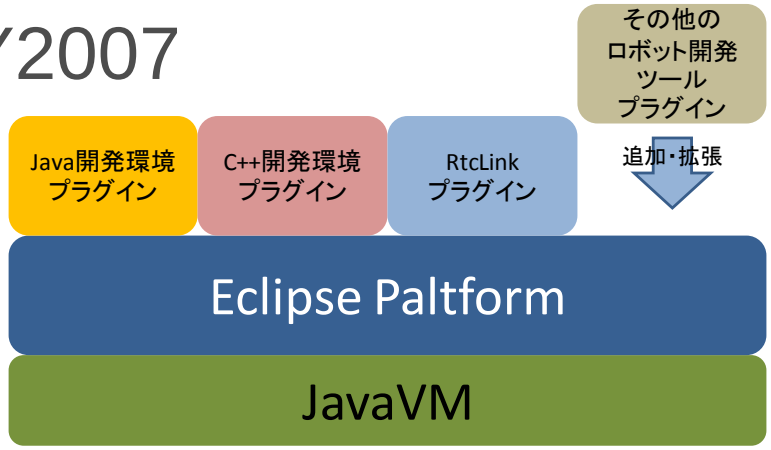
NEDO基盤PJ

FY2003-FY2007

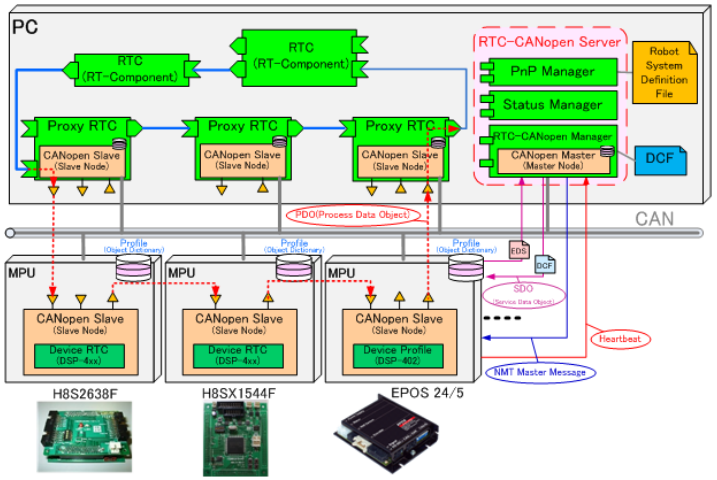
- 名称:「運動制御デバイスおよびモジュールの開発」
- 目的:
  - 運動制御デバイスの開発
  - デバイスに搭載するRTCの開発
  - その他モーションコントロールに資するRTM/RTCの開発
- 予算規模:
  - 15百万円/年
  - 371百万円、全体1,259百万円



dsPIC版RTC-Liteの開発



ツールのEclipseプラグイン化

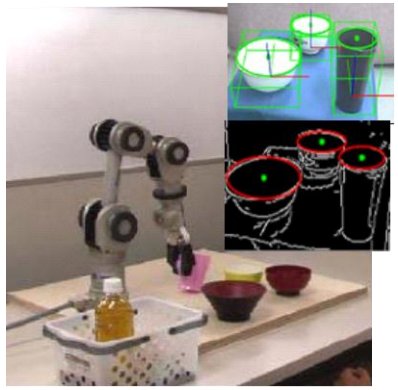


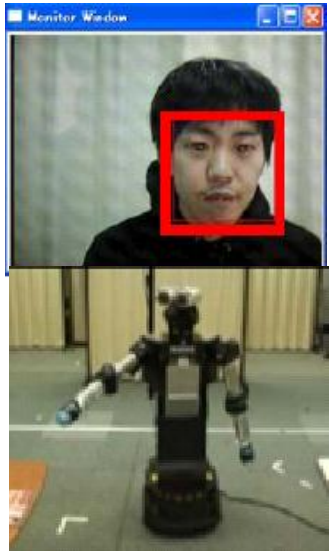
RTC-CANの開発

知能化PJ

FY2007-FY2012

- 名称:「次世代ロボット知能化技術開発プロジェクト」
- 目的
  - ソフトウエアプラットフォームの開発
  - 作業知能、移動知能、コミュニケーション知能に関するモジュールの開発
- 予算:
  - 400百万円
  - 全体7,000百万円
- 研究グループ
  - 15グループ





RTミドルウエアの広がり

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 592 人+22人
利用組織（Google Map）	46 組織

プロジェクト登録数

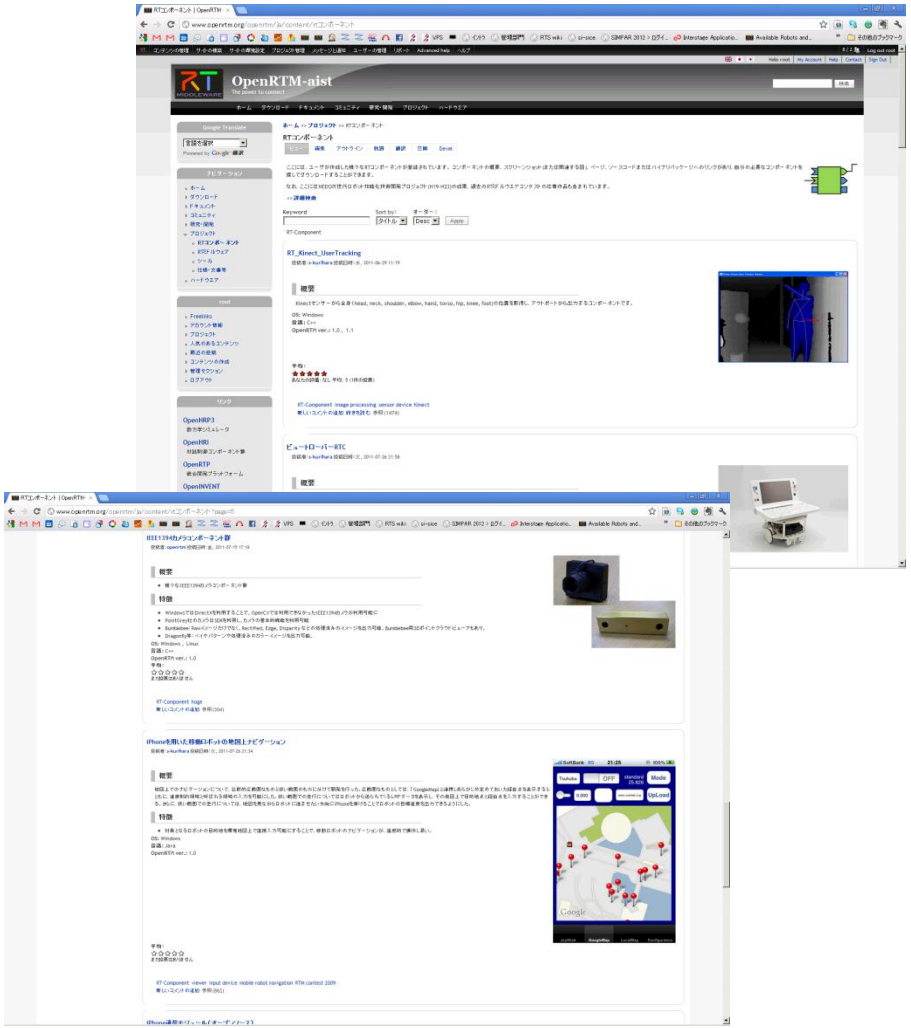
タイプ	登録数
RTコンポーネント群	287
RTミドルウエア	14
ツール	19
仕様・文書	4
ハードウエア	28

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

プロジェクトページ

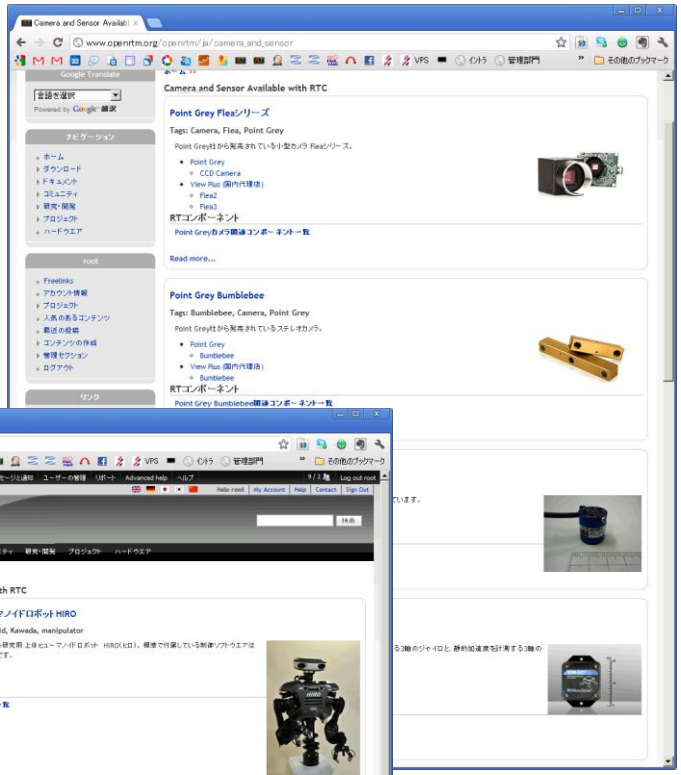
- ユーザが自分の作品を登録
- 他のユーザの作ったRTCを探すことができる

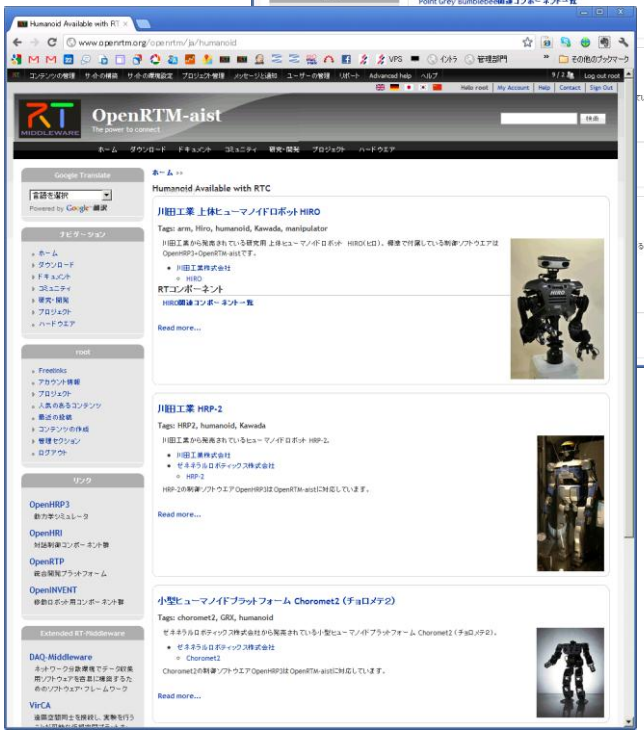


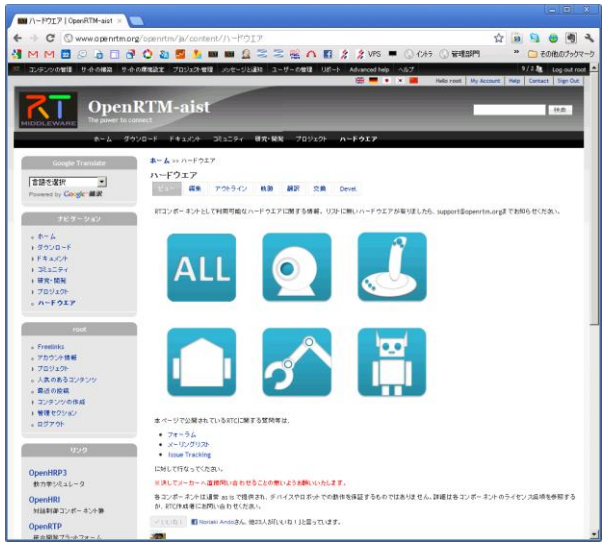
タイプ	登録数
RTコンポーネント群	287
RTミドルウエア	14
ツール	19
仕様・文書	4
ハードウエア	28

ハードウエア集

- OpenRTMで利用可能なハードウエアのリスト
- ハードウエアを利用するために利用できるコンポーネントのリスト







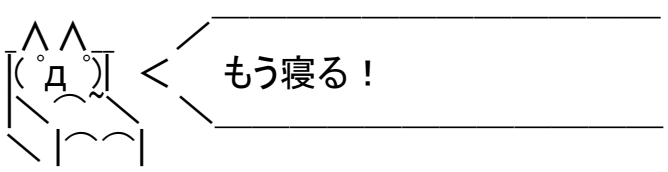
NEDO RTコンポーネント集

- www.openrtm.org に NEDO知能化PJ成果物の特別ページを設置
  - ツール
  - 作業知能モジュール
  - 移動知能モジュール
  - 対話知能モジュール
  - 商用ライセンスモジュール

  の5カテゴリに分けて掲載

サマーキャンプ

- 毎年夏に1週間開催
- 今年：7月29日～8月2日
- 募集人数：10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
  - エントリー〆切：9月ごろ
  - ソフトウエア登録：10月ごろ
  - 講演原稿〆切：10月ごろ
  - 発表・授賞式：12月ごろ
- 2012年度実績
  - 応募数：17件
  - 計測自動制御学会RTミドルウエア賞（副賞10万円）
  - 奨励賞（賞品協賛）：3件
  - 奨励賞（団体協賛）：10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

ROS

ROSの良い点

- UNIXユーザに受けるツール（特にCUI）が豊富
- 独自のパッケージ管理システムを持つ
- 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

ROSの問題点

- Ubuntu以外の対応が今一つ
  - Windows対応はいろいろな人が試したがいまだに公式には含まれない
- コアライブラリの仕様が固まっていない
  - 他の言語で実装する際の妨げ
  - サードパーティー実装が出にくい
  - 品質を保証しづらい
- コンポーネントモデルがない

OpenRTM

OpenRTMの良い点

- 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
- 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

OpenRTMの問題点

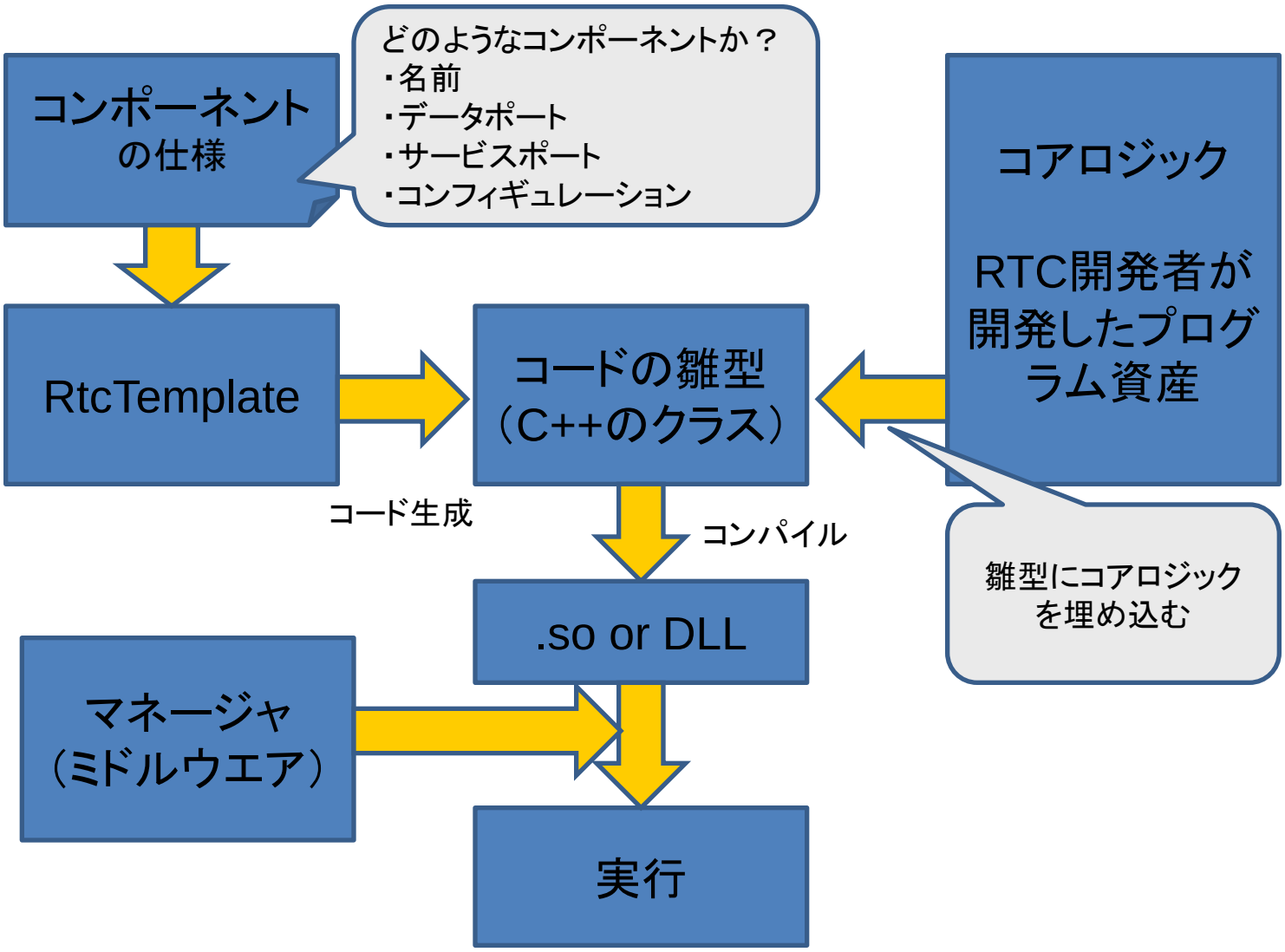
- コンポーネントの数が少ない
- パッケージ管理システムがない
- CUIツールが少ない
  - UNIXユーザ受けしない
- ユーザ数が少ない
- 英語のドキュメントが少ない
- 知名度が低い
- 開発速度が遅い
  - 現在は開発者一人

提言

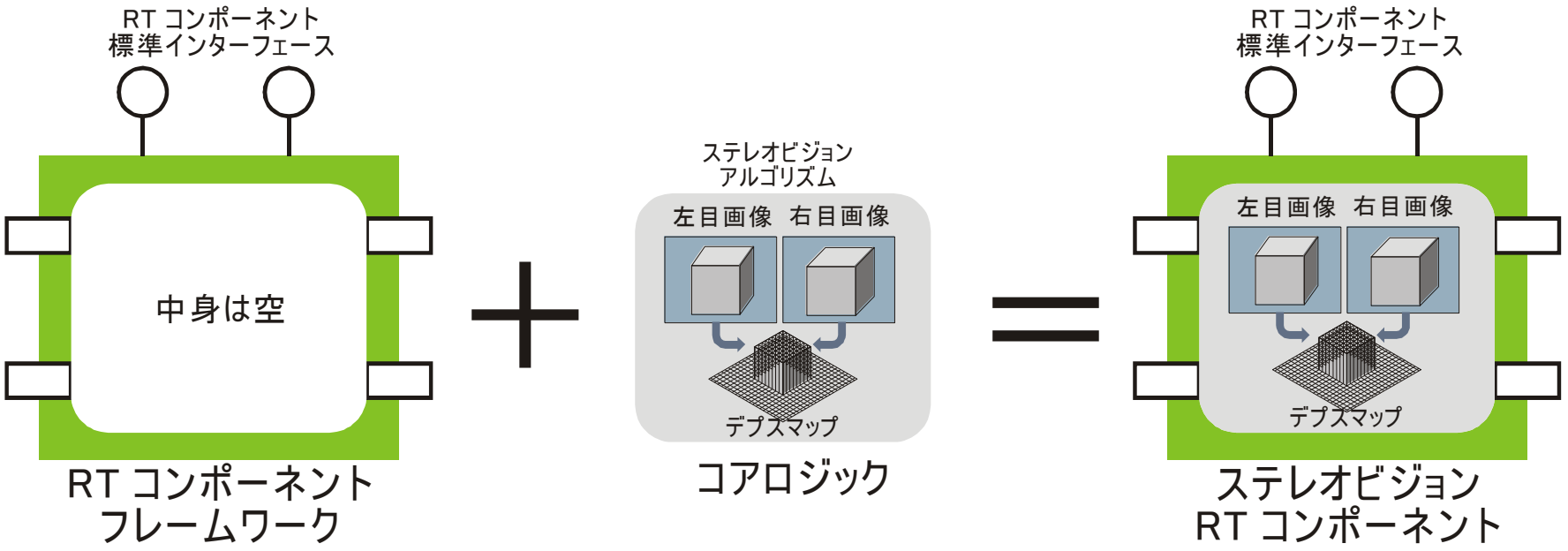
- 自前主義はやめよう！！
  - 書きたてのコードより、いろいろな人に何万回も実行されたコードのほうが動くコードである！！
  - 自分にとって本質的でない部分は任せて、本当にやりたい部分・やるべき部分のコードを書こう！！
  - 誰かがリリースしたプログラムは一度は動いたことがあるプログラムである！！
  - 人のコードを読むのが面倒だからと捨ててしまうのはもったいない！！
- オープンソースにコミットしよう！！
  - 臆せずMLやフォーラムで質問しよう！！
  - どんなに初歩的な質問でも他の人にとっては価値ある情報である。
  - 要望を積極的にあげよう！！
  - できればデバッグ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

RTコンポーネントの開発

OpenRTMを使った開発の流れ

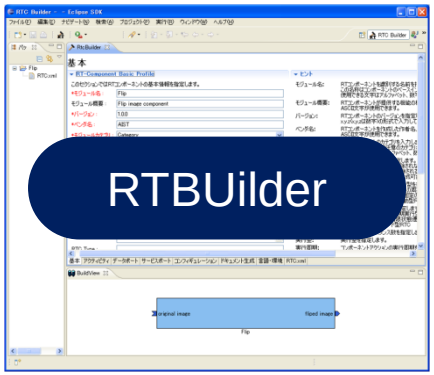


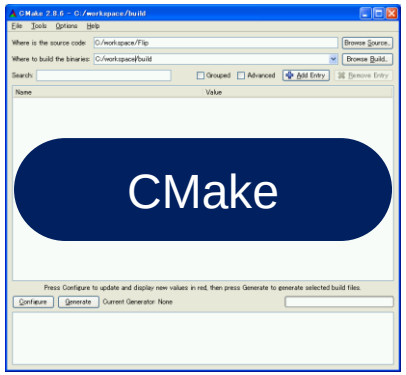
フレームワークとコアロジック

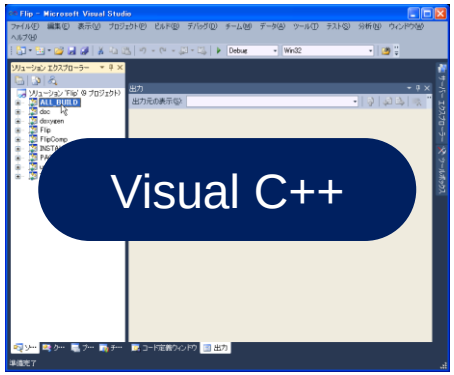


RTCフレームワーク＋コアロジック＝RTコンポーネント

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

テンプレートコード
の生成

VCのプロジェクト
ファイルの生成

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

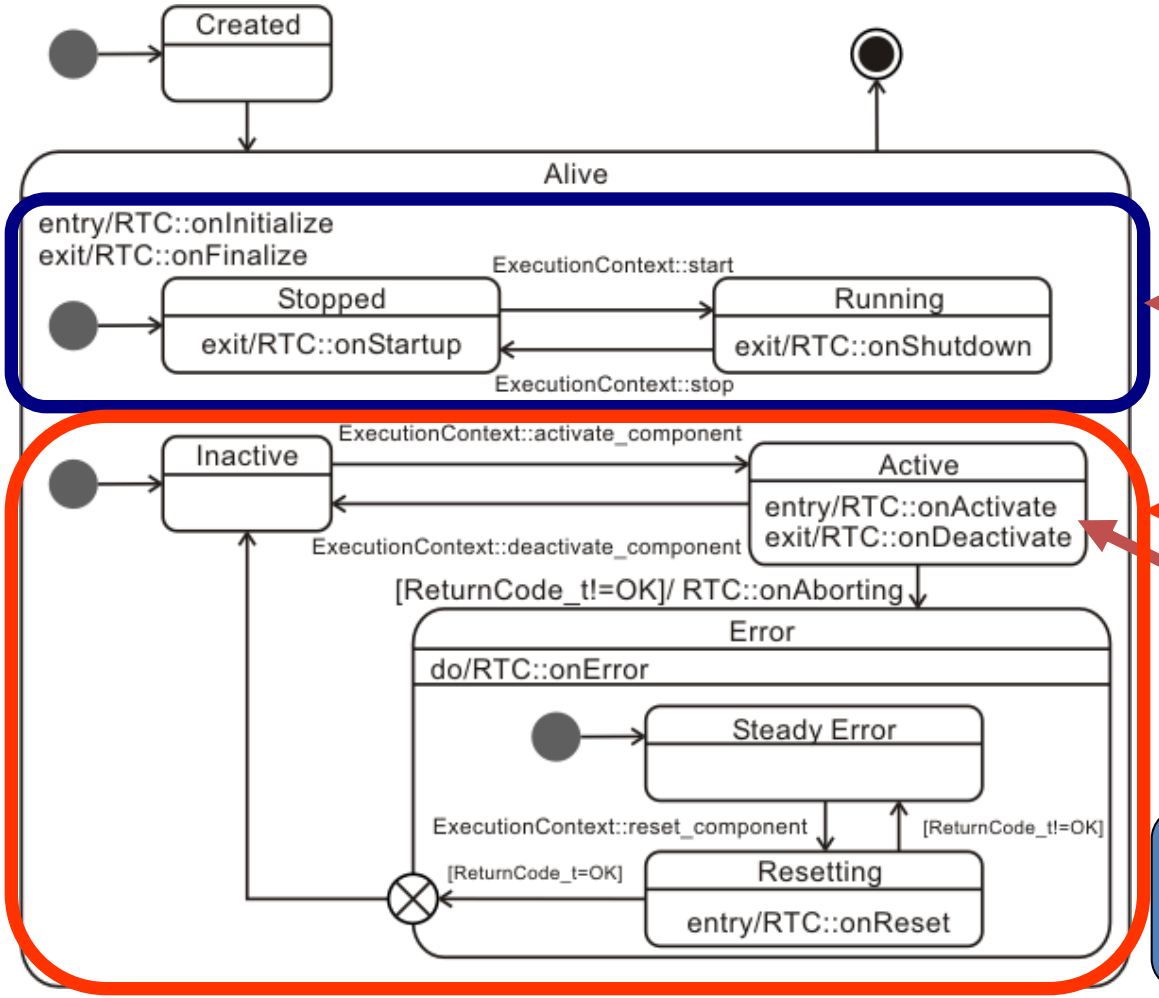
- 生成されたクラスのメンバー関数に必要な処理を記述
- 主要な関数
  - onExecute (周期実行)
- 処理
  - InPortから読む
  - OutPortへ書く
  - サービスを呼ぶ
  - コンフィギュレーションを読む

```
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



ユーザがあまり
意識しなくてよい部分

コンポーネント開発時に
必要な部分

ActiveDo/RTC::onExecuteはここに入る
（DataFlow型のコンポーネントのとき）

コールバック関数

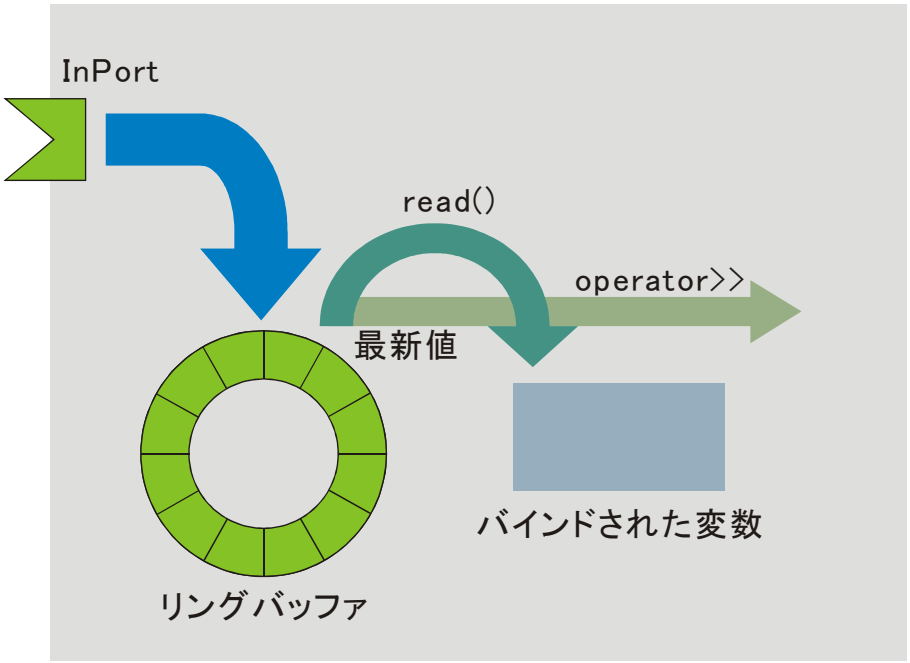
RTCの作成＝コールバック関数に処理を埋め込む

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

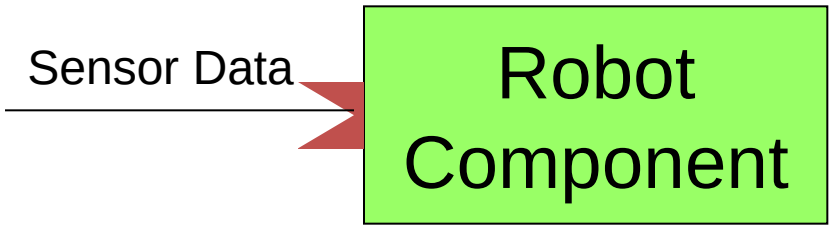
- InPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- InPortのメソッド
  - read(): InPort バッファからバインドされた変数へ最新値を読み込む
  - >> : ある変数へ最新値を読み込む



基本的にOutPortと対になる

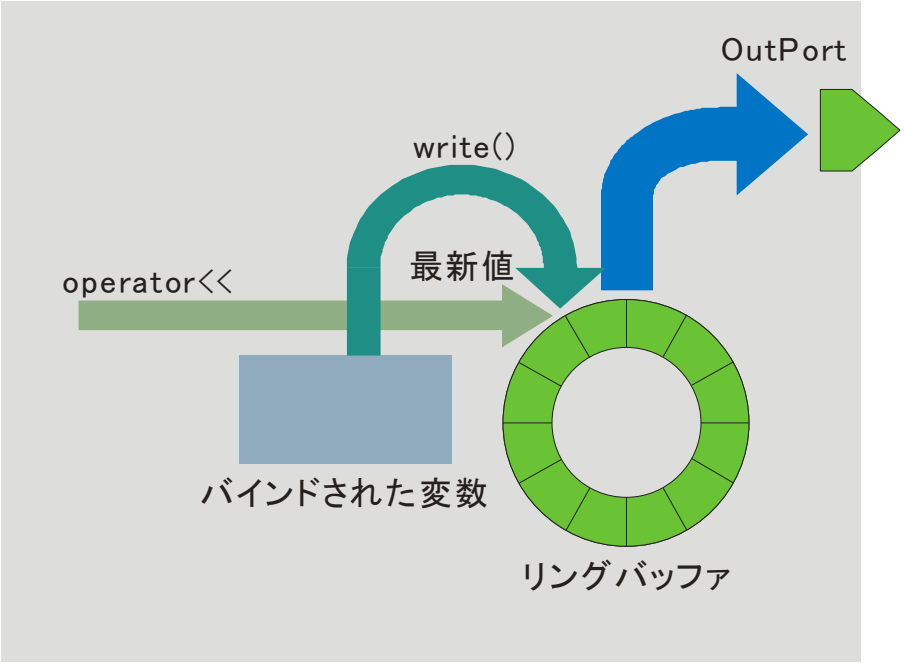
データポートの型を
同じにする必要あり

例



OutPort

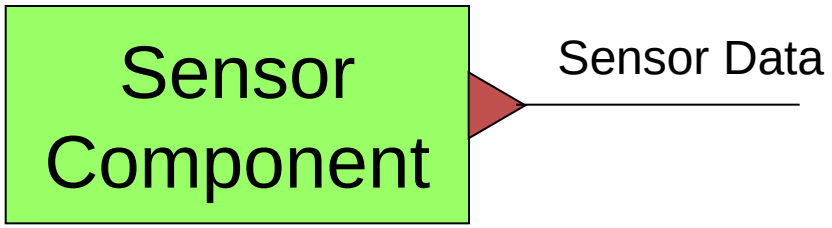
- OutPortのテンプレート第2引数：バッファ
  - ユーザ定義のバッファが利用可能
- OutPortのメソッド
  - write(): OutPort バッファへバインドされた変数の最新値として書き込む
  - >> : ある変数の内容を最新値としてリングバッファに書き込む



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

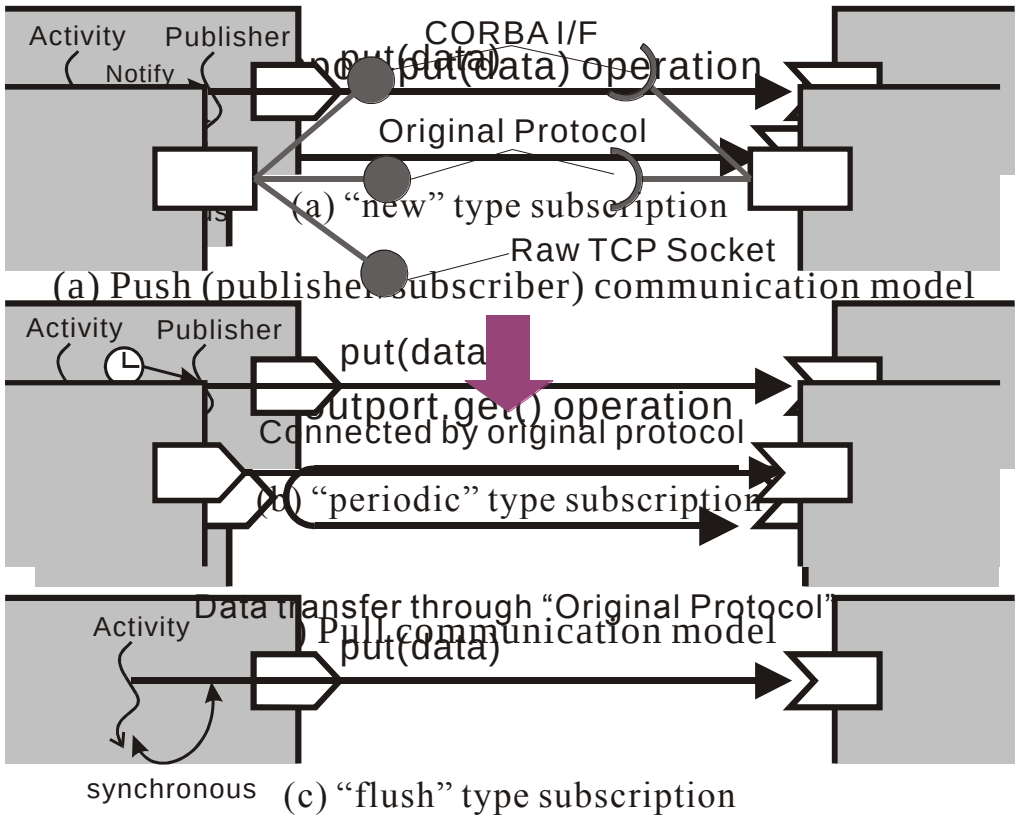
```
struct TimedShortSeq
 {
      Time tm;
      sequence<short> data;
 };
```

- 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

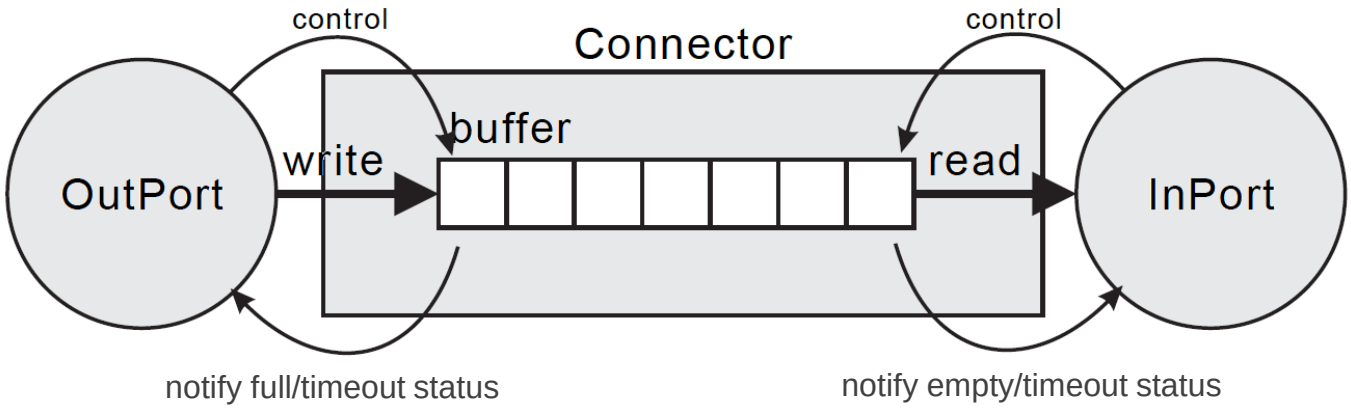
データポート

- 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



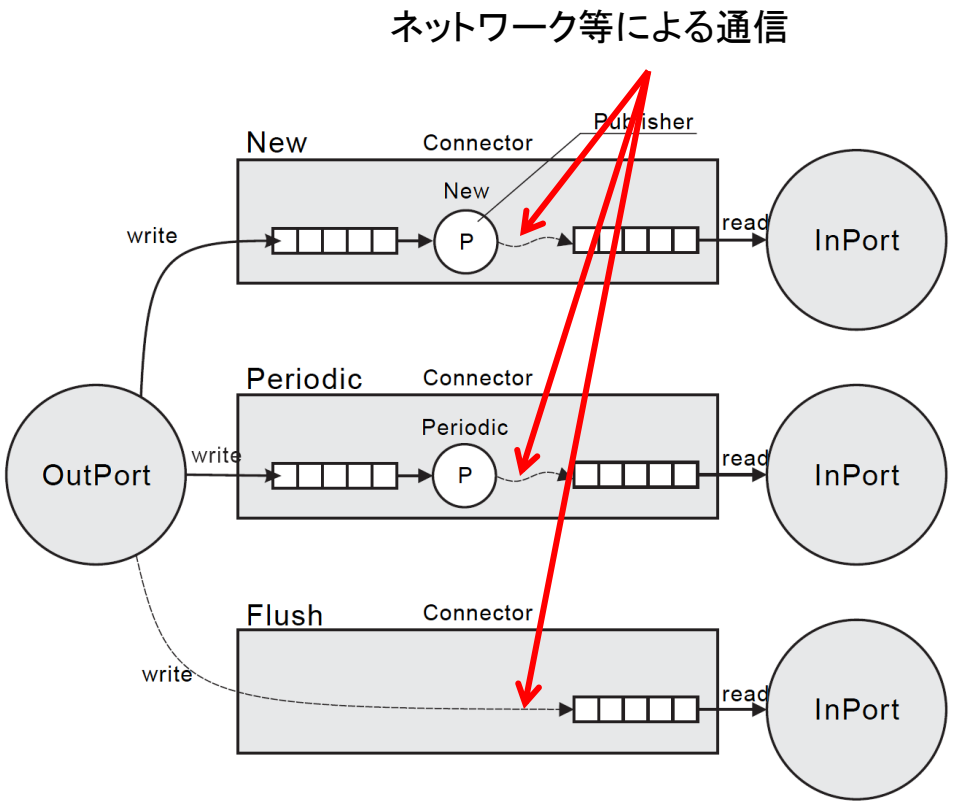
データポートモデル

- 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

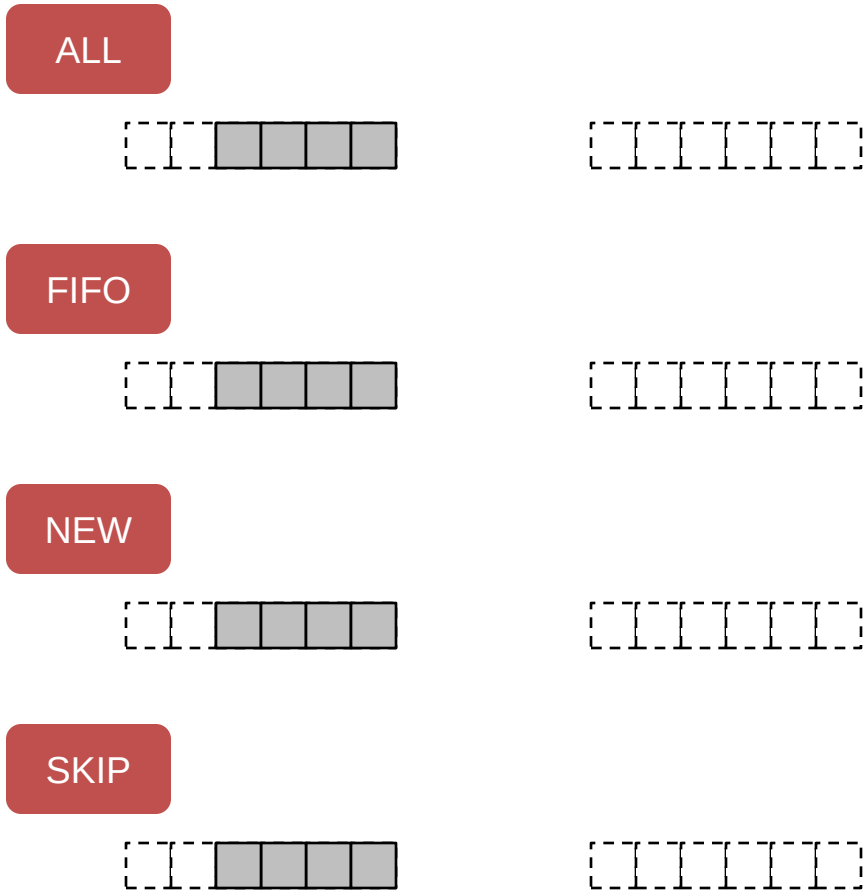
- BUFFER OK 正常終了
- BUFFER ERROR エラー終了
- BUFFER FULL バッファフル状態
- BUFFER EMPTY バッファ空状態
- NOT SUPPORTED サポート外の要求
- TIMEOUT タイムアウト
- PRECONDITION NOT MET 事前状態を満たさない

Pushポリシー

- バッファ残留データ
  - 送り方のポリシ

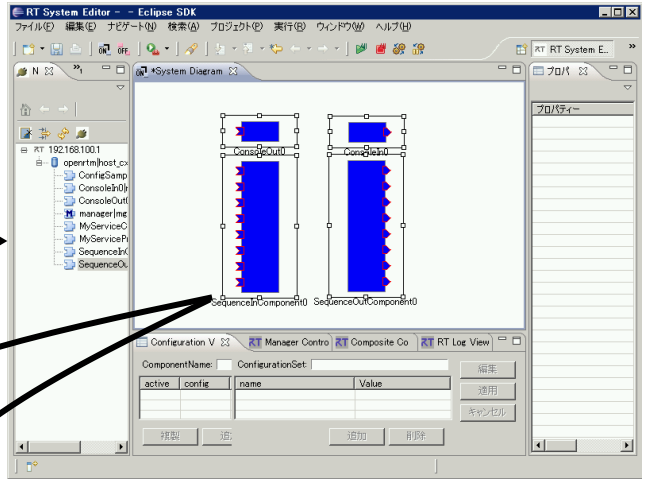
ポリシ	送り方
ALL	全部送信
FIFO	先入れ後だしで1個ずつ送信
NEW	最新値のみ送信
SKIP	n個おきに間引いて送信

  - データ生成・消費速度を考慮して設定する必要がある。



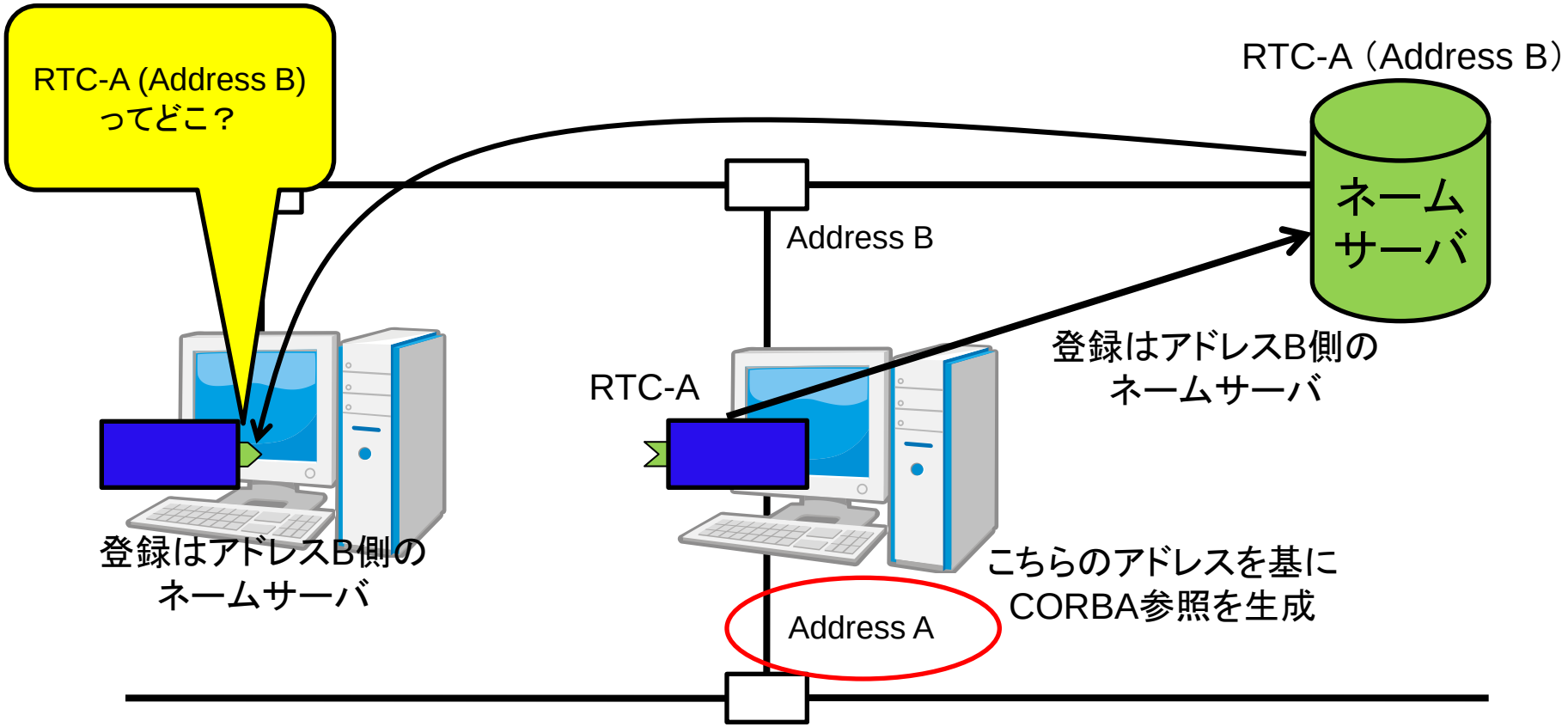
予稿にはNEWの説明にLIFOと記述していましたが正確にはLIFOではなく最新値のみの送信です。LIFO形式のポリシを導入するかどうかは検討中です。ご意見ください。

動作シーケンス


RT System Editor - Eclipse SDK
*System Diagram
Configuration V
Manager Contro
Composite Co
RT Log View
ComponentName:
ConfigurationSet:
active
config
name
Value

ネームサーバ
③ 参照を取得
④ ポートを接続
① 参照を登録
② 参照を登録

ネットワークインターフェースが2つある場合の注意



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

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

まとめ

- RTミドルウエアの概要
  - 背景、目的、利点
  - 標準化、適用例
  - 過去のプロジェクト、Webページ
- RTコンポーネントの開発
  - 開発の流れ
  - 動作シーケンス
  - コールバック、データポート、rtc.conf