

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

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

安藤慶昭



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ミドルウェア

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ェア
- RT-Component (RTC)
 - RT-Middlewareにおけるソフトウェアの基本単位

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

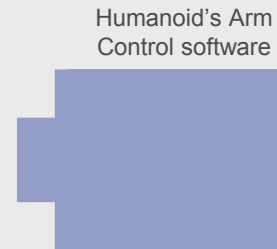
従来のシステムでは...



Joystick



Joystick
software



Humanoid's Arm
Control software



Humanoid's Arm



Robot Arm

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

Robot Arm
Control software

RTミドルウェアでは...

RTミドルウェアは別々に作られたソフトウェ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

Arm A
Control software



compatible
arm interfaces



Arm B
Control software



Humanoid's Arm



Robot Arm

ソフトウェアの再利用性の向上
RTシステム構築が容易になる

ミドルウェア、コンポーネント、etc...

- ミドルウェア
 - 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ェア
 - 例: RDBMS、ORB等。定義は結構曖昧
- 分散オブジェクト(ミドルウェア)
 - 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ェアの断片(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)
 - ソフトウェア・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ェア開発手法

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

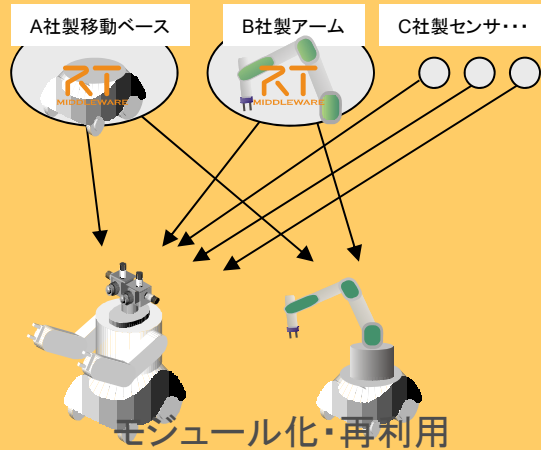
RTコンポーネント化のメリット

モジュール化のメリットに加えて

- ソフトウェアパターンを提供
 - ロボットに特有のソフトウェ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
- 分散ミドルウェア
 - 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ミドルウェアの目的 モジュール化による問題解決

コストの問題



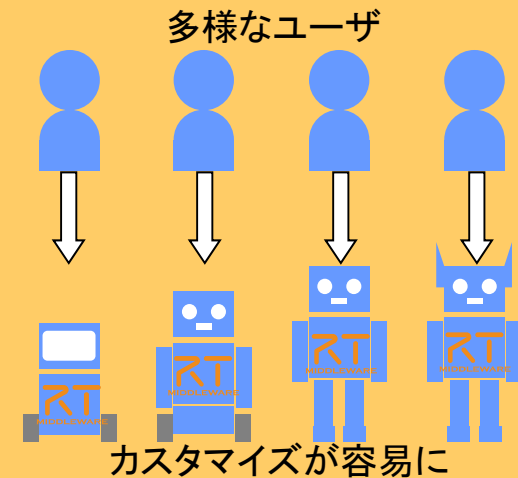
ロボットの低コスト化

技術の問題



最新技術を利用可能

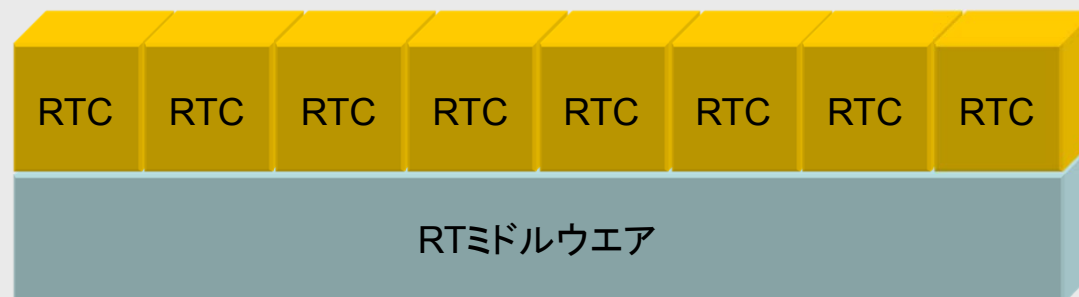
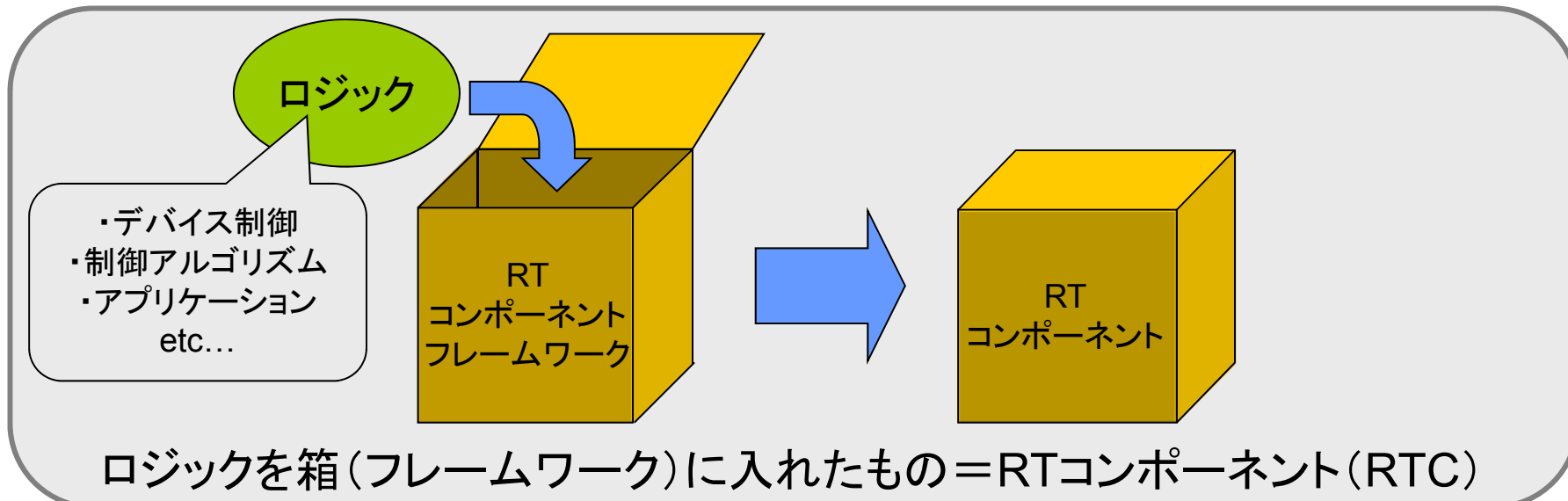
ニーズの問題



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ミドルウェアとRTコンポーネント

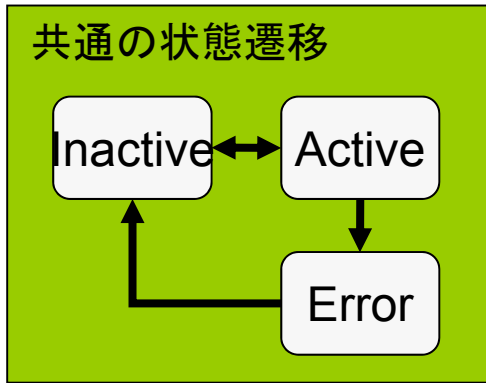


RTCの実行環境(OSのようなもの) = RTミドルウェア(RTM)
※RTCはネットワーク上に分散可能

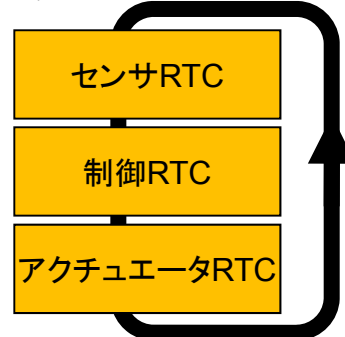
RTコンポーネントの主な機能

アクティビティ・実行コンテキスト

共通の状態遷移



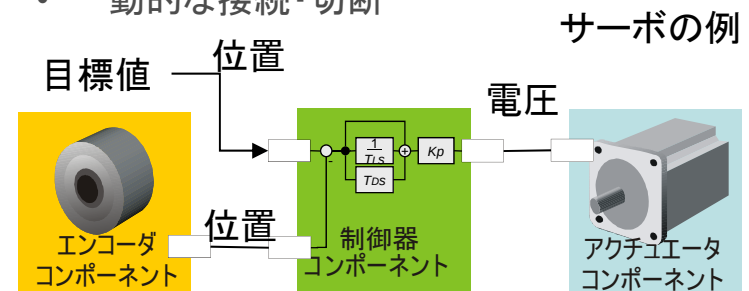
複合実行



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

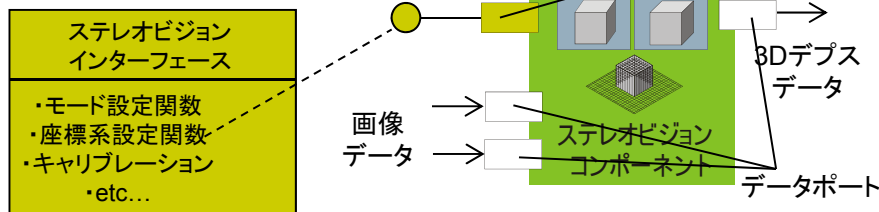
- データ指向ポート
- 連続的なデータの送受信
- 動的な接続・切断



データ指向通信機能

サービスポート

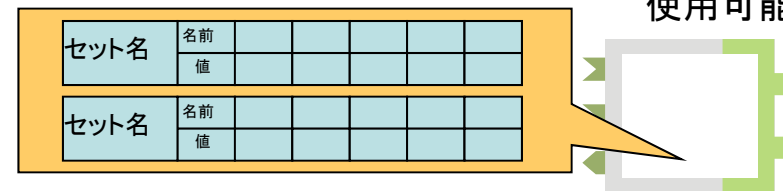
- 定義可能なインターフェースを持つ
- 内部の詳細な機能にアクセス
 - パラメータ取得・設定
 - モード切替
 - etc...



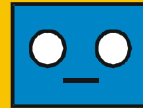
サービス指向相互作用機能

コンフィギュレーション

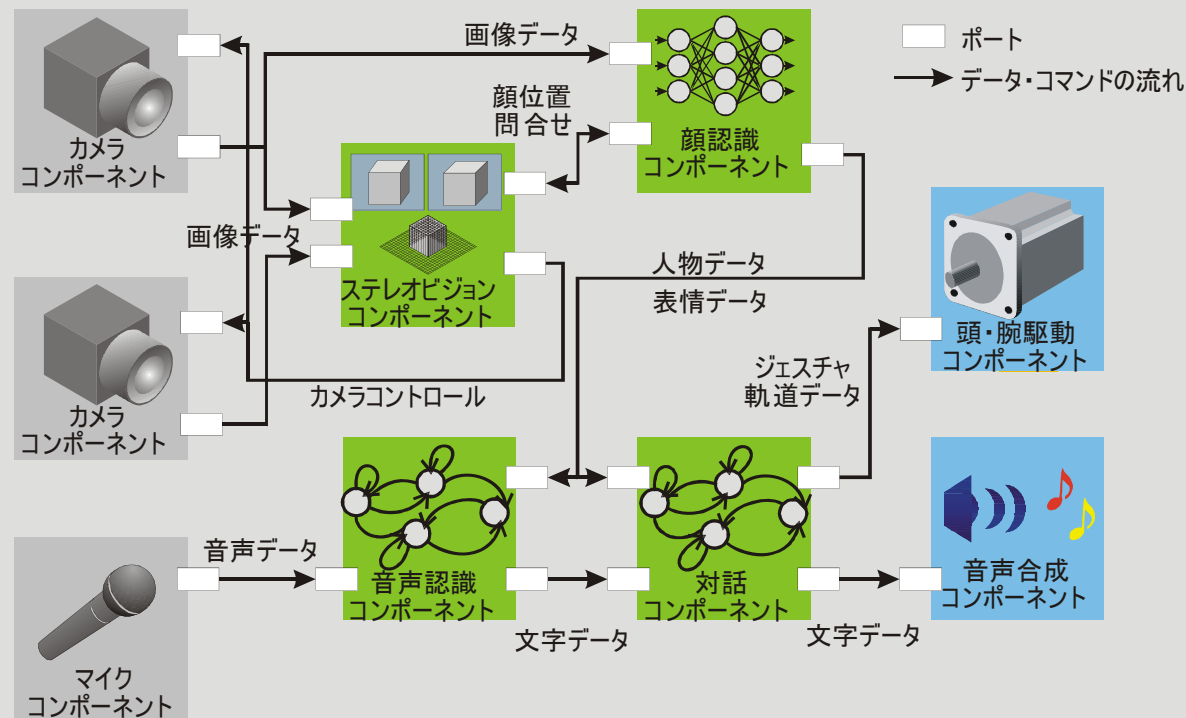
- パラメータを保持する仕組み
 - いくつかのセットを保持可能
 - 実行時に動的に変更可能
- 複数のセットを動作時に切り替えて使用可能



RTCの分割と連携

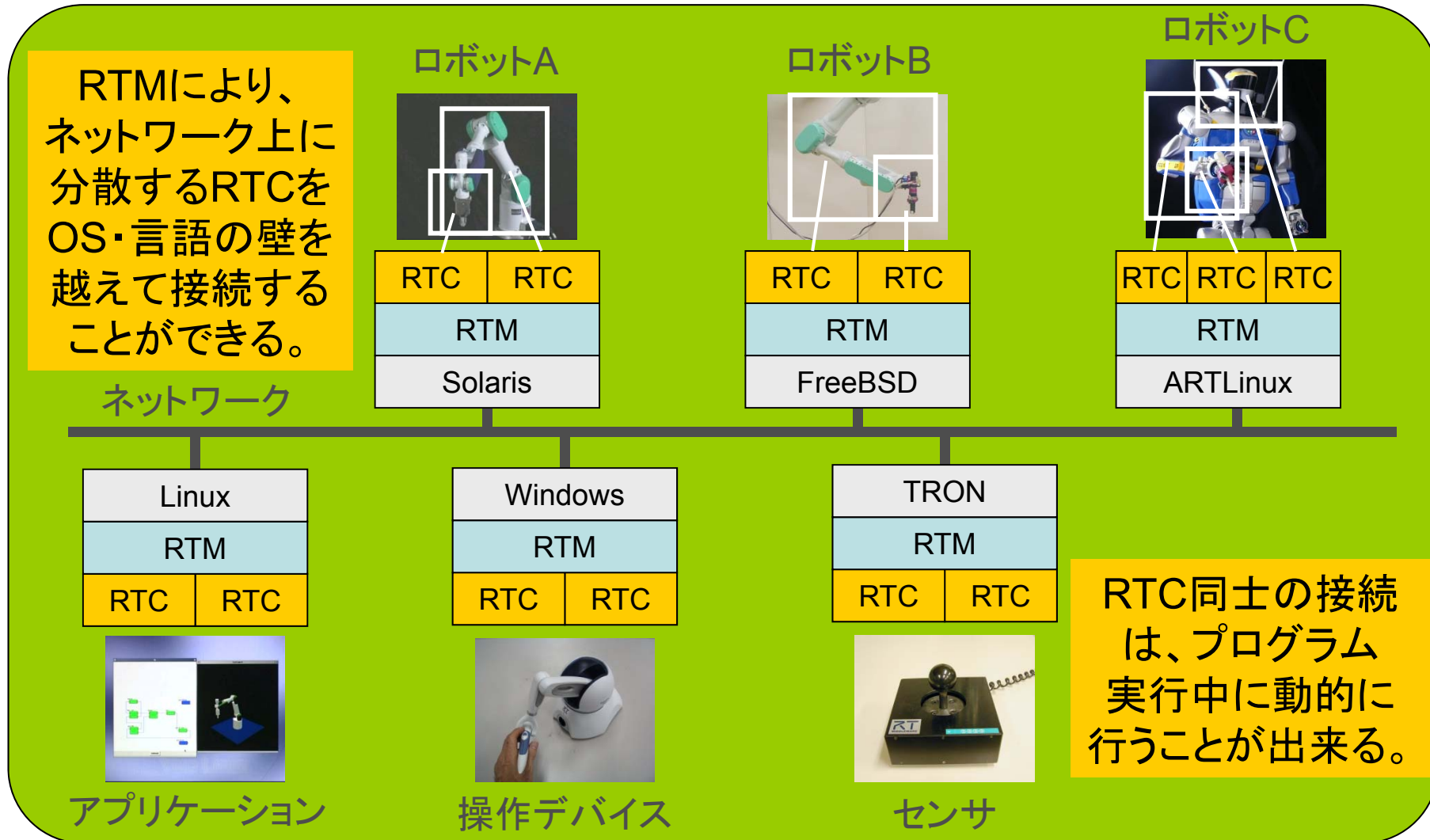


ロボット体内のコンポーネントによる構成例



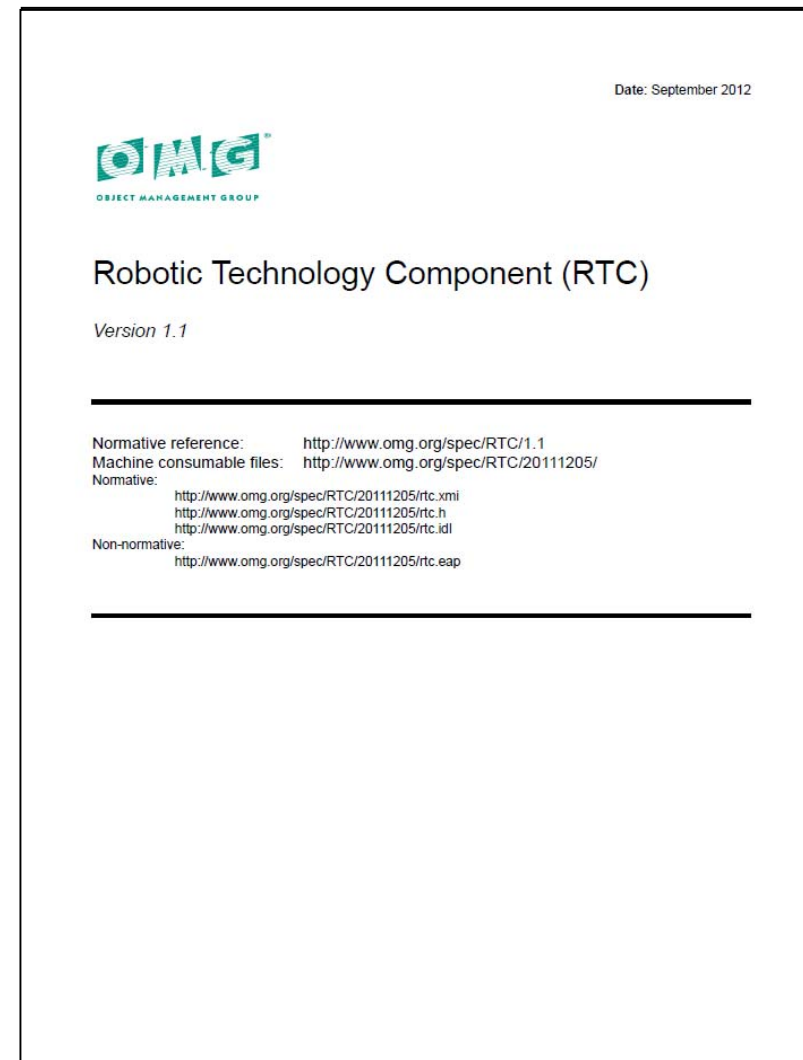
(モジュール)情報の隠蔽と公開のルールが重要

RTミドルウェアによる分散システム



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 ver.1.0公式リリース
- 2010年1月
OpenRTM-aist-1.0リリース
- 2012年9月
ver. 1.1改定
- 2014年12月
FSM4RTC(FSM型RTCとデータポート標準) Beta1



OMG RTC ファミリ

名称	ベンダ	特徴	互換性
OpenRTM-aist	AIST	C++, Python, Java	---
OpenRTM.NET	SEC	.NET(C#,VB,C++/CLI, F#, etc..)	◎
RTM on Android	SEC	Android版RTミドルウェア	◎
H-RTM	本田R&D	OpenRTM-aist互換、FSM型コンポーネントをサポート	◎
RTC-Lite	AIST	PIC, dsPIC上の実装	○(ブリッジ)
miniRTC, microRTC	SEC	CAN・ZigBee等を利用した組込用RTC実装	○(ブリッジ)
RTMSafety	SEC/AIST	機能安全認証 (IEC61508) capableなRTM実装, 商用	○(ブリッジ)
RTC CANOpen	SIT, CiA	CANOpen-RTCマッピングを定めたCiA 標準	○(ブリッジ)
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

応用例（研究用プラットフォーム）



HRP-2(川田工業)



HRP-4(川田工業)



HIRO(川田工業)



ビュートローバーRTC/RTC-BT(VSTONE)

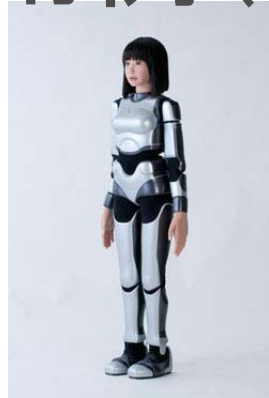


OROCHI(アールティ)

応用例(実応用その他)



S-ONE: SCHAFT



HRP-4C: AIST

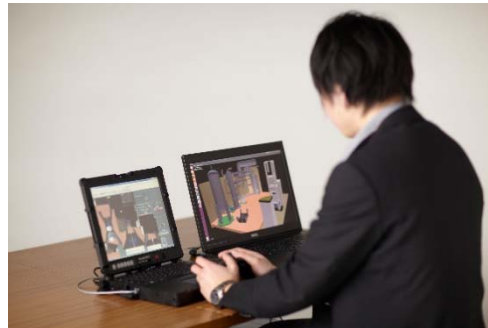


TAIZOU: GRX



DAQ-Middleware: KEK/J-PARC

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



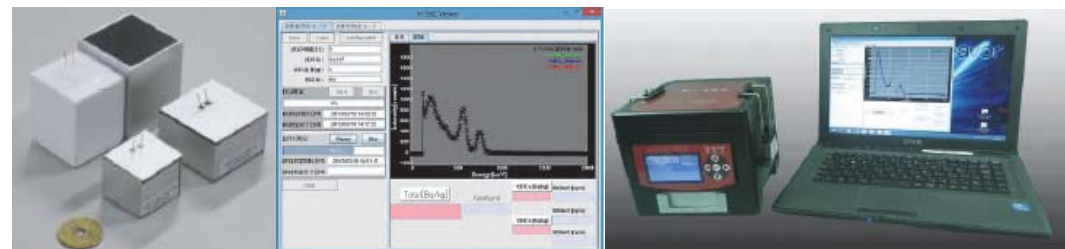
災害対応ロボット操縦シミュレータ:
NEDO/千葉工大



RAPUDA: Life Robotics



新日本電工他: Mobile SEM

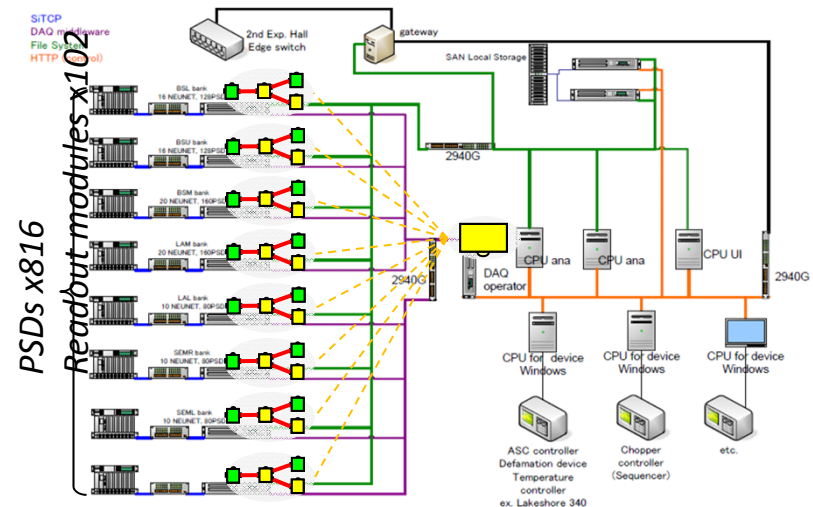
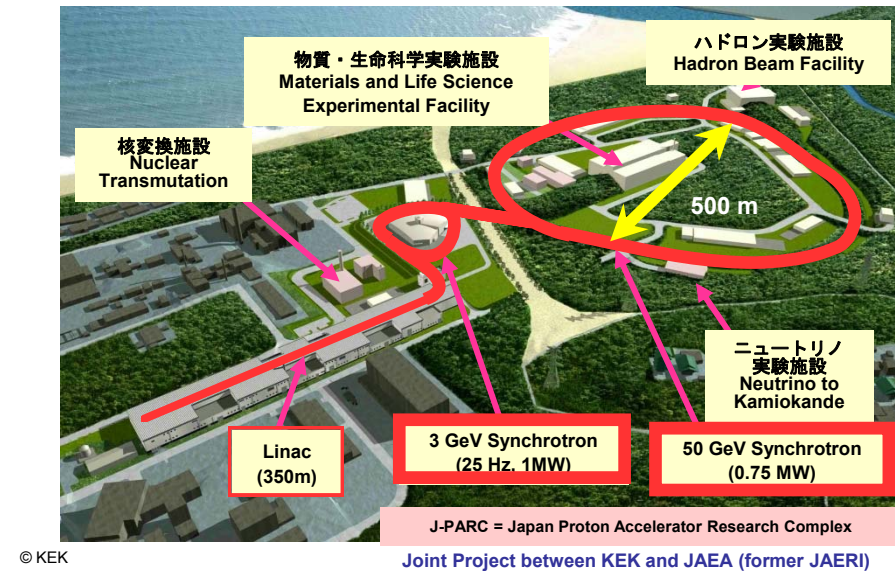
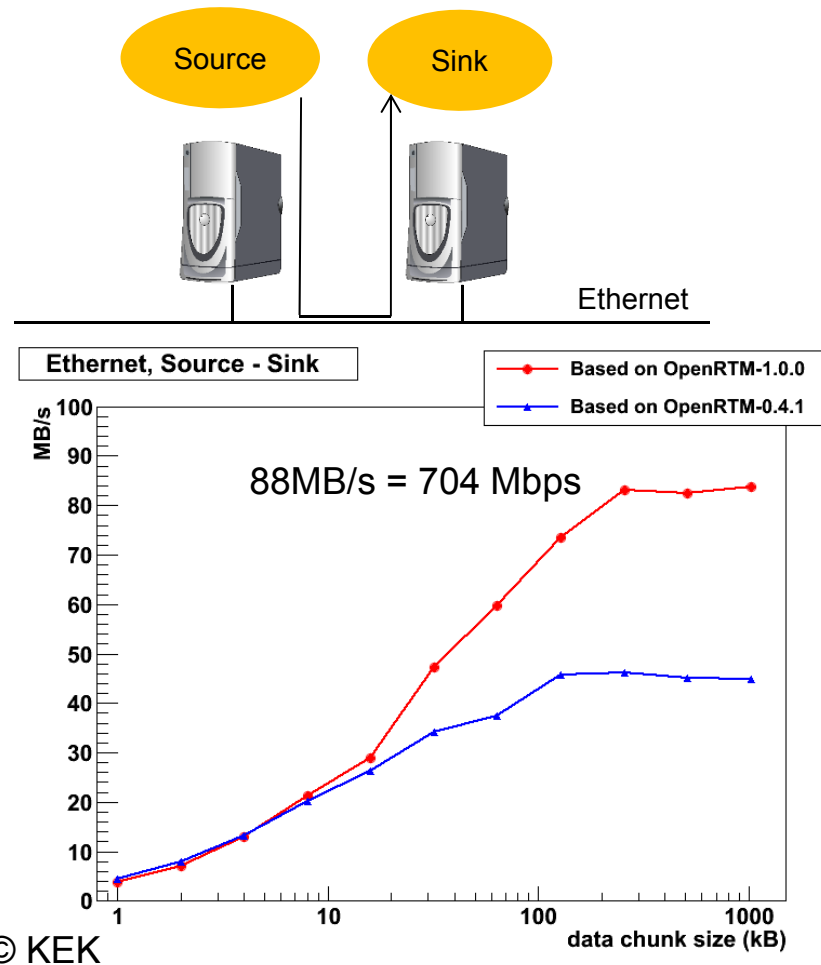


新日本電工他: 小型ベクレルカウンタ

Japan Proton Accelerator Research Complex

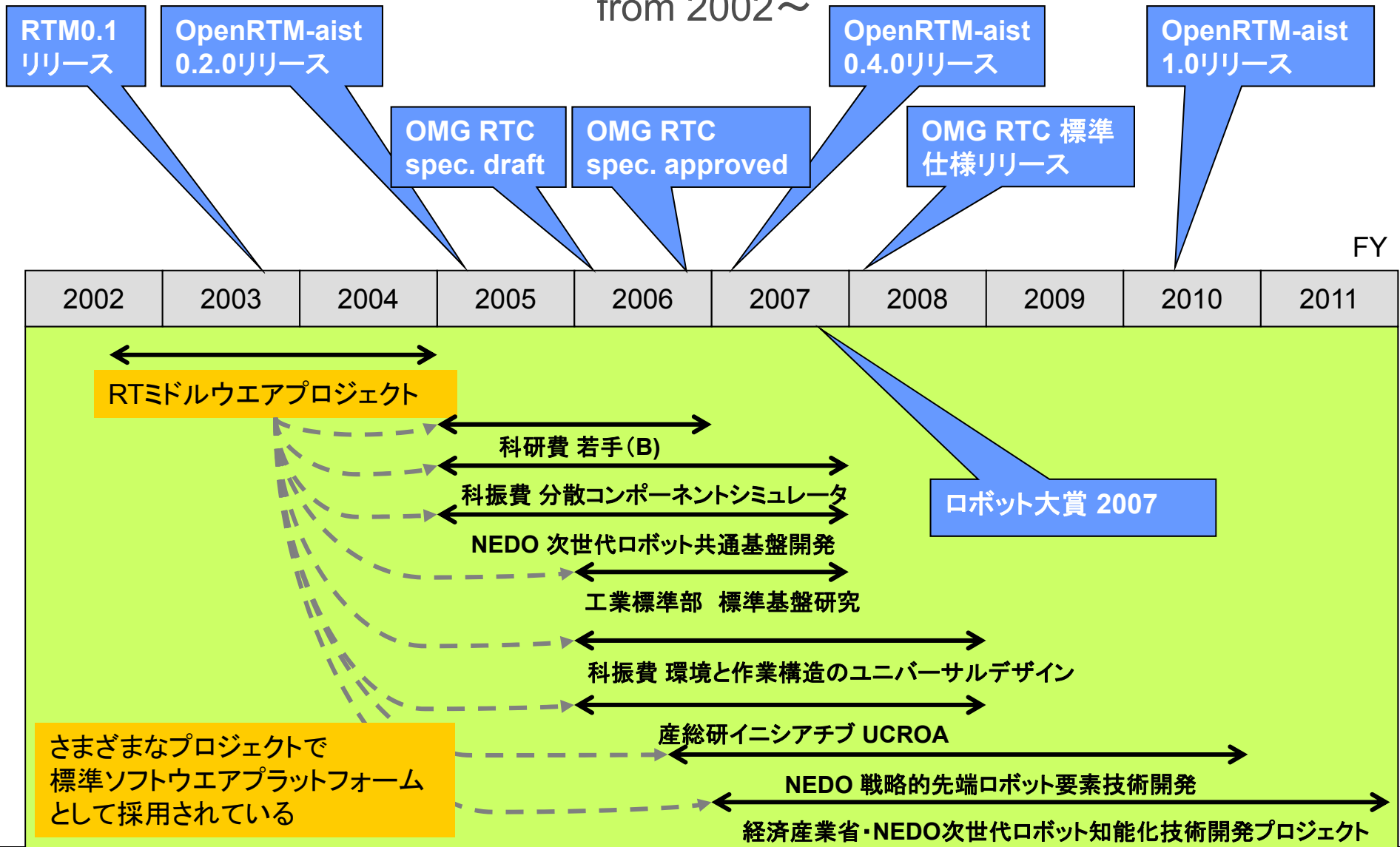
(J-PARC, 大強度陽子加速器施設)

Model : Dell PowerEdge SC1430
 CPU : Intel Xeon 5120 @ 1.86GHz 2 Cores × 2
 Memory: 2GB
 NIC: Intel Pro 1000 PCI/e (1GbE)
 OS: Scientific Linux 5.4 (i386)



RT-Middleware関連プロジェクト

from 2002~



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 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ミドルウェア	14
ツール	19
仕様・文書	4
ハードウェア	28

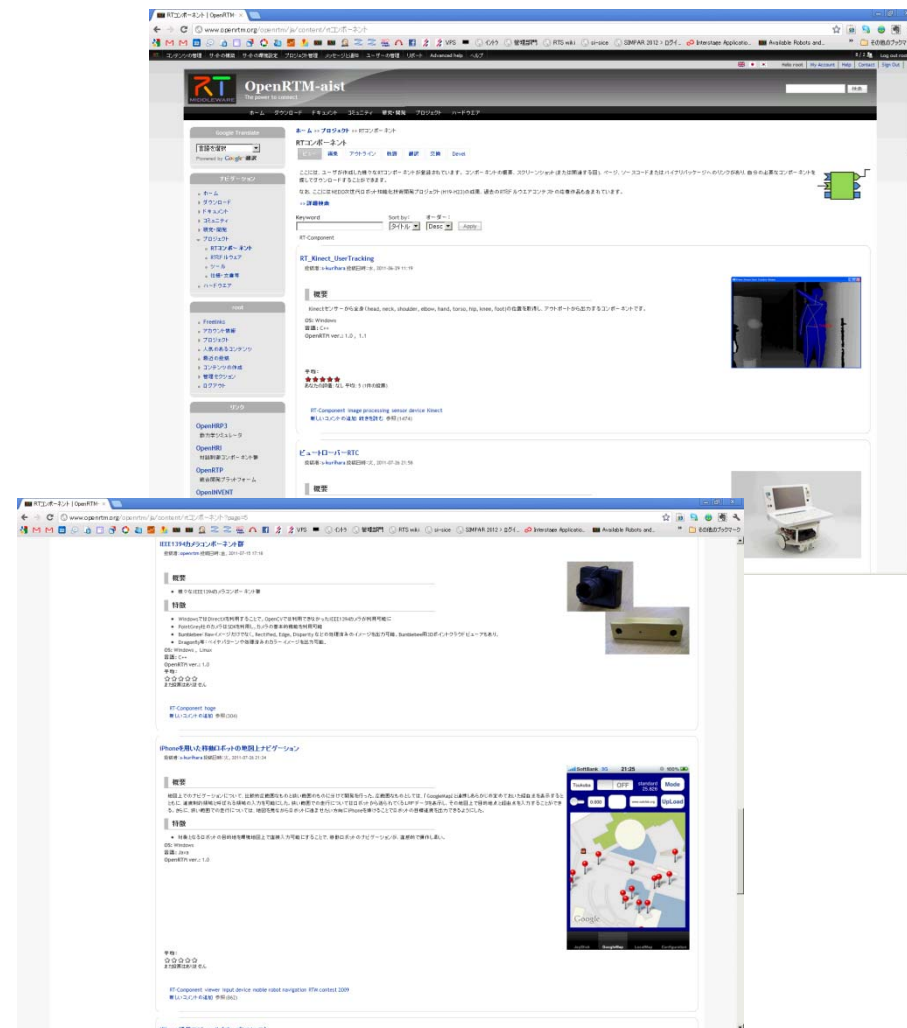
OMG RTC規格実装 (11種類)

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

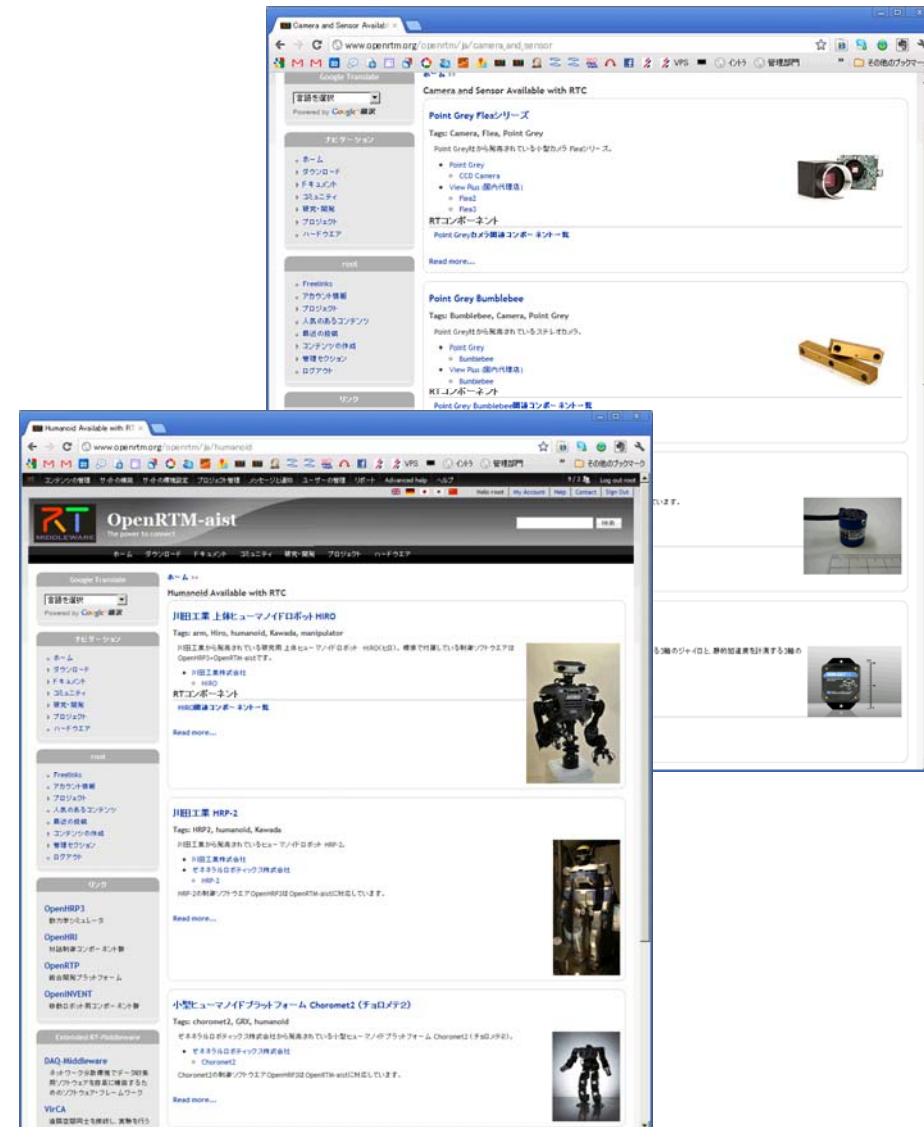
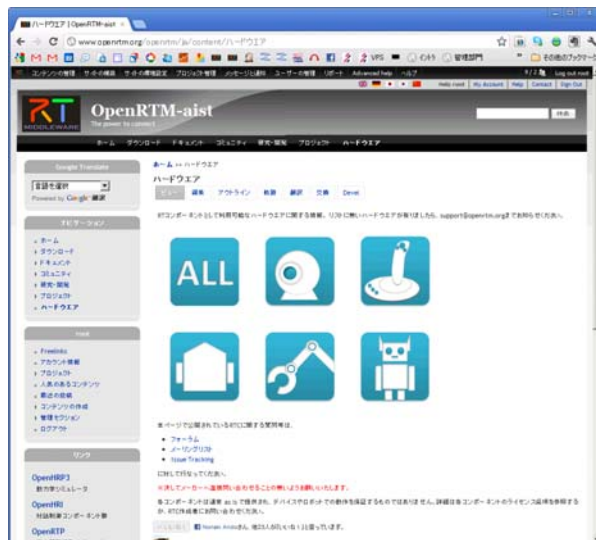
- ユーザが自分の作品を登録
- 他のユーザの作ったRTCを探することができる

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



ハードウェア集

- OpenRTMで利用可能なハードウェアのリスト
- ハードウェ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:8月3日～8月7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ミドルウェアコンテスト

- 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(SI2015締切)
 - ソフトウェア登録:10月ごろ
 - 講演原稿〆切:9月25日
 - オンライン審査:11月下旬～
 - 発表・授賞式:12月ごろ
- 2014年度実績
 - 応募数:20件
 - 計測自動制御学会学会RTミドルウェア賞（副賞10万円）
 - 奨励賞（賞品協賛）:2件
 - 奨励賞（団体協賛）:11件
 - 奨励賞（個人協賛）:7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

ロボットOS: 現状と課題

ロボットOSの覇権争い



日本 産総研・NEDO発
2002年から開発

OMG国際標準を獲得し10以上の
実装が様々なベンダからリリース



韓国

OPRoS

OROCOS

ROS.org

米国ベンチャー Willow Garage発
2007年から開発
ロボットを配布するプログラム等により
一気にユーザを獲得
現在はOSRF財団が管理

ユーザ数: 約10倍の開き

課題



HONDA



OROSはデファクト標準であり, OSRFにより仕様を変えられてしまう恐れあり。

⇒ バージョンが上がるごとに, ユーザが振り回される

⇒ ROSは技術的に古く, 次期バージョンアップで大幅に変わる可能性あり。

○国内ユーザと海外ユーザの利用OSの差(国内:Windows, ITRON, 海外:Linux)

⇒ Linuxのみ対応するROSと, 様々なOSに対応するRTMの利用比率は半々程度

○国内でのソフトウェアの再利用が進んでいない。

⇒ オープンソース文化の欠如。

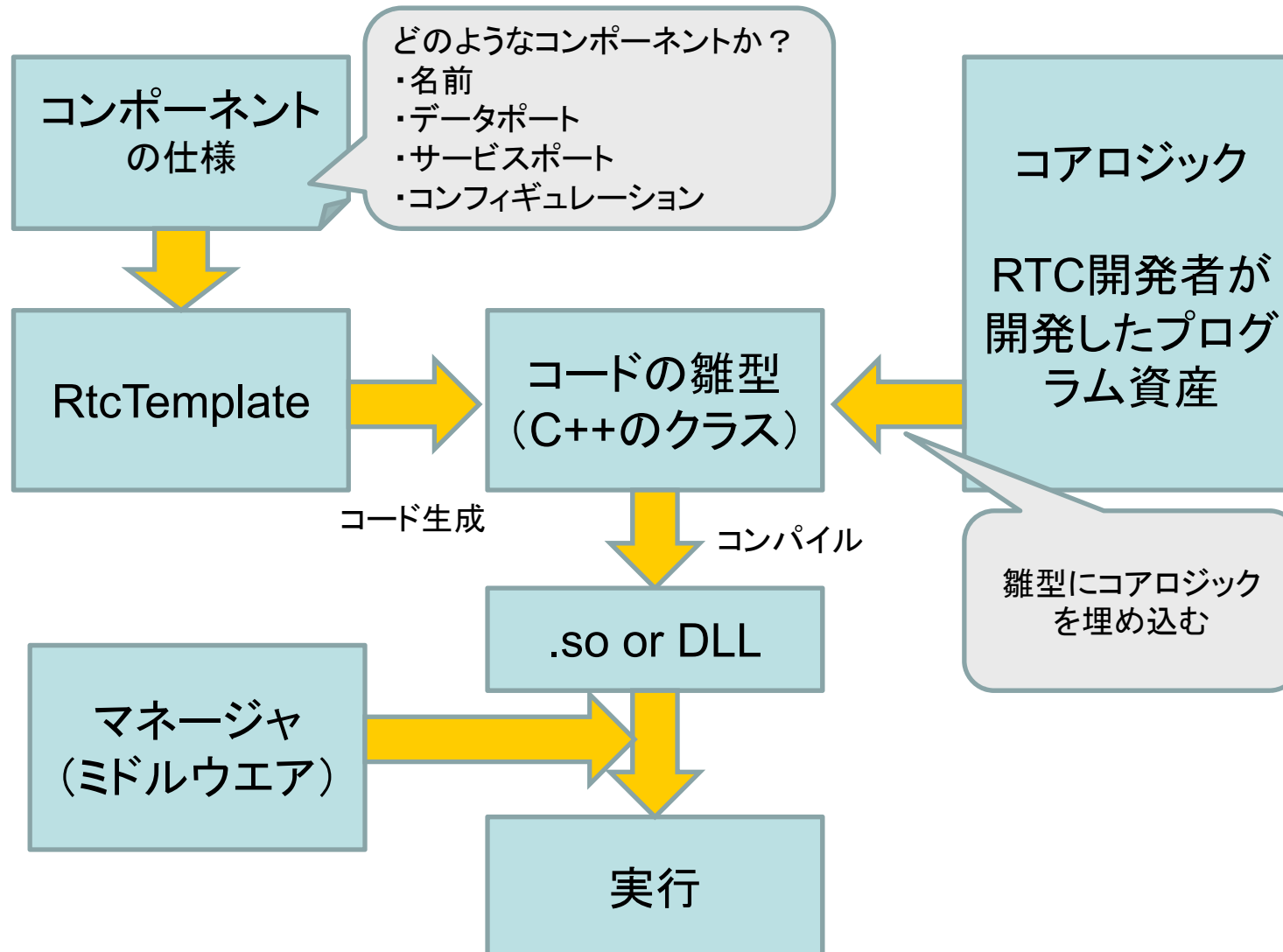
⇒ NEDO知能化PJで蓄積したソフトウェア資産も再利用が進んでいない。

提言

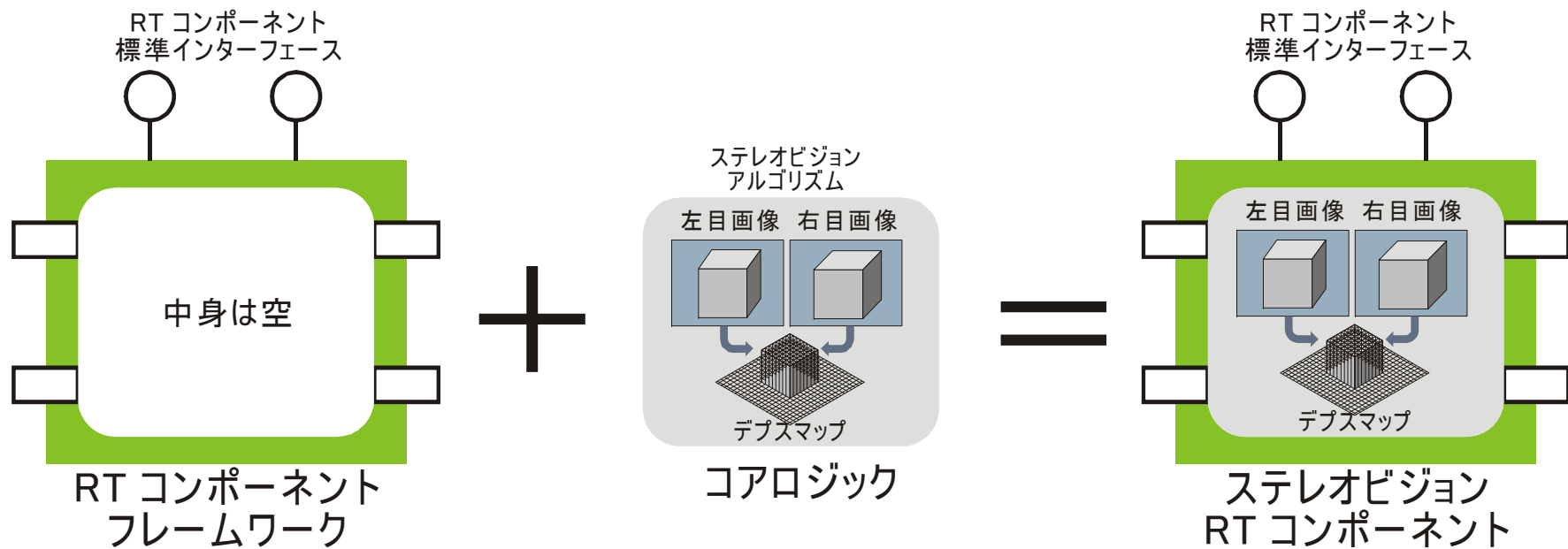
- 自前主義はやめよう！！
 - 書きたてのコードより、いろいろな人に何万回も実行されたコードのほうが動くコードである！！
 - 自分にとって本質的でない部分は任せて、本当にやりたい部分・やるべき部分のコードを書こう！！
 - 誰かがリリースしたプログラムは一度は動いたことがあるプログラムである！！
 - 人のコードを読むのが面倒だからと捨ててしまうのはもったいない！！
- オープンソースにコミットしよう！！
 - 臆せずMLやフォーラムで質問しよう！！
 - どんなに初歩的な質問でも他の人にとっては価値ある情報である。
 - 要望を積極的にあげよう！！
 - できればデバッグしてパッチを送ろ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

RTコンポーネントの開発

OpenRTMを使った開発の流れ

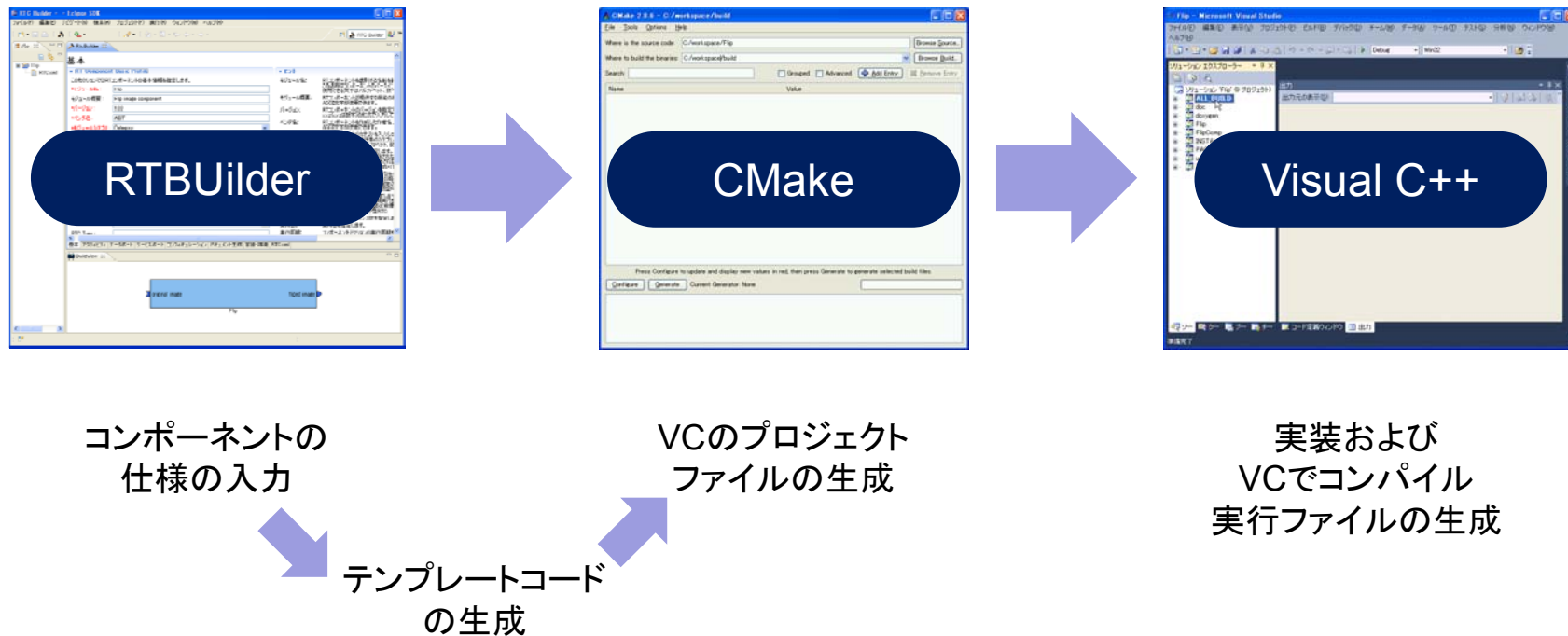


フレームワークとコアロジック



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

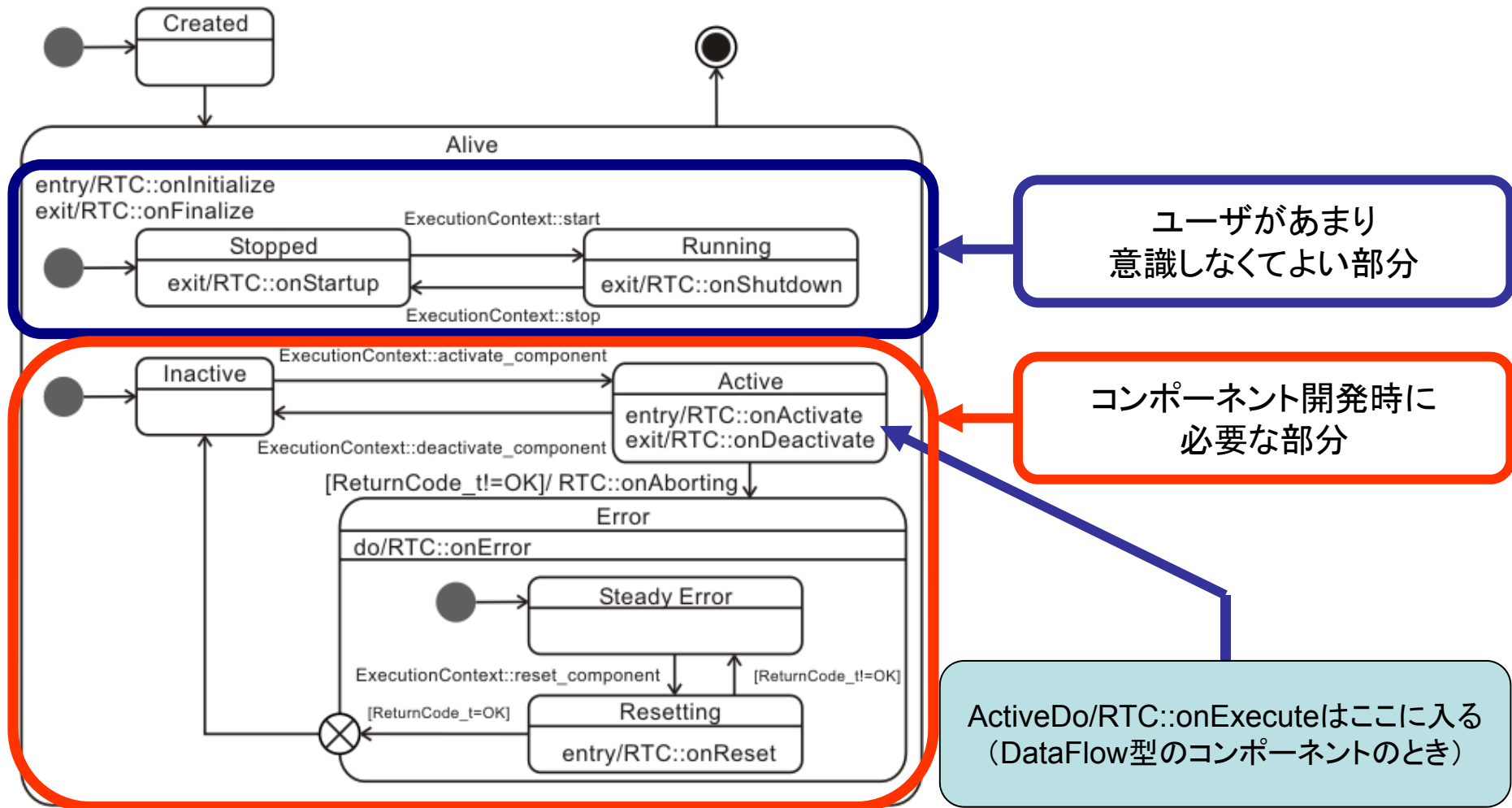
- 生成されたクラスのメンバー関数に必要な処理を記述
- 主要な関数
 - onExecute (周期実行)
- 処理
 - InPortから読む
 - OutPortへ書く
 - サービスを呼ぶ
 - コンフィギュレーションを読む

```
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_something())
            return RTC::RTC_OK;
        return 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

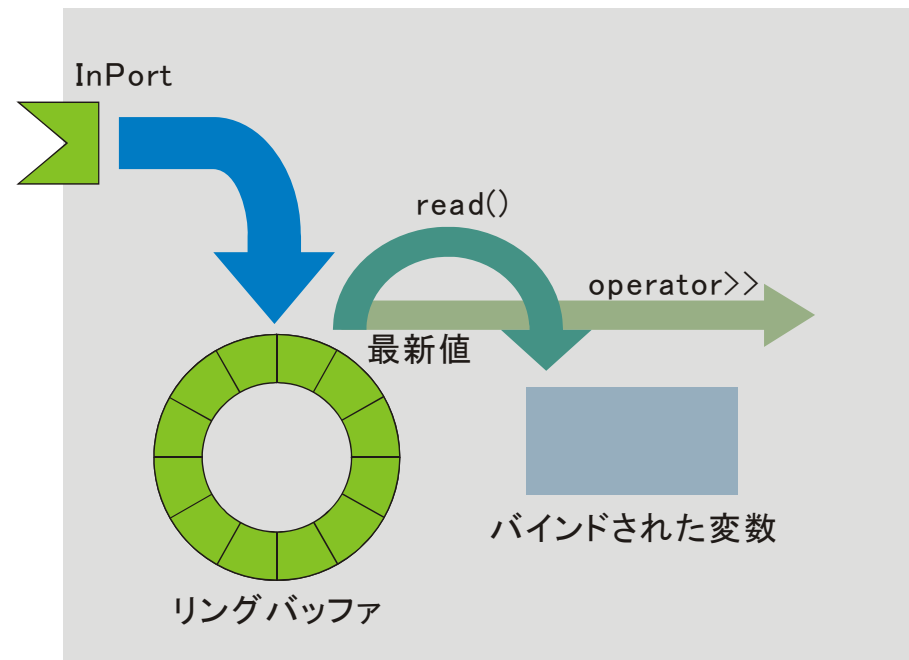
RTCの作成＝コールバック関数に処理を埋め込む

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

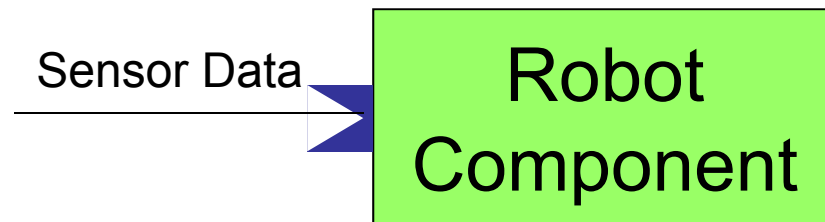
- InPortのテンプレート第2引数: バッファ
 - ユーザ定義のバッファが利用可能
- InPortのメソッド
 - read(): InPort バッファからバインドされた変数へ最新値を読み込む
 - >>: ある変数へ最新値を読み込む



基本的にOutPortと対になる

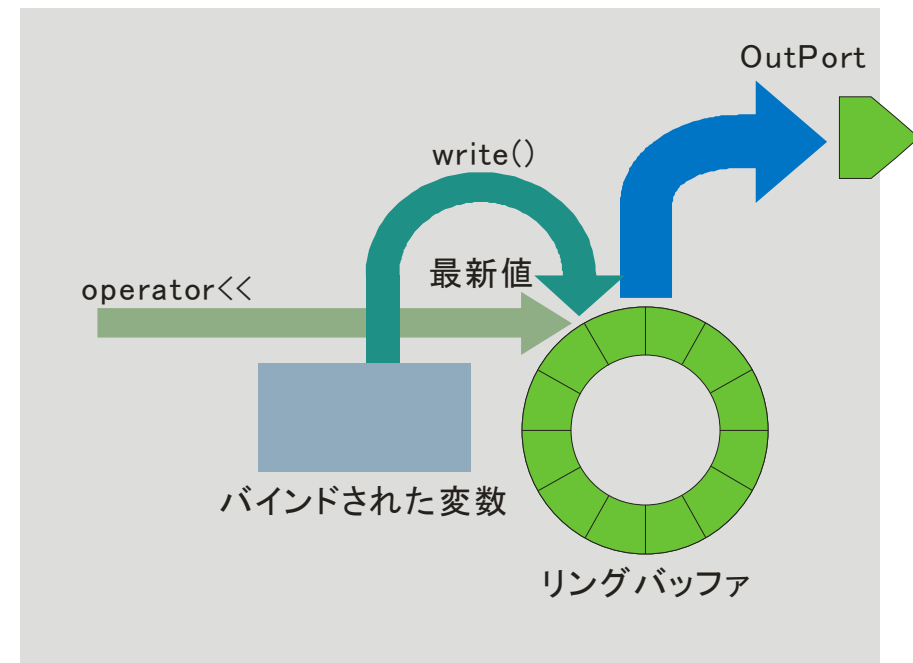
データポートの型を
同じにする必要あり

例



OutPort

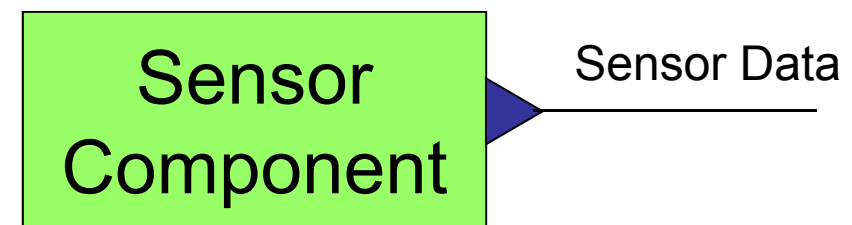
- OutPortのテンプレート第2引数:
バッファ
 - ユーザ定義のバッファが利用
可能
- OutPortのメソッド
 - write(): OutPort バッファへ
バインドされた変数の最新値
として書き込む
 - >>: ある変数の内容を最新
値としてリングバッファに書き
込む



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;
};
```

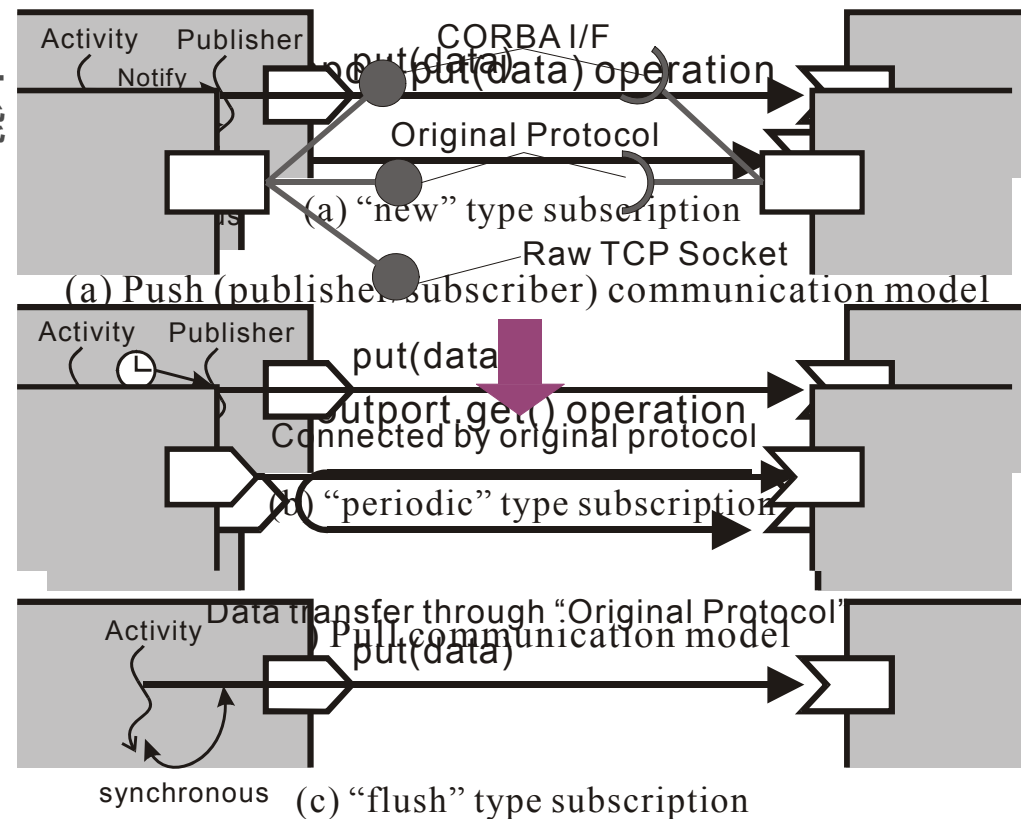
- 基本型
 - tm: 時刻
 - data: データそのもの

```
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

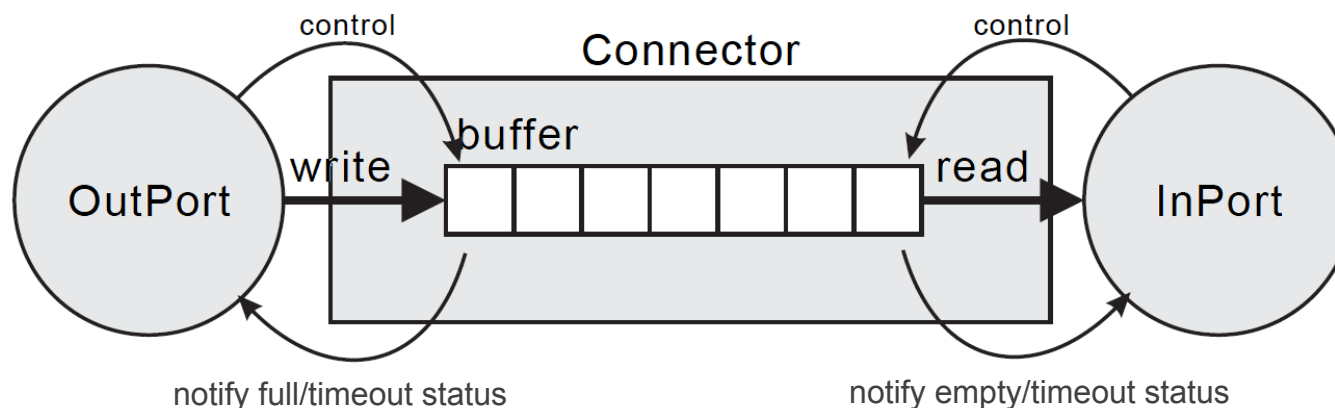
データポート

- 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

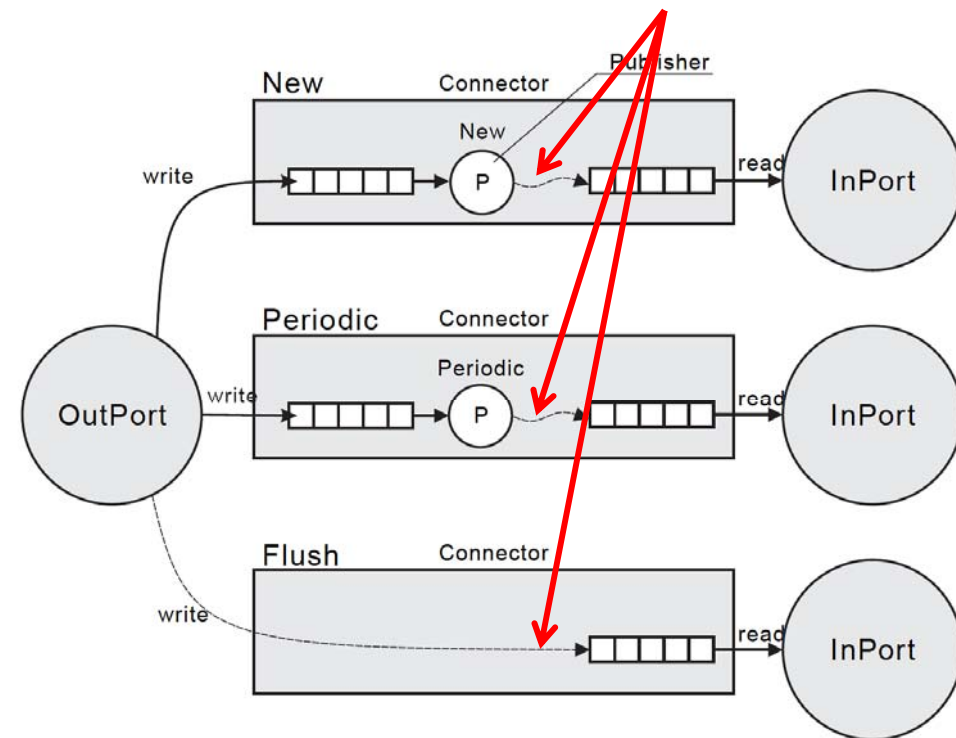
- Connector:
 - バッファと通信路を抽象化したオブジェクト。OutPortからデータを受け取りバッファに書き込む。InPortからの要求に従いバッファからデータを取り出す。
 - OutPortに対してバッファフル・タイムアウト等のステータスを伝える。
 - InPortに対してバッファエンプティ・タイムアウト等のステータスを伝える。
- OutPort:
 - アクティビティからの要求によってデータをコネクタに書き込むオブジェクト
- InPort:
 - アクティビティからの要求によってデータをコネクタから読み出すオブジェクト



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

ネットワーク等による通信



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

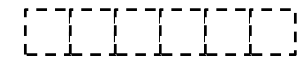
- バッファ残留データ
– 送り方のポリシー

ポリシー	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

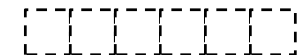
- データ生成・消費速度を考慮して設定する必要がある。

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形式のポリシーを導入するかどうかは検討中です。ご意見ください。

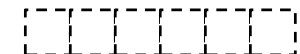
ALL



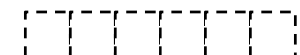
FIFO



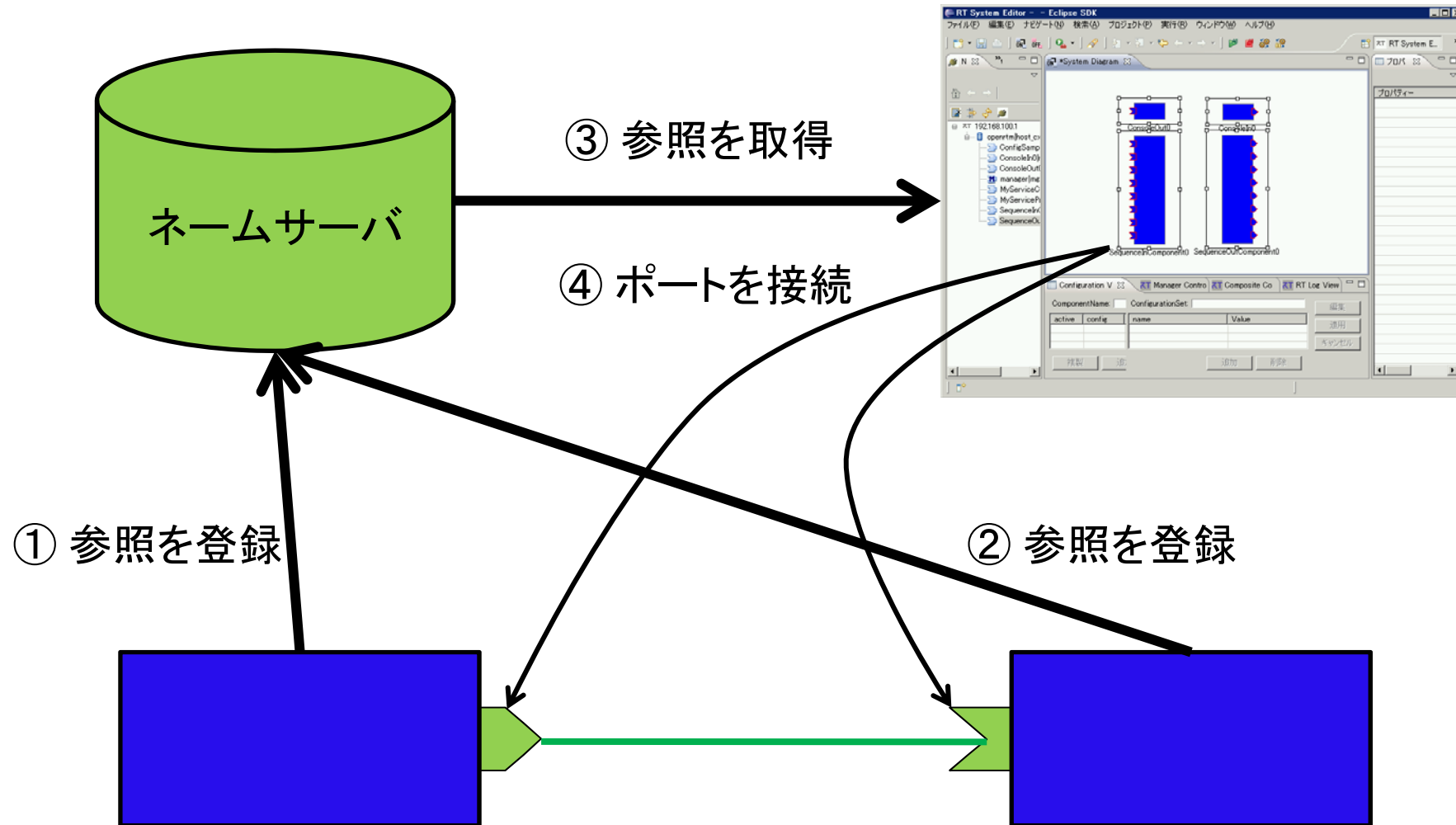
NEW



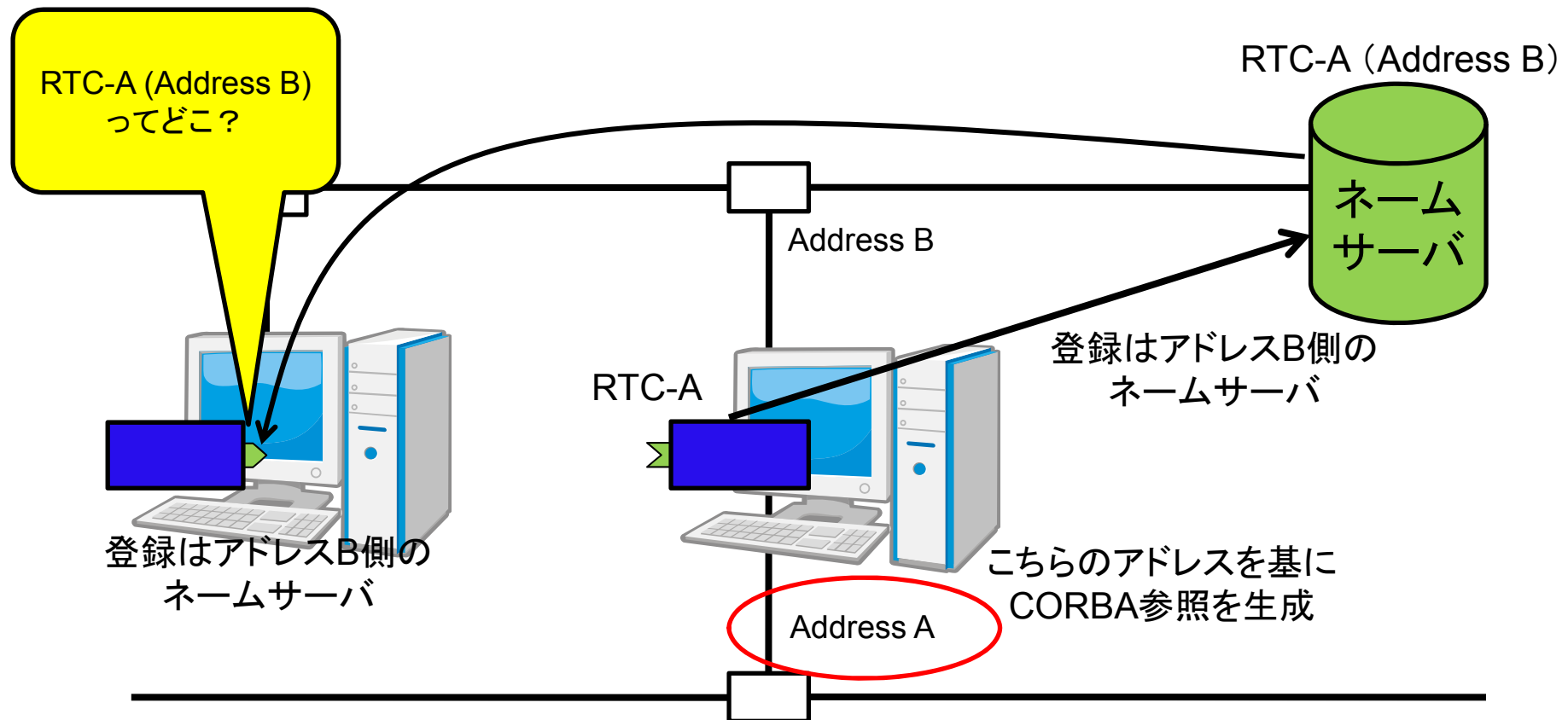
SKIP



動作シーケンス



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



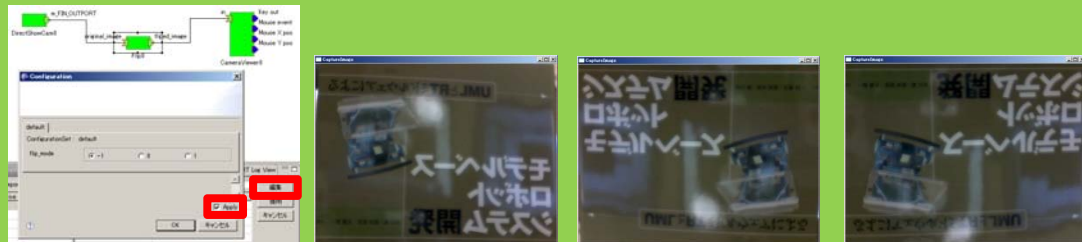
必須でないOption設定

まとめ

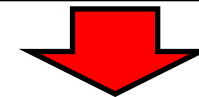
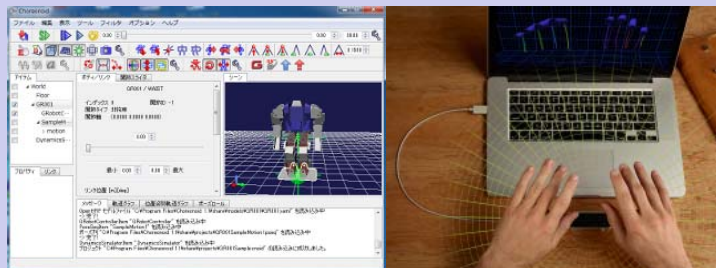
- RTミドルウェアの概要
 - 背景、目的、利点
 - 標準化、適用例
 - 過去のプロジェクト、Webページ
- RTコンポーネントの開発
 - 開発の流れ
 - 動作シーケンス
 - コールバック、データポート、rtc.conf

本日の実習の流れ

13:00 - 14:30
RTコンポーネント作成
(Flipコンポーネント)
講師: 安藤



14:45 - 16:45
ロボット制御コース
講師: ビグズ ジェフ氏



14:45 - 16:45
G-ROBOT & Choreonoidコース
講師: 木村哲也先生(長岡技科大)
松坂要佐氏

