A Programming Model for Multithreading with Real-time Allocations of VC++ Controls

Gao-Wei Chang*

Department of Mechatronic Engineering, National Taiwan Normal University,Taipei, Taiwan, ROC.

*Corresponding author: gwchang@ntnu.edu.tw

Abstract— **The goal of this paper is to propose a programming model for supporting the technique of multithreading that performs real-time allocation of VC++ controls. In our approach, the design of task-interfacing class (TIC) plays an essential role in meeting the requirement of real-time, spatially distributed processing. Thru constructing the class, its object integrates multiple tasks as the class members, in preparation for the work of multithreading, where each task performs with an individual thread. Our experiments exhibit that the TIC model performs satisfactory results for the Windows programming.**

***Keywords*—multithreading, real-time, VC++, controls.**

I. INTRODUCTION

Multithreading has been regarded as an important technique in Windows programming, to perform nearly parallel processing for multiple tasks. With the development of object-oriented programming (OOP), the design of delegate class in Microsoft Visual C++ assemblies or VC++/CLR (common language runtime) has offered a useful solution to realization of multithreading [1]-[3].

In this paper, besides the delegation, design of interfacing with multiple tasks is necessary for meeting the requirement of real-time, spatially distributed processing in Windows programming. To do this, a task-interfacing class (TIC) model is proposed to support the work of multithreading for real-time allocation of VC++ controls (or button arrays, BAs). Figure 1.1 shows a conceptual diagram for the TIC model that integrates the tasks for multithreading.

The proposed model integrates the UI (user-interface or form) and time-delay (TD) tasks for multithreading. To demonstrate its effectiveness, the projects with event-handling are presented for multithreading with message flows within BA, where the controls are arranged at specified locations.

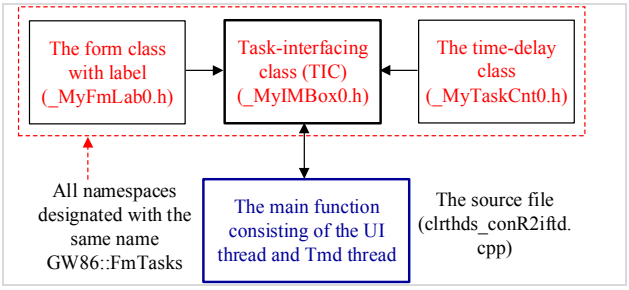


Fig. 1.1 A conceptual diagram for the TIC model that integrates multiple tasks for multithreading.

II. DESIGN OF TASK-INTERFACING CLASS

Figure 2.1(a) shows that the content of the source file clrthds_conR2iftd.cpp, where the header files for the UI (form) and time-delay tasks are included, as well as the namespaces required for the project are used. In Fig. 2.1(b), the main function consists of the threads in a while loop, for the same TIC object (or the instance pointed by the handle hnd).

```
//clrthds_conR2iftd0.cpp
//Gao-Wei Chang, 180501, 180927.
#pragma once
#include "c:\ ⋯ \_MyTaskCnt0.h"
#include "c:\ ⋯ \_MyFmLab0.h"
#include "c:\ ⋯ \_MyIMBox0.h"

using namespace System::Threading;
using namespace GW86::FmTasks;

void main() { ⋯ }
```

Header files at specified folders

For VC++/CLR and the user-defined assemblies

(a)

```
void main() {
  while(1) {
    Console::Write("Num. cnt(to cnt+99) : ");
    int cnt = Convert::ToInt32(Console::ReadLine());
    MyIMBox0^ hnd = gcnew MyIMBox0(cnt); //(t1.0)
    ThreadStart^ theThrdDlg1 = //(t1.1)
      gcnew ThreadStart(hnd, &MyIMBox0::show);
    Thread^ newThread1 =
      gcnew Thread(theThrdDlg1); //(t1.2)
    newThread1->Start(); //(t1.3)

    ThreadStart^ theThrdDlg2 = //(t2.1)
      gcnew ThreadStart(hnd, &MyIMBox0::tmd);
    Thread^ newThread2 = //(t2.2)
      gcnew Thread(theThrdDlg2);
    newThread2->Start(); //(t2.3)
  };
  Console::Write("Press any key."); Console::ReadKey();
}
```

UI or form thread

Time-delay thread

(b)

Fig. 2.1 (a) Illustration of the content of the source file clrthds_conR2iftd.cpp; (b) the implementation of the main function in that file.

Figure 2.2(a) shows the implementation of the TIC model, where the class constructor assigns the attribute cnt with its argument of the same variable name. Figures 2.2(b1) and (b2)

978-1-5386-7036-1/18/$31.00 ©2018 IEEE
DOI 10.1109/IS3C.2018.00017

illustrate the individual contents of the header files for the UI and TD tasks, respectively. In the TD task, there are configurations of for-loop with the pre-defined constants.

In Fig. 2.3, each number inputted to the end of a command line in the console initiates a child form, as well as starts and counts up with such an integer, say cnt, until the final number (cnt+99) is reached.

```
//_MyIMBox0.h
#pragma once
namespace GW86 {
namespace FmTasks {
public ref class MyIMBox0 {
public:
  int cnt; //(i0.1)
  MyFmLab0^ hnd; //(i0.2)
  MyIMBox0(int cnt) {//(i1.1)
    this->cnt=cnt; };
  void show() { //(i2.1)-(i2.2)
    hnd=gcnew MyFmLab0(this->cnt);
    Application::Run(hnd); };
  void tmd() {//(i3.x)
    for (int i=0; i<100; i++) {//(i3.0)
      MyTaskCnt0::tmd(5);//(i3.1)
      hnd->label1->Text = //(i3.2)
        Convert::ToString((this->cnt)+i); };
  };
}; } }
```

Constructor with argument cnt to set the attribute cnt

For the form thread

Upper limit

Time delay

For the time-delay thread

(a)

```
//_MyFmLab0.h
#pragma once
namespace GW86 {
namespace FmTasks {
using namespace System;
using namespace System::Windows::Forms;
using namespace System::Drawing;

public ref class MyFmLab0 : public Form
{ … };
} }
```

Using namespaces for VC++/CLR assemblies

Including a label with text to display a number

(b1)

```
//_MyTaskCnt0.h
#pragma once
namespace GW86 {
namespace FmTasks {
#define OUTER_CNT 1
#define INNER_CNT 100000000
public ref class MyTaskCnt0 {
  public:
  static void tmd(int cnt) { … };
};
} }
```

The constants for time delay.

There is for-loop with the pre-defined constants

(b2)

Fig. 2.2 (a) Implementation of the TIC model; individual illustrations for the contents of the header files (b1) _MyFmLab0.h and (b2) _MyTmdCnt.h that are included in the source file of the project clrthds_conR2iftd.

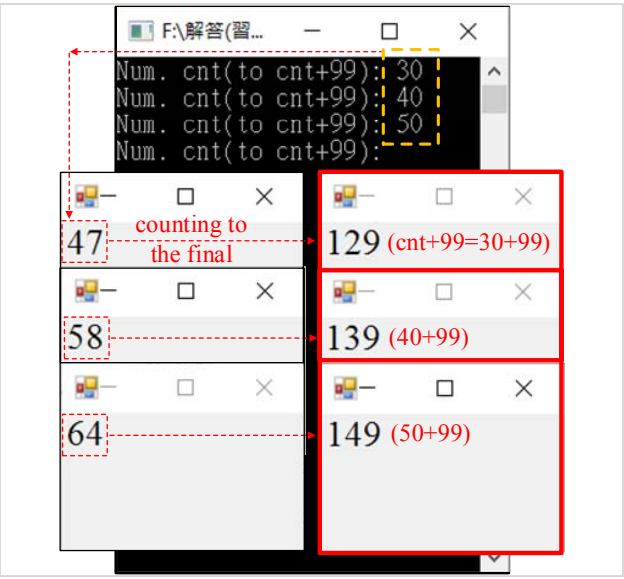


Fig. 2.3 Experimental results for the project clrthds_conR2iftd.

III. MULTITHREADING DUE TO HANDLING EVENTS

In Sec. II, the tasks with multithreading are directly initiated in the main function. This section starts them with a button-clicking event from a main form, which is an object of the user-defined form, say MyForm0. Such a form class is implemented and executed in the project fhthds_cnt2iiftd_ini0.

Figure 3.1(a) shows the content of the source file MyForm0.cpp in the project. The implementation of the header file MyForm0.h included in the source file is illustrated in Fig. 3.1(b).

```
//MyForm0.cpp
#include "MyForm0.h"
using namespace fhthds_cnt2iiftd_ini0;

void main() {
  Application::Run(gcnew MyForm0);
}
```

Main form

Dynamic mem. alloc.

(a)

```
//MyForm0.h
Including the headers and using namespaces
as the same as those in clrthds_conR2iftd.cpp
namespace fhthds_cnt2iiftd_ini0 {
public ref class MyForm0 : public Form {
  Constructor, members, and the delegate for
  handling events (embedded with button1_Click)
}
```

(b)

Fig. 3.1 In the project fhthds_cnt2iiftd_ini0, (a) code of the main function in the source file MyForm0.cpp; (b) illustration for the header file MyForm0.h that is included in the source file.

Figure 3.2(a) illustrates the user-defined function button1_Click of handling a button-clicking event in the main form of the project fhthds_cnt2iiftd_ini0, where the UI and TD threads are the same as those in the project clrthds_conR2iftd. The execution examples of this project are exhibited in Fig. 3.2(b).

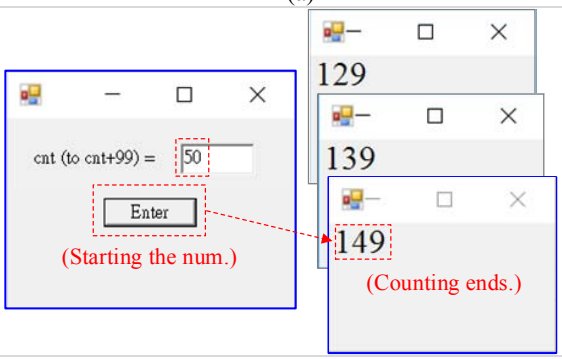
```
void button1_Click(Object^ s, EventArgs^ e) {
  int cnt, tot=100, td=5; //(b1.1)
  try {cnt = Convert::ToInt32(textBox1->Text);}
  catch (FormatException^ e) { //(b1.2)
    MessageBox::Show(e->Message); return; };
  MyIMBcm0^ hnd =
    gcnew MyIMBcm0(cnt, tot, td); //(t1.0)
}
```

Reading/converting data

Stats. (t1.1)-(t1.3) in the main func. of clrthds_conR2iftd.cpp — UI (dialog) thread

Stats. (t1.4)-(t1.6) in the same file as above. — Time delay thread

(a)



(b)

Fig. 3.2 (a) Illustration of the function button1_Click of in the main form of the project fhthds_cnt2iiftd_ini0; (b) execution examples of this project.

IV. MULTITHREADING FOR REAL-TIME ALLOCATION OF THE CONTROLS

In this section, we extend the concept of the TIC model to enhance the multithreading technique for spatially-distributed processing the controls (or BA) to meet the real-time requirements.

Figure 4.1(a) shows the content of the form container in the project fhmct_bAthdcs0. In Fig. 4.1(b), the content of the child-form class is illustrated for allocating the elements of a BA. Figure 4.2(a) depicts the configuration of the source file of the child form (MyDialog1.cpp). Figure 4.2(b) lists the code of the TIC model in the project fhmct_bAthdcs0. Figure 4.2(c1) illustrates the member function of MyDialog1 _allocBA that allocates and adds the element of BA to the main form, as well as evaluates and set up the coordinates of those elements. In Fig. 4.2(c2), the function button1_Click is analogous to that of the project fhthds_cnt2iiftd_ ini0 in Sec. 3, except for using BA in the project fhmct_bAthdcs0, instead of using forms.

```
//MyContainer0.h
//Gao-Wei Chang, 180520.
#pragma once
namespace fhmct_bAthdcs0 {
ref class MyForm0;
ref class MyDialog1;
public ref class MyContainer0 {
public:
  MyForm0^ theForm0;
  MyDialog1^ theDialog1;
};
}
```

Form container

(a)

```
//MyDialog1.h.
#pragma once
namespace fhmct_bAthdcs0 {…}
public ref class MyDialog1 : public Form {
public:
  MyContainer0^ theComponents0;
  MyDialog1(MyContainer0^ theComponents0) { … };
  …
  void button1_Click( … );
  private: void _allocBA(int cnt);
#define BTN_NO 20
#define MAX_VAL 99
  public: static array<Button^>^ btnArr = gcnew
    array<Button^>(BTN_NO);
}; }
```

Using namespaces of CLR form and drawing assemblies

Including the setting of form container

The members and their setting

Declaring and allocating BA

(b)

Fig. 4.1. (a) The content of the form container in the project fhmct_bAthdcs0; (b) illustration of the class MyDialog1 for allocating the elements of BA.

```
//MyDialog1.cpp
#include "MyForm0.h"
#include "c:\_dll86\_GW86_FmTasks\_MyTaskCnt0.h"
using namespace System::Threading;
using namespace fhmct_bAthdcs0;

namespace fhmct_bAthdcs0 {
public ref class MyIBtnA0 { … }; }

void MyDialog1::_allocBA(int cnt) { … }
void MyDialog1::button1_Click( … ){ … }
```

Analog to GW86:: FmTask:: MyIMBox0

(a)

```
namespace fhmct_bAthdcs0 {
public ref class MyIBtnA0 {
public:
  Button^ btn; int n;
  MyIBtnA0(Button^ btn, int n) {
  this->btn = btn; this->n=n; };
  void bTmd() {
    for (int i=this->n; i<=MAX_VAL; i++) {
      this->btn->Text=i.ToString();
      GW86::FmTasks::MyTaskCnt0::tmd(1); };
  };
}; }
```

Analog to GW86:: FmTask::MyIMBox0::tmd()

(b)

```
void MyDialog1:: _allocBA(int cnt) {
  btnArr[cnt] = gcnew Button;
  theComponents0->theForm0->
    Controls->Add(btnArr[cnt]);
  Eval. and setting the coordinate
  of each element of BA
}
```
Allocating and adding the element of BA to the main form

(c1)

```
void MyDialog1::button1_Click( ··· ) {
  static int cnt=0;
  if (cnt>=BTN_NO) {
    MessageBox::Show("No. of btns exceeds the limit");
    return; };
  allocBA(cnt);
  int start_num =Convert::ToInt32
    (theComponents0->theDialog1->textBox1->Text);
  MyIBtnA0^ hnd = gcnew
    MyIBtnA0(btnArr[cnt], start_num);
  ThreadStart^ theThrdDlg1 = gcnew
    ThreadStart(hnd, &MyIBtnA0::bTmd);
  Thread^ newThread = gcnew Thread(theThrdDlg1);
  newThread->Start();
  cnt++;
}
```
Allocating the specified element of BA and setting its location

Reading/converting data

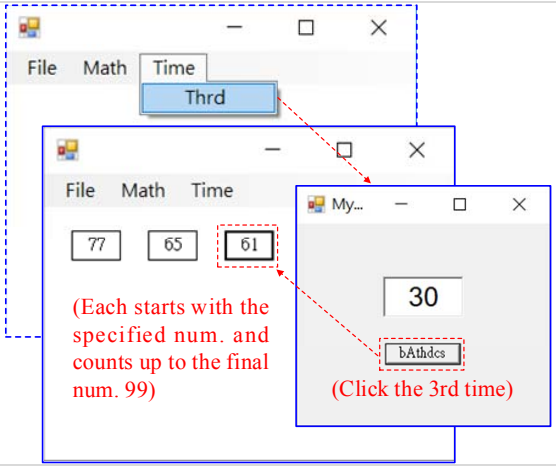
Time delay thread

(c2)

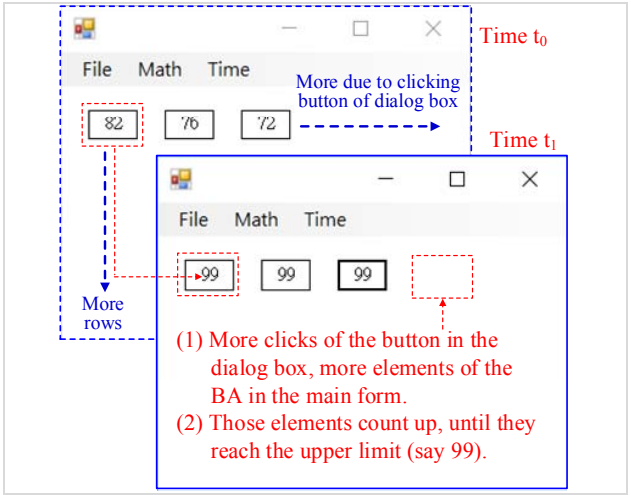
Fig. 4.2 Implementation and illustration for the source file of the child form (MyDialog1.cpp) in the project fhmct_bAthdcs0.

In the project fhmct_bAthdcs0, one can click the item Time>Thrd to generate the child form (MyDialog1), as shown in Fig. 4.3(a). To demonstrate the effectiveness of the TIC model, we may sequentially click the button of the dialog box, to allocate the elements of BA.

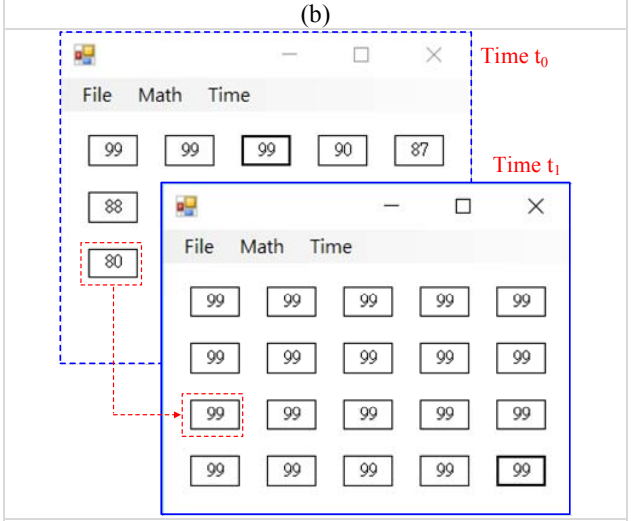
Specifically, Figs 4.3(b) and (c) respectively depict that more clicks of the button in the dialog box, more elements of the BA allocated in the main form. In the figures, those elements individually count up, until they reach the upper limit (say 99).



(a)



(b)



(c)

Fig. 4.3 The execution examples of the project fhmct_bAthdcs0 to allocate the elements of BA.

V. Conclusion

This paper has proposed a TIC model to support the multithreading with the capability of real-time, spatially distributed processing. The TIC object integrates the UI and TD tasks as the class members, for the work of multithreading, where each task performs with an individual thread. The experimental results show that the proposed model performs well for allocating the elements of BA in the Windows application projects.

References

[1] I Horton, *Ivor Horton's beginning Visual C++ 2010*, John Wiley and Sons, Inc, 2010.

[2] G. W. Chang, "A series of programming models for command-driven Windows platforms with VC++ Assemblies," *Proceedings of the 15th International Conference on Automation Technology (Automation 2017)*, no. 1066, pp. 1-6, Dec. 2017.

[3] Microsoft Developer Network. https://msdn.microsoft.com/en-us