Hệ thống điều khiển phân tán

Chương 8: Công nghệ hướng đối tượng trong điều khiển phân tán

13.09.06

Chương 8: Công nghệ đối tượng trong điều khiển phân tán

8.1 Qui trình công nghệ phần mềm

8.2 Công nghệ đối tượng là gì

8.3 Ngôn ngữ mô hình hóa thống nhất UML

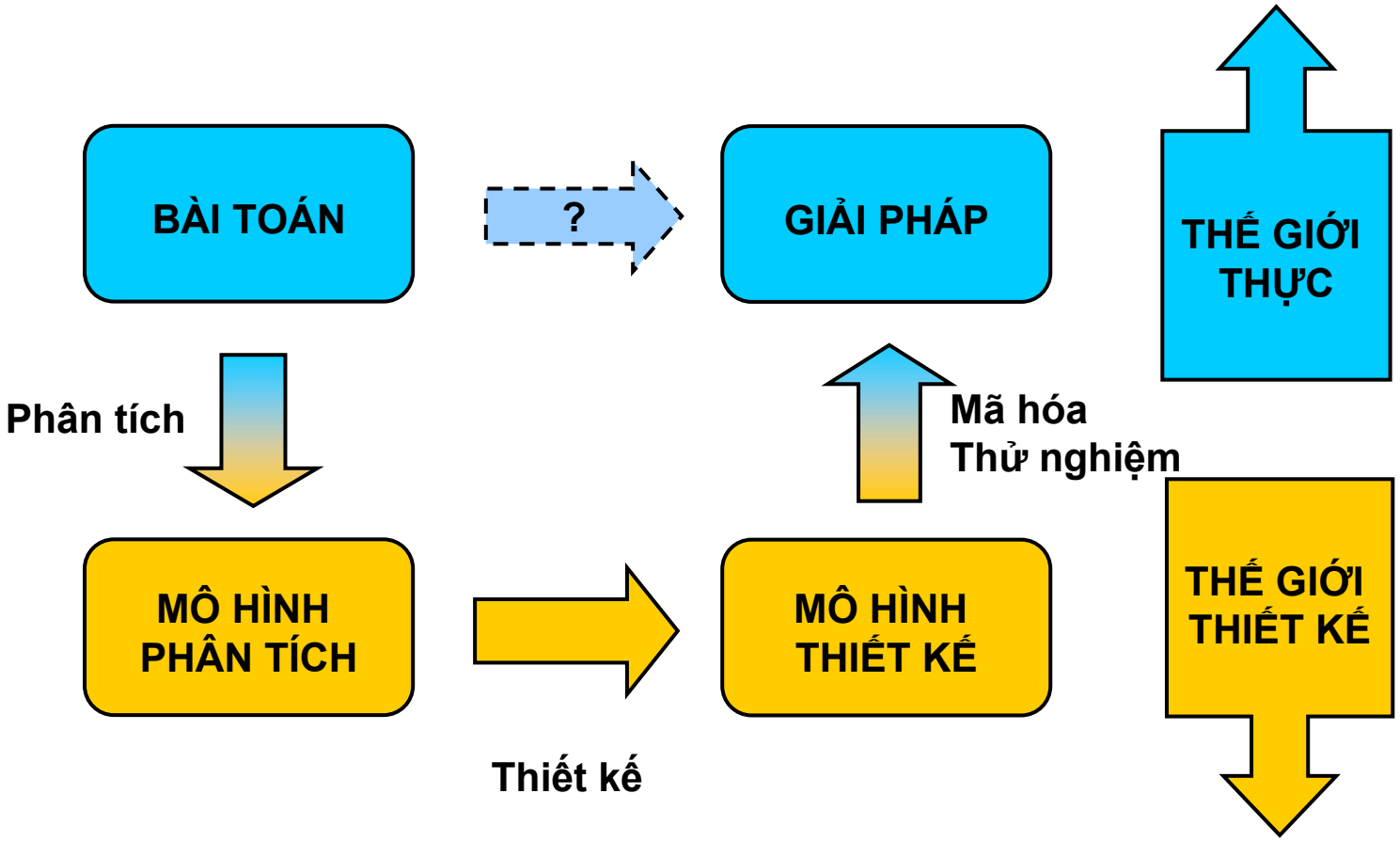
8.4 Khái niệm đối tượng phân tán

8.5 Mô hình COM/DCOM

8.6 Lập trình với COM/DCOM

Tài liệu: *Tự động hóa ngày nay* 5/04-8/04 (CD: \papers\ )
UML Reference Manual (CD:\UML\refman.pdf)
Dr. GUI on COM and ATL (CD: \com-dcom\)

8.1 Qui trình công nghệ phần mềm



Phân tích yêu cầu (Requirement analysis)

- Bởi vì: Khách hàng thường không biết là họ muốn gì, nhưng họ biết chắc chắn là họ không muốn gì
- Cho nên: Cần phải cùng với khách hàng làm rõ những yêu cầu về phạm chức năng, về giao diện sử dụng
- Kết quả: Mô hình đặc tả (*Specification Model*), một phần của hợp đồng
- Cần một ngôn ngữ mô hình hóa dễ hiểu để trao đổi giữa khách hàng và nhóm phân tích

⇨ Trả lời câu hỏi: **Khách hàng cần những gì**

Phân tích hệ thống (System analysis)

- Phân tích mối liên hệ của hệ thống với môi trường xung quanh
- Tìm ra cấu trúc hệ thống và các thành phần quan trọng
- Định nghĩa chức năng cụ thể của các thành phần
- Nhận biết các đặc điểm của từng thành phần
- Phân loại các thành phần, tổng quát hóa, đặc biệt hóa
- Nhận biết mối liên hệ giữa các thành phần
- Kết quả: Mô hình hệ thống (*System model*)
- Cần một ngôn ngữ mô hình hóa để trao đổi giữa các thành viên trong nhóm phân tích và với nhóm thiết kế

⇨ Trả lời câu hỏi: **Những gì sẽ phải làm**

Thiết kế hệ thống (System Design)

- Dựa trên mô hình hệ thống, xây dựng các mô hình chi tiết phục vụ sẵn sàng mã hóa/cài đặt
- Bao gồm:
  - Thiết kế cấu trúc (*structured design*): chương trình, kiểu dữ liệu, đối tượng, quan hệ cấu trúc giữa các đối tượng và kiểu)
  - Thiết kế tương tác (*interaction design*): quan hệ tương tác giữa các đối tượng
  - Thiết kế hành vi (*behaviour design*): sự kiện, trạng thái, phép toán, phản ứng
  - Thiết kế chức năng (*funtional design*): tiến trình hành động, hàm, thủ tục)
- Kết quả: Mô hình thiết kế (các bản vẽ và lời văn mô tả)

⇨ Trả lời câu hỏi: **Làm như thế nào**

Các bước khác

- Mã hóa/cài đặt (*Coding/Implementation*): Thể hiện mô hình thiết kế bằng một ngôn ngữ/công cụ lập trình cụ thể
- Thử nghiệm (*Testing, Verification*): Chạy thử, phân tích và kiểm chứng:
  - Thử đơn vị (*Unit Test*)
  - Thử tích hợp (*Integration Test*)
- Gỡ rối (*Debugging*): Tìm ra và sửa các lỗi chương trình chạy (các lỗi logic)
- Xây dựng tài liệu (*Documenting*): Xây dựng tài liệu phát triển, tài liệu hướng dẫn sử dụng
- Đào tạo, chuyển giao
- Bảo trì, bảo dưỡng

8.2 Công nghệ (hướng) đối tượng là gì?

Các nội dung của công nghệ phần mềm, được xây dựng trên cơ sở phương pháp luận hướng đối tượng

- Mô hình hóa hướng đối tượng
- Phân tích, thiết kế hướng đối tượng
- Lập trình hướng đối tượng
- Phần mềm thành phần
- Đối tượng phân tán
- ...

Công nghệ hướng đối tượng có vai trò then chốt trong công nghiệp phần mềm hiện nay và trong tương lai

Đối tượng là gì?

- Mô hình/đại diện của một đối tượng vật lý:
  - Tank, Heater, Furnace
  - Motor, Pump, Valve
  - Sensor, Thermometer, Flowmeter
  - Control Loop, Control System
- Hoặc một đối tượng logic ("conceptual object):
  - Trend, Report, Button, Window
  - Matrix, Vector, Polynomial
- Đóng gói dữ liệu + phép toán áp dụng

Một đối tượng có...

- Các thuộc tính (attributes)
- Trạng thái (state)
  - Dữ liệu
  - Quan hệ
- Hành vi (behavior)
  - Các phép toán
  - Đặc tính phản ứng
- Căn cước (identity)
- Ngữ nghĩa/trách nhiệm (semantic/responsibilities)

Nguyên lý cơ bản của phương pháp luận hướng đối tượng

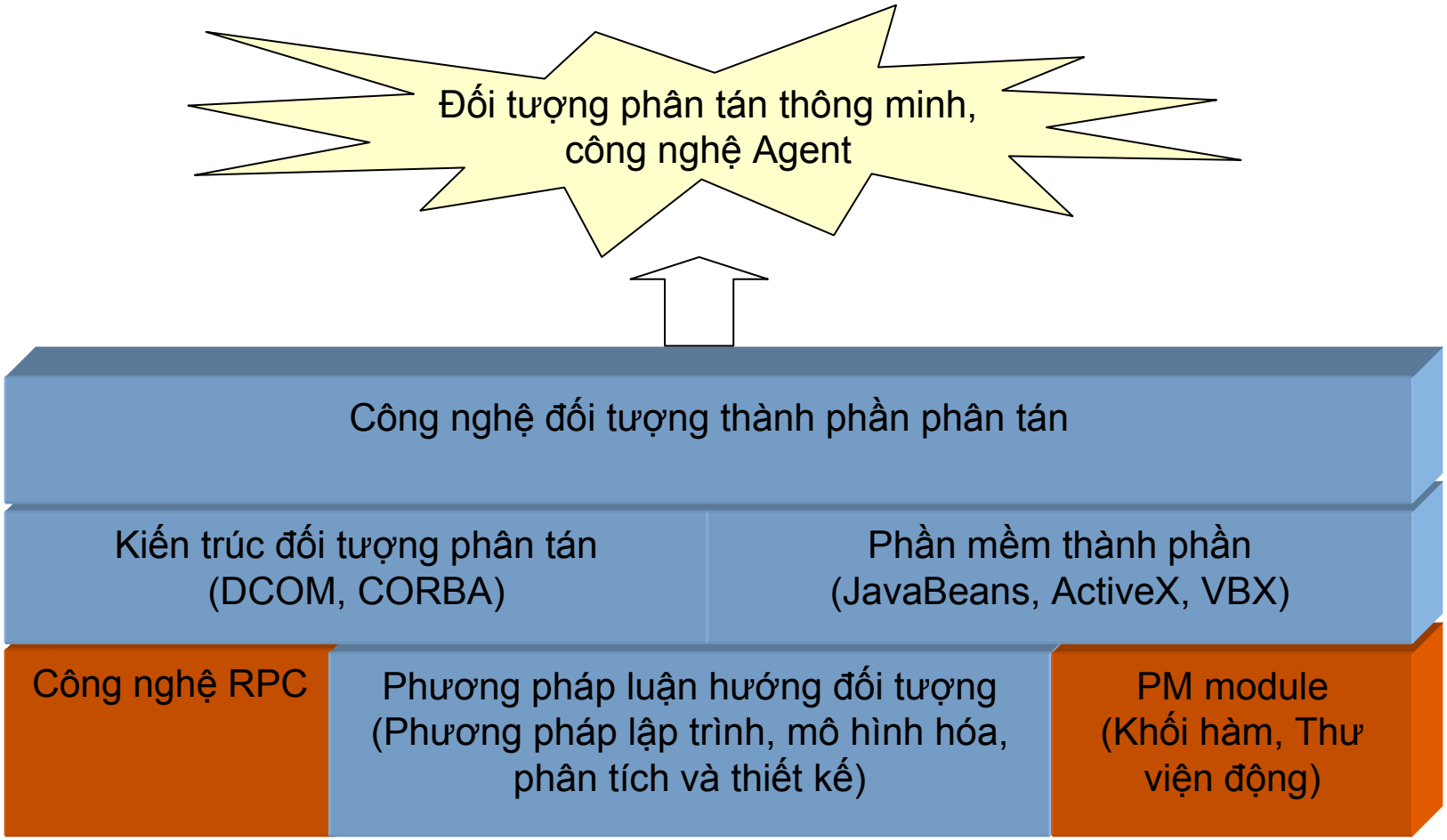
- Trừu tượng hóa (*abstraction*): giúp đơn giản hóa vấn đề, dễ sử dụng lại
- Đóng gói dữ liệu/che dấu thông tin (*data encapsulation/ information hiding*): nâng cao giá trị sử dụng lại và độ tin cậy của phần mềm
- Dẫn xuất/thừa kế (*subtyping/inheritance*): giúp dễ sử dụng lại mã phần mềm và thiết kế
- Đa hình/đa xạ (*polymorphism*): giúp phản ánh trung thực thế giới thực và nâng cao tính linh hoạt của phần mềm

Tại sao lại "hướng đối tượng"

*Phương pháp luận hướng đối tượng cho phép tư duy ở mức **trừu tượng cao** nhưng **gần với thế giới thực***

- Thế giới thực cấu thành bởi các đối tượng và mỗi liên hệ giữa chúng
- Mô hình nhất quán cho toàn bộ qui trình công nghệ phần mềm
- Trừu tượng hóa vấn đề tốt hơn
- Bền vững hơn với thay đổi
- Khả năng sử dụng lại cao
- Khả năng phù hợp với nhiều qui mô khác nhau
- Hỗ trợ tốt hơn cho phát triển các hệ tin cậy và an toàn
- Hỗ trợ tốt hơn cho xử lý cạnh tranh

Sự tiến hóa của công nghệ đối tượng



Vai trò của công nghệ đối tượng trong các hệ thống điều khiển?

- Vai trò của công nghệ phần mềm trong các hệ thống điều khiển?
- Có một công cụ phần mềm nào trong hệ thống điều khiển không được lập trình hướng đối tượng?
- Ví dụ về các đối tượng cụ thể:
  - Các khối chức năng: PID, AI, AO,...
  - Các khối đồ họa Windows Controls, ActiveX-Controls: Đồ thị, phím bấm, cửa sổ, bình chứa, van điều khiển, băng tải,...
  - OPC server, Web server,...

8.3 Ngôn ngữ mô hình hóa UML

Mô hình là gì?

- Một ánh xạ thế giới thực (đang tồn tại hoặc cần xây dựng)
- Mô tả thế giới thực từ một góc nhìn
- Các dạng mô hình:
  - Mô hình toán học
  - Mô hình đồ họa
  - Mô hình máy tính
- Một mô hình tốt cần đơn giản nhưng thể hiện được các đặc tính quan trọng cần quan tâm của thế giới thực
- *"Không có mô hình nào chính xác, nhưng có một số mô hình có ích!"*

Mô hình để làm gì?

- Trừu tượng hóa (đơn giản hóa) vấn đề
- Phương tiện giao tiếp trong nhóm phát triển
- Phương tiện giao tiếp giữa nhóm phát triển và khách hàng
- Phương tiện phân tích, thiết kế và kiểm chứng
- Tài liệu phần mềm

➲ Cần một ngôn ngữ mô hình hóa tốt và một phương pháp mô hình hóa thích hợp !

Thế nào là một ngôn ngữ mô hình hóa tốt

- Đơn giản, trực quan, dễ hiểu, dễ xây dựng (đồ họa)
- Khả năng biểu diễn mạnh (toán, văn bản, đồ họa)
- Khả năng thực thi (máy tính, văn bản, đồ họa máy tính)
- Linh hoạt, khả mở
- Nhất quán: cho suốt qui trình công nghệ phần mềm
- Chuẩn hóa quốc tế

UML: Unified Modeling Language

- Ngôn ngữ mô hình hóa rất mạnh, có đầy đủ các đặc tính tốt đã nêu
- Hỗ trợ mô hình hóa hướng đối tượng, hướng thành phần và các phương pháp luận khác
- Thống nhất Rumbaugh‘s *OMT*, *Booch‘94* và Ivar Jacobson‘s *Use Case*
- Chắt lọc, thừa kế nhiều phương pháp luận khác
- Ngôn ngữ mô hình hóa trung lập
- Kết hợp biểu tượng đồ họa + văn bản
- Chuẩn công nghiệp (OMG consortium: www.omg.org), đặc tả hiện tại V1.5

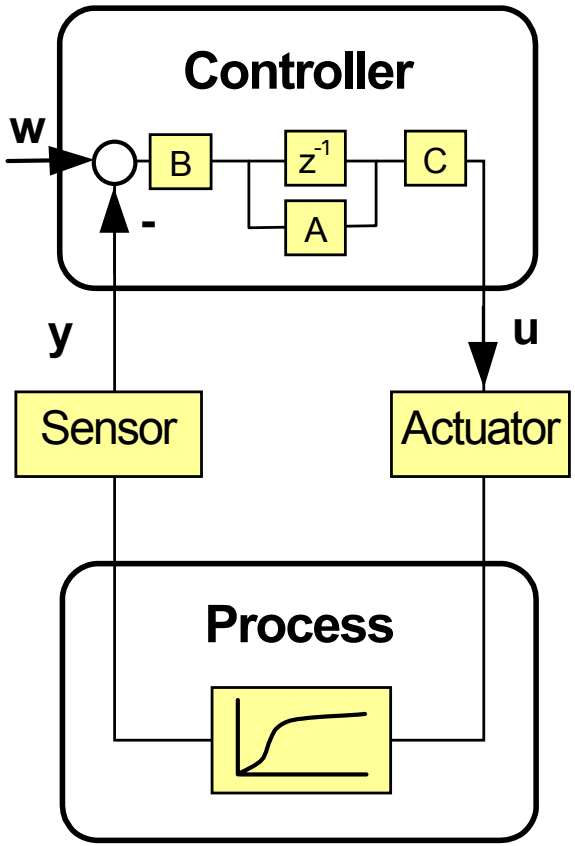
Mô hình hóa cấu trúc

- Static view
  - ⇨ Biểu đồ lớp: class, interface, inheritance, association, ...
- Use case view
  - ⇨ Biểu đồ use case: use case, scenario, ...
- Implementation view
  - ⇨ Biểu đồ thành phần: component, package, module, ...
- Deployment view
  - ⇨ Biểu đồ phân bố: node, processor, component,...

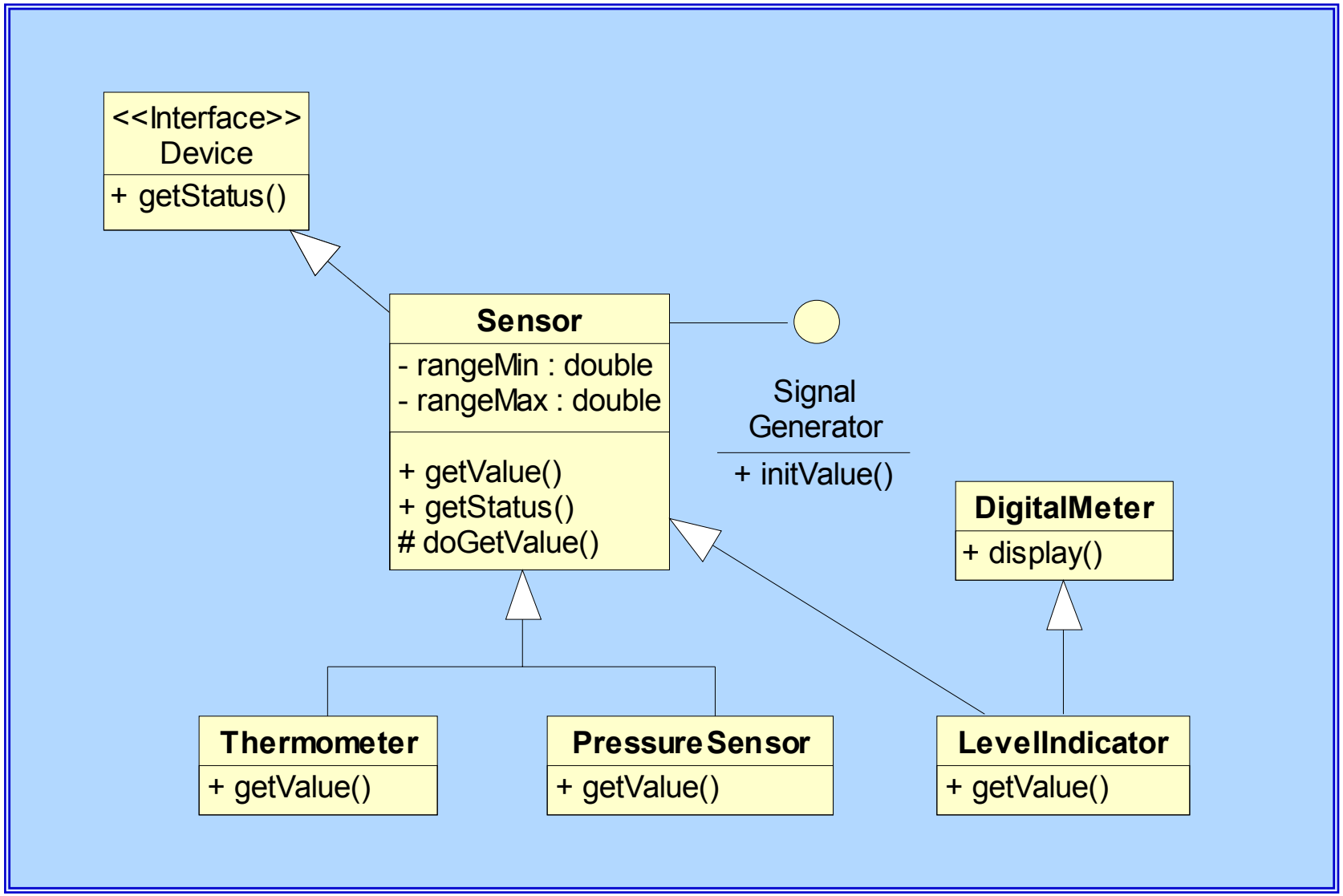
Lớp, đối tượng và giao diện

- Một lớp là thực thi của các đối tượng có chung:
  - Ngữ nghĩa
  - Thuộc tính
  - Quan hệ
  - Hành vi
- Một giao diện là một kiểu dịch vụ của đối tượng, ví dụ
  - Truy nhập thuộc tính
  - Thực hiện các phép toán

Ví dụ: Hệ thống điều khiển



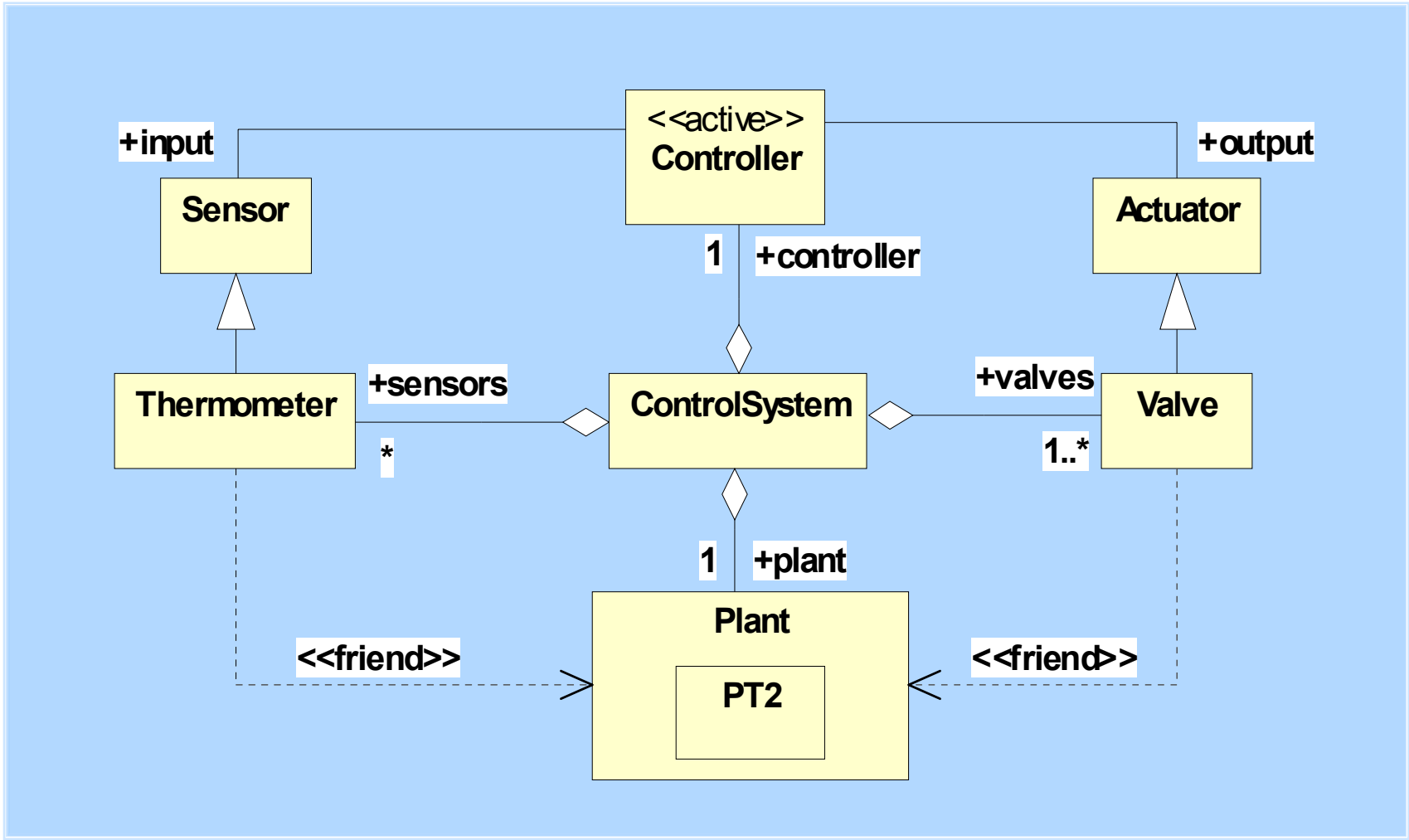
Lớp và giao diện trong UML



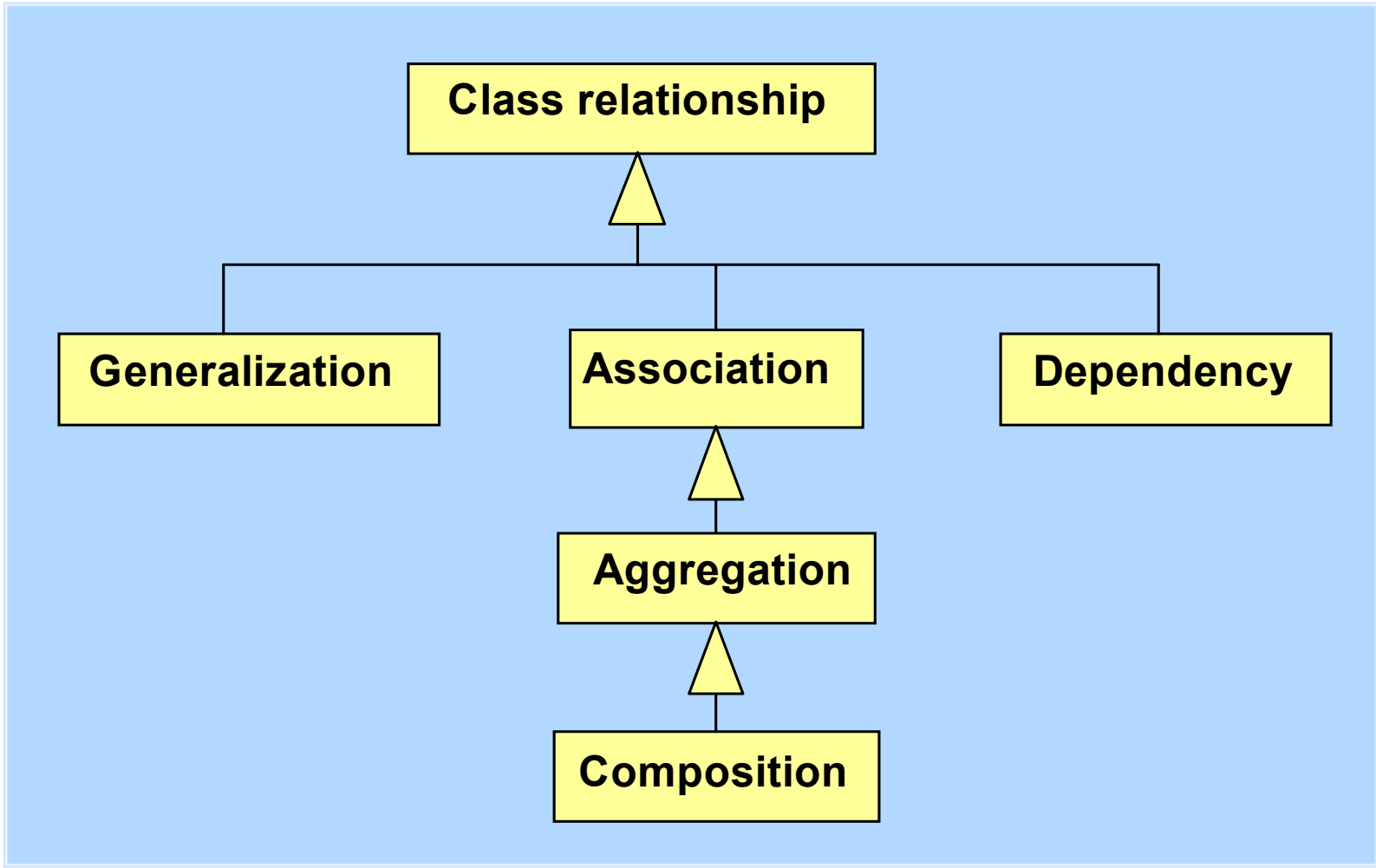
Quan hệ lớp/đối tượng

- Generalization/Specialization: Thừa kế (Inheritance), Dẫn xuất (Subtyping)
- Association: Quan hệ chung chung
- Aggregation: Quan hệ sở hữu
- Composition: Quan hệ cấu thành
- Dependency: Quan hệ sử dụng

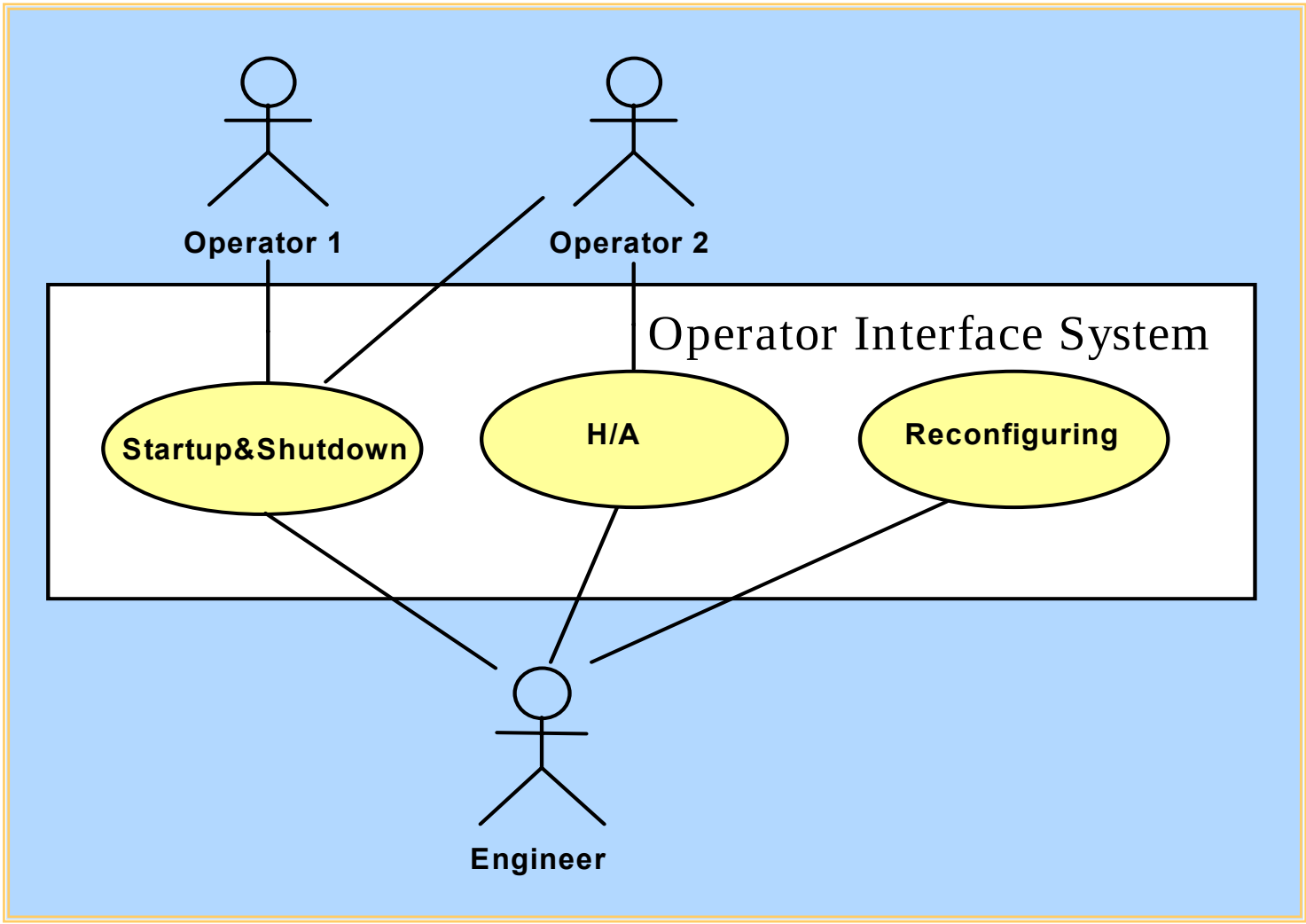
Quan hệ lớp trong UML



Quan hệ lớp - Meta model



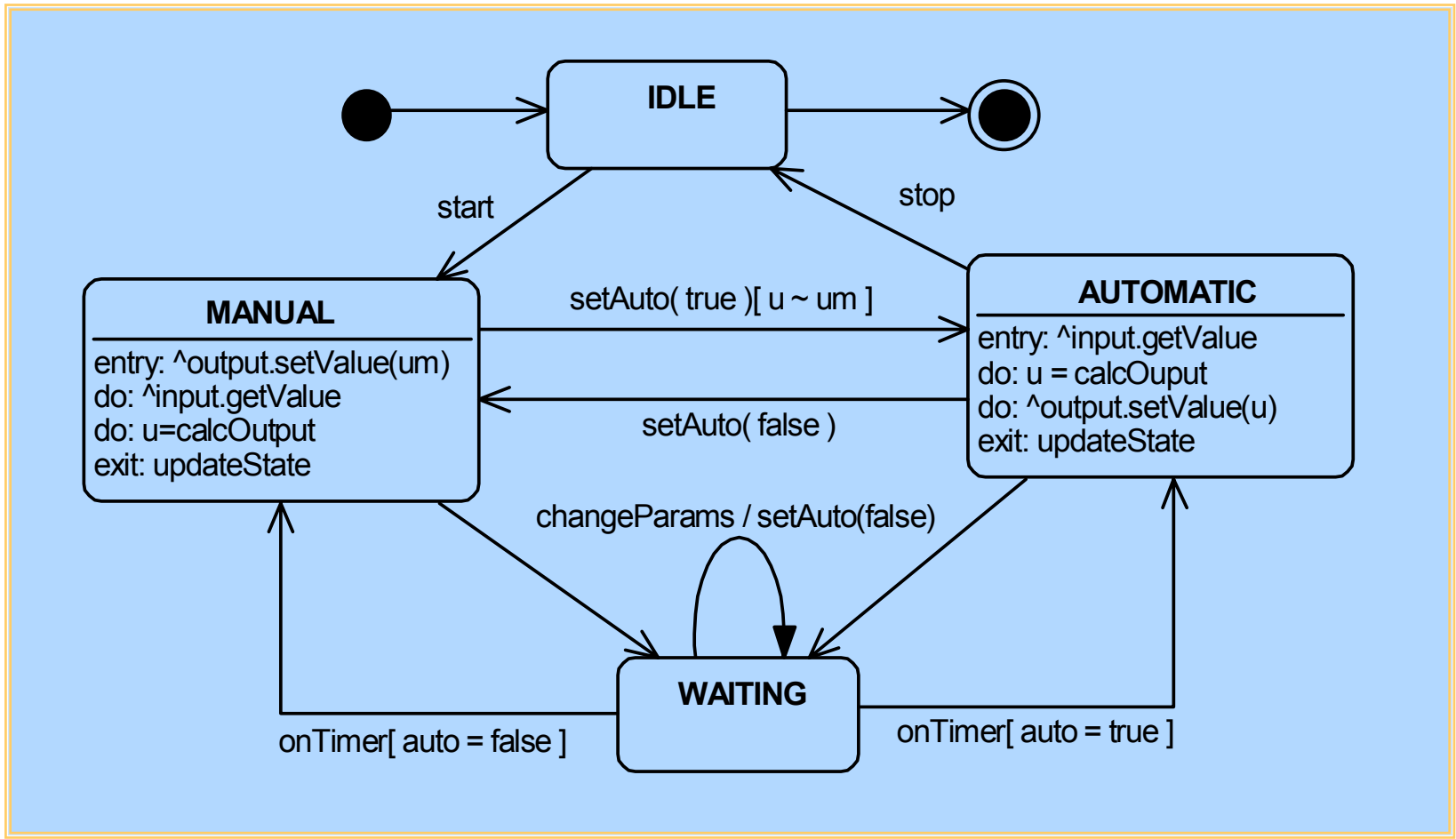
Use case



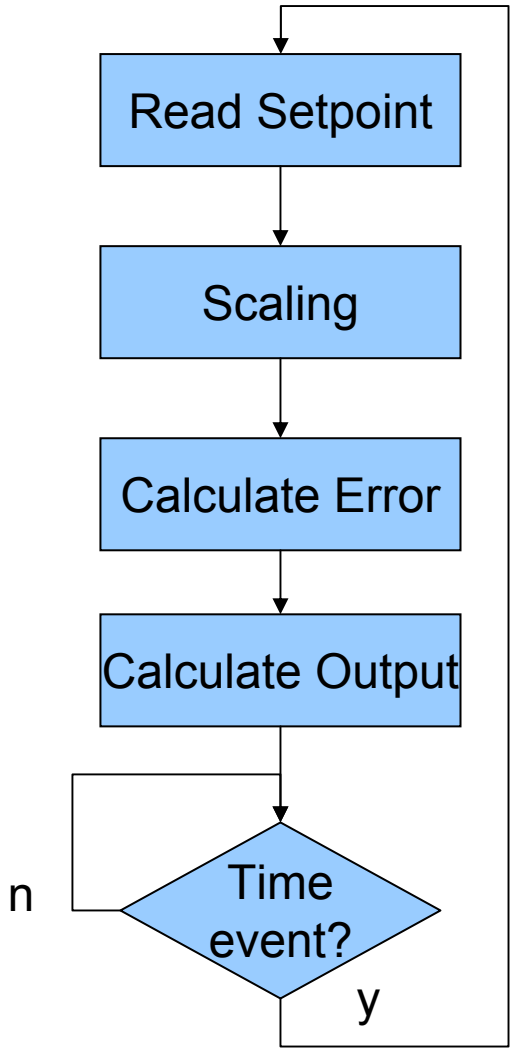
Mô hình hóa hành vi

- Hành vi đối tượng:
  - Biểu đồ trạng thái (*Statecharts*)
  - Biểu đồ hành động (*Activity diagramm*)
- Tương tác giữa các đối tượng
  - Biểu đồ trình tự (*Sequence diagram)*
  - Biểu đồ cộng tác (*Collaboration diagram*)

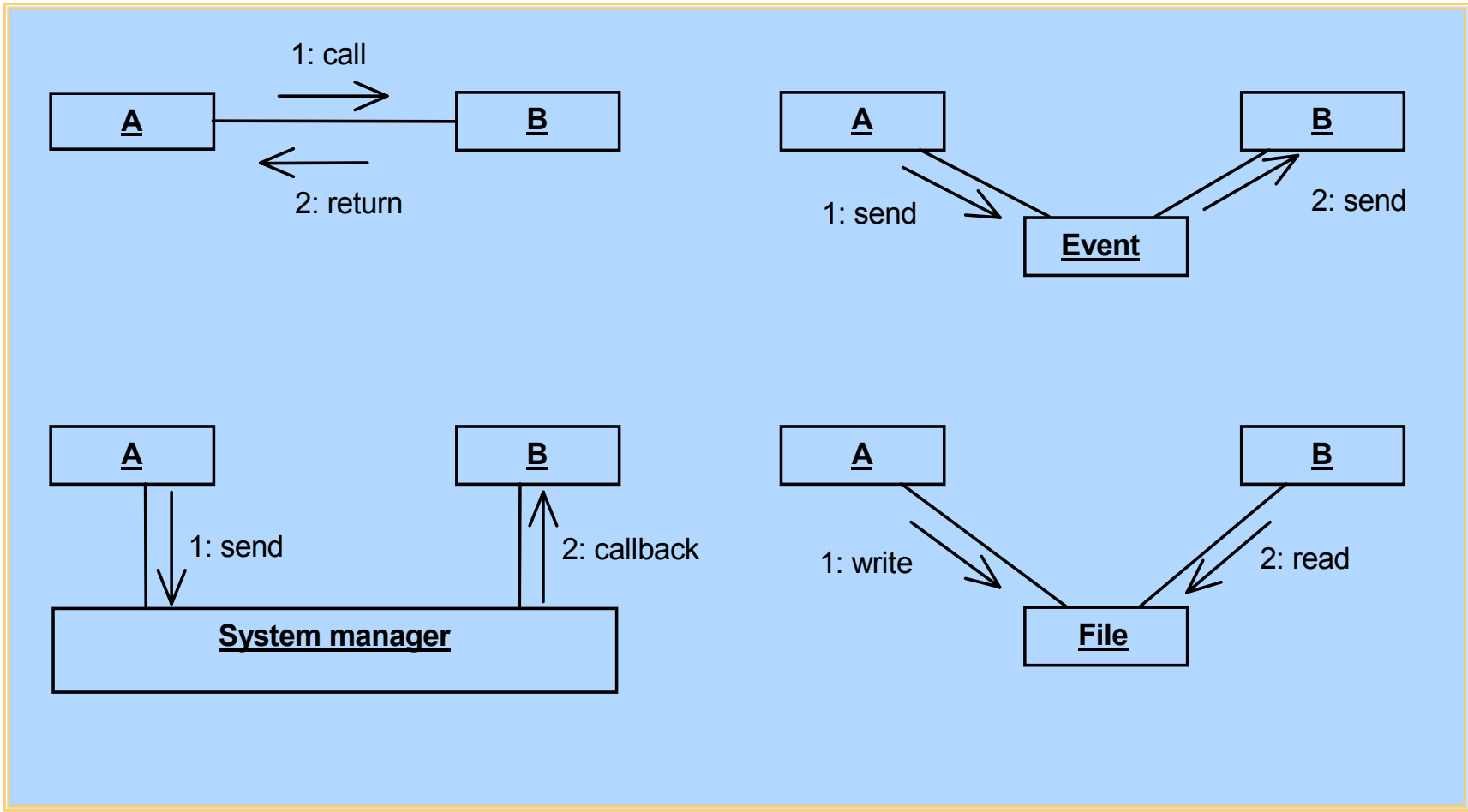
Biểu đồ trạng thái (bộ điều khiển)



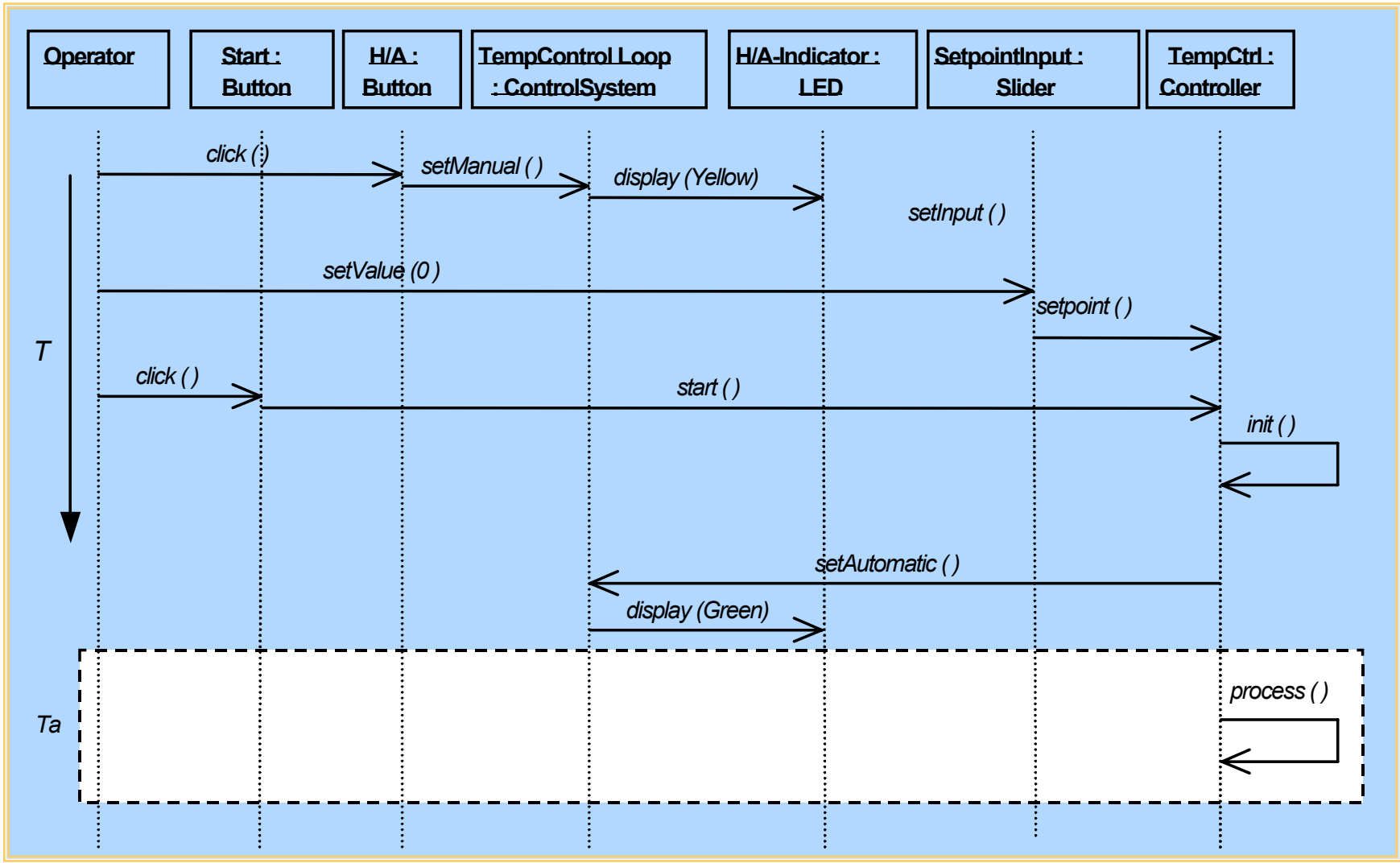
Biểu đồ hành động (calcOutput)



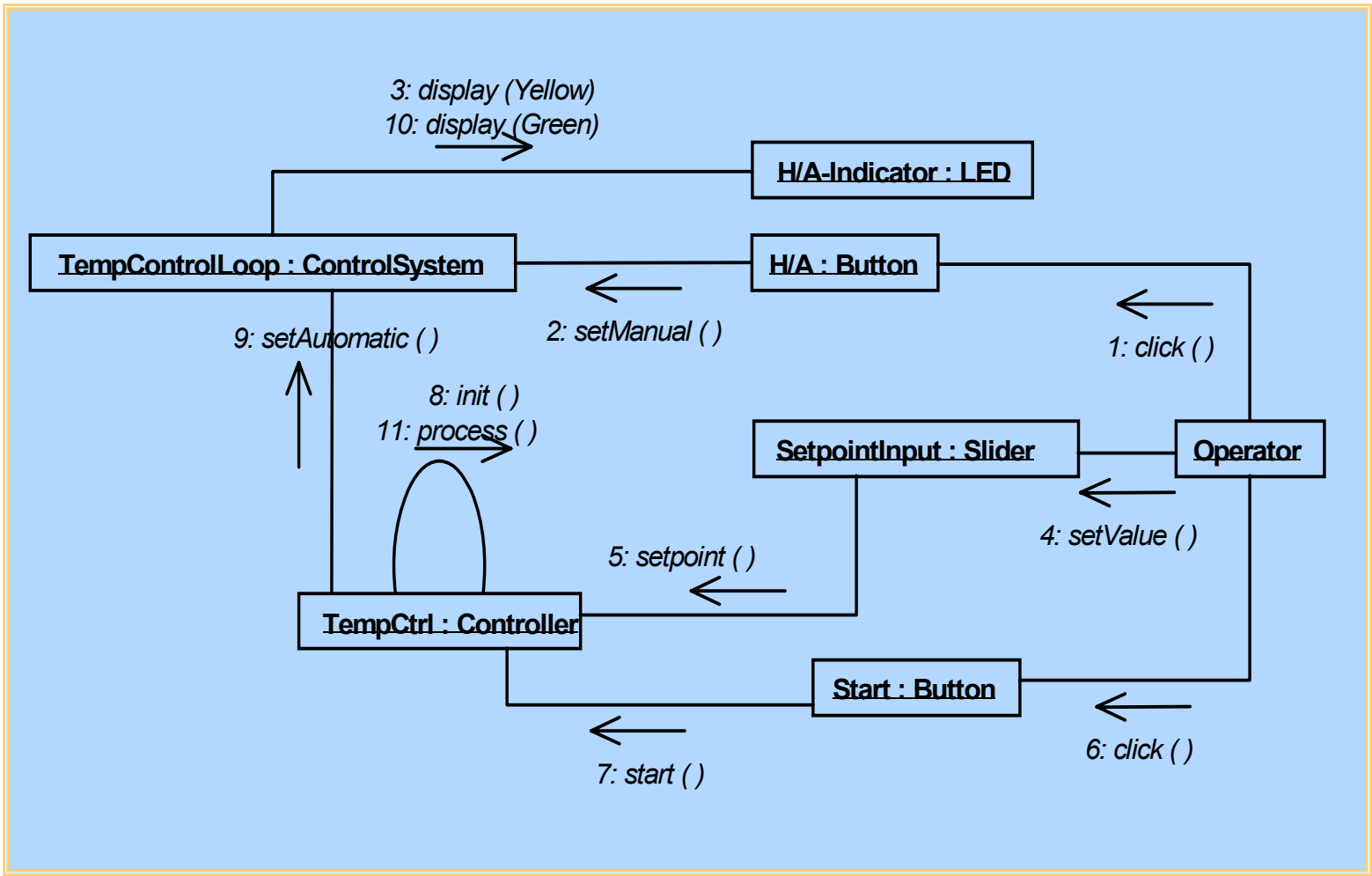
Tương tác đối tượng



Biểu đồ trình tự



Biểu đồ cộng tác



Tóm tắt

- Use case và kịch bản cho phân tích yêu cầu, phân tích hệ thống
- Biểu đồ lớp cho thiết kế cấu trúc
- Biểu đồ tương tác cho thiết kế giao diện và quan hệ tương tác giữa các đối tượng
- Biểu đồ trạng thái cho thiết kế hành vi đối tượng và thiết kế cụ thể các phép toán
- Biểu đồ hành động cho thiết kế thuật toán, chi tiết thực thi phép toán

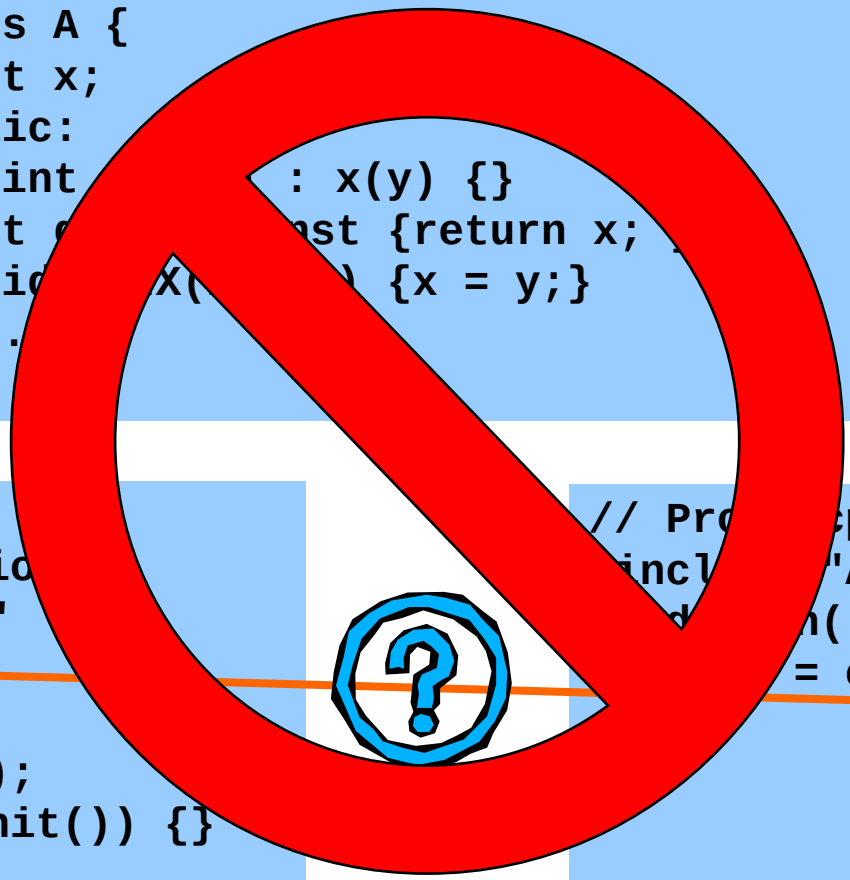
8.4 Khái niệm đối tượng phân tán

- Câu hỏi: Làm thế nào để gọi một hàm thành viên của một đối tượng viết trên C++ từ một chương trình khác?

```
// File A.h
class A {
  int x;
public:
  A(int y) : x(y) {}
  int getX() const {return x;}
  void setX(int y) {x = y;}
  ...
};
```

```
// Prog1.cpp
#include <conio.h>
#include "A.h"
A obj(5);
void main() {
   obj.setX(6);
   while (!kbhit()) {}
}
```

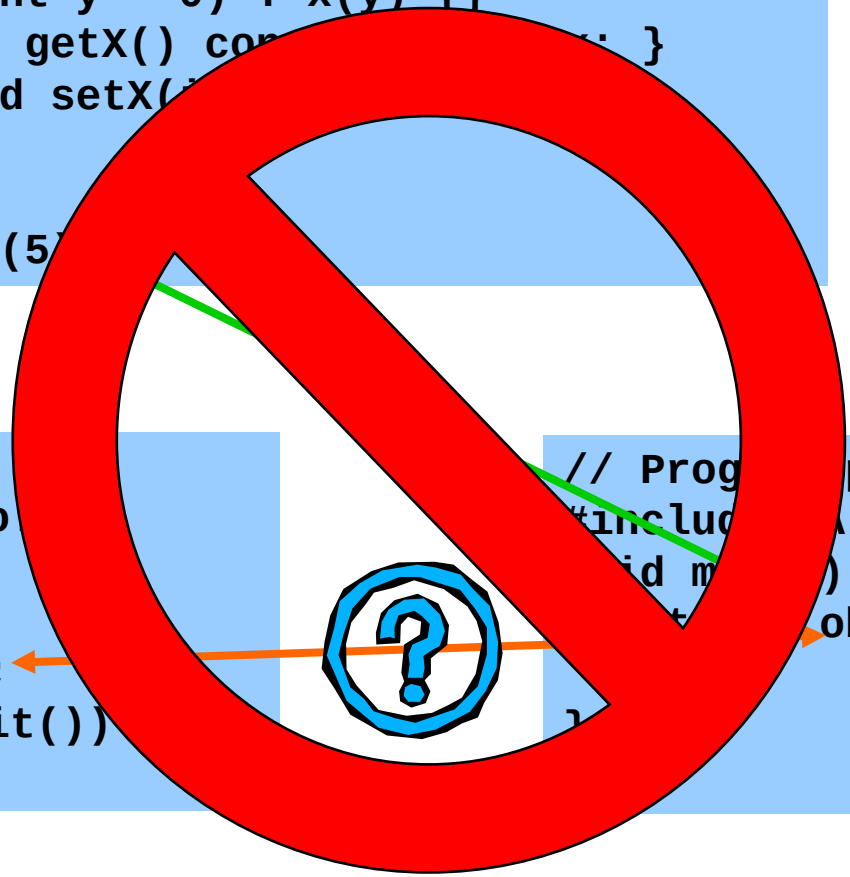
```
// Prog2.cpp
#include "A.h"
void main() {
   int x = obj.getX();
```



```
// File A.h
class A {
  int x;
public:
  A(int y = 0) : x(y) {}
  int getX() co
  void setX(i
  ...
};
A obj(5)
```

```
// Prog1.cpp
#include <conio
#include "A.h"
void main() {
   obj.setX(6);
   while (!kbhit())
}
```

```
// Prog
#inclu .h"
id m ) {
   obj.getX();
}
```



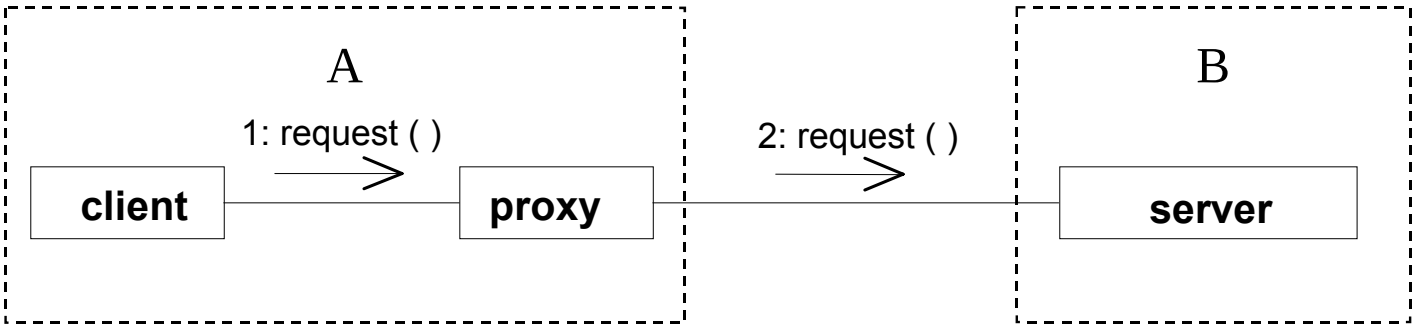
Đối tượng phân tán là gì?

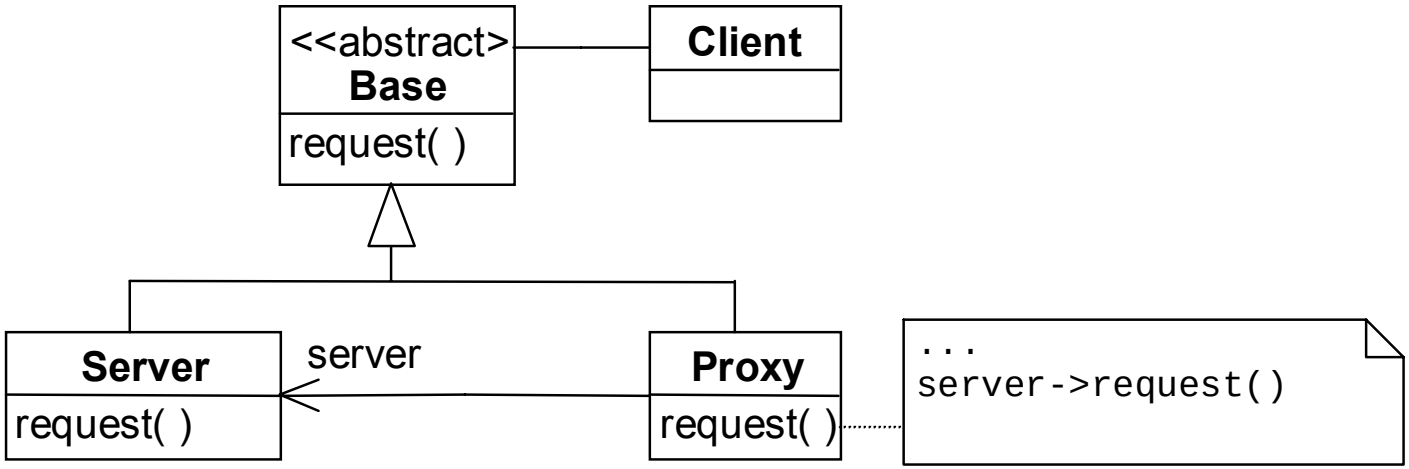
Đối tượng phân tán là các đối tượng phần mềm trong một hệ thống phân tán, có khả năng giao tiếp ngầm, có thể sử dụng từ xa (từ một chương trình khác, từ một nút mạng khác)

- Giống với đối tượng cổ điển
  - Có những đặc tính của đối tượng cổ điển (thuộc tính, phép toán, hành vi, trạng thái, căn cước)
- Khác với đối tượng cổ điển:
  - Không gắn với một ngôn ngữ lập trình
  - Không gắn với một nền cài đặt, nền mạng
  - Có thể tạo, hủy và gọi hàm từ xa

Mẫu thiết kế: Proxy

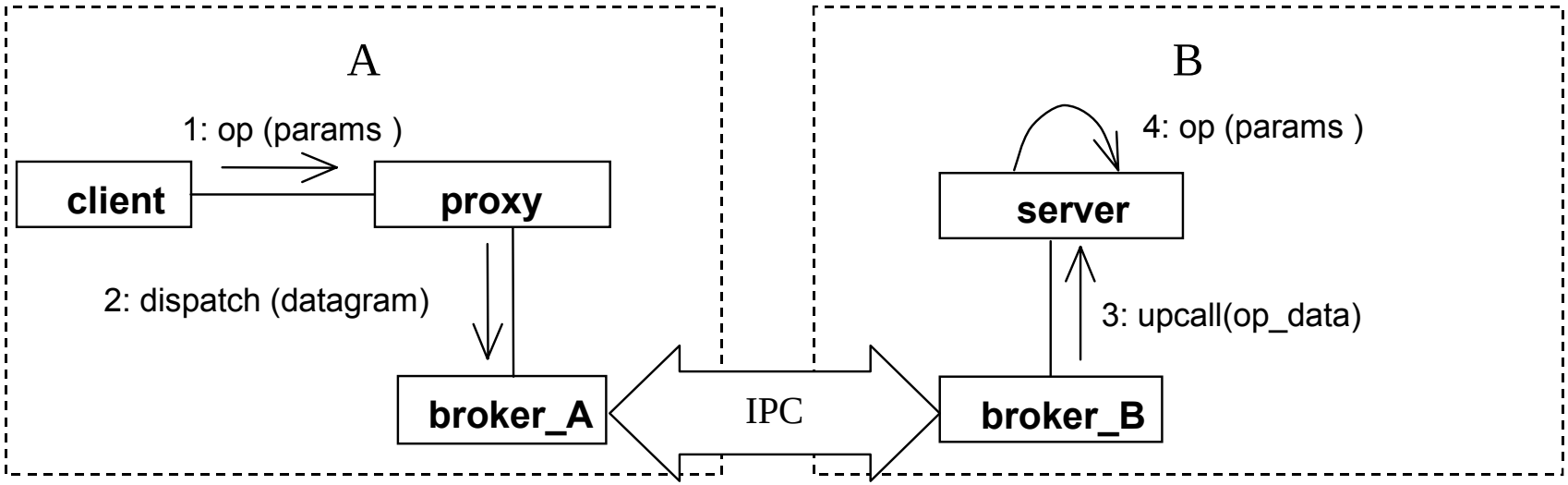
Sử dụng đối tượng từ xa thông qua một đại diện (Proxy)





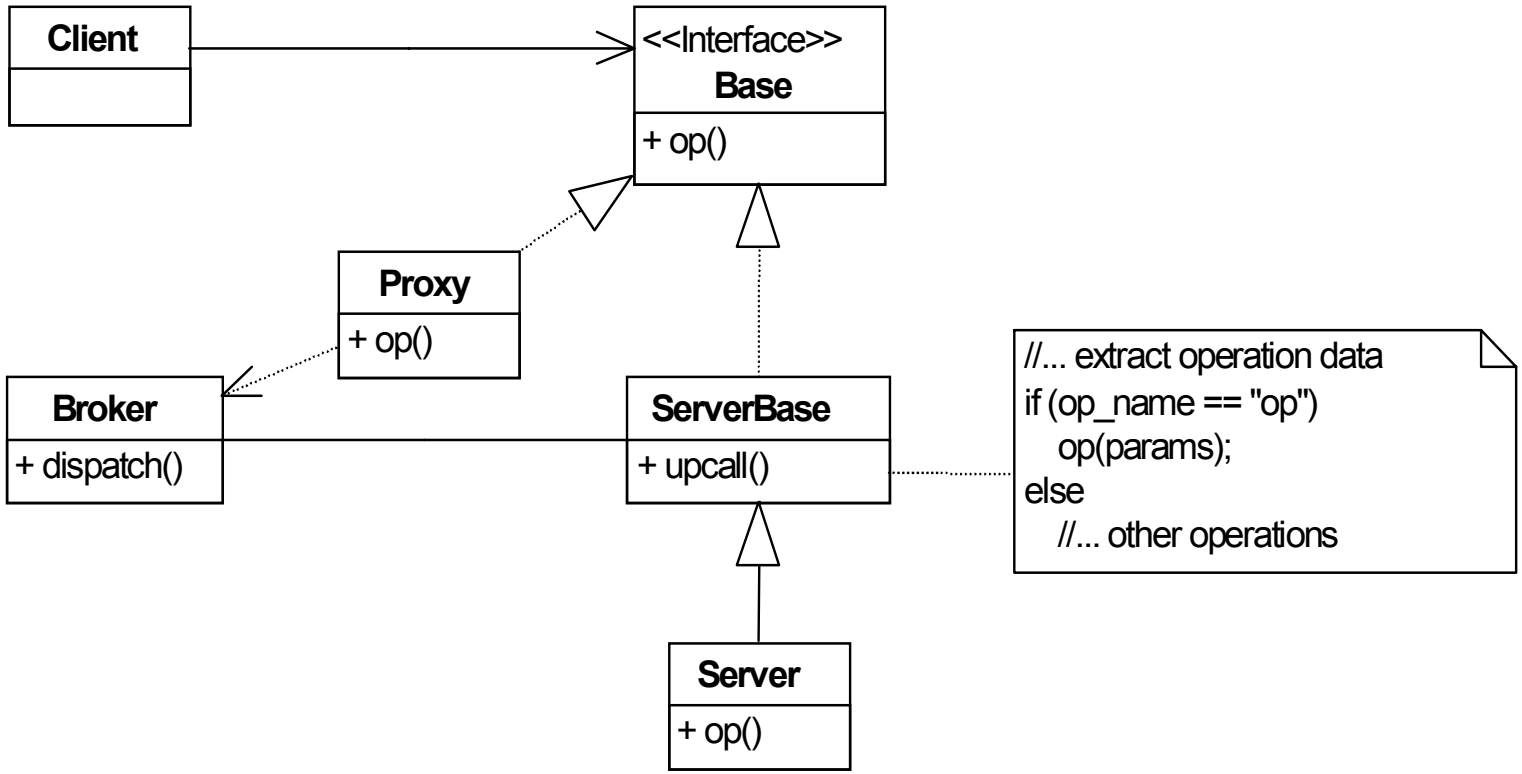
Mẫu thiết kế: Broker + Marshaling/Unmarshaling

Sử dụng đối tượng trung gian (Broker) để Proxy thực hiện giao tiếp với Server một cách trong suốt, không phụ thuộc cơ chế truyền thông cụ thể phía dưới



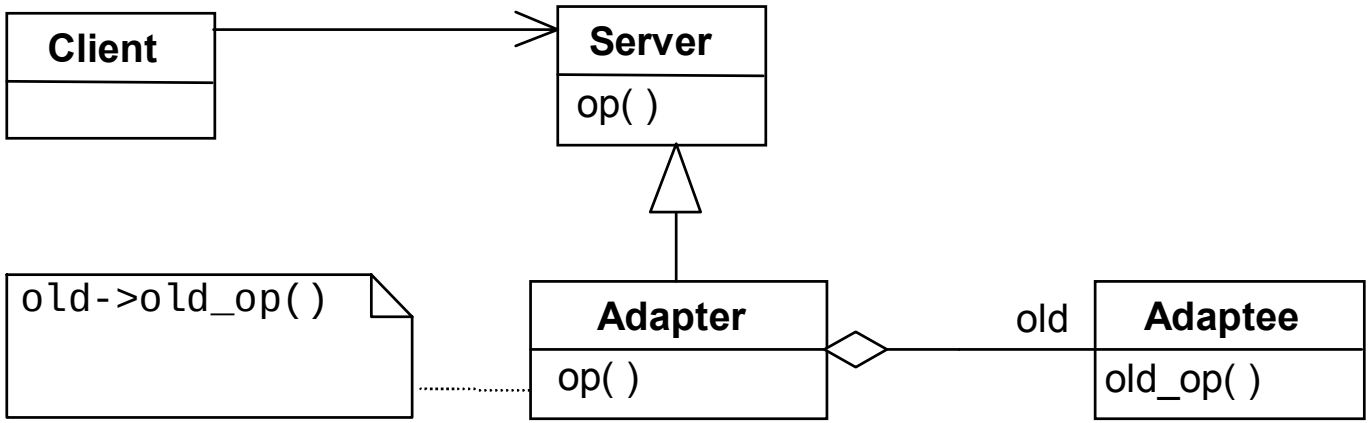
Marshaling: Đóng gói dữ liệu thành một bức điện, mã hóa căn cước đối tượng, tên hàm cần gọi và các tham số
Unmarshaling: Ngược lại với Marshaling.

Mẫu thiết kế: Broker + Marshaling/Unmarshaling



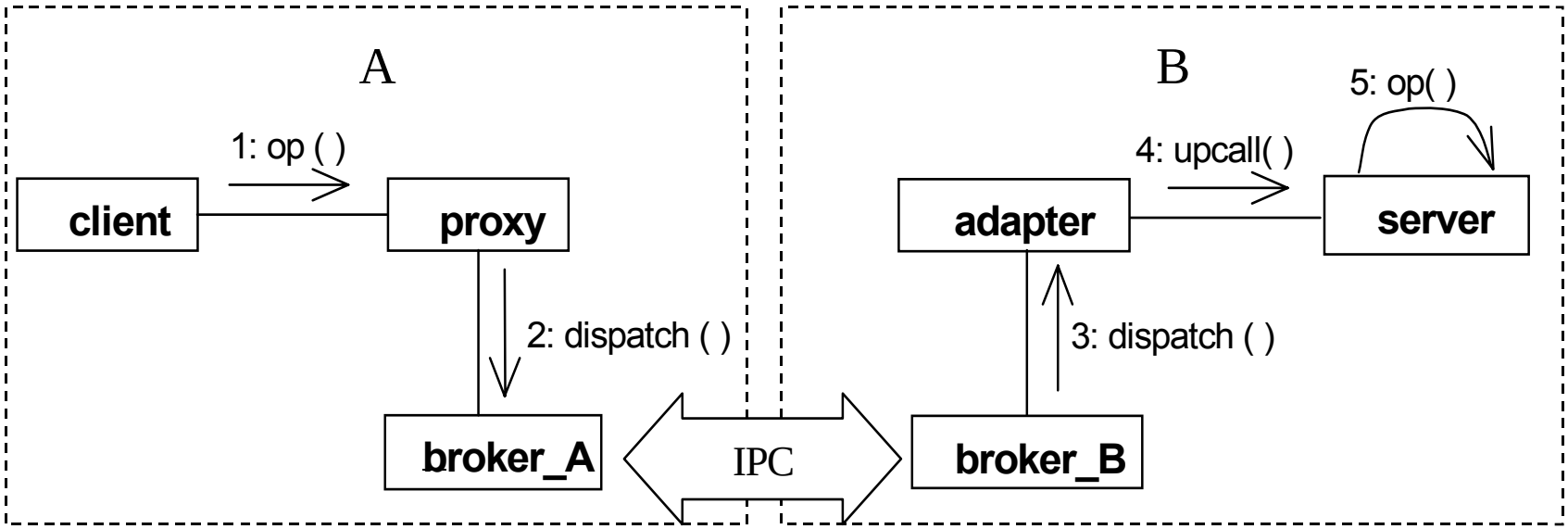
Mẫu thiết kế: Adapter

Sử dụng một đối tượng có sẵn thông qua giao diện thích ứng (Adapter)




```
class Adapter : public Server {
   Adaptee *old;
public:
  void op() { old->old_op(); }
  ...
};
```

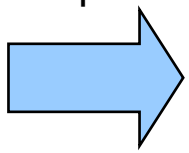
Kiến trúc tổng thể



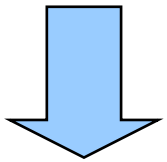
Mẫu thiết kế: Interface Mapping

```
// Server.idl
// Interface Definition
// Language
interface Server {
  bool op();
};
```

IDL-to-C++ Compiler



```
// Proxy.h
class Server_p {
public:
  boolean op() {
    // Marshaling Codes
  }
  ...
}
```



IDL-to-C++ Compiler

Broker

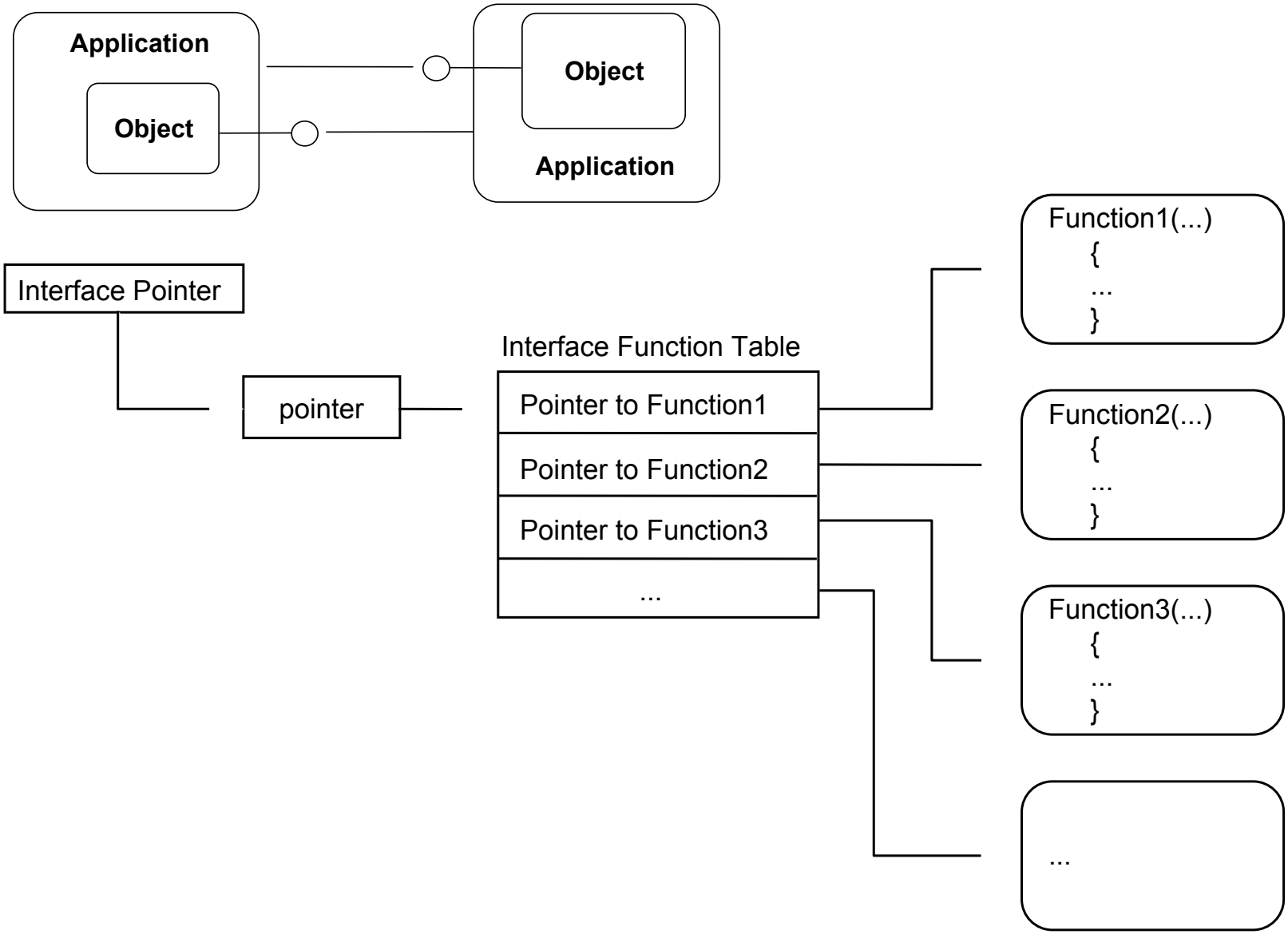
```
// ServerImp.cpp
class Server_s {
  virtual bool op()=0;
  virtual upcall() {
    ...
    op();
  }
};
class MyServer:
      public Server_s {
  bool op() { ... }
};
```

```
// Client Program
// Client.cpp
#include <Proxy.h>
void main() {
  Server_p *obj =
  NewObject("ServerID");
  bool result = obj->op();
  ...
  obj->DeleteObject();
}
```

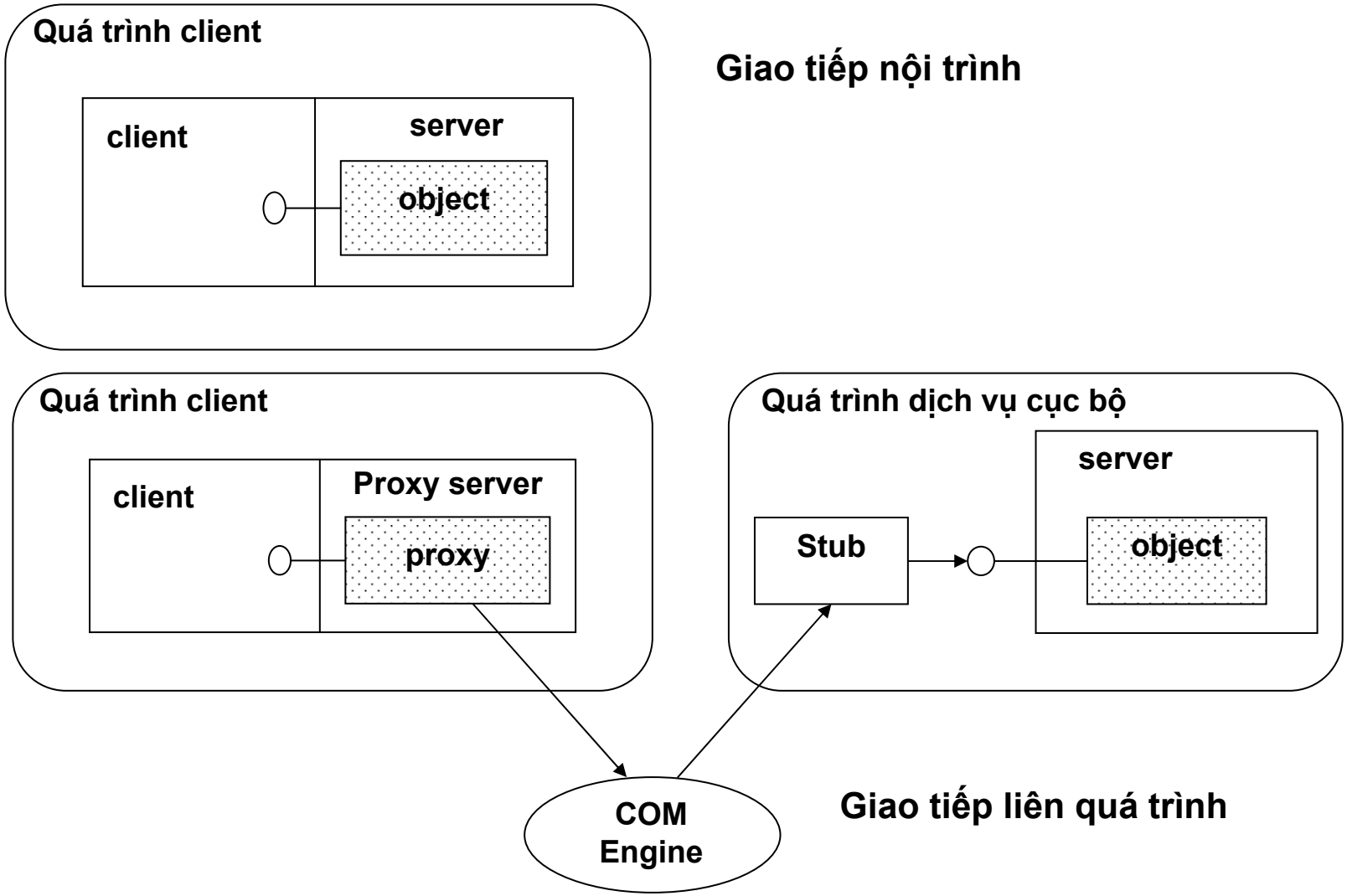
8.5 Mô hình COM và DCOM

- COM (Component Object Model)
  - Chuẩn của Microsoft, chủ yếu thực hiện trên nền Windows
  - Kiến trúc giao tiếp bậc cao giữa các thực thể phần mềm (đối tượng thành phần) trong hệ thống
  - Là nền tảng cho các công nghệ khác: OLE, ActiveX-Control, ASP, ADO, ...
  - Công nghệ then chốt trong các sản phẩm của Microsoft ngày nay
  - Hỗ trợ rất mạnh trong các sản phẩm phần mềm khác
- DCOM (Distributed COM)
  - Giao thức hỗ trợ giao tiếp với COM qua mạng
  - Kiến trúc đối tượng phân tán (so sánh với CORBA)

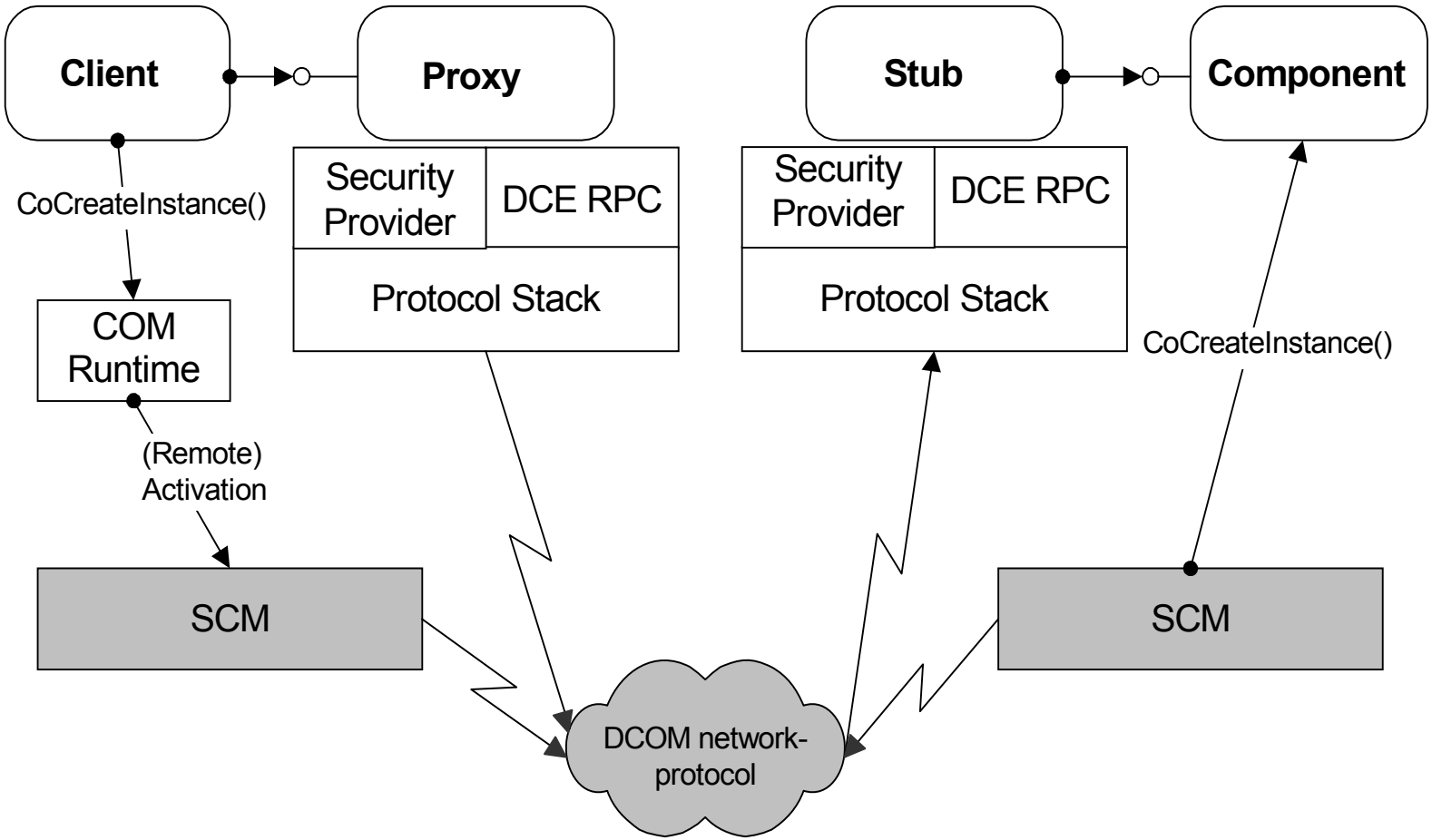
Đối tượng COM và giao diện COM



Giao tiếp với COM



Giao tiếp qua mạng với DCOM



8.6 Lập trình với COM/DCOM

- Tạo một đối tượng COM
- Sử dụng một đối tượng COM

Tạo một đối tượng COM

- Định nghĩa giao diện và căn cước đối tượng bằng IDL

```
/* COM-IDL */
[ object,
  uuid(793D8ABD-3E1B-11D3-A3E3-00A0C910AB98) ]
interface ISensor : IUnknown
{
// Eigenschaften
[propget] HRESULT status([out, retval] short *pVal);
[propget] HRESULT rangeMin([out, retval] double *pVal);
[propget] HRESULT rangeMax([out, retval] double *pVal);

// Methoden
HRESULT getValue([out, retval] double *pVal);
    // ...
};

[ uuid(793D8ABE-3E1B-11D3-A3E3-00A0C910AB98) ]
coclass Sensor
{
[default] interface ISensor;
};
```

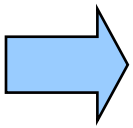
- Dịch sang C++ với IDL-Compiler

```
/* C++ */
class ISensor : public IUnknown
{
public:
virtual HRESULT STDMETHODCALLTYPE get_status(short *pVal) = 0;
virtual HRESULT STDMETHODCALLTYPE get_rangeMin(double *pVal) = 0;
virtual HRESULT STDMETHODCALLTYPE get_rangeMax(double *pVal) = 0;
virtual HRESULT STDMETHODCALLTYPE getValue(double *pVal) = 0;
... // Marshaling Codes
};
```

- Dẫn xuất lớp và thực thi đối tượng COM

```
/* C++ */
class Thermometer : public ISensor
{
public:
  virtual HRESULT get_status(short *pVal)     {...}
  virtual HRESULT get_rangeMin(double *pVal)  {...}
  virtual HRESULT get_rangeMax(double *pVal)  {...}
  virtual HRESULT getValue(double *pVal)      {...}
};
```

- Biên dịch mã thực thi đối tượng
- Đăng ký với hệ điều hành Windows trên các trạm cài đặt và trạm sử dụng



Các bước trên có thể thực hiện đơn giản với sự hỗ trợ của Visual C++ và ATL/COM Wizard

Sử dụng một đối tượng COM

```
/* C++ */
ISensor *pSensor;
HRESULT hr = CoCreateInstance(CLSID_Sensor, 0, CLSCTX_ALL,
  IID_ISensor,(void**)&pSensor);
if (SUCCEEDED(hr)) {
  double value = 0;
  pSensor->getValue(&value);
    //...
}
//...
pSensor->Release();
```