ROOTを用いたMonitorコンポーネント開発

仲吉一男

June 2009

概 要

この文章では、ROOTを用いたオンラインモニタ用コンポーネント開発について説明します。この文章ではROOTそのものの解説や説明は行いません。すでにROOTを使用している方を対象に1次元ヒストグラムによるオンラインモニタの実装方法を説明します。

目 次

1 はじめに

DAQ ミドルウエアは、ネットワーク分散環境でデータ収集用ソフトウェアを容易に構築するためのソフトウェア・フレームワークです。ユーザは、DAQ コンポーネントと呼ばれるソフトウェア・コンポーネンを組み合わせて DAQ システムを構築することができます。この DAQ コンポーネントの中には取得した実験データをオンラインでデコードしてヒストグラム等にして表示を行う Monitor コンポーネントがあります。CERN で開発が行われている ROOT[1] を使用して、ヒストグラムの作成、表示を行います。この文章では、ROOT そのものの解説や説明は行いません。ROOT 経験者が、モニタ・コンポーネントで ROOT を使用したオンライン・モニタリング行うための実装方法を説明します。

2 開発環境の整備

この文章を作成する際に使用した開発環境は次の通りです。

- OS: Scientific Linux SL release 5.2 (Boron)
- gcc: gcc version 4.1.2 20071124 (Red Hat 4.1.2-42)
- ROOT: Version 5.20/00
- DAQ ミドルウェア: 2009.7 月版

2.1 ROOT のインスト ール

ROOT の開発環境がない方は、[1] からパッケージをダウンロードしてください。バイナリパッケージは、OS の種類とその中で CPU のアーキテクチャ、gcc のバージョンにより複数存在するので、自身の環境に合ったものをダウンロードしてインストールしてください。バイナリパッケージは、tar.gz 形式のファイルを展開すれば使用できます。自分の環境に合ったパッケージが存在しない場合は、ソースファイルをダウンロードしてコンパイルしてください。コンパイルオプション等は、[1] にあるドキュメントを参照してください。

2.2 環境変数の設定

インストールが終了したら環境変数をセットします。$ はプロンプトです。

1. 各自のシェルに応じて環境変数 ROOTSYS を展開したディレクトリにセットする（bash の例）

```
$ export ROOTSYS=/usr/local/root (/usr/local の下へ展開した場合) }
```

2. LD_LIBRARY_PATH の設定（bash の例）

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
```

3. PATH に追加（bash の例）

```
$ export PATH=$PATH:$ROOTSYS/bin
```

4. ROOT の起動テスト

```
$ root
```

root のスプラッシュがでて、root [0] というプロンプトがでれば、インストールは正常に行われたでしょう。”.q”と入力して終了します。

2.3 DAQ ミドルウェアのインストール

「MLF 中性子用 DAQ ミドルウェア インストールおよび操作マニュアル [4]」にしたがって DAQ ミドルウェア for MLF 2009 年 7 月版のパッケージ [3] をインストールしてください。

3 実装の手順

ROOT を利用してオンラインでヒストグラムを生成し、表示するための実装を行います。Monitor コンポーネントの一般的な説明は [2] を参照してください。ここでは、DAQ ミドルウェア for MLF 2009 年 7 月版のパッケージ [3] に入っている Skeleton コンポーネントを利用して、オンライン・モニタのためのコンポーネントを作成します。Skeleton コンポーネントは、例題コンポーネントでその動作のためのロジックは実装されていません。新しいコンポーネントを開発する際に利用できます。

```
$ cp ./DaqComponents.2009.07.tar.gz your_directory
$ cd your_directory
$ tar xvfz DaqComponents.2009.07.tar.gz
```

展開後、DaqComponents というディレクトリができます。その下の src というディレクトリにある下記の 4 つのファイルを使用します。

- Skeleton.h
- Skeleton.cpp
- SkeletonComp.cpp
- Makefile.Skeleton

この例題では、ROOT で 1 次元ヒストグラムを作成して表示します。ROOT の実装に集中するため、コンポーネント自身がデータを生成します。生成するデータは正規分布を示します。

3.1 ROOT 関連ファイルのインクルード、ヒストグラムの宣言

Skeleton.h で関連する ROOT のヘッダファイルを 30 行目から追加します。

```
#include "TH1.h"
#include "TCanvas.h"
#include "TRandom.h"
#include "TApplication.h"
```

Skeleton.h で使用する 1 次元ヒストグラム関連のメンバ変数を 71 行目から次のように追加します。

```
private:
    TRandom* m_rand;       //乱数発生に使用
    TCanvas* m_canvas;     //ROOT Canvas
    TH1F*    m_histo;      //1 次元ヒストグラム
    int      m_bin;        //ヒストグラムのビン数
    double   m_min;        //ヒストグラムの最小値
    double   m_max;        //ヒストグラムの最大値
    double   m_mean;       //正規分布の平均値
    double   m_sigma;      //正規分布の σ
```

3.2 入出力ポート関連コードの変更

この例題では、自分自身でデータを生成するためデータ入力ポートは使用しません。また、データ出力ポートも使用しないため Skeleton.h の 49 行目〜53 行目をコメントアウトします。

```
private:
    //TimedOctetSeq m_in_data;
    //InPort<TimedOctetSeq, MyRingBuffer> m_InPort;

    //TimedOctetSeq m_out_data;
    //OutPort<TimedOctetSeq, MyRingBuffer> m_OutPort;
```

Skeleton.cpp のコンストラクタに次のような変更を加えます。入出力ポート関連のコードをコメントアウトします。

```
Skeleton::Skeleton(RTC::Manager* manager)
    : DAQMW::MlfComponent(manager),
      ///m_InPort("skeleton_in",   m_in_data),
      ///m_OutPort("skeleton_out", m_out_data),

      ///m_in_status(BUF_SUCCESS),
      ///m_out_status(BUF_SUCCESS),

      m_rand(0),           //乱数の初期化
      m_canvas(0),         //キャンバスの初期化
      m_histo(0),          //ヒストグラムの初期化
      m_bin(50),           //ビン数の初期化
      m_min(0.0),          //ヒストグラム最小値の初期化
      m_max(200.0),        //ヒストグラム最大値の初期化
      m_mean(100.0),       //正規分布平均値の初期化
      m_sigma(20.0),       //正規分布 σ の初期化

      m_debug(false)
{
    // Registration: InPort/OutPort/Service

    // Set InPort buffers
    ///registerInPort ("skeleton_in",   m_InPort);
    ///registerOutPort("skeleton_out", m_OutPort);
```

```
    init_command_port();
    init_state_table( );
    set_comp_name(COMP_NONAME);
}
```

3.3 SkeletonComp.cpp の変更

SkeletonComp.cpp の 77 行目に次の ROOT 関連の 1 行を追加します。

```
int main (int argc, char** argv)
{
    RTC::Manager* manager;
    manager = RTC::Manager::init(argc, argv);

    TApplication theApp("App", &argc, argv);     //ROOT GUI 用を追加

    // Initialize manager
    manager->init(argc, argv);
    ...
```

3.4 Makefile.Skeleton の編集

下記の行を Makefile.Skeleton の 23 行目に加える

```
ROOTLIBS = ‘$(ROOTSYS)/bin/root-config --glibs‘
ROOTINC = -I ‘$(ROOTSYS)/bin/root-config --incdir‘
```

下記のように $(ROOTINC) を CXXFLAGS に加える

```
CXXFLAGS = ‘/usr/bin/rtm-config --cflags‘ $(ROOTINC)
```

下記のように $(ROOTLIBS) を加える

```
SkeletonComp: SkeletonComp.o $(OBJS)
        $(CXX) -o $@ $(OBJS) SkeletonComp.o $(LDFLAGS) $(ROOTLIBS)
        cp $@ $(BINDIR)/$@
```

3.5 ビルド

これまでの一連の変更後、ビルドを行う。

```
$ cd your_directory/DaqComponents
$ make -f Makefile.Skeleton
```

エラーが発生した場合は、原因を調べて修正してください。

3.6 動作確認

ビルドが成功したら、次の手順でコンポーネントを起動してコマンドを送って動作を確認します。

3.6.1 コンフィグレーション・ファイル

テスト用に次のコンフィグレーション・ファイルを使用します。テストに使用する PC の IP アドレスが 192.168.91.130 の場合です。各自の環境に合わせて変更してください。

```
<?xml version="1.0"?>
<configInfo>
   <daqOperator>
      <hostAddr>192.168.91.130</hostAddr>
   </daqOperator>
   <daqGroups>
      <daqGroup gid="group0">
         <components>
            <component cid="Skeleton0">
               <hostAddr>192.168.91.130</hostAddr>
               <hostPort>50000</hostPort>
               <instName>Skeleton0.rtc</instName>
               <execPath>/home/daq/bin/SkeletonComp</execPath>
               <confFile>/home/daq/rtc.conf</confFile>
               <startOrd>1</startOrd>
               <inPorts>
               </inPorts>
               <outPorts>
               </outPorts>
               <params>
               </params>
            </component>
         </components>
      </daqGroup>
   </daqGroups>
</configInfo>
```

上記 XML ドキュメントを例えば（名前は任意）config-skeleton.xml という名前で config.xml があるディレクトリに保存します。その後 config.xml へシンボリックを張ります。

```
$ rm config.xml
$ ln -s config-skeleton.xml config.xml
```

3.6.2 起動スクリプト

テスト用に次の起動スクリプトを使用します。hostaddr は各自の環境に合わせて変更してください。

```
#! /bin/bash

hostaddr='192.168.91.130'
nsport='9876'
. setup.sh

#kill old omniORB name-servers and Components
pkill omniNames
pkill Comp

#start new name-server on port nsport
rtm-naming $nsport

sleep 8
echo 'corba.nameservers: '$hostaddr':'$nsport > ./rtc.conf
echo "naming.formats: ${hostaddr}.host_cxt/%n.rtc" >> ./rtc.conf
echo 'logger.enable: NO' >> ./rtc.conf
echo 'exec_cxt.periodic.rate: 1000000000' >> ./rtc.conf

#start Skeleton Component
xterm -geometry 40x10+0+500 -T Skeleton -e "bin/SkeletonComp -f rtc.conf;bash"&
```

```
##start DAQ Operator
sleep 8
bin/DaqOperatorComp -x ./config.xml -h $hostaddr -p $nsport -c
```

上記スクリプトを例えば（名前は任意） run-xterm.sh という名前で保存します。ターミナルを立ち上げて、スクリプトを実行して動作を確認します。

```
$ sh run-xterm.sh
```

上記スクリプト実行後、図 1 のようなウインドウが現れます。左側のターミナルは、Skeleton コンポーネントが起動されたものです。右側のターミナルは、スクリプトを実行したもので最終的に DAQ-Operator と呼ばれるコントローラが起動します。図の状態では、Skeleton コンポーネントが起動しており、現在のステートは LOADED です。configure コマンドを受け付けることができ、その後 CONFIGURED 状態へ遷移します。画面上のコマンドに対応した番号を入力してリターンキーを押すことで、DAQ-Operator から Skeleton コンポーネントへコマンドを送り、状態を遷移させることができることを確認してください。

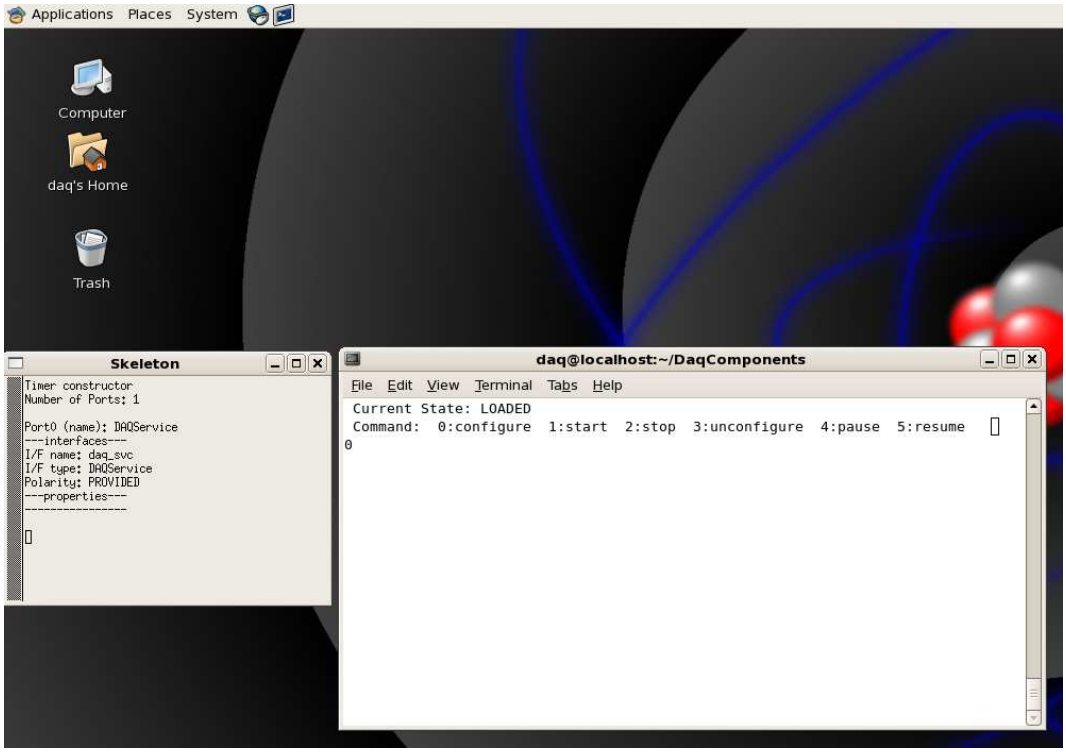

図 1:

3.7 ヒストグラムの実装

ヒストグラムとそれを表示する Canvas はラン毎に初期化を行うことにします。Skeleton.cpp の Skeleton::daq_start() に次のコードを追加します。

```
if (m_rand) {
  delete m_rand;
  m_rand = 0;
}
m_rand = new TRandom(); // TRandom を生成

if (m_canvas) {
  delete m_canvas;
  m_canvas = 0;
}
m_canvas =  new TCanvas("c1", "canvas", 10, 10, 300, 320); // TCanvas 生成

if (m_histo) {
  delete m_histo;
  m_histo = 0;
}
m_histo = new TH1F("histo", "histo", m_bin, m_min, m_max); // 1D Histo 生成
```

3.8 データの生成、フィリング、アップデート

ROOT の TRandom クラスを使用して、Gauss 分布データを生成します。Skeleton.cpp の Skeleton::daq_run() に次のコードを追加します。これで、平均値が 100.0、σ が 20.0 の正規分布データが得られます。

```
double dat = m_rand->Gaus(m_mean, m_sigma);  // 平均値 m_mean, σ m_sigma の正規分布データ生成
m_histo->Fill(dat);                          // ヒストグラムへフィル
m_histo->Draw();                             // ヒストグラムを描く
m_canvas->Update();                          // Canvas をアップデートする
m_total_event++;
```

これまでの一連の変更後、コンパイルをします。

```
$ cd your\_directory/DaqComponents
$ make -f Makefile.Skeleton
```

エラーが発生した場合は、原因を調べて修正します。実行時のスクリーンショットを図 2 に示します。

3.9 ヒストグラム関連パラメータの変更

ここまでは、発生する正規分布データの平均値や σ は固定値で、ヒストグラムのビンや最小、最大値も固定値でした。DAQ ミドルウェアには、XML で記述されたコンディション・ファイルによりランのスタート時に装置やオンライン解析用のパラメータを更新する機構があります。詳細は、コンディションデータベースの開発マニュアル [5] を参照してください。これによりラン毎にパラメータを変更することができます。ラン中はコンディション・ファイルによりパラメータを変更することはできません。

4 さいごに

Skeleton コンポーネントを変更して、ROOT を用いたオンライン・モニタの開発の手順を説明しました。非常に簡単な 1 次元ヒストグラムの実装を行いましたが、これを元により複雑なオンラインモニタが実装できると考えています。ヒストグラムの数を多くする、その他より進んだ ROOT を利用した解析は ROOT のホームページ [1] 等を参考にしてください。

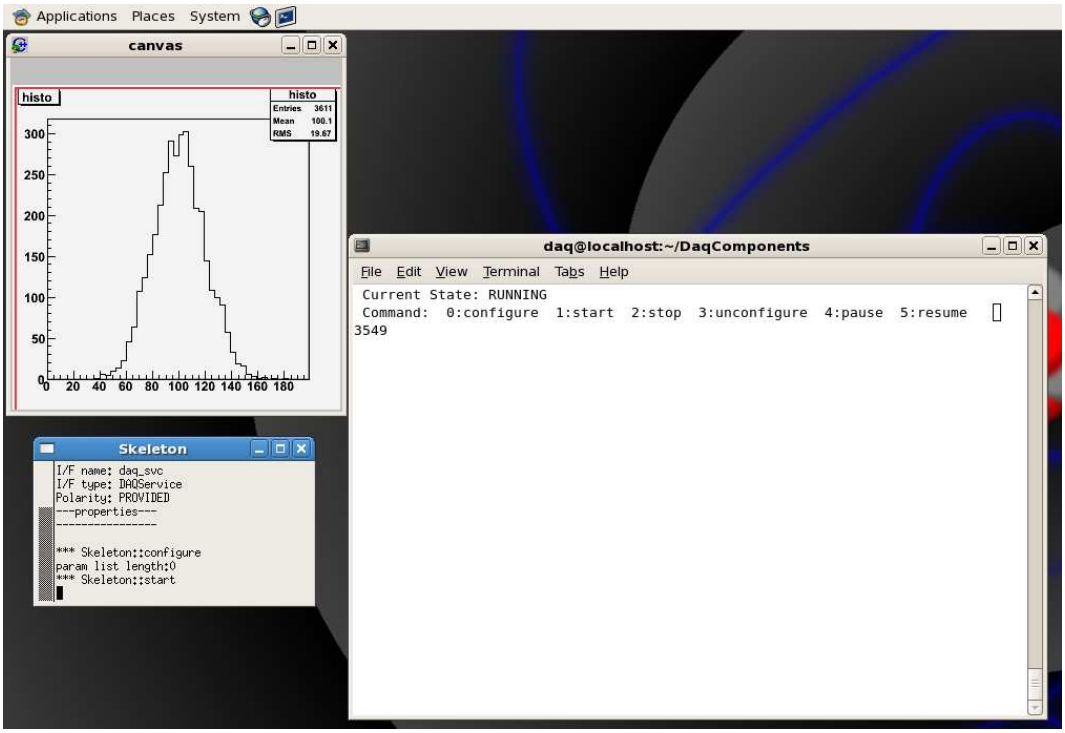

図 2:

参考文献

[1] ROOT のホームページ参照
http://root.cern.ch/

[2] 千代浩司、モニターコンポーネント開発マニュアル http://greentea.kek.jp/daqm/seminar2009/

[3] http://www-jlc.kek.jp/~sendai/OpenRTM/EL5/tars.2009.07/

[4] 千代浩司、仲吉一男、安 芳次、MLF 中性子用 DAQ ミドルウェアインストールおよび操作マニュアル、2009. http://www-jlc.kek.jp/~sendai/OpenRTM/EL5/tars.2009.07/DAQMM.pdf

[5] 安 芳次、コンディションデータベースの開発マニュアル、2009 年 6 月 http://www-jlc.kek.jp/~sendai/OpenRTM/EL5/tars.2009.07/