第12回 公益社団法人 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

The 12th SICE System Integration Division
Annual Conference

SI 2011

講演概要集

SICE SO since 1961

SICE SI
SYSTEM INTEGRATION DIVISION

2011.12.23-25 京都大学 吉田キャンパス

OpenRTM-aist によるビジョンシステムのためのフレームワーク
– ユースケース分析と設計 –

産業技術総合研究所 ○安藤慶昭,
ミュンヘン工科大学 ウィトマイヤー シュテファン, ヤンツ ミヒャエル, クノール アーロイス

Vision Systems Framework based on OpenRTM-aist
– Use-case analysis and its architecture –

*Noriaki ANDO, AIST
Steffen Wittmeier, Michael Jäntsch, Alois Knoll, TUM

Abstract—
OpenRTM-aist is a generic middleware for component based system development for robotic systems. If it is applied to specific domain such as vision, motion control, manipulation and so on, the domain specific frameworks improve its usability, flexibility and performance of generic middleware. In this paper, we propose a vision framework for OpenRTM-aist. We enumerate use-case of vision systems and analyze them. From the results some interfaces which are commonly used in vision related systems are defined.

Key Words: RT-Middleware, RT-Component, vision system, framework

1. はじめに

近年ソフトウエアプラットフォームやフレームワークを用いたロボットシステムの研究開発が盛んに行われている。これらには、OpenCV[1][1], OpenTL[2], XVision[3] 等のような特定の領域に特化したものから、OpenRTM-aist[4, 5] や ROS[6] 等のような汎用のものがある。

特定領域向けのプラットフォームは、対象領域 (例えばビジョン) でシステムを構築する場合、汎用のものに比べて、より容易に早く、また一般的に効率の良いシステムを実現することができる利点がある。しかしながら、ロボットシステムは制御、ビジョン、音声、ナビゲーション等さまざまな領域の技術を組み合わせる場合、汎用フレームワークを利用する必要がある。

両者は対立するものではなく、うまく組み合わせることで、特定領域のフレームワークの持つ機能性や効率性と、汎用フレームワークのもつ柔軟性、再利用性の両者を併せ持つロボットシステムを容易に構築可能となる。

本稿では, 汎用コンポーネントフレームワークとしての OpenRTM-aist の利点を生かしつつ、ビジョンシステムを効率的かつ柔軟に構築するためのフレームワークを提案する。ビジョンシステムにおけるユースケースを列挙・分析し、ここから共通的に利用される機能を抽出する。共通的なインターフェースやデータ型として定義することで、コンポーネントの再利用性能向上を目指す。

2. ユースケース分析

ビジョンシステムにおいて一般的に利用されるであろうコンポーネントを利用したシステム構成を列挙し、そこから共通的に行われる機能や構成の特徴を抽出する。

[1]OpenCV 等は画像処理ライブラリとしての側面と画像処理システムを容易に構築する highgui 等のフレームワークの側面がある。

2·1 単眼カメラとフィルタ

単一のカメラデバイスから画像を取り込み、複数の画像処理を施し、最終的に処理画像や抽象化された値等を出力する場合である (図 1)。カメラ、画像処理、出力はそれぞれ RTC (RT コンポーネント [5]) として構成することができ、画像やその他のデータはデータポート間でやり取りされることになる。これはいわゆるフィルタパターン [7] と呼ばれる形式であり、画像処理において最も一般的なユースケースと言える。

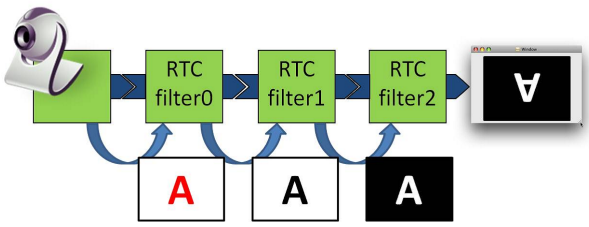

Fig.1 A single camera with filter components.

RTC 間で送受信されるデータは画像や、画像処理から得られた抽象的な値となるが、後者はビジョンという領域外のものもあるため、ここでは議論しない。前者の画像データ型は、多様な画像データを格納できる汎用性の高い型があれば、フィルタの数の増減、順序の入れ替えなどを自由の行うことができ、再利用性も高まる。画像データの表現方法は多くの種類が有り、処理に応じて使い分けられる場合があるため、多様なデータを効率よく一つのデータ型で表現できることが望ましい。

2·2 ステレオカメラ

三次元画像処理においては、2 つまたはそれ以上のカメラからの画像を用いて処理をすることが一般的である (図 2)。

このとき、複数のカメラ間で画像取得の同期が取れ

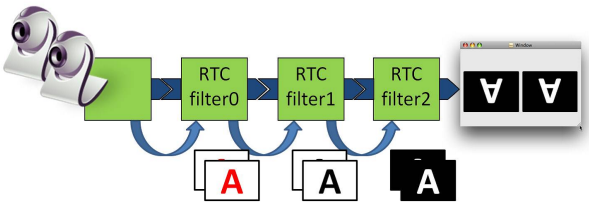

Fig.2 One or more images through filter components.

ていることが重要であり、ハードウエアレベルおよびソフトウエアレベルで実現できる必要がある。

同一の IEEE1394 バス上のカメラ画像の同期取得がハードウエアレベルで自動に行われる Point Grey 社のカメラを利用することで画像間の同期は実現できるが、これを利用するソフトウエア側においても同期を考慮する必要がある。

例えば ROS1 ではひとつのコンポーネント (ノード) は原則として一つのプロセスであるため、2 つのカメラノードを同期させることは難しい。これはステレオ画像用のコンポーネント (ノード) をつくることで解決できるが、カメラデバイスの数に応じてコンポーネントを作成する必要が有り柔軟性に欠ける。デバイスとコンポーネントの対応関係を調整する亜ある種の仕組みが必要となる。

2·3 分散カメラ

多数のセンサを利用したロボット制御 [8] や、知能化空間 [9] 等のシステムにおいて、カメラは最もよく用いられるセンサのひとつである (図 3)。通常、天井などに設置されたカメラからの画像あるいは処理済のデータを、ネットワークを介して 1 箇所に集め処理を行う。RT コンポーネントや ROS のノードは分散コンポーネントのための枠組みであるため、分散カメラを扱うことは比較的容易にできるが、カメラ間の同期などは別途考慮する必要がある。

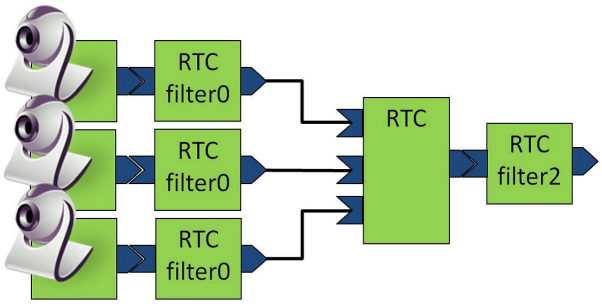

Fig.3 Distributed image source and gatherer component.

2·4 同期並列・マルチレート

一つの画像に対して、並列に複数の処理を行い、かつ並列の処理が同期する必要がある場合が有る。さらに複雑な状況として並列処理部分の周期が異なる場合も考えられる (図 4)。

具体的な例として、多数の画像認識処理を並列で行い、処理結果を統合して一つの認識処理を出力するシステム等がある。また、一部の処理に長時間かかったり、全く別の処理を行いたいために、他の処理と異なるレートで処理を行う場合もあり、このとき一つの画像から処理が開始されるため、複数のフィルタの処理の開始を同期させた方が効率よく処理が可能となる。

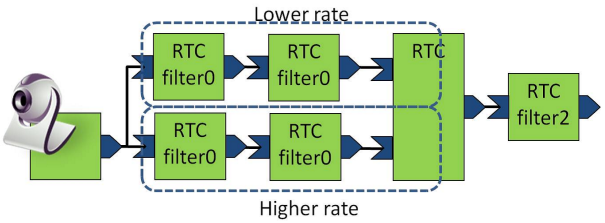

Fig.4 Parallel, synchronized and multi rate processing.

3. フレームワークに求められる機能

本節では、上述のユースケースから、ビジョンフレームワークが持つべき機能について議論する。

3·1 デバイス管理フレームワーク

上記のステレオ画像処理システム等では、一つのコンポーネントに対して複数のカメラが関連付けられており、それらのカメラが同期的に動作することが期待される。さらに多くの複数のカメラ画像を同期的に扱ったり、いくつかのグループとして利用するケースなどが考えられる。

こうした、動的なコンポーネントとカメラの関連付けを行うためには、RT コンポーネントにサービスインターフェースを提供するだけでは十分ではない。なぜなら、RT コンポーネントのライフサイクル開始以前に、予めカメラデバイスの状況を知り、それに基づいてコンポーネントの実体化を行う必要があるためである。したがって、例えば RT コンポーネントのライフサイクルを管理するコンポーネントマネージャ等にサービスインターフェースを設置する必要がある。

3·2 共通インターフェース

カメラデバイスには、明るさ、露出、シャッタースピードなど調整可能な様々なパラメータがある。また、デバイスドライバによってはビデオストリームを保存したりチューナ機能を提供するものなども存在する。RTC においてこうした、詳細な機能にアクセスする方法としてはサービスポートおよびそこに設置されるインターフェースが利用される。再利用性向上のために、必要十分な機能を提供するカメラデバイスのためのインターフェースを定義する必要がある。

3·3 共通データ型

コンポーネント間では、データポートを介して画像データの送受信が行われる。画像データには、様々な表現方法 (フォーマット) が有り、処理の内容や目的に応じて使い分ける必要がある。画像データは一般的に、画像に関する基本情報 (幅、高さ、色数、フォーマット等) およびデータ列といった形で表現され、これを共通化することができれば、コンポーネントの再利用性が向上する。

3·4 データポート

画像処理システムにおいては、一般に 1 画面あたり 1∼ 数 MB 程度の生画像データを 15FPS ∼ 60FPS 程度のフレームレートで処理する。フィルタ型構造のシステムにおいては、RTC が多数接続されているため、

データポート間でのコピーが多数発生すれば、現在の一般的なコンピュータの処理速度でもパフォーマンスの低下は無視できないものとなる。

さらに、RTC では、言語間や CPU アーキテクチャ間の相互運用実現のためマーシャリング (データの直列化) が行われるためより多くの処理が必要となる。こうした観点から、画像処理フレームワークでは、そのデータの大きさから、より効率的なデータ転送の機能が求められる。

4. アーキテクチャ

本節では、上述の議論に基づいて、ビジョンフレームワークの具体的機能について議論する。

4·1 デバイス管理レイヤ

上述のカメラのデバイス情報の取得や、デバイスと RTC 間の関連付けは、RTC のライフサイクル外で行われるため、RTC を管理する側すなわちコンポーネントマネージャ側に何らかの仕組みが必要となる。そこで、こうしたデバイスの管理をになうデバイス管理レイヤを提案する (図 5)。デバイス管理レイヤは RTC マネージャの拡張モジュールとして実装され、マネージャにロードされると一種のローカルサービスとしてデバイスの管理を行う。また、CORBA サービスとして公開することでこのサービスを外部から利用することも可能となる。

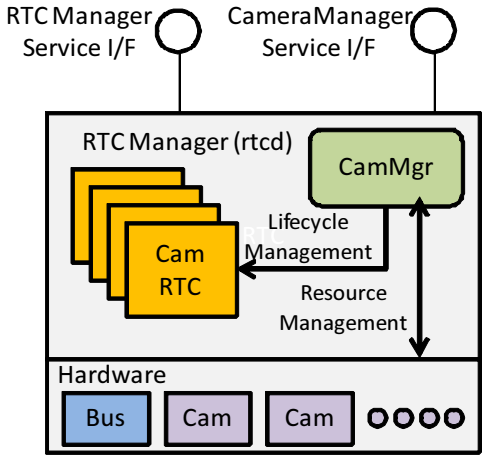

Fig.5 Device management layer and provided CORBA service.

4·2 共通データ型

共通となるデータ型として画像データ型がある。

データ型について、OpenRTM-aist に付属するもの、また OpenRTM-aist ユーザが独自に定義して使用しているもの、および他のプラットフォーム [1, 6] で使われているものを調査した。ほぼすべてのデータ型が、画像のサイズやデプス、フォーマットなどのフィールドを含み、若干の違いがあるもののほぼ同等の表現能力を持つ。ただし、問題点として、画像フォーマットを表現する方法がデータ型の間で若干異なることが分かった。

OpenRTM-aist の CameraImage 型や ROS で使用されている方は文字列による表現、他のものは enum 列挙子による表現である。前者は、様々なフォーマットを事前の制約なしに利用することができる一方で、フォーマットの特定に時間がかかる問題点がある。逆に列挙子による方法では、フォーマットの特定が定数時間で行える一方で、後からフォーマットの種類を変更することが難しいといった問題もある。

```
struct TimedImage
{
  Time tm;
  unsigned short width;
  unsigned short height;
  unsigned short bpp;
  unsigned short channel;
  unsigned short widthStep;
  FourCC format;
  ImageData image_data;
};

typedef sequence<TimedImage> TimedImageSeq;
```

Fig.6 Image data type definition.

4·2.1 FourCC

そこで、画像のフォーマットを表現する方法として FourCC と呼ばれる標準を利用することとした。FourCC は画像や動画像のフォーマットを表す 4 バイト (4 文字) による表現方法であり、主に、動画コーデックなどで利用されている、FourCC.org が管理するデファクト標準である [10]。

4·2.2 ステレオ画像

上述したとおり、ステレオ画像を含む複数枚の画像をデータポートで送受信するケースがある。複数枚の画像を一つのデータとして送る方法としては、一つの構造体中に複数枚の画像を保持する方法や、単純に 1 枚の画像データをシーケンス (動的配列) として保持する方法が考えられる。前者は 1 枚の画像を保持するデータ型別のデータ型として定義する必要があるのに対し、後者は 1 枚の画像データ型の単なる配列となるため、データ間の独立性が高まり扱いやすくなるため、RTC 内の処理が簡潔にできる可能性が有る。そこで、ステレオ画像を扱う方法としてはシーケンス型を採用した (図 6)。

4·3 デバイスインターフェース

デバイスの詳細機能を利用するためのインターフェースである。カメラデバイスは、多種多様なデバイスが市販されており、その機能も様々である。従って、共通カメラデバイスインターフェースに各機能ごとに細かなオペレーションを定義することは難しい。従って、各機能の詳細 (feature) を一種のパラメータ列とみなし、そのリストをやり取りすることで、feature の設定を変更するものとした。以上から定められたカメラデバイスインターフェースを図 7 に示す。

4·4 実行コンテキスト

画像処理システムを上述の RT コンポーネントを用いたフィルタ構造として実現する場合、コンポーネント間の処理の同期が問題となる。通常、個々の RT コンポーネントはそれぞれ自身のロジックを駆動する実行コンテキストを一つ持つ。単にコンポーネントのポー

```
interface CameraDevice
{
  ReturnCode init(in DeviceProfile profile);
  ReturnCode getProfile(out DeviceProfile profile);
  ReturnCode getFeatures(out NVList features);
  ReturnCode setFeatures(in NVList features);
  ReturnCode getOneShotImage(out TimedImage image);
  ReturnCode startCapture();
  ReturnCode stopCapture();
  ReturnCode getServiceProfile(
          out ServiceProfileList service_list);
  ReturnCode getService(in string service_name,
                        out CameraService svc);
};
```

Fig.7 CameraDevice interface definition.

トを接続した場合、個々のロジックとデータ転送は独立して行われ、仮にコンポーネント数を n、実行周期を t とした場合、処理により最悪 $n \times t$ の遅延が発生する。RT コンポーネントは、任意の実行コンテキストに参加して、その実行コンテキストに参加できるため、図 8 のように、一つの実行コンテキストに複数の RTC を関連付けることで、処理とデータ転送を直列的に実行し、すべての処理を同期的に行うことが可能である。

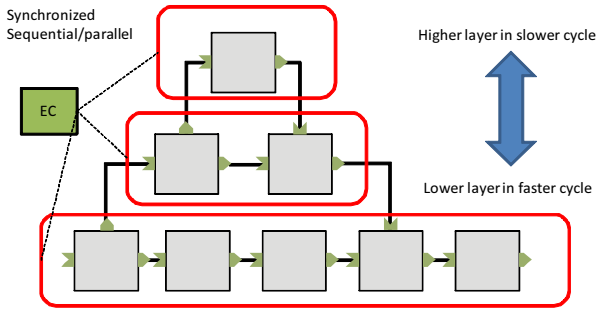

Fig.8 Synchronized andmultirate execution by Execution Context.

さらに、上述した並列・マルチレート実行も、実行コンテキストが複数の異なる実行レートのスレッドを内部で制御することにより、希望する実行形態を実現することが可能である。利用しやすくするためには、RTC と実行コンテキストの関連付けをより簡単にするサービスインターフェースやツールを提供する必要がある。

4.5 データポート

RT コンポーネントは複数のコンポーネントを同一プロセス上で実体化できるため、原理上データポート間のデータ転送を変数間のコピーにすることが可能である。通常データポート (InPort, OutPort) はそれぞれが固有に管理する変数を持つ。 したがって、入力ポート InPort が管理するメモリに対して、出力ポートである OutPort からデータを直接書き込むことができる。これを DMT (Direct Memory Transfer) Port と呼ぶ。

接続時に ConnectorProfile と呼ばれる接続設定情報をポート間で交換するが、交換情報に InPort のポインタないしは InPort の変数のポインタを OutPort に渡すことで、OutPort から直接 InPort の変数に書き込むことが可能となる。また、共有メモリを利用すれば別プロセス間での通信もメモリコピーと同等の速度で行えることになる。

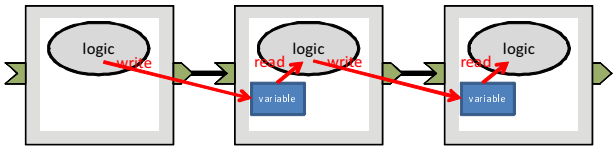

Fig.9 DMT (Direct Memory Transfer) Port.

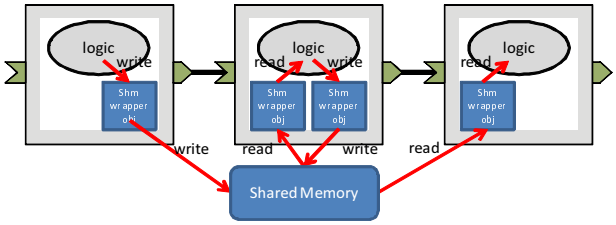

Fig.10 SMT (Shared Memory Transfer) Port.

しかしながら、データポートの変数を共有メモリに配置する際には、構造体内でポインタで表現される領域等も同時に共有メモリ上に置く必要があるため、何らかの直列化を行う必要がある。

5. おわりに

本稿では、OpenRTM-aist を基盤とした画像処理に特化したフレームワークについて述べた。画像処理システムの一般的なユースケースを列挙し、フレームワークまたはミドルウエアが持つべき機能について議論した。主なものとして、デバイス管理フレームワーク、共有メモリ型データポート、特殊化された実行コンテキスト、共通データ型・インターフェースを提案した。今後は、提案したそれぞれの機能を実装し OpenRTM-aist に組み込むとともに、パフォーマンスの向上や、ユーザビリティの向上などに関連して定量的・定性的評価を行う予定である。

参考文献

[1] OpenCV, http://opencv.willowgarage.com/
[2] OpenTL, http://www.opentl.org/
[3] XVision2, http://www.cs.jhu.edu/CIRL/XVision2/
[4] OpenRTM-aist, http://www.openrtm.org
[5] Noriaki ANDO, et.al, "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)", IEEE/RSJ IROS2005, pp.3555-3560, 2005.08
[6] ROS, http://www.ros.org/
[7] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Professional, 1994
[8] Alois Knoll, "Distributed contract networks of sensor agents with adaptive reconfiguration - modelling, simulation, implementation", Journal of the Franklin Institute, Elsevier Science, 338(6):669-705, September 2001.
[9] Lee, Joo-Ho; Hashimoto, Hideki, "Intelligent Space - concept and contents", Advanced Robotics, Vol.16, No.3, pp. 265-280, 2002
[10] FourCC, http://www.fourcc.org/