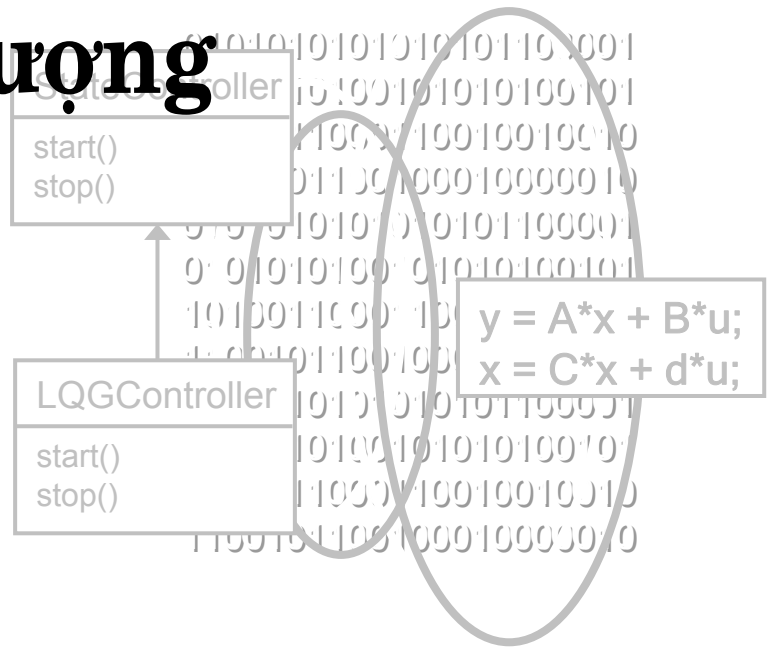
Kỹ thuật lập trình

Chương 8: Tiến tới tư duy lập trình hướng đối tượng



8/13/2007

Nội dung chương 8

8.1 Đặt vấn đề

8.2 Giới thiệu ví dụ chương trình mô phỏng

8.3 Tư duy "rất" cổ điển

8.4 Tư duy hướng hàm

8.5 Tư duy dựa trên đối tượng (object-based)

8.6 Tư duy thực sự hướng đối tượng

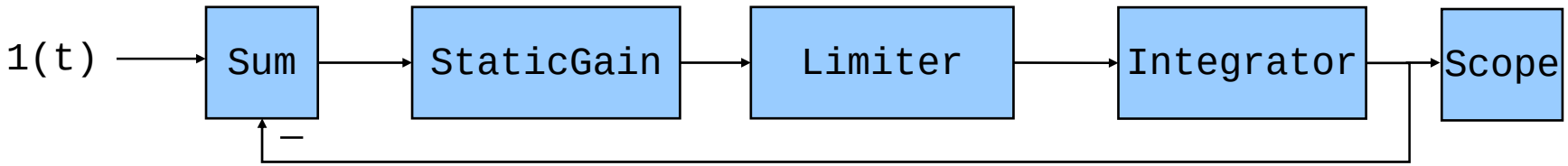
8.1 Đặt vấn đề

„Designing object-oriented software is hard, and designing reusable object-oriented software is even harder...It takes a long time for novices to learn what object-oriented design is all about. Exprienced designers evidently know something inexperienced ones don't...

One thing expert designers know *not* to do is solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past. When they find a good solution, they use it again and again. Such experience is part of what makes them experts. Consequently, you'll find recurring patterns of classes and communicating objects in many object-oriented systems. These patterns solve specific design problems and make object-oriented design more flexible, elegant, and ultimately reusable...“

Erich Gamma et. al.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

8.2 Phần mềm mô phỏng kiểu FBD



Nhiệm vụ:

Xây dựng phần mềm để hỗ trợ mô phỏng thời gian thực một cách linh hoạt, mềm dẻo, đáp ứng được các yêu cầu của từng bài toán cụ thể

Trước mắt chưa cần hỗ trợ tạo ứng dụng kiểu kéo thả bằng công cụ đồ họa

8.3 Tư duy rất cổ điển

```
// SimProg1.cpp
#include <iostream.h>
#include <conio.h>
#include <windows.h>
void main() {
    double K =1,I=0, Ti = 5;
    double Hi = 10, Lo = -10;
    double Ts = 0.5;
    double r =1, y=0, e, u, ub;
    cout << "u\ty";
    while (!kbhit()) {
        e = r-y;                    // Sum block
        u = K*e;                    // Static Gain
        ub = max(min(u,Hi),Lo); // Limiter
        I += ub*Ts/Ti;              // Integrator state
        y  = I;                     // Integrator output
        cout << '\n' << u << '\t' << y;
        cout.flush();
        Sleep(long(Ts*1000));
    }
}
```

Vấn đề?

- Phần mềm dưới dạng chương trình, không có giá trị sử dụng lại
- Rất khó thay đổi hoặc mở rộng theo yêu cầu cụ thể của từng bài toán
- Toàn bộ thuật toán được gói trong một chương trình => khó theo dõi, dễ gây lỗi, không bảo vệ được chất xám

8.4 Tư duy hướng hàm

```
// SimProg2.cpp
#include <iostream.h>
#include <conio.h>
#include <windows.h>
#include "SimFun.h"
void main() {
   double K  = 5.0, double Ti = 5.0;
   double Hi = 10, Lo = -10;
   double Ts = 0.5;
   double r =1, y=0, e, u, ub;
   cout << "u\ty";
   while (!kbhit()) {
     e = sum(r,-y);       // Sum block
     u = gain(K,e);       // Static Gain
     ub= limit(Hi,Lo,u);     // Limiter
     y = integrate(Ti,Ts,ub);    // Integrator output
     cout << '\n' << u << '\t' << y;
     cout.flush();
     Sleep(long(Ts*1000));
   }
}
```

```
// SimFun.h
inline double sum(double x1, double x2) { return x1 + x2; }
inline double gain(double K, double x)  { return K * x; }
double limit(double Hi, double Lo, double x);
double integrate(double Ti, double Ts, double x);
```

```
// SimFun.cpp
double limit(double Hi, double Lo, double x) {
       if (x > Hi) x = Hi;
       if (x < Lo) x = Lo;
       return x;
}

double integrate(double Ti, double Ts, double x) {
       static double I = 0;
       I += x*Ts/Ti;
       return I;
}
```

Vấn đề?

- Vẫn chưa đủ tính linh hoạt, mềm dẻo cần thiết
- Thay đổi, mở rộng chương trình mô phỏng rất khó khăn
- Các khâu có trạng thái như khâu tích phân, khâu trễ khó thực hiện một cách "sạch sẽ" (trạng thái lưu trữ dưới dạng nào?)
- Rất khó phát triển thành phần mềm có hỗ trợ đồ họa kiểu kéo thả

8.5 Tư duy dựa đối tượng

```
// SimClass.h
class Sum {
public:
    double operator()(double x1, double x2) {
        return x1 + x2;
    }
};
class Gain {
    double K;
public:
    Gain(double k = 1) : K(k) {}
    double operator()(double x){ return K * x; }
};
class Limiter {
    double Hi, Lo;
public:
    Limiter(double h=10.0, double l= -10.0);
    double operator()(double x);
};
```

```
class Integrator {
   double Ki, Ts;
   double I;
public:
   Integrator(double ti = 1.0, double ts = 0.5);
   double operator()(double x);
};

class Delay {
   double* bufPtr;
   int      bufSize;
   double  Td, Ts;
public:
   Delay(double td = 0, double ts = 1);
   Delay(const Delay&);
   Delay& operator=(Delay&);
   ~Delay();
   double operator()(double x);
private:
   void createBuffer(int sz);
};
```

```
#include <math.h>
#include "SimClass.h"

Limiter::Limiter(double h, double l) : Hi(h), Lo(l) {
   if (Hi < Lo) Hi = Lo;
}
double Limiter::operator()(double x) {
   if (x > Hi) x = Hi;
   if (x < Lo) x = Lo;
   return x;
}
Integrator::Integrator(double ti, double ts)
   : Ts(1), Ki(1), I(0) {
   if (ts > 0)
      Ts = ts;
   if (ti > 0)
      Ki = ts/ti;
}
double Integrator::operator()(double x) {
    I += x*Ki;
    return I;
}
```

```
Delay::Delay(double td, double ts) : Td(td), Ts(ts) {
    if (Td < 0) Td = 0;
       if (Ts < 0) Ts = 1;
       createBuffer((int)ceil(Td/Ts));
}

double Delay::operator()(double x) {
       if (bufSize > 0) {
              double y = bufPtr[0];
              for (int i=0; i < bufSize-1; ++i)
                     bufPtr[i] = bufPtr[i+1];
              bufPtr[bufSize-1] = x;
              return y;
       }
       return x;
}
void Delay::createBuffer(int sz) {
       bufSize = sz;
       bufPtr =  new double[bufSize];
       for (int i=0; i < bufSize; ++i)
              bufPtr[i] = 0.0;
}
...
```

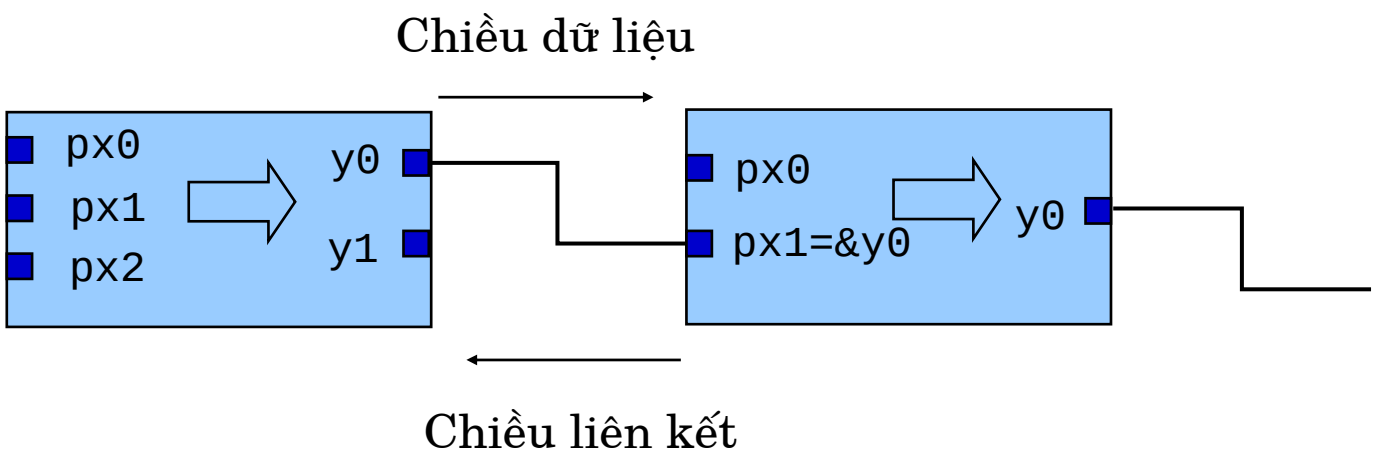
```
// SimProg3.cpp
#include <iostream.h>
#include <conio.h>
#include <windows.h>
#include "SimClass.h"

void main() {
    double Ts = 0.5;
    Sum sum;
    Gain gain(2.0);
    Limiter limit(10,-10);
    Integrator integrate(5,Ts);
    Delay delay(1.0);
    double r =1, y=0, e, u, ub;
    cout << "u\ty";
    while (!kbhit()) {
        e = sum(r,-y);     // Sum block
        u = gain(e);       // Static Gain
        ub= limit(u);      // Limiter
        y = integrate(ub);// Integrator output
        y = delay(y);
        cout << '\n' << u << '\t' << y;
        cout.flush();
        Sleep(long(Ts*1000));
    }
}
```

Vấn đề?

- Khi số lượng các khối lớn lên thì quản lý thế nào?
- Khi quan hệ giữa các khối phức tạp hơn (nhiều vào, nhiều ra) thì tổ chức quan hệ giữa các đối tượng như thế nào?
- Làm thế nào để tạo và quản lý các đối tượng một cách động (trong lúc chương trình đang chạy)?
- Lập trình dựa đối tượng mới mang lại ưu điểm về mặt an toàn, tin cậy, nhưng chưa mang lại ưu điểm về tính linh hoạt cần thiết của phần mềm => giá trị sử dụng lại chưa cao.

8.6 Tư duy hướng đối tượng

```
class FB {
public:
   virtual void execute() = 0;
private:
   virtual double* getOutputPort(int i=0) = 0;
   virtual void setInputPort(double* pFromOutputPort,
                             int i=0)= 0;
   friend class FBD;
};
```



```
class Sum : public FB {
public:
     Sum(bool plus_sign1 = true, bool plus_sign2 = false);
     void execute();
private:
     bool   sign[2];
     double *px[2];
     double y;
     double* getOutputPort(int i=0);
     void setInputPort(double* pFromOutputPort, int i=0);
};
Sum::Sum(bool plus_sign1, bool plus_sign2): y(0) {
     px[0] = px[1] = 0;
     sign[0] = plus_sign1;
     sign[1] = plus_sign2;
}
void Sum::execute() {
     if (px[0] != 0) y = sign[0] ? *(px[0]) : - *(px[0]);
     if (px[1] != 0) y += sign[1] ? *(px[1]) : - *(px[1]);
}
double* Sum::getOutputPort(int) {
     return &y;
}
void Sum::setInputPort(double* pFromOutputPort, int i) {
     if(i < 2)
          px[i] = pFromOutputPort;
}
```

```
class Limiter: public FB {
public:
   Limiter(double h=10.0, double l = -10.0);
   void execute();
private:
   double Hi, Lo;
   double *px;
   double  y;
   double* getOutputPort(int i=0);
   void setInputPort(double* pFromOutputPort, int i=0);
};
Limiter::Limiter(double h, double l) : Hi(h), Lo(l), y(0), px(0) {
if (Hi < Lo)   Hi = Lo; }
void Limiter::execute() {
   if (px != 0) {
      y = *px;
      if (y > Hi) y = Hi;
      if (y < Lo) y = Lo;
   }
}
double* Limiter::getOutputPort(int) {
   return &y;
}
void Limiter::setInputPort(double* pFromOutputPort, int i) {
   px = pFromOutputPort;
}
```

```
#include <vector>
#include <windows.h>
class FBD : public std::vector<FB*> {
   double Ts;
   bool stopped;
public:
   FBD(double ts = 0.5): Ts (ts > 0? ts : 1), stopped(true) {}
   void addFB(FB* p) { push_back(p); }
   void connect(int i1, int i2, int oport=0, int iport = 0) {
      FB *fb1= at(i1), *fb2= at(i2);
      fb2->setInputPort(fb1->getOutputPort(oport),iport);
   }
   void start();
   ~FBD();
};
FBD::~FBD() {
   for (int i=0; i < size(); ++i)
      delete at(i);
}
void FBD::start() {
   while(!kbhit()) {
      for (int i=0; i < size(); ++i)
         at(i)->execute();
      Sleep(long(Ts*1000));
   }
}
```

```
#include <iostream>
#include "SimFB.h"
void main() {
   double Ts=0.5;
   FBD fbd(0.5);
   fbd.addFB(new Step(1.0));       // 0
   fbd.addFB(new Sum);             // 1
   fbd.addFB(new Gain(5.0));       // 2
   fbd.addFB(new Limiter(10,-10));   // 3
   fbd.addFB(new Integrator(5,Ts));  // 4
   fbd.addFB(new Delay(0.0, Ts));    // 5
   fbd.addFB(new Scope(std::cout));  // 6

   for(int i=0; i < fbd.size()-1; ++i)
      fbd.connect(i,i+1);
   fbd.connect(5,1,0,1);
   fbd.connect(3,6,0,1);

   std::cout << "y\tu";
   fbd.start();

}
```

Bài tập về nhà

- Luyện tập lại trên máy tính các ví dụ từ phần 8.3 – 8.5
- Dựa trên các ví dụ lớp đã xây dựng ở phần 8.6 (Limiter, Sum), bổ sung các lớp còn lại (Step, Scope, Gain, Integrator, Delay)
- Chạy thử lại chương trình ở phần 8.6 sau khi đã hoàn thiện các lớp cần thiết.
- Bổ sung lớp Pulse để mô phỏng tác động của nhiễu quá trình (dạng xung vuông biên độ nhỏ, chu kỳ đặt được). Mở rộng chương trình mô phỏng như minh họa trên hình vẽ.