2007 JSME Conference on Robotics and Mechatronics

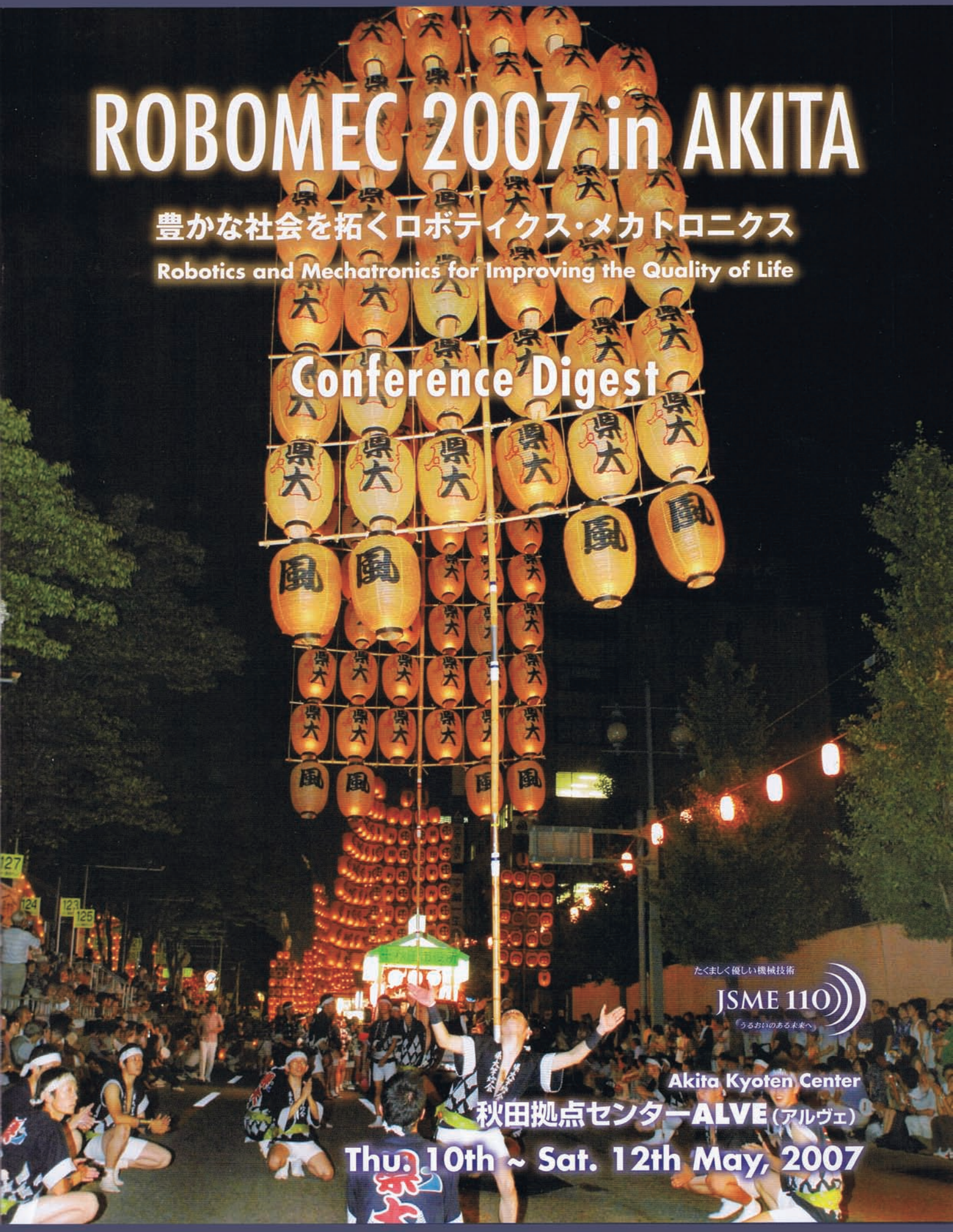
主催　社団法人 日本機械学会 ロボティクス・メカトロニクス部門

ROBOMEC 2007 in AKITA

豊かな社会を拓くロボティクス・メカトロニクス

Robotics and Mechatronics for Improving the Quality of Life

Conference Digest



たくましく優しい機械技術

JSME 110

「うるおいのある未来へ」

Akita Kyoten Center

秋田拠点センターALVE（アルヴェ）

Thu. 10th ~ Sat. 12th May, 2007

社団法人 日本機械学会

〒160-0016 東京都新宿区信濃町35番地 信濃町煉瓦館5階 TEL. (03) 5360-3500　FAX (03) 5360-3508

OpenRTM_aist による実時間制御を考慮した RT コンポーネント

Real Time RT Component on OpenRTM-aist

○尹祐根　安藤慶昭　末廣尚士　北垣高成　神徳徹雄（産総研）

Woo-Keun YOON, Noriaki ANDO, Takashi SUEHIRO, Kosei KITAGAKI and Tetsuo KOTOKU

Intelligent Systems Research Institute, AIST

Abstract— OpenRTM-aist that has been developed in our research group is one of RT-middleware frameworks. In this paper, we report a RT component that includes a real time scheduling on OpenRTM-aist. A real time scheduling is implemented in ARTLinux. To demonstrate our method, we make a real time RT component for master arm.

Key Words: RT (Robot Technology), Middleware, Component, Real Time

1. はじめに

複雑な RT システム (ロボット技術を用いたシステムの総称) の分析，設計，実装から成る，体系的なシステムインテグレーションを支援する実装プラットフォーム「RT ミドルウエア」およびソフトウエアのモジュール化のフレームワークとして「RT コンポーネント」の研究開発を実施してきた [1]–[3].

システムインテグレーションの知識は，個々の研究者のスタンスに依存している．したがって，知識の蓄積と再利用を実現するには，共通のプラットフォームを用意し，そのプラットフォーム上で様々なシステムを構築していく手法が有効である．そこで，我々は共通プラットフォーム上での RT ミドルウエアと RT コンポーネントの実装を実現する OpenRTM-aist を作成し，多くの研究者に利用して頂くことにより改良を進めてきた [4], [5].

ロボットには，多くのモータやセンサが実装されるため，ロボット制御には実時間 OS を利用することが多い．実時間 OS としては，RT-Linux, ARTLinux, ITRON, QNX, VxWorks などを挙げることができる．本論文では，実時間制御プログラムが簡単に記述できるなど様々なメリットの多い ARTLinux を利用する．

これまで開発してきた多数の実時間制御 OS を利用したロボット制御プログラムを RT コンポーネント化する場合，多大な労力が必要となれば RT コンポーネント化に抵抗を感じてしまう．そこで，本論文では，既に開発済みであるロボット制御プログラムを，最小限の労力で OpenRTM-aist のフレームワーク上に実時間制御 RT コンポーネント化する手順を紹介する．

2. ソフトウエア構成

2·1 ARTLinux

現在リリースされている ARTLinux は，Kernel 2.4 用である．ARTLinux では，実時間制御に必要な関数がいくつか用意されているが，最低限以下の 3 つの関数を利用することで実時間制御が可能となる．

- int art_enter(art_prio_t prio, art_flags_t flags, int usec)
  実時間処理を設定し，実時間処理に入る．
  引数 prio は，実時間タスクの優先度を設定する．ART_PRIO_MIN から ART_PRIO_MAX 間の値を入れる．最優先処理をさせる場合には，ART_PRIO_MAX となる．
  引数 flags は，実時間タスクの実行を制御するフラグを設定する．周期実行させる場合には，ART_TASK_PERIODIC となる．
  引数 usec は，周期実行時間を設定し，マイクロ秒単位の値を入れる．1 [ms] 周期で実行させる場合には，1000 となる．
  戻り値は，正常に終了した場合には 0，エラーが発生した場合には-1 となる．
- int art_wait()
  art_enter で設定した usec 毎に呼ばれる．
  戻り値は，正常に終了した場合には 0，エラーが発生した場合には-1 となる．
- int art_exit()
  実時間処理から抜け，非実時間処理に戻る．
  戻り値は，正常に終了した場合には 0，エラーが発生した場合には-1 となる．

2·2 実時間制御プログラム

前提として，ARTLinux 上で動作するマスタアームのプログラムが既に開発されているとする．以下に，簡単化したサンプルプログラムを記述する．なお，実時間制御周期は 1 [ms] である．

```
int main()
{
  // MasterArm の初期化
  master_init();

  // ARTLinux の初期化
  if (art_enter(ART_PRIO_MAX, ART_TASK_PERIODIC,
                1000) == -1){
    perror("art_enter");
    exit(1);
  }

  // 実時間周期関数
```

```
  while(1){
    if (art_wait() == -1) {
      // 制御プログラムの記述
      ActiveDoProcess()
    }
  }

  // ARTLinux の終了
  if (art_exit() == -1) {
    perror("art_exit");
    exit(1);
  }
}
```

2·3 OpenRTM-aist

現在リリースされている OpenRTM-aist のバージョンは，0.2.0 である．OpenRTM-aist-0.2.0 のインストール後，コンポーネントのテンプレートジェネレータ (rtm-template) を使用して，以下のように実時間 RT コンポーネントのテンプレートを作成する．

```
> rtc-template --c++ --module-name=MasterArm \
  --module-desc='Master Arm' \
  --module-version=0.1 --module-author=YOON \
  --module-category=MasterArm \
  --module-comp-type=COMMUTATIVE \
  --module-act-type=SPORADIC \
  --module-max-inst=10 \
  --module-inport=InData:TimedFloatSeq \
  --module-outport=OutData:TimedFloatSeq
  MasterArm.h was generated.
  MasterArm.cpp was generated.
  MasterArmComp.cpp was generated.
  Makefile.MasterArm was generated.
>
> ls
Makefile.MasterArm MasterArm.h MasterArm.cpp
MasterArmComp.cpp
```

MasterArm.h の中には，様々な関数が用意されているが，最低限以下の 3 つの関数を利用することで実時間制御が可能となる．コンポーネントの操作には，GUI を使った直感的でわかりやすい RTCLink を利用する．

- virtual RtmRes rtc_init_entry();
  rtc-link 上でコンポーネントを表示させる時にコールされる関数であり，実時間制御の初期化に利用する．
- virtual RtmRes rtc_active_do();
  rtc-link 上でコンポーネントを start させるときにコールされる関数であり，実時間周期に利用する
- virtuak RtmRes rtc_exiting_entry();
  rtc-link 上でコンポーネントを stop させるときにコールされる関数であり，実時間制御終了に利用する

2·4 実時間制御 RT コンポーネント

2.2 で紹介したプログラムを RT コンポーネント化する．

まず,2.3 で作成した MasterArm.h 内で使用する関数 (rtc_init_entry, rtc_active_do, rtc_exiting_entry) のコメントアウトを外す．

```
public:
  MasterArm(RtcManager* manager);

  // [Initializing state]
  virtual RtmRes rtc_init_entry();

  // [Ready state]
  //  virtual RtmRes rtc_ready_entry();
  //  virtual RtmRes rtc_ready_do();
  //  virtual RtmRes rtc_ready_exit();

  // [Starting state]
  //  virtual RtmRes rtc_starting_entry();

  // [Active state]
  //  virtual RtmRes rtc_active_entry();
  virtual RtmRes rtc_active_do();
  //  virtual RtmRes rtc_active_exit();

  // [Stopping state]
  //  virtual RtmRes rtc_stopping_entry();

  // [Aborting state]
  //  virtual RtmRes rtc_aborting_entry();

  // [Error state]
  //  virtual RtmRes rtc_error_entry();
  //  virtual RtmRes rtc_error_do();
  //  virtual RtmRes rtc_error_exit();

  // [Fatal Error state]
  //  virtual RtmRes rtc_fatal_entry();
  //  virtual RtmRes rtc_fatal_do();
  //  virtual RtmRes rtc_fatal_exit();

  // [Exiting state]
  virtual RtmRes rtc_exiting_entry();
```

次に，MasterArm.cpp 内で使用する関数 (rtc_init_entry, rtc_active_do, rtc_exiting_entry)

のコメントアウトを外し，プログラムを追加する．

```
#include "art_task.h"
extern "C" void master_init();
extern "C" void ActiveDoProcess();
 .
 .
RtmRes MasterArm::rtc_init_entry()
{
  // master arm の初期化
  master_init();
  // ARTLinux の初期化
  if (art_enter(ART_PRIO_MAX, ART_TASK_PERIODIC,
      1000) == -1){
    std::perror("art_enter");
    std::exit(1);
  }
  return RTM_OK;
}
 .
 .
RtmRes MasterArm::rtc_active_do()
{
  // 実時間周期関数
  if (art_wait() == -1) {
    std::perror("art_wait");
    std::exit(1);
  }
  // 制御プログラムの記述
  ActiveDoProcess();
  return RTM_OK;
}
 .
 .

RtmRes MasterArm::rtc_exiting_entry()
{
  // ARTLinux の終了
  if (art_exit() == -1) {
    std::perror("art_exit");
    std::exit(1);
  }
  // manager stop
  handler(0);
  return RTM_OK;
}
```

マスタアームのプログラム修正はここまでであり，マスタームの実時間 RT コンポーネント化が終了となる．

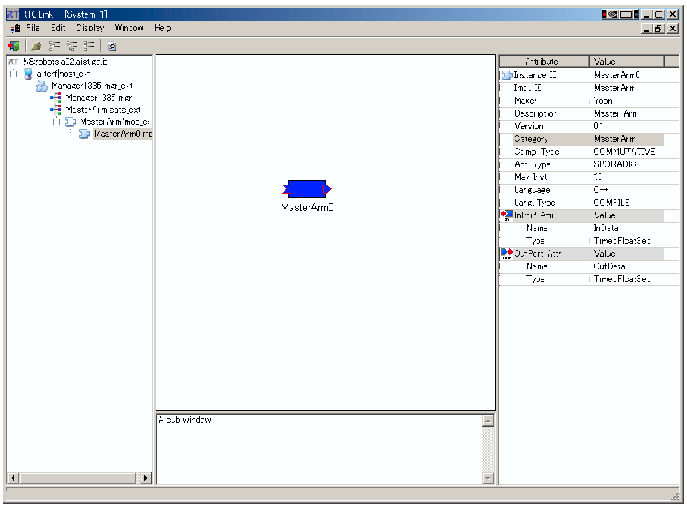


Fig.1 Real time RT component on RTCLink

図 1 に作成した実時間 RT コンポーネントを示す．コンポーネントには，InPort, OutPort それぞれひとつ実装されている．コンポーネントは，RTCLink 上に置かれた時点で MasterArm::rtc_init_entry() を実行する．この状態は，コンポーネントの色をブルーにすることで表現している．

3. まとめと今後の課題

本論文では，既に開発されている実時間ロボット制御プログラムを，最小限の労力で実時間 RT コンポーネント化する手法を紹介した．

産総研から公開されている OpenRTM-aist に用意されている rtc-template や RTCLink を利用すれば，既存の実時間制御プログラムを簡単にコンポーネント化することができることがわかった．

今後，既存のロボット制御プログラムを RT コンポーネント化する際に，本論文が多少の参考となれば幸いである．

参考文献

[1] 末廣尚士, 北垣高成, 神徳徹雄, 尹祐根, 安藤慶昭, “RT コンポーネントの実装例.RT ミドルウエアの基本機能に関する研究開発 (その 1)”, 第 21 回日本ロボット学会学術講演会予稿集, 1F27, 2003.09

[2] 末廣尚士, 北垣高成, 神徳徹雄, 尹祐根, 安藤慶昭, “RT コンポーネントの実装例.RT ミドルウエアの基本機能に関する研究開発 (その 2)”, 第 21 回日本ロボット学会学術講演会予稿集, .1F28, 2003.09

[3] 安藤慶昭, 末廣尚士, 北垣高成, 神徳徹雄, 尹祐根, “RT 要素のモジュール化および RT コンポーネントの実装”, 第 9 回ロボティクスシンポジア, pp.288-293, 2004.03

[4] 土屋裕, 水川真, 安藤吉伸, 末廣尚士, 安藤慶昭, 中本啓之, 池添明宏, “RT ミドルウェアを用いた CAN モジュール構成型ロボットの構築”, 日本機械学会 ロボティクス・メカトロニクス講演会 2006, 1P1-C31, 2006.05

[5] 原佑輔, 國井康晴, 安藤慶昭, “RT ミドルウェアを利用した遠隔移動ロボットシステムの構築”, 日本機械学会 ロボティクス・メカトロニクス講演会 2006, 1P1-C36, 2006.05