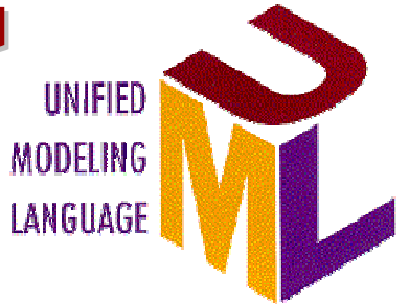
PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



CHỦ ĐỀ

Tiến trình phát triển phần mềm theo hướng đối tượng

1. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Tài liệu tham khảo chính

1. Đặng Văn Đức, *Phân tích thiết kế hướng đối tượng bằng UML*, Nhà xuất bản Giáo dục, 287 trang. 2002.
2. Zhiming Liu, *Object-Oriented Software Development with UML*, UNU/IIST, 169 pp, 2002.
3. Phần mềm: *Rational Rose Enterprise Edition* 2002, IBM Rational Software. 2002.

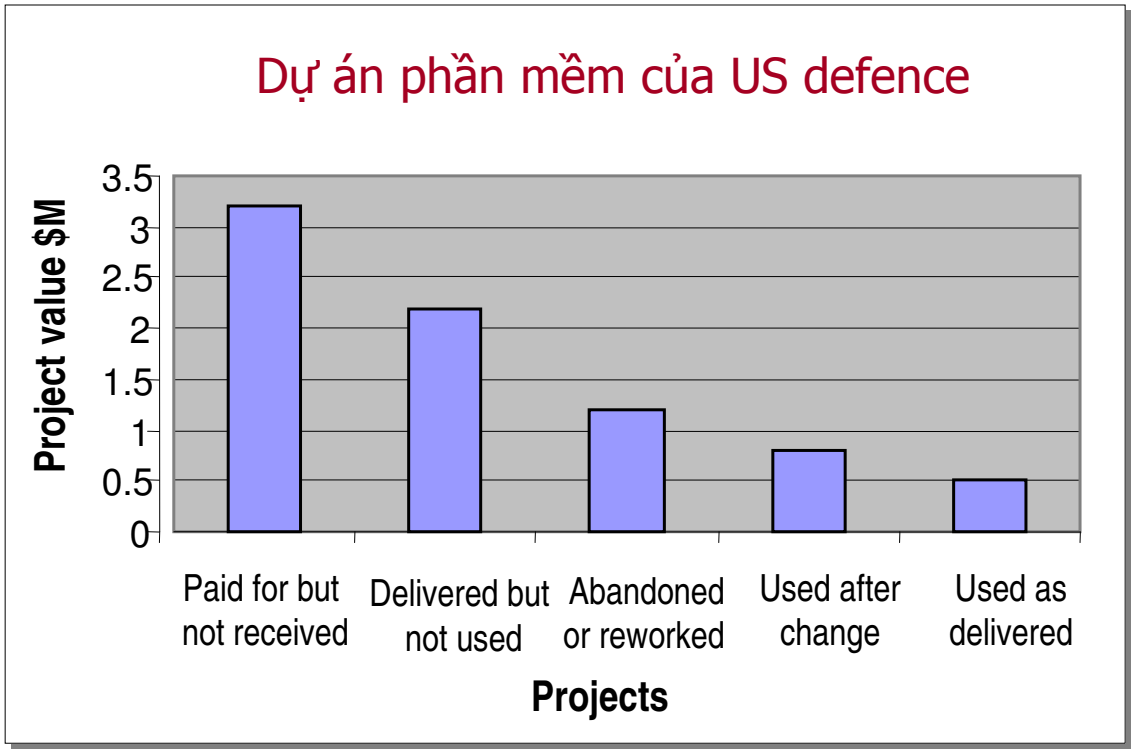
Bài 1

Tiến trình phát triển phần mềm theo hướng đối tượng

Lịch sử phương pháp hướng đối tượng

- Khủng hoảng phần mềm
  - NATO Software Engineering Conference, Germany, 1968
  - Thống kê của chính phủ Mỹ về các dự án SW của Bộ quốc phòng, 1970.

(E. Balagurusamy)



Kỹ nghệ phần mềm

- Khái niệm kỹ nghệ phần mềm (software engineering) xuất hiện vào cuối 1960 – khi bắt đầu có máy tính thế hệ 3
- Các đặc tính chủ yếu của hệ thống phần mềm hiện nay
  - Nó mô hình hóa các phần của thế giới thực
  - Rất lớn và phức tạp
  - Nó là trừu tượng
  - Phải có tính độc lập cao
  - Phải dễ bảo trì:
    - khi thế giới thực thay đổi, phần mềm phải đáp ứng các yêu cầu thay đổi
  - Phải thân thiện với người sử dụng
    - UI là phần rất quan trọng của hệ thống phần mềm

Kỹ nghệ phần mềm

- Phát triển phần mềm bị khủng hoảng vì không có phương pháp đủ tốt
  - Kỹ thuật áp dụng cho các hệ thống nhỏ trước đây không phù hợp cho các hệ thống lớn
  - Các dự án lớn thường bị kéo dài hàng năm do vậy làm tăng kinh phí
  - Phần mềm không tin cậy, khó bảo hành
- Thực tế: Giá phần cứng giảm nhanh, giá phần mềm tăng cao
- Để đáp ứng đòi hỏi của phần mềm cần có
  - Lý thuyết, kỹ thuật, phương pháp, công cụ mới để điều khiển tiến trình phát triển hệ thống phần mềm
- Kỹ nghệ phần mềm: Liên quan tới lý thuyết, phương pháp và công cụ cần để phát triển phần mềm
- Mục tiêu: Sản xuất phần mềm độc lập, đúng hạn, phù hợp kinh phí và đáp ứng mọi yêu cầu người sử dụng

Sản phẩm phần mềm

- Kỹ nghệ phần mềm để sản xuất
  - Hệ thống phần mềm
  - Các tài liệu
    - Thiết kế hệ thống
    - Tài liệu sử dụng: Cài đặt? và Sử dụng phần mềm?
- Các đặc tính cơ bản của phần mềm
  - Có thể sử dụng được
    - Cần có UI phù hợp, tài liệu rõ ràng
  - Tính dễ bảo hành
    - Dễ dàng mở rộng để đáp ứng các yêu cầu thay đổi (phần mềm mềm dẻo)
  - Tính độc lập
    - Các tính chất cơ bản như tin cậy, an toàn
    - Không gây tác hại về vật lý, kinh tế ngay cả khi hệ thống hỏng
  - Tính hiệu quả
    - Không tiêu tốn quá nhiều tài nguyên hệ thống như bộ nhớ, thời gian CPU

Sản phẩm phần mềm

- Để thỏa mãn đồng thời mọi tính chất của sản phẩm phần mềm như nói trên là rất khó khăn
  - Thí dụ giữa giá cả với tính năng
- Để xây dựng hệ thống phần mềm tốt ta cần
  - Xác định đúng đắn tiến trình phát triển phần mềm
    - Các pha của hoạt động
    - Sản phẩm của mỗi pha
  - Phương pháp và kỹ thuật áp dụng trong từng pha và mô hình hóa sản phẩm của chúng
  - Công cụ phát sinh ra sản phẩm

Sản phẩm phần mềm được xem như mô hình của thế giới thực. Nó phải được duy trì để luôn luôn phản ánh chính xác sự thay đổi trong thế giới thực

Tiến trình phát triển phần mềm

- Mọi kỹ nghệ (engineering) đều đề cập đến sản xuất sản phẩm theo tiến trình
- Tổng quát thì tiến trình (process) xác định ai (Who) làm gì (What) và làm khi nào (When) và làm như thế nào (How) để đạt tới mục đích mong muốn.
- Tiến trình phát triển phần mềm (Software Development Process - SDP) là tiến trình xây dựng sản phẩm phầm mềm hay nâng cấp phần mềm đang có.
- Thí dụ tiến trình phát triển phần mềm:
  - Rational Unified Process - RUP

Tiến trình phát triển phần mềm

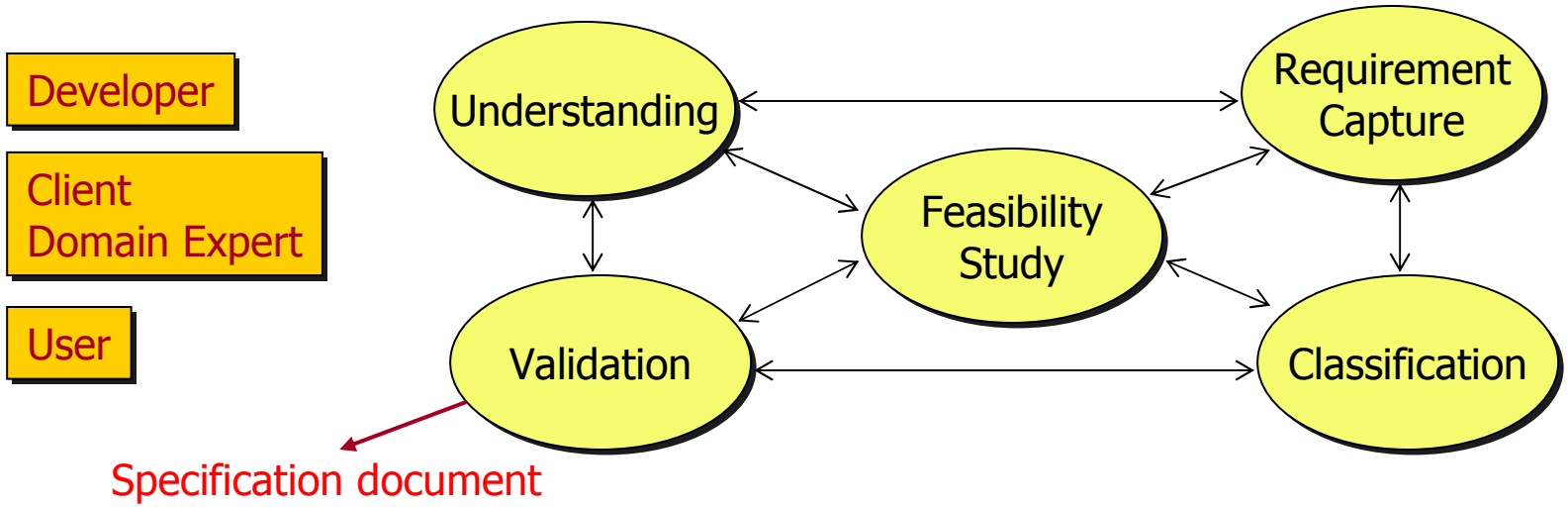
- Tiến trình phát triển phần mềm mô tả tập các hoạt động cần thiết để chuyển đổi từ yêu cầu người sử dụng sang hệ thống phần mềm
- Yêu cầu người sử dụng xác định mục tiêu phát triển phần mềm
  - Khách hàng và kỹ sư tin học xác định các dịch vụ mà hệ thống cần có (yêu cầu chức năng của hệ thống)
- Yêu cầu chức năng mô tả cái mà hệ thống phải làm (What) không mô tả hệ thống làm như thế nào (How)
  - Khách hàng cũng có các ràng buộc phi chức năng: thời gian đáp ứng, chuẩn ngôn ngữ...

Tiến trình phát triển phần mềm

- Thu thập và phân tích yêu cầu là công việc rất khó khăn
  - Các yêu cầu thường là không hoàn chỉnh
  - Yêu cầu của khách hàng thường được mô tả bằng khái niệm, đối tượng và các thuật ngữ khó hiểu với kỹ sư tin học
  - Các yêu cầu của khách hàng thường thiếu cấu trúc, thiếu chính xác, dư thừa, phỏng chừng, thiếu nhất quán
  - Các yêu cầu thiếu tính khả thi
- Do vậy
  - Bất kỳ tiến trình phát triển nào đều bắt đầu từ thu thập và phân tích yêu cầu
- Các hoạt động trong SDP và các kết quả liên quan hình thành pha đầu tiên của tiến trình và gọi nó là Phân tích yêu cầu

Thu thập và phân tích yêu cầu

- Mục tiêu
  - Hình thành tài liệu đặc tả yêu cầu (Requirement Specification)
- Tài liệu đặc tả yêu cầu được sử dụng như
  - Cam kết giữa khách hàng và tổ chức phát triển hệ thống về cái mà hệ thống có thể làm (và cái mà hệ thống không thể làm)
  - Cơ sở để đội ngũ phát triển phát triển hệ thống
  - Mô hình tương đối đầy đủ về cái hệ thống đòi hỏi
- Tiến trình phân tích yêu cầu bao gồm các hoạt động lặp



Các hoạt động của phân tích yêu cầu

- Hiểu lĩnh vực vấn đề
  - Phân tích viên trình bày hiểu biết về lĩnh vực vấn đề
  - Khám phá các quan niệm
  - Suy ra các yêu cầu khách hàng
- Thu thập yêu cầu
  - Phân tích viên cần có cách thu thập nhu cầu khách hàng sao cho họ có thể cùng tham gia vào dự án
  - Phân tích viên, khách hàng, chuyên gia lĩnh vực ứng dụng và người sử dụng hệ thống cùng phát hiện và thu thập yêu cầu
  - Kỹ năng trừu tượng là rất quan trọng để thu thập những cái chính, bỏ qua cái không cần thiết
- Phân lớp
- Đánh giá
- Nghiên cứu khả thi

Các hoạt động của phân tích yêu cầu

- Hiểu lĩnh vực vấn đề
- Thu thập yêu cầu
- Phân lớp
  - Đầu vào của hoạt động này là tập hợp phi cấu trúc của các yêu cầu thu thập được trong pha trước để tổ chức chúng thành các nhóm dính liền nhau
  - Gắn mức ưu tiên cho các yêu cầu theo tầm quan trọng của chúng đối với khách hàng và người sử dụng
- Đánh giá
  - Kiểm tra xem các yêu cầu có nhất quán và đầy đủ
  - Giải quyết các mâu thuẫn giữa các yêu cầu
- Nghiên cứu khả thi
  - Dự báo khả năng thỏa mãn sử dụng phần cứng, phần mềm của các yêu cầu đã nhận ra
  - Quyết định các bước tiếp theo nếu nếu hệ thống đề xuất có hiệu quả

Phân tích yêu cầu

- Khi nào kết thúc phân tích yêu cầu?
  - Không có quy luật nhất định
- Để tiến tới bước phát triển phần mềm tiếp theo hãy trả lời các câu hỏi sau:
  - Khách hàng, người sử dụng cuối cùng và người phát triển đã hiểu trọn vẹn hệ thống?
  - Mô hình của hệ thống đòi hỏi xây dựng đã được hình thành đầy đủ?
    - có đầy đủ các chức năng (dịch vụ)
    - có đầy đủ đầu vào- đầu ra
    - cần loại dữ liệu nào
- Chú ý: Chưa mô tả quyết định cài đặt nào ở mô hình này
- Đặc tả yêu cầu và mô hình của hệ thống tại mức này cần phải được hiệu chỉnh, bổ sung khi cần thiết trong các pha phát triển tiếp theo.

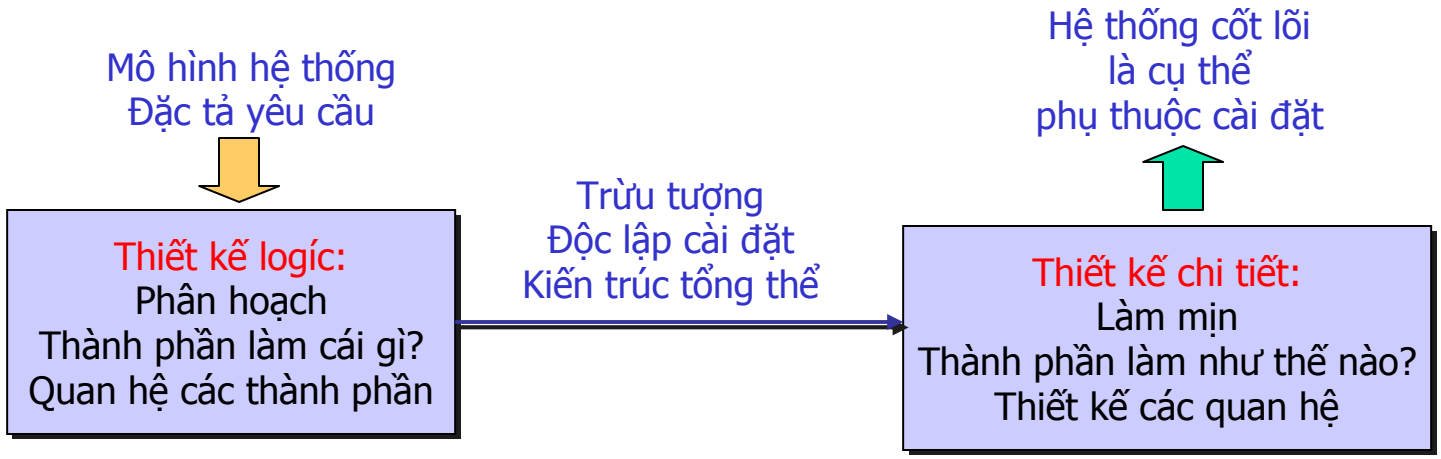
Phân tích yêu cầu

- Đặc tả yêu cầu
  - là thông báo chính thức cái đòi hỏi hệ thống phải được phát triển
  - Nó không phải là tài liệu thiết kế
- Mô tả đặc tả yêu cầu
  - Ngôn ngữ đặc tả
  - Ký pháp đồ họa

Pha thu thập và phân tích yêu cầu rất quan trọng. Nếu không phát hiện ra lỗi tại pha này thì rất khó và tốn kém để phát hiện ra nó ở pha tiếp theo.

Thiết kế hệ thống

- Sau khi có đặc tả yêu cầu, hai tiến trình thiết kế hệ thống tiếp theo
  - Thiết kế kiến trúc (logíc)
    - Phân hoạch các yêu cầu thành các thành phần
    - Tài liệu thiết kế kiến trúc mô tả mỗi thành phần cần làm gì và chúng tương tác với nhau như thế nào để hình thành các chức năng hệ thống
  - Thiết kế chi tiết (vật lý)
    - Thiết kế từng thành phần
    - Tài liệu thiết kế chi tiết mô tả mỗi thành phần và cả hệ thống phải làm cái nó cần làm như thế nào
- Các hoạt động của thiết kế



Thiết kế hệ thống

- Tài liệu của pha thiết kế kiến trúc là mô hình kiến trúc
  - Đặc tả thành phần, mô tả cái mà thành phần phải làm bằng cách chỉ ra giao diện giữa các thành phần
  - Mô hình hệ thống ở đây chủ yếu mô tả "what", ít mô tả "how"
- Thiết kế chi tiết thực hiện nhiều bước làm mịn mô hình kiến trúc
- Mô hình thiết kế chi tiết mô tả:
  - thiết kế chức năng của mỗi thành phần
  - thiết kế giao diện của mỗi thành phần
- Mô hình hệ thống tại mức này được xem như hệ thống cốt lõi
  - nó là cụ thể
  - phụ thuộc cài đặt
  - xác định "How"

Lập trình và kiểm thử mođun

- Mỗi thành phần trong pha thiết kế được hiện thực thành một mođun chương trình
- Kiểm chứng hay kiểm thử mỗi mođun chương trình theo đặc tả có từ pha thiết kế

Tích hợp và kiểm thử hệ thống

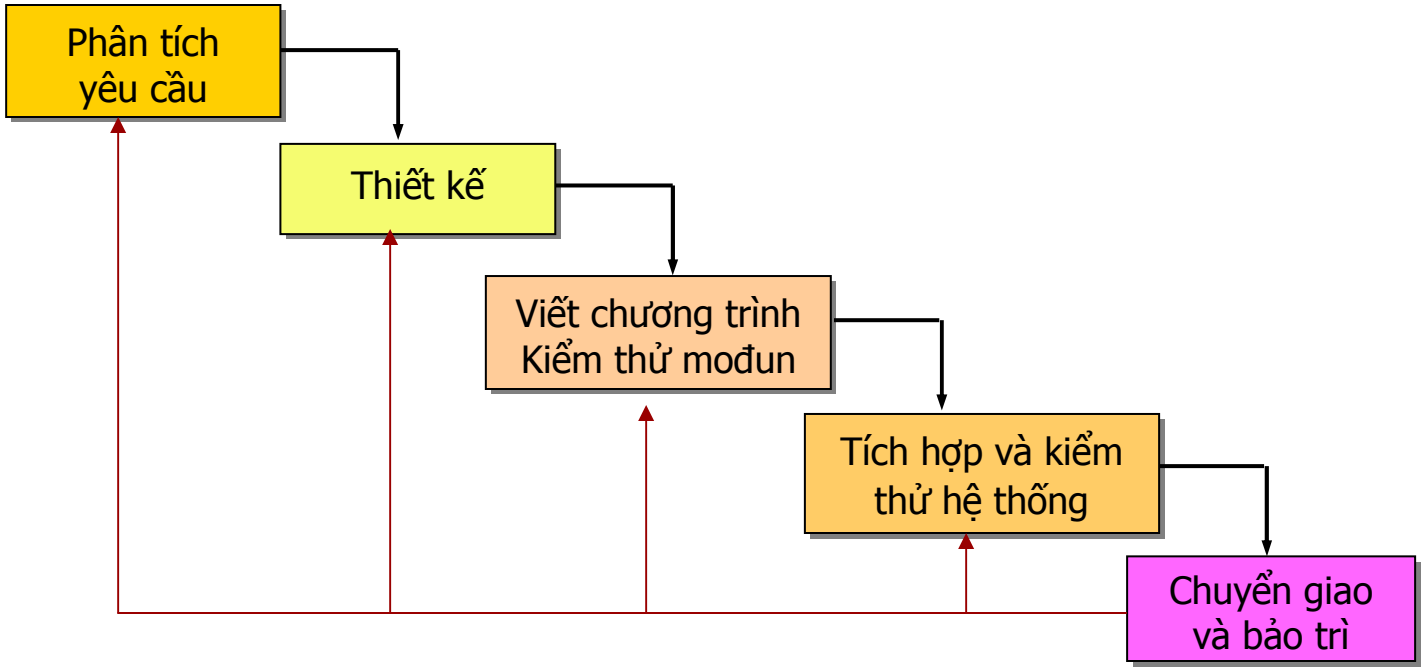
- Tổ hợp các modun chương trình thành hệ thống
- Kiểm thử hệ thống chương trình để đảm bảo đáp ứng đầy đủ yêu cầu
- Khi người phát triển thỏa mãn với sản phẩm
  - khách hàng kiểm thử hệ thống
- Pha này kết thúc khi khách hàng chấp nhận sản phẩm

Bảo trì hệ thống

- Pha này bắt đầu khi hệ thống được cài đặt sử dụng thực tế, sau khi đã cấp phát sản phẩm cho khách hàng
- Bảo trì bao gồm mọi thay đổi sản phẩm để khách hàng đồng ý rằng họ đã thỏa mãn với sản phẩm.
- Bảo trì bao gồm
  - sửa phần mềm
    - loại bỏ các lỗi mà không phát hiện trong các pha trước đó
  - nâng cấp phần mềm
    - Hiệu năng: Bổ sung chức năng, tăng tốc độ thực hiện chương trình
    - Thích nghi: Các thay đổi cho phù hợp với môi trường phần mềm hoạt động thay đổi, thí dụ yêu cầu mới của chính phủ
- Thời gian trung bình:
  - sửa lỗi 17,5%, hiệu năng 60%, thích nghi 18%.

Mô hình thác nước

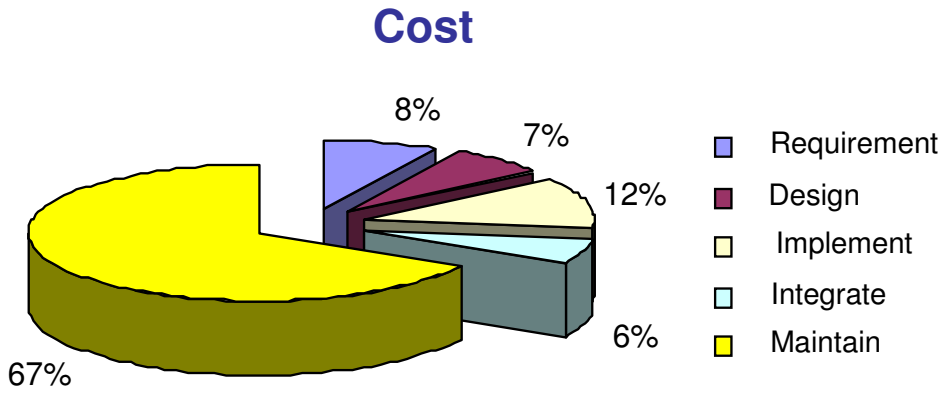
- Các hoạt động phát triển phần mềm có thể biểu diễn bằng mô hình thác nước
- Vòng đời (life cycle) phần mềm
  - Tiến trình phát triển sản phẩm phần mềm

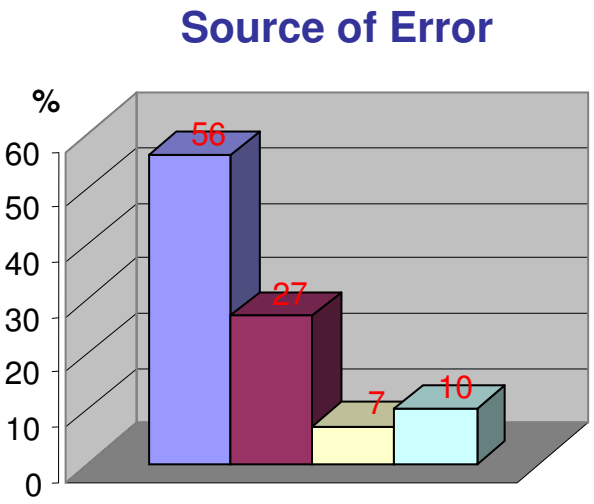


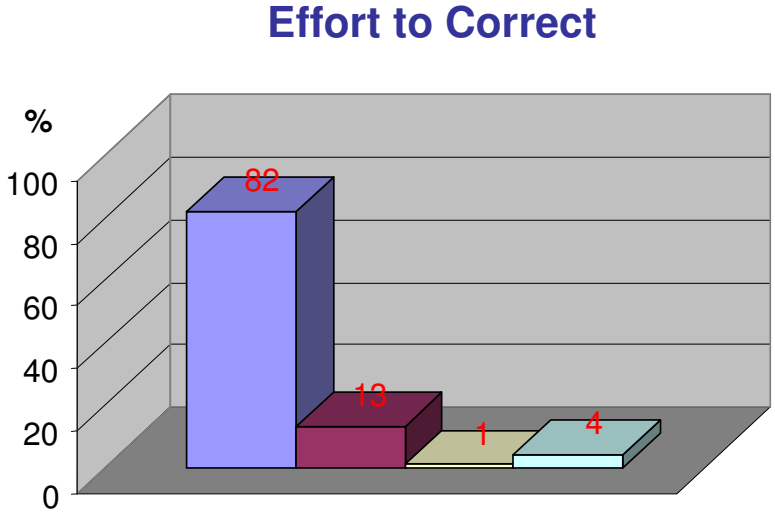
Mô hình thác nước

- Nhận xét mô hình thác nước
  - Khó phân biệt rõ ràng giới hạn các pha, nhiều pha gối lên nhau và cung cấp thông tin cho nhau
    - Khi thiết kế mới nhận ra các yêu cầu mới
    - Khi viết mã trình nhận thấy một vài thiết kế có vấn đề...
    - Khi bảo trì hiệu năng, có thể thực hiện lại một vài hay toàn bộ các bước trước đó
  - Tiến trình phát triển không phải là mô hình tuyến tính mà là trình tự lặp các hoạt động phát triển
  - Tiến trình phát triển bao gồm các lặp thường xuyên
    - Khó nhận ra các điểm mấu chốt để lập kế hoạch và báo cáo kết quả
    - Do vậy, sau một vài lần lặp thường phải đưa ra các vật phẩm như đặc tả... để tiếp tục các bước sau.
  - Đôi khi rất khó phân hoạch các hoạt động phát triển trong dự án thành các bước trong mô hình.

Mô hình thác nước





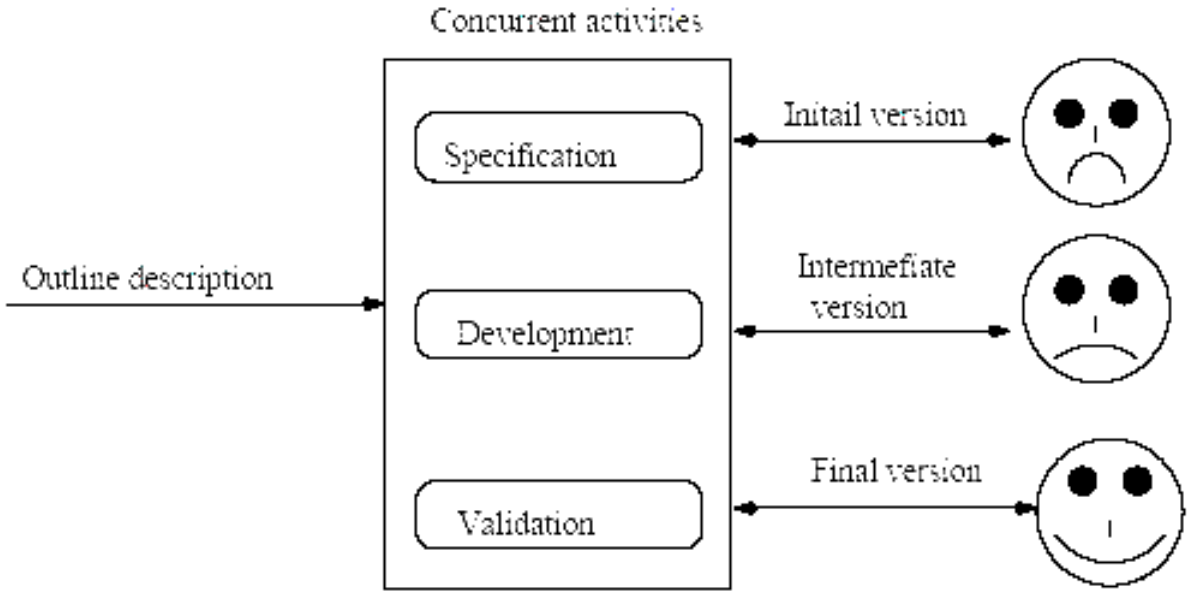


Phát triển tiến hóa

- Vấn đề của mô hình thác nước
  - Một vài dự án phát triển phần mềm rất khó phân hoạch thành các giai đoạn khác nhau như phân tích yêu cầu, thiết kế...
  - Đôi khi rất khó khăn trong việc hình thành đặc tả chi tiết yêu cầu
- Tiến trình phát triển tiến hóa (Evolutionary Development)
  - Dựa trên ý tưởng phát triển mã trình khởi đầu
  - Thu thập ý kiến người sử dụng
  - Làm mịn dần thông qua nhiều phiên bản cho đến khi có hệ thống hoàn chỉnh
  - Cho phép phát triển đồng thời các hoạt động phát triển phần mềm

Phát triển tiến hóa

- Tiến trình phát triển bắt đầu từ mô tả outline hệ thống
- Không phân chia tách biệt thành các hoạt động đặc tả, phát triển (thiết kế, cài đặt) và đánh giá (thử nghiệm hoặc/và kiểm chứng hoặc/và làm prototyping)
- Thực hiện tương tranh với phản hồi các hoạt động phát triển phần mềm



Phát triển tiến hóa

- Các kỹ thuật sử dụng trong phát triển tiến hóa
  - Lập trình thăm dò (Exploratory programming)
    - Làm việc cùng khách hàng để thăm dò các yêu cầu của họ và dãu bày hệ thống cuối cùng
    - Phát triển bắt đầu từ những phần của hệ thống đã hiểu rõ ràng
    - Hệ thống tiến hóa bằng bổ sung các đặc trưng mới do khách hàng đề xuất
  - Prototyping
    - Mục đích là để hiểu yêu cầu khách hàng
    - Prototype tập trung vào thực nghiệm những phần yêu cầu của khách hàng mà chưa được hiểu rõ

Phát triển tiến hóa

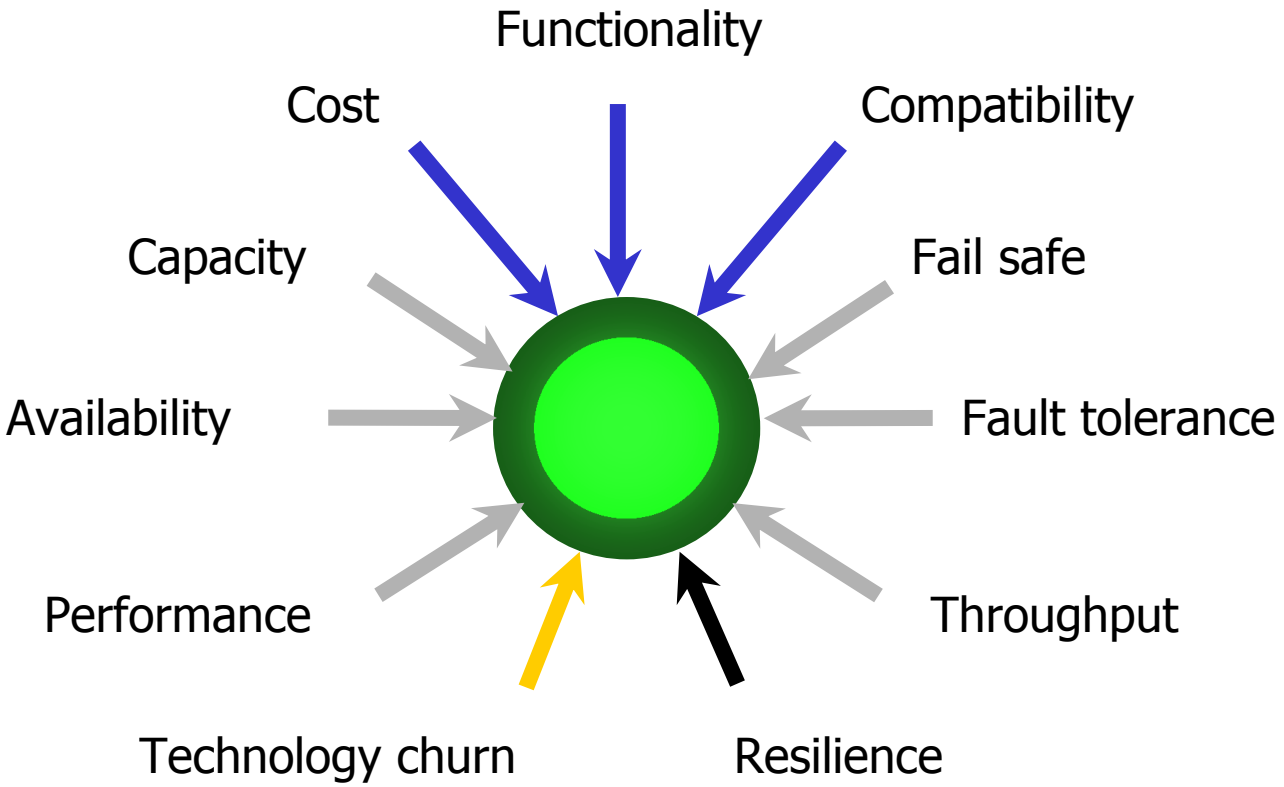
- Vấn đề của các hoạt động phát triển trong tiến trình này
  - Tiến trình không rõ ràng
    - Rất khó hình thành tài liệu phản ảnh từng phiên bản của hệ thống
  - Hệ thống không có cấu trúc tốt
    - Thay đổi liên tục kéo theo việc phá hỏng cấu trúc hệ thống
  - Không luôn luôn khả thi
    - Với hệ thống lớn: việc thay đổi ở phiên bản cuối cùng thường là khó khăn hoặc không có thể
    - Yêu cầu mới, đòi hỏi mới đòi hỏi người phát triển bắt đầu lại toàn bộ dự án
    - Prototyping thường xuyên rất tốn kém
- Tiến hóa phần mềm có thể là khó khăn và đắt

Phát triển tiến hóa

- Khuyến cáo ứng dụng mô hình phát triển tiến hóa
  - Phát triển hệ thống tương đối nhỏ
  - Phát triển hệ thống với vòng đời ngắn
    - Trong trường hợp này vấn đề bảo trì không quan trọng
  - Phát triển hệ thống hay những phần của hệ thống mà chúng không thể biểu diễn trước các đặc tả chi tiết

Các ý tưởng, nguyên tắc và kỹ thuật của tiến trình phát triển tiến hóa luôn có ích và có thể áp dụng trong các pha khác nhau của tiến trình phát triển rộng lớn hơn như hiểu biết và đánh giá yêu cầu trong mô hình thác nước

Phát triển phần mềm theo OO



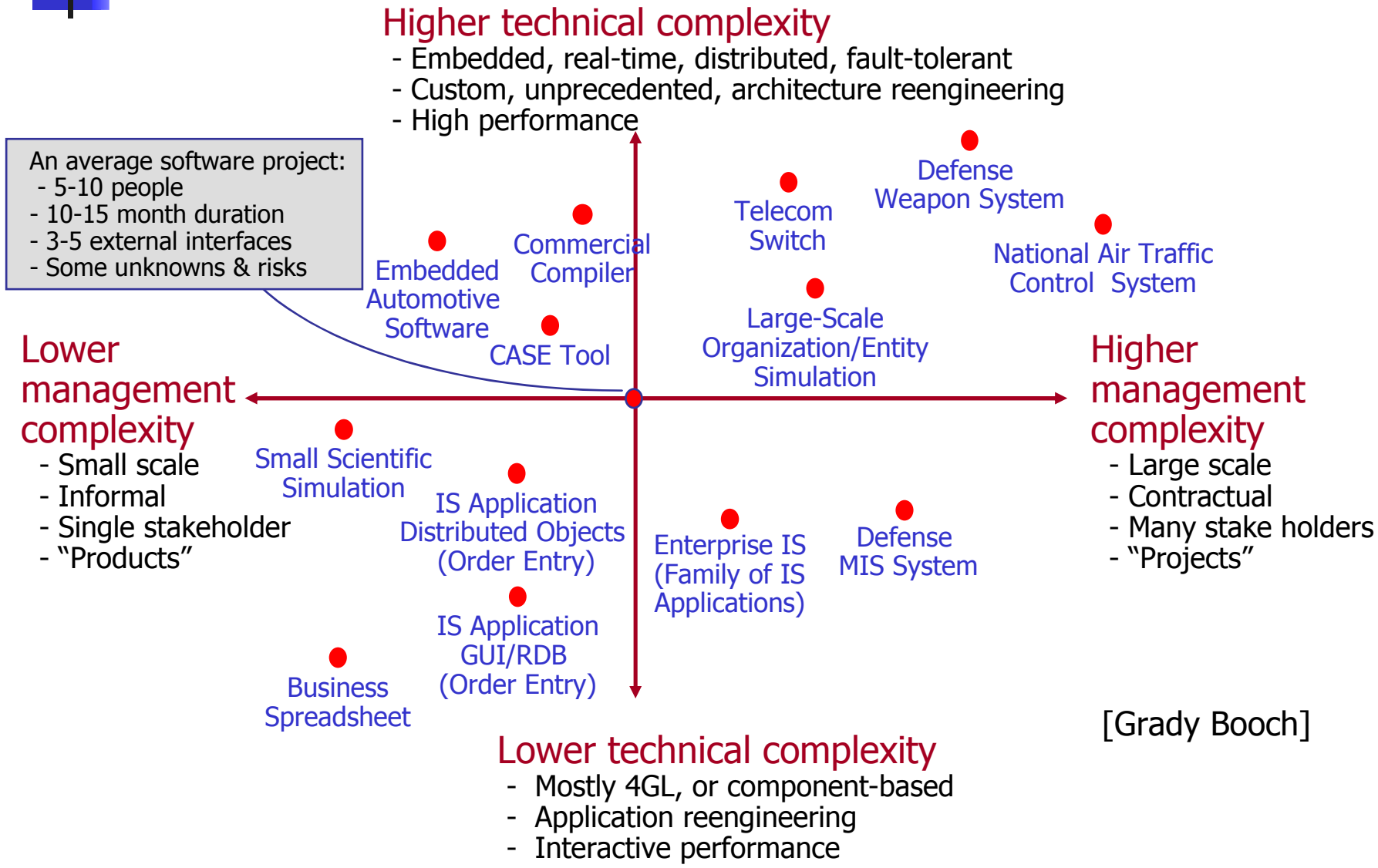
The challenge over the next 20 years will not be speed or cost or performance; it will be a question of complexity.

Bill Raduchel, Chief Strategy Officer, Sun Microsystems

Our enemy is complexity, and it's our goal to kill it.

Jan Baan

Phát triển phần mềm theo OO



[Grady Booch]

Tính phức tạp cố hữu của phần mềm

- Chúng ta vẫn đang trong giai đoạn "khủng hoảng" phần mềm
  - Do tính phức tạp cố hữu của phần mềm
- Tính phức tạp của lĩnh vực vấn đề
  - Xuất phát từ sự không hiểu nhau giữa người sử dụng và người phát triển hệ thống
    - Người sử dụng thường gặp khó khăn khi diễn đạt chính xác nhu cầu dưới hình thức người phát triển có thể hiểu
    - Người sử dụng có thể chỉ có ý tưởng mơ hồ về cái họ muốn có trong hệ thống
  - Các yêu cầu trái ngược nhau: giữa yêu cầu chức năng và yêu cầu phi chức năng
  - Thay đổi yêu cầu thường xuyên khi phát triển hệ thống
- Khó khăn trong quản lý tiến trình phát triển
- Vấn đề xác định đặc điểm hành vi hệ thống

Tính phức tạp cố hữu của phần mềm

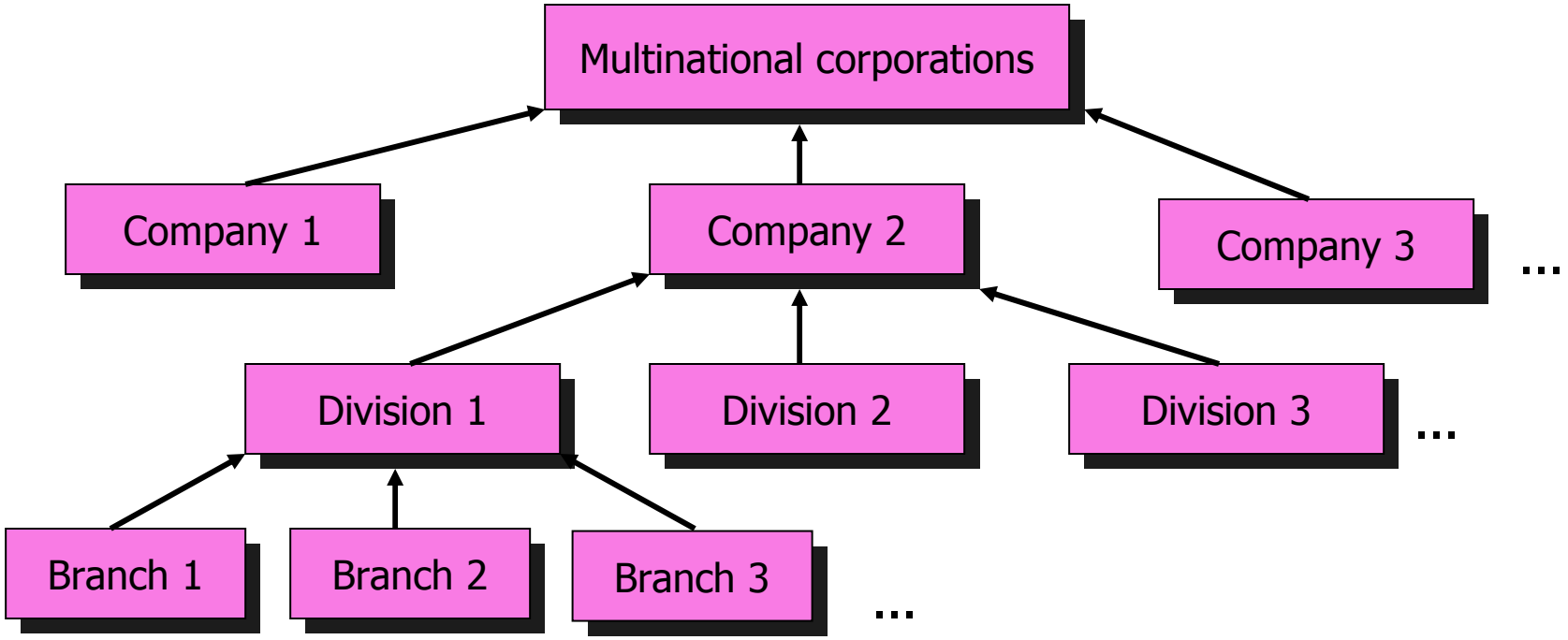
- Tính phức tạp của lĩnh vực vấn đề
- Khó khăn trong quản lý tiến trình phát triển
  - Nhiệm vụ cơ bản của đội ngũ phát triển phần mềm là
    - chỉ ra hình ảnh đơn giản để người sử dụng không bị rối vì độ phức tạp quá lớn của hệ thống
  - Hệ thống lớn và phức tạp đòi hỏi viết hàng nghìn, hàng triệu dòng lệnh
    - Cần có đội ngũ phát triển
  - Nhiều người phát triển
    - giao tiếp phức tạp, điều phối phức tạp hơn
- Vấn đề xác định đặc điểm hành vi hệ thống
  - Trong hệ thống ứng dụng lớn
    - có đến hàng nghìn biến và nhiều luồng điều khiển
  - Hành vi hệ thống là nó thay đổi thế nào từ trạng thái này sang trạng thái khác
    - Tổng số trạng thái rất lớn
    - Mỗi sự kiện bên ngoài có thể làm thay đổi trạng thái hệ thống
    - Hệ thống phản ứng với sự kiện ngoài một cách không xác định trước

Làm chủ hệ thống phức tạp

- Nhiệm vụ cơ bản của kỹ nghệ phần mềm là làm chủ độ phức tạp trong tiến trình phát triển phần mềm
- Thí dụ hệ thống phức tạp
  - Máy vi tính
    - Máy tính PC tương đối phức tạp
    - Có thể phân rã thành các đơn vị
    - Các đơn vị được phân rã thành các linh kiện...
  - Bản chất phân cấp của hệ thống phức tạp
    - Máy PC hoạt động được nhờ các hoạt động cộng tác của các đơn vị thành phần
    - Các tầng phân cấp biểu diễn mức độ trừu tượng khác nhau, mỗi tầng hình thành trên cơ sở tầng khác
    - Tại mỗi tầng trừu tượng ta tìm ra các bộ phận hợp tác để hình thành các dịch vụ cho tầng cao hơn
    - Tầng lựa chọn để nghiên cứu phụ thuộc nhu cầu hiện tại

Làm chủ hệ thống phức tạp

- Thí dụ hệ thống phức tạp
  - Tổ chức xã hội
    - Là những nhóm người liên kết với nhau để thực hiện nhiệm vụ mà từng nhóm riêng lẻ không thể hoàn thành
    - Cấu trúc của tổ chức lớn là phân cấp, thí dụ công ty đa quốc gia

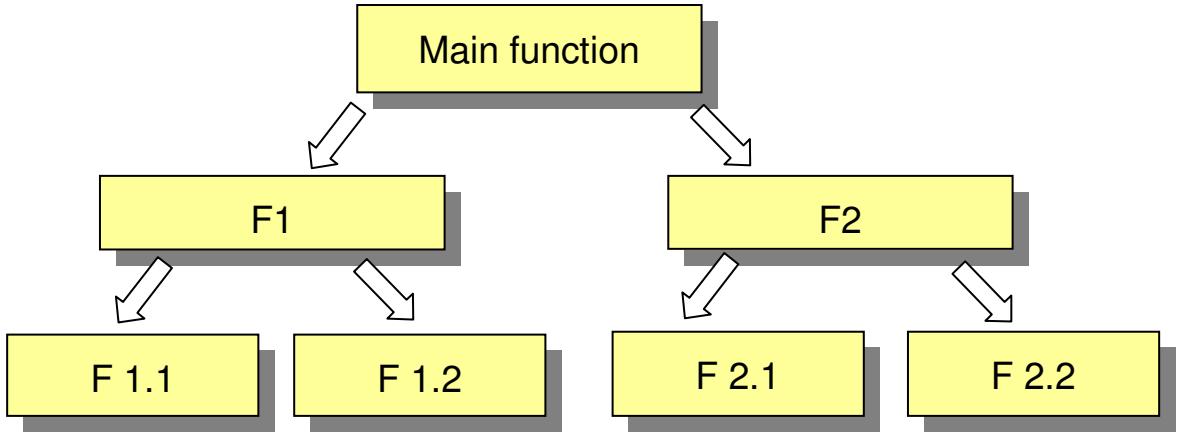


Năm tính chất của hệ thống phức tạp

- Tính phức tạp có hình thức phân cấp
  - do vậy, hệ thống phức tạp được hình thành từ các phân hệ quan hệ với nhau, chúng lại có các phân hệ nhỏ hơn cho đến mức thấp nhất là các thành phần cơ sở.
- Việc chọn thành phần nào làm cơ sở trong hệ thống là tương đối tùy ý
  - phụ thuộc vào người quan sát hệ thống.
- Kết nối bên trong thành phần mạnh hơn kết nối giữa các thành phần
  - Các thành phần trong hệ thống không hoàn toàn độc lập
  - Hiểu biết liên kết giữa các thành phần của hệ thống là quan trọng.
- Thông thường các hệ thống phân cấp hình thành từ vài loại phân hệ khác nhau, theo các tổ hợp và sắp xếp khác nhau
  - Các hệ thống phức tạp có mẫu chung trong việc xây dựng và phát triển.
- Mọi hệ thống phức tạp được tiến hóa từ hệ thống đơn giản
  - Hệ thống phức tạp luôn tiến hóa theo thời gian. Các đối tượng được xem là hệ thống phức tạp sẽ trở thành các đối tượng cơ sở để hình thành hệ thống phức tạp.

Phương pháp hướng chức năng

- Cho đến giữa 1990: Phần lớn các kỹ sư phần mềm sử dụng phương pháp thiết kế chức năng top-down (thiết kế kiến trúc)
  - Bị ảnh hưởng bởi các ngôn ngữ lập trình ALGOL, Pascal, C
  - Các hàm của hệ thống phần mềm được xem như tiêu chí cơ sở khi phân dã
  - Tách chức năng khỏi dữ liệu
    - Chức năng có hành vi
    - Dữ liệu chứa thông tin bị các chức năng tác động
  - Phân tách top-down chia hệ thống thành các hàm để chuyển sang mã trình, dữ liệu được gửi giữa chúng.



Phương pháp hướng chức năng

- Tiến trình phát triển tập trung vào thông tin mà hệ thống quản lý
  - Người phát triển hệ thống hỏi người sử dụng cần thông tin gì
  - Thiết kế CSDL để lưu trữ thông tin
  - Xây dựng màn hình nhập liệu
  - Hiển thị báo cáo
- Chỉ tập trung vào thông tin, ít quan tâm đến cái gì thực hiện với thông tin hay hành vi hệ thống
- Tiệm cận này gọi là tiệm cận hướng dữ liệu
  - Đã được áp dụng nhiều năm và tạo ra hàng ngàn hệ thống
  - Thuận tiện cho thiết kế CSDL
  - Bất tiện cho xây dựng các hệ thống tác nghiệp
    - yêu cầu hệ thống thay đổi theo thời gian

Phương pháp hướng chức năng

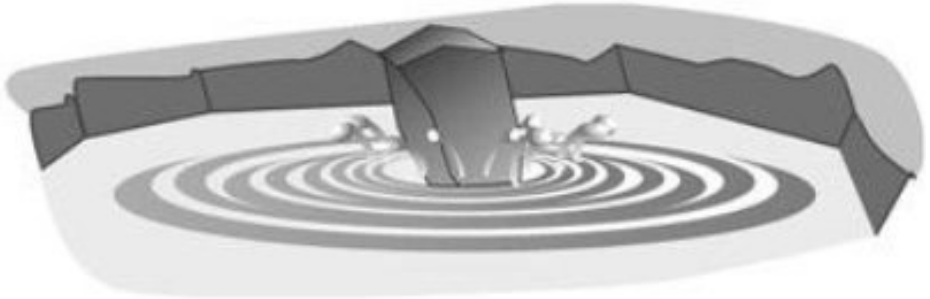
- Công nghệ hướng chức năng có các hạn chế sau
  - Sản phẩm hình thành từ giải pháp này khó bảo trì
    - Mọi chức năng đều chia sẻ khối dữ liệu lớn
    - Các chức năng phải hiểu rõ dữ liệu được lưu trữ thế nào
    - Khi thay đổi cấu trúc dữ liệu kéo theo thay đổi mọi hàm liên quan
  - Tiến trình phát triển không ổn định
    - Thay đổi yêu cầu kéo theo thay đổi các chức năng
    - Rất khó bảo toàn kiến trúc thiết kế ban đầu khi hệ thống tiến hóa
  - Tiệm cận này không hỗ trợ lập trình bằng ngôn ngữ hướng đối tượng như C++, Java, Smalltalk, Eiffel.

Phương pháp hướng đối tượng

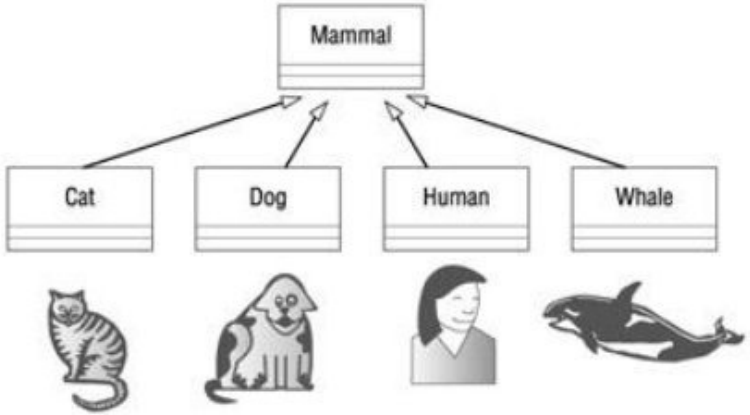
- Chiến lược phát triển phần mềm hướng đối tượng là quan sát thế giới như tập các đối tượng
  - Các đối tượng tương tác và cộng tác với nhau để hình thành hành vi mức cao
- Các tính chất của đối tượng
  - Đối tượng có thể là
    - thực thể nhìn thấy được trong thế giới thực (trong pha phân tích yêu cầu)
    - biểu diễn thực thể hệ thống (trong pha thiết kế)
  - Đối tượng có trách nhiệm quản lý trạng thái của mình, cung cấp dịch vụ cho đối tượng khác khi có yêu cầu
    - do vậy, dữ liệu và hàm cùng gói trong đối tượng
  - Chức năng hệ thống:
    - các dịch vụ được yêu cầu và cung cấp như thế nào giữa các đối tượng, không quan tâm đến thay đổi trạng thái bên trong đối tượng
  - Các đối tượng được phân thành class
    - Các đối tượng thuộc cùng lớp đều có đặc tính (thuộc tính và thao tác) chung

Phương pháp hướng đối tượng

- Tiệm cận hướng đối tượng tập trung vào cả thông tin và hành vi
- Cho khả năng xây dựng hệ thống mềm dẻo, "co dãn"
- Phương pháp này dựa trên các nguyên tắc sau
  - Tính gói
  - Kế thừa
  - Đa trị

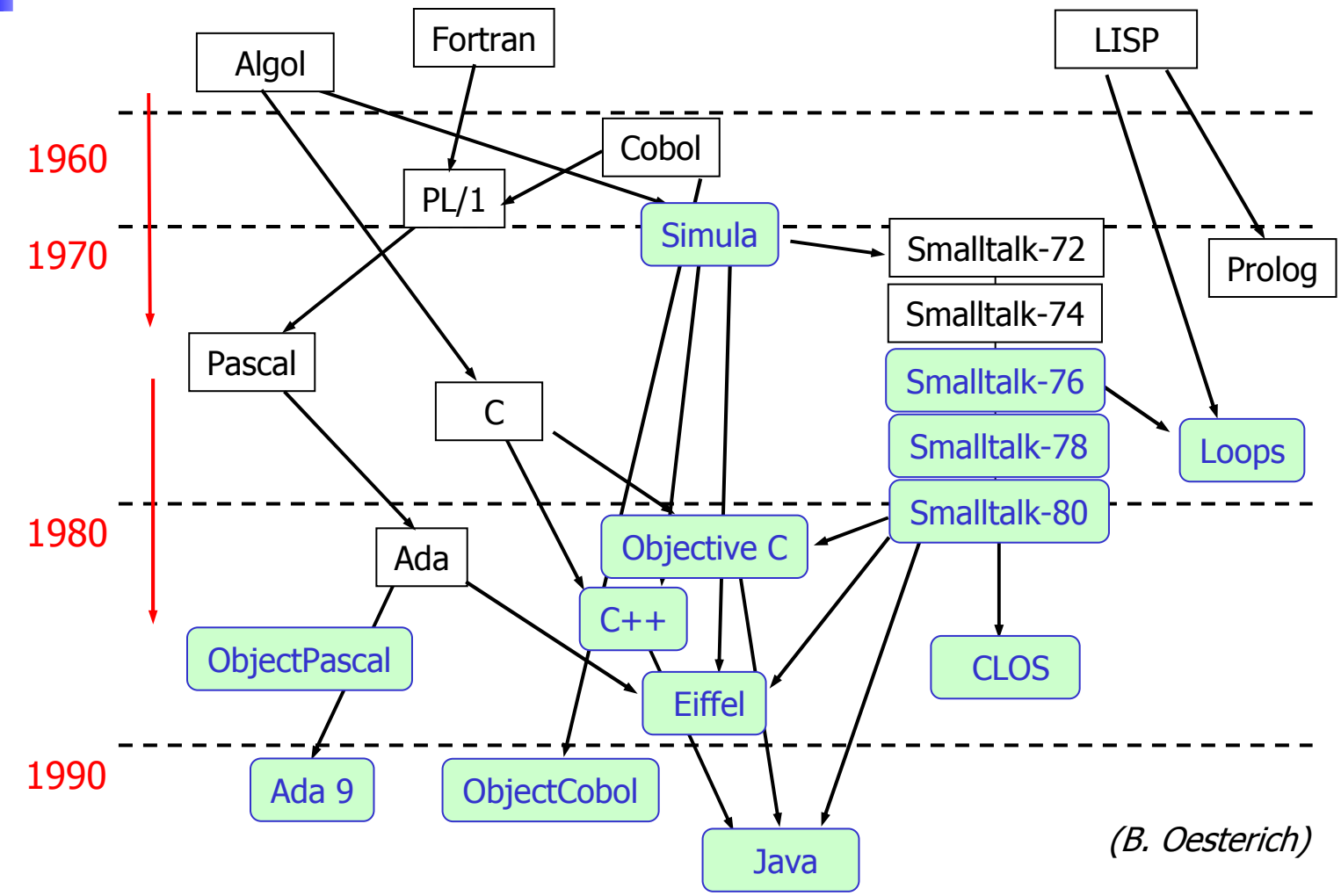


Lake Model



Natural Model

Ngôn ngữ lập trình



Thí dụ tiến trình lặp và tăng dần

- Tiến trình thống nhất (Rational Unified Process - RUP)
  - Là Software Engineering process
  - Là sản phẩm tiến trình (process product) do Rational Software phát triển và bảo trì
  - RUP nâng cao team productivity
  - Các hoạt động RUP tạo lập và quản lý models
  - Là hướng dẫn cách sử dụng hiệu quả UML
  - Được nhiều công cụ hỗ trợ, chúng tự động phần lớn tiến trình
  - Là configurable process
    - Không một tiến trình nào là phù hợp cho mọi công việc phát triển phần mềm.
    - Phù hợp với các đội ngũ phát triển nhỏ và các tổ chức phát triển lớn.

Các khái niệm cơ bản của RUP

- Phase, Iterations

When does architecture happen?

- Process Workflows
  - Activity, steps

What does happen?

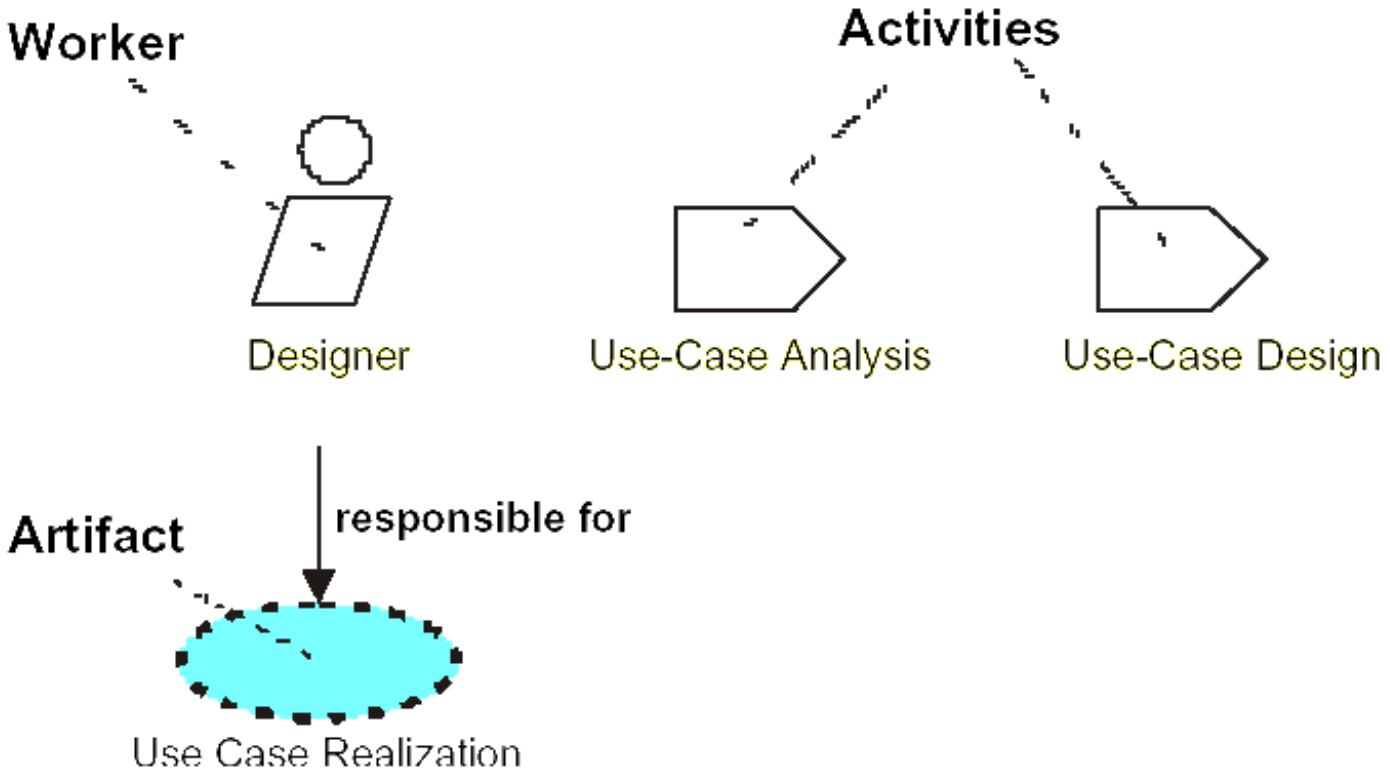
- Artifacts
  - models
  - reports, documents

What is produced?

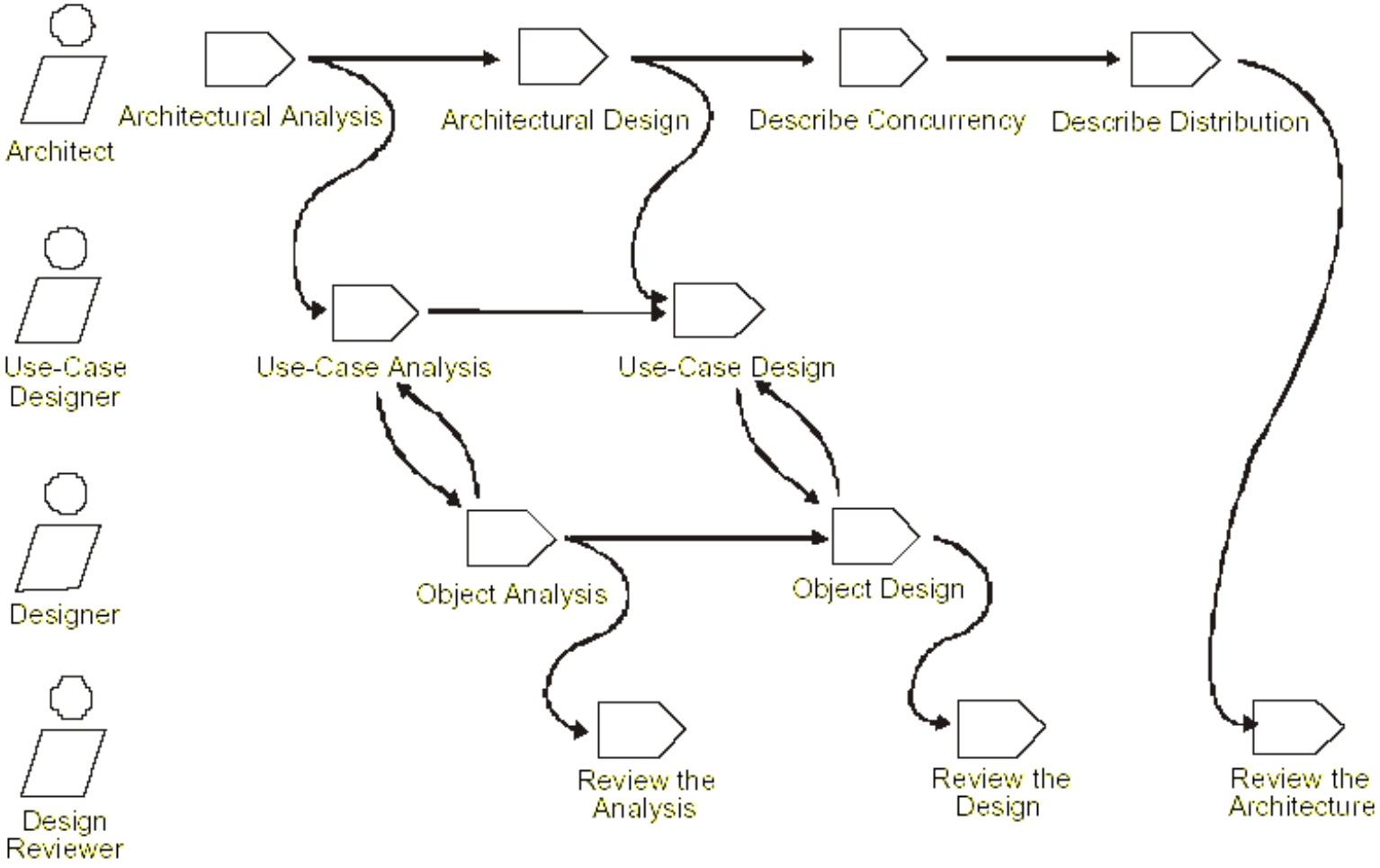
- Worker: Architect

Who does it?

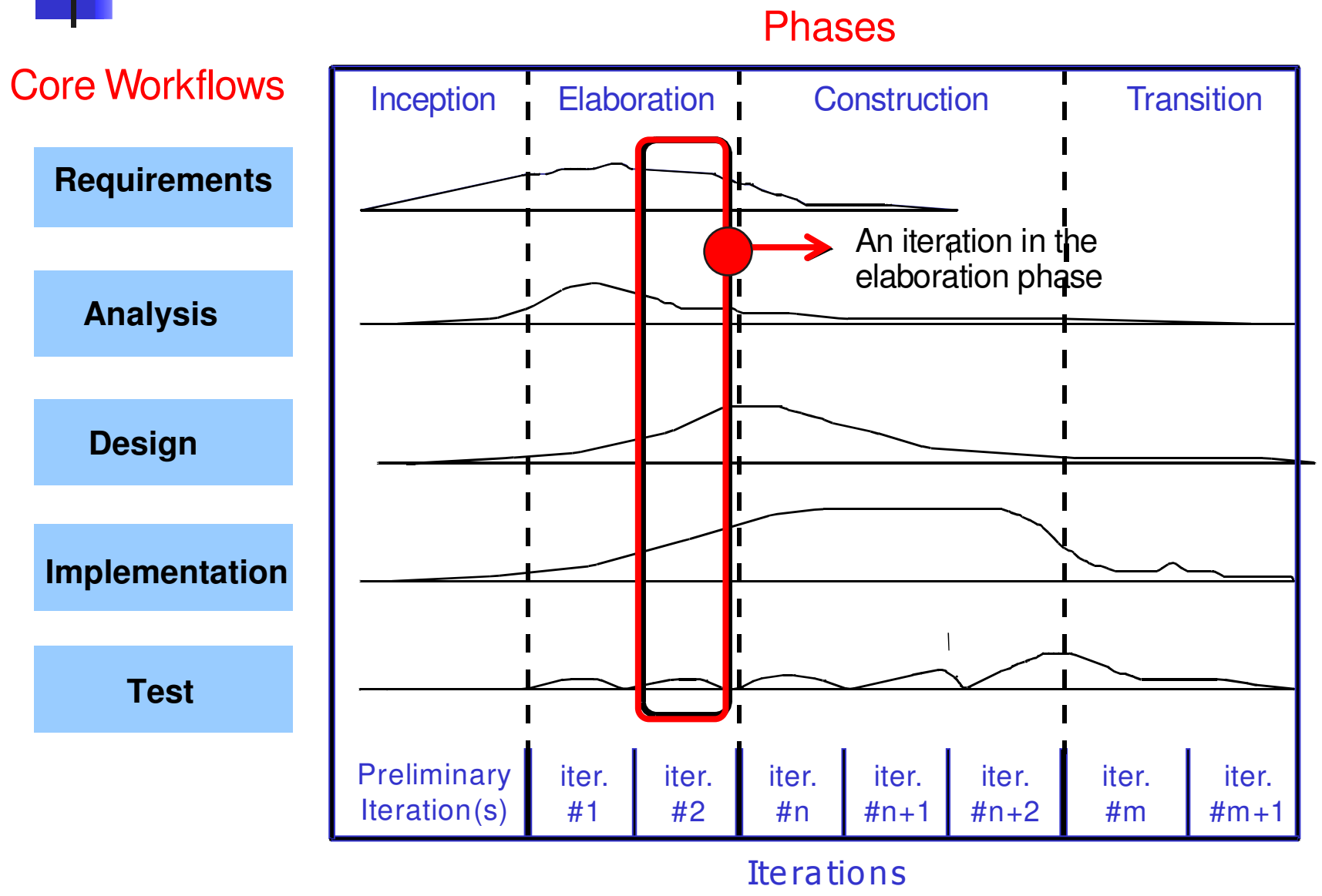
Worker-Activities-Artifact



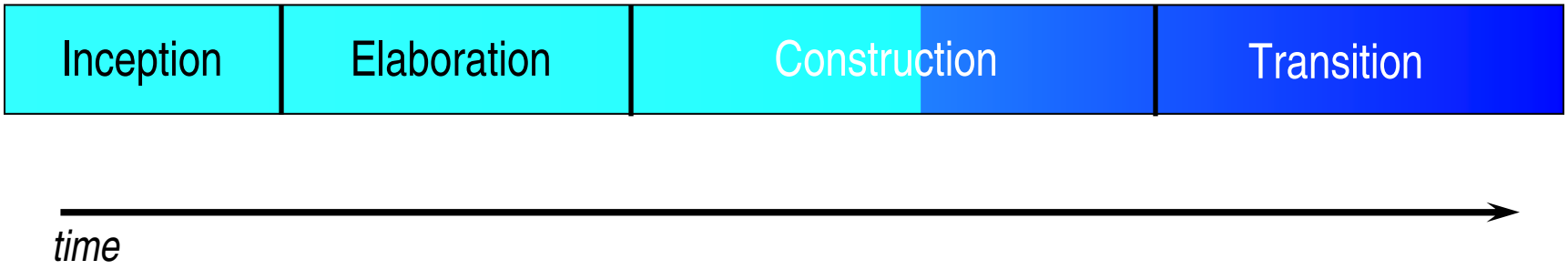
Thí dụ luồng công việc



Các lặp và luồng công việc

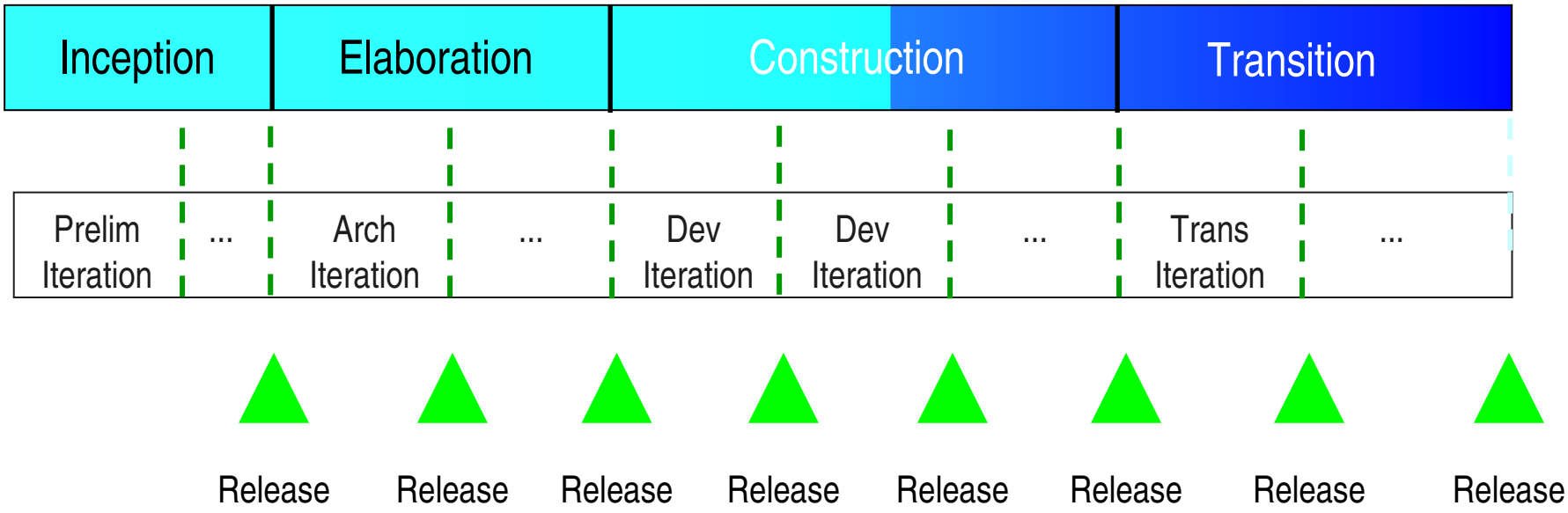


Các pha của vòng đời



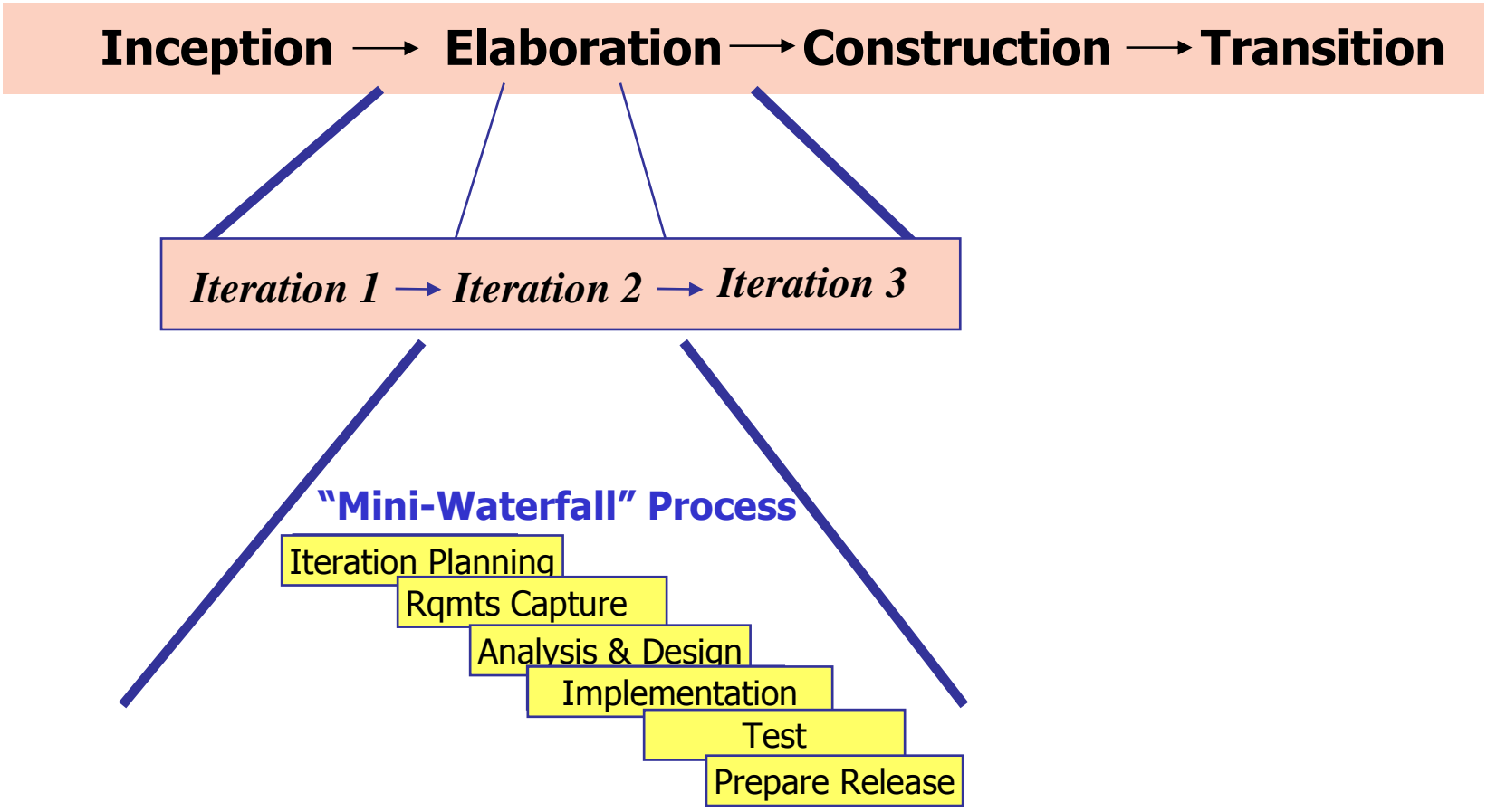
- **Inception** — Define the scope of the project and develop business case
- **Elaboration** — Plan project, specify features, and baseline the architecture
- **Construction** — Build the product
- **Transition** — Transition the product to its users

Các pha và lặp



Một vòng lặp (iteration) là trình tự các hoạt động với kế hoạch xây dựng trước và tiêu chí đánh giá, cho kết quả là các phiên bản chạy được.

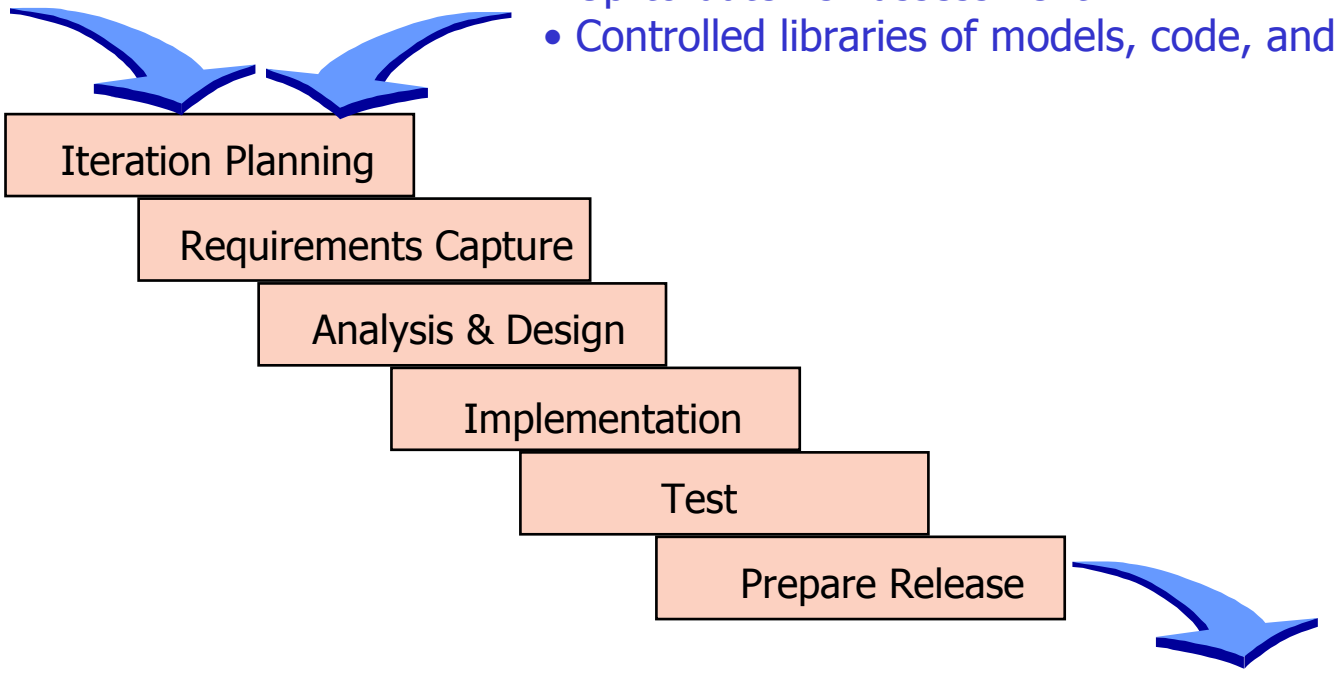
Tiến trình lặp



Vòng đời của lặp: A Mini-Waterfall

Selected scenarios

- Results of previous iterations
- Up-to-date risk assessment
- Controlled libraries of models, code, and tests



Release description
Updated risk assessment
Controlled libraries

Các hoạt động của lặp

- Kế hoạch lặp
  - Trước khi lặp bắt đầu thực hiện, mục tiêu chính của lặp cần được hình thành trên cơ sở
    - Các kết quả của các lặp trước (nếu có)
    - Cập nhật đánh giá rủi ro của dự án
  - Xác định tiêu chí đánh giá cho lặp này
  - Chuẩn bị kế hoạch chi tiết cho lặp
    - Bao gồm intermediate milestones để điều khiển tiến trình
    - Bao gồm walkthroughs và reviews

Các hoạt động của vòng đời lặp

- Requirements Capture
  - Select/define the use cases to be implemented in this iteration
  - Update the object model to reflect additional domain classes and associations discovered
  - Develop a test plan for the iteration
- Analysis & Design
  - Determine the classes to be developed or updated in this iteration
  - Update the object model to reflect additional design classes and associations discovered
  - Update the architecture document if needed
  - Begin development of test procedures
- Implementation
- Test
- Prepare the release description

Các hoạt động của vòng đời lặp

- Requirements Capture
- Analysis & Design
- Implementation
  - Automatically generate code from the design model
  - Manually generate code for operations
  - Complete test procedures
  - Conduct unit and integration tests
- Test
  - Integrate and test the developed code with the rest of the system (previous releases)
  - Capture and review test results
  - Evaluate test results relative to the evaluation criteria
  - Conduct an iteration assessment
- Prepare the release description
  - Synchronize code and design models
  - Place products of the iteration in controlled libraries

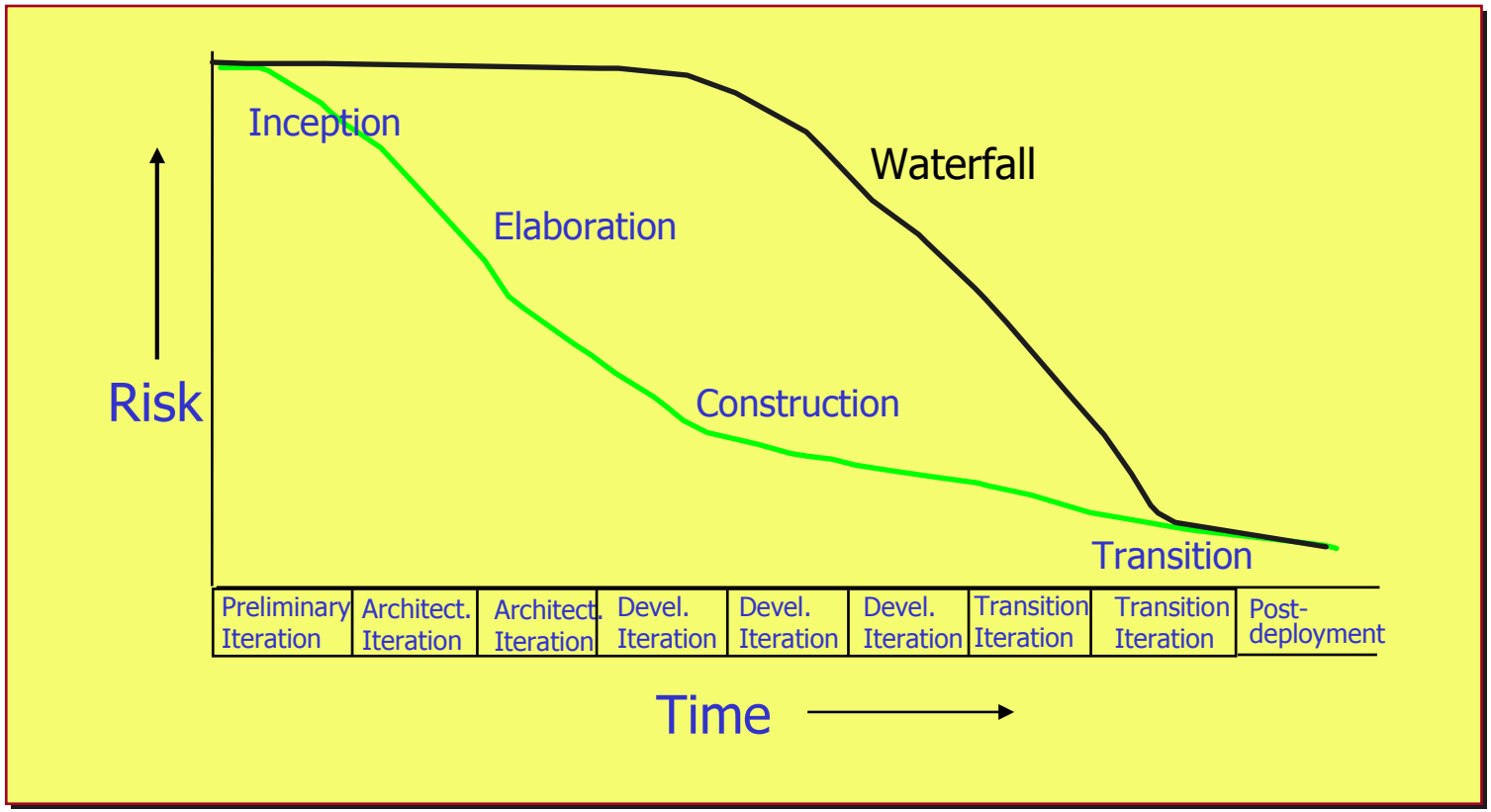
Chọn lựa lặp

- How many iterations do I need?
  - On projects taking 18 months or less, 3 to 6 iterations are typical
- Are all iterations on a project the same length?
  - Usually
  - Iteration length may vary by phase. For example, elaboration iterations may be shorter than construction iterations

Ích lợi của tiệm cận lặp

- Compared to the traditional waterfall process, the iterative process has the following advantages:
  - Risks are mitigated earlier
  - Change is more manageable
  - Higher level of reuse
  - The project team can learn along the way
  - Better overall quality

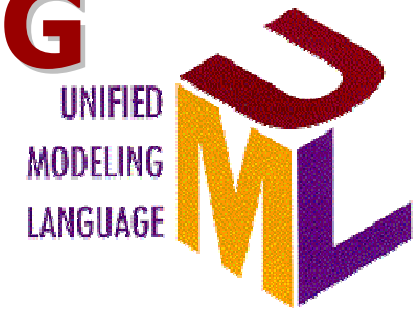
Biểu đồ rủi ro của tiến trình lặp



Tóm tắt

- Các vấn đề đã nghiên cứu
  - Khái quát về tiến trình phát triển phần mềm
  - Các hoạt động chính trong phát triển phần mềm
  - Mô hình thác nước của tiến trình phát triển phần mềm
  - Phát triển tiến hóa
  - Tính phức tạp cố hữu của phần mềm
  - Phát triển hệ thống theo phương pháp hướng đối tượng
  - Giới thiệu RUP

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



NỘI DUNG

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. ✓ *Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML*
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Bài 2

Giới thiệu
Ngôn ngữ mô hình hóa thống nhất

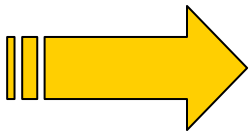
Mô hình là gì?

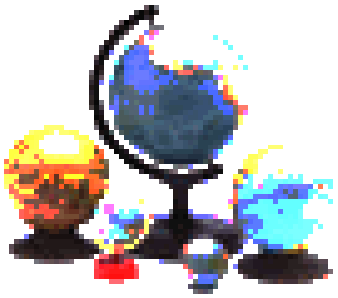
- Mô hình
  - là bức tranh hay mô tả vấn đề đang cố gắng giải quyết hay mô tả chính giải pháp vấn đề
  - là ngôn ngữ của người thiết kế (trong nhiều lĩnh vực)
  - là trình diễn hệ thống sẽ xây dựng
  - là phương tiện giao tiếp giữa các stakeholders
  - là kế hoạch chi tiết (blueprints)
- Mô hình cho khả năng suy diễn một số đặc tính của hệ thống thực
- Mô hình hóa trực quan
  - Bằng các phần tử đồ họa
- Ngôn ngữ mô hình hóa là ngôn ngữ mô tả hệ thống hay tác nghiệp

An abstraction is an intellectual simplification

Thí dụ mô hình

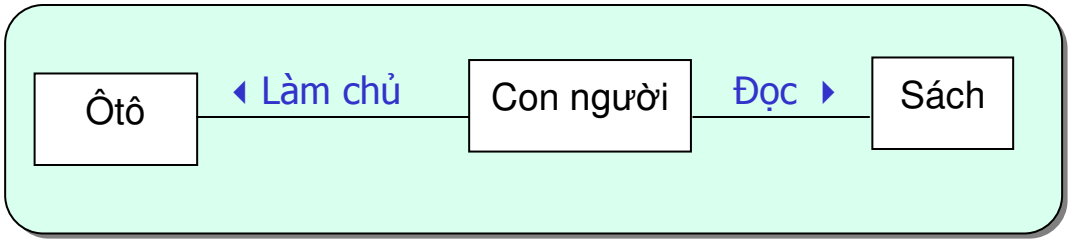
Thế giới thực





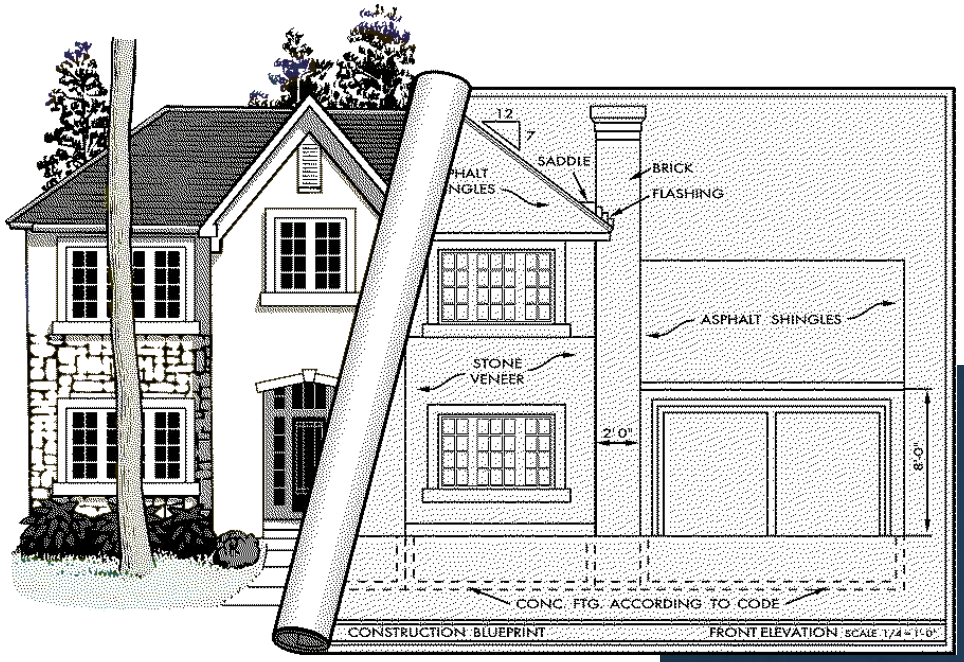
Mô hình: Quả địa cầu học sinh

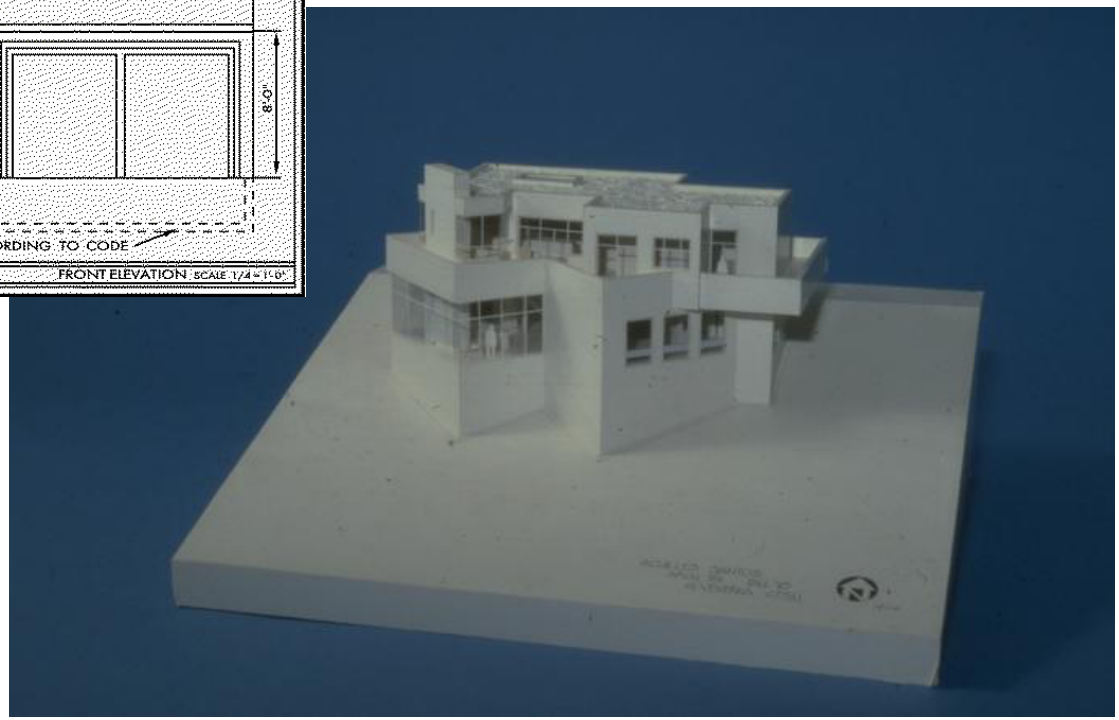
Thế giới thực



Mô hình

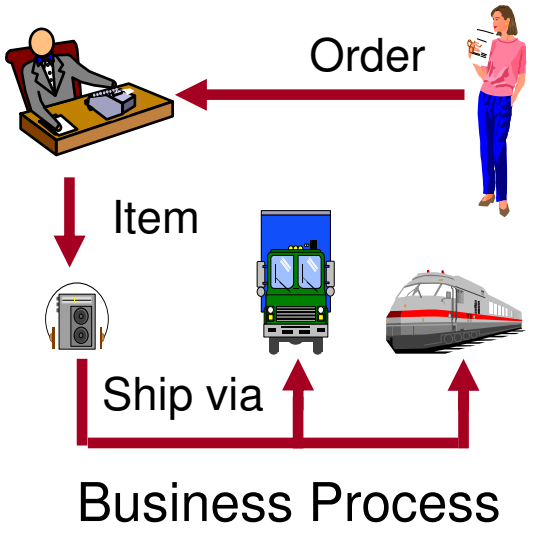
Thí dụ mô hình





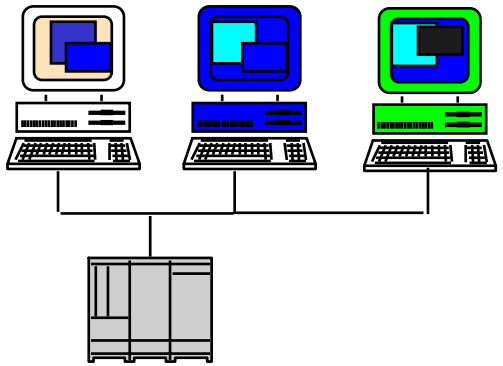
A **model** is a complete description of a system from a particular perspective

Mô hình hóa trực quan?



"Modeling captures essential parts of the system."

Dr. James Rumbaugh

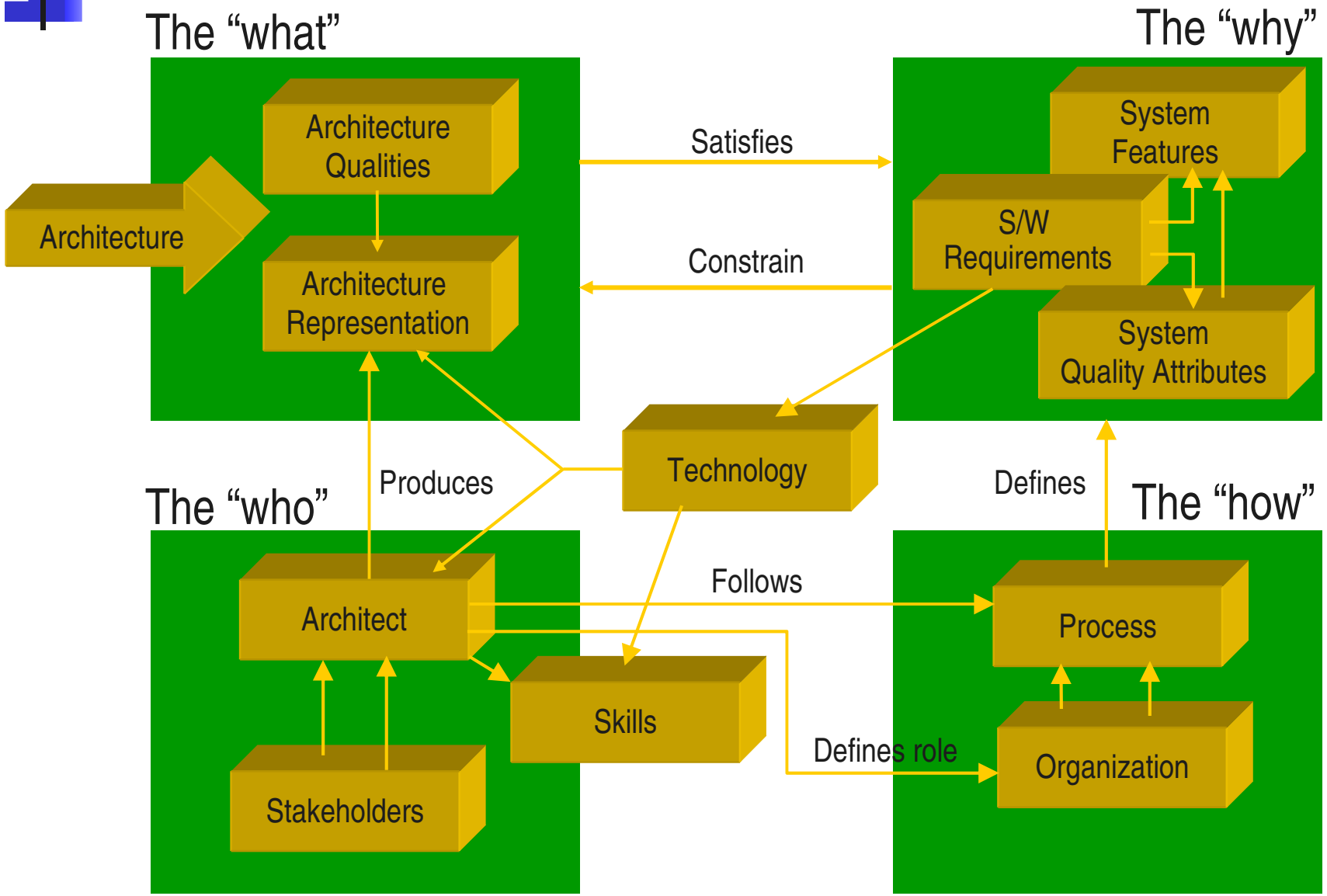


Visual Modeling is modeling using standard graphical notations

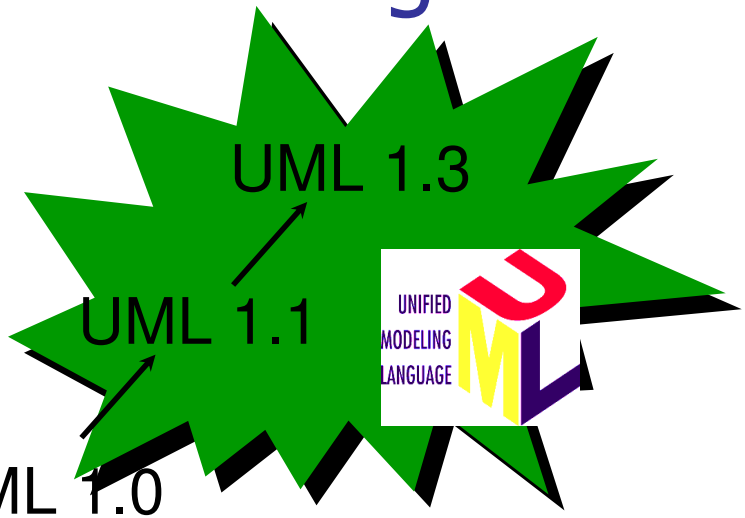
Bốn nguyên tắc mô hình hóa

- Việc chọn mô hình nào để tạo lập có ảnh hưởng sâu sắc đến cách giải quyết vấn đề và cách hình thành các giải pháp
- Mỗi mô hình biểu diễn hệ thống với mức độ chính xác khác nhau
- Mô hình tốt nhất phải là mô hình phù hợp với thế giới thực
- Không mô hình nào là đầy đủ. Mỗi hệ thống thường được tiếp cận thông qua tập mô hình gần như độc lập nhau.

Thiết kế kiến trúc



Ngôn ngữ mô hình hóa thống nhất



OMG Acceptance, Nov 1997

Final submission to OMG, Sep '97

public feedback

First submission to OMG, Jan ´97

UML partners

UML 1.0

June ´96

UML 0.9

June 12, 2003
UML 2.0

OOPSLA ´95

Unified Method 0.8

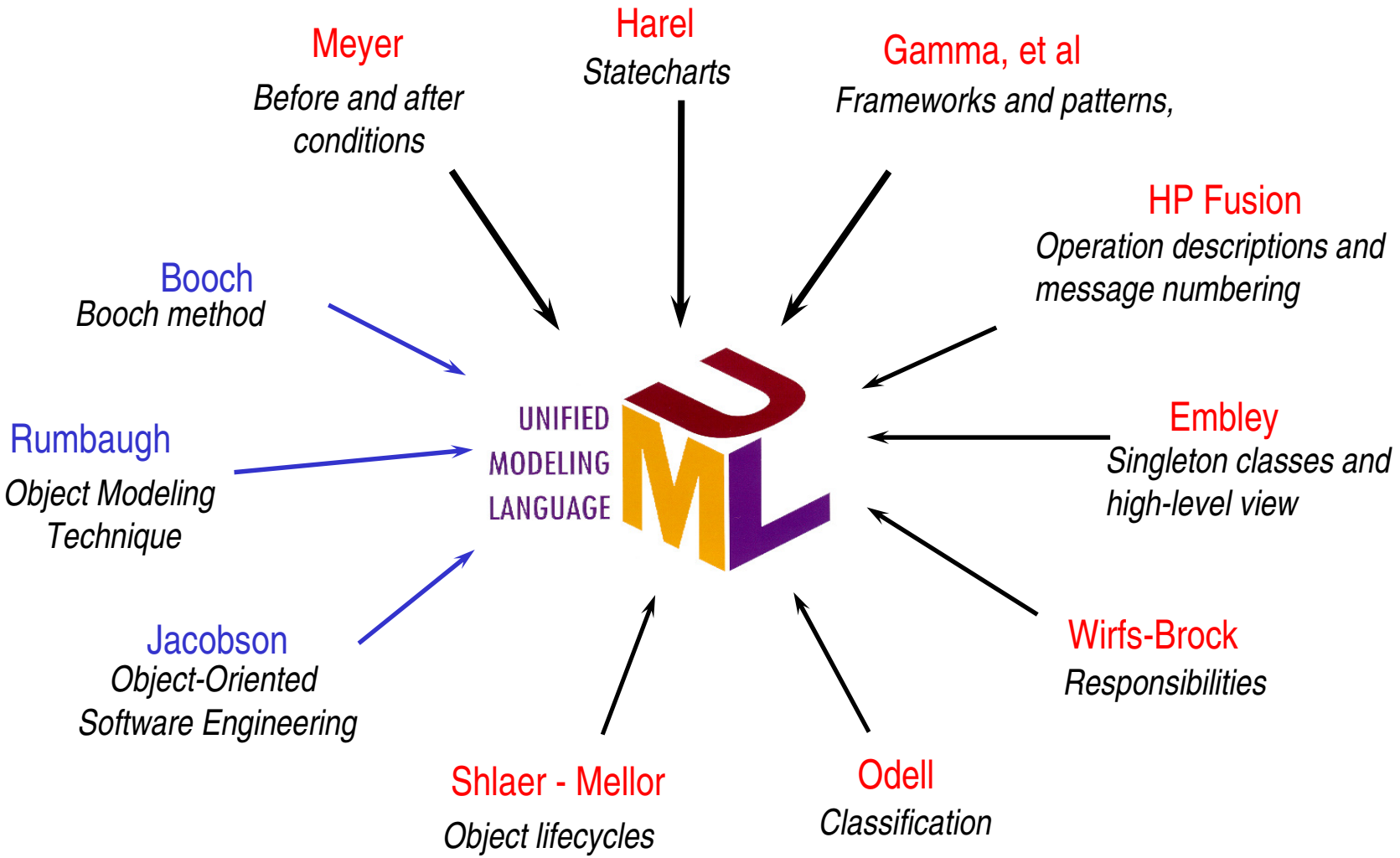
Other methods Booch method OMT OOSE

UML stands for Unified Modeling Language

UML Partners

- Rational Software Corporation
- Hewlett-Packard
- I-Logix
- IBM
- ICON Computing
- Intellicorp
- MCI Systemhouse
- Microsoft
- ObjecTime
- Oracle
- Platinum Technology
- Taskon
- Texas Instruments/Sterling Software
- Unisys

Contributions to the UML



Khái quát về UML

- UML là ngôn ngữ để
  - visualizing
  - specifying
  - constructing
  - documenting

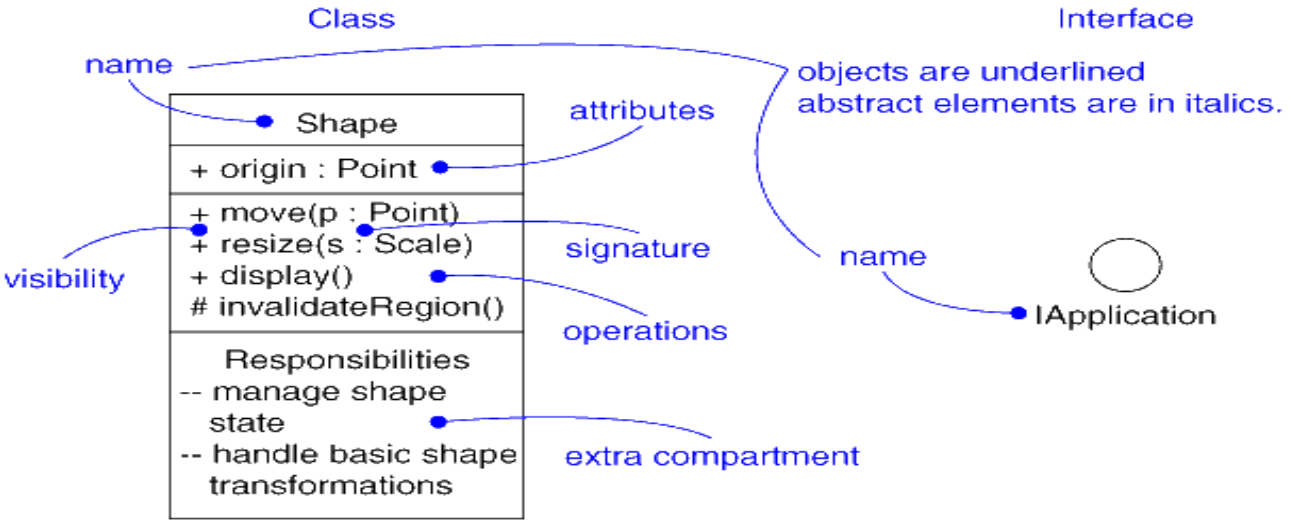
  các vật phẩm (artifacts) của hệ thống phần mềm
- Nó có thể sử dụng trong mọi tiến trình, xuyên suốt vòng đời phát triển và trải qua các công nghệ cài đặt khác nhau.

Khái quát về UML

- Mô hình hóa các phần tử
- Các quan hệ
- Cơ chế mở rộng
- Các biểu đồ

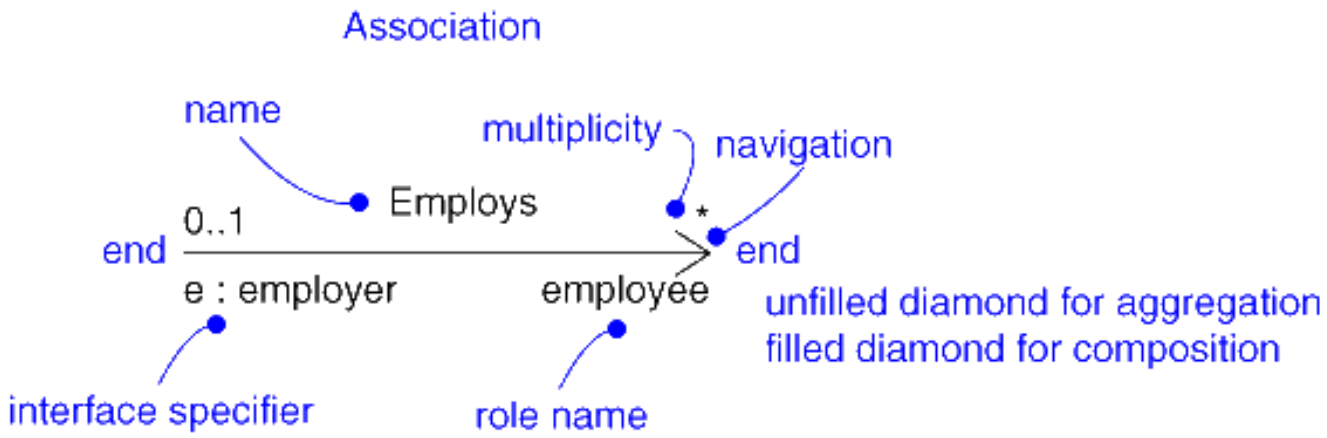
Mô hình hóa các phần tử

- Các phần tử cấu trúc
  - class, interface, collaboration, use case, active class, component, node
- Các phần tử hành vi
  - interaction, state machine
- Nhóm các phần tử
  - package, subsystem
- Các phần tử khác
  - note



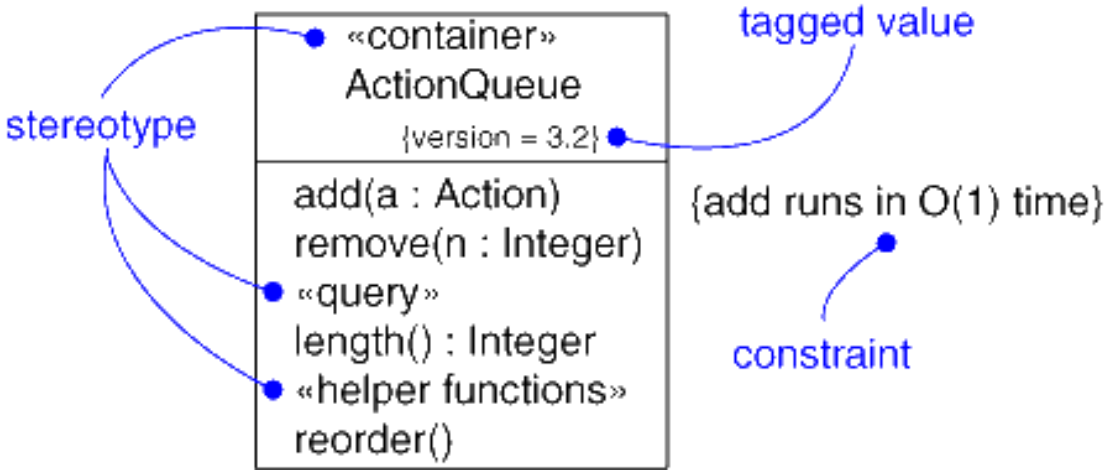
Các quan hệ

- Dependency
- Association
- Generalization
- Realization

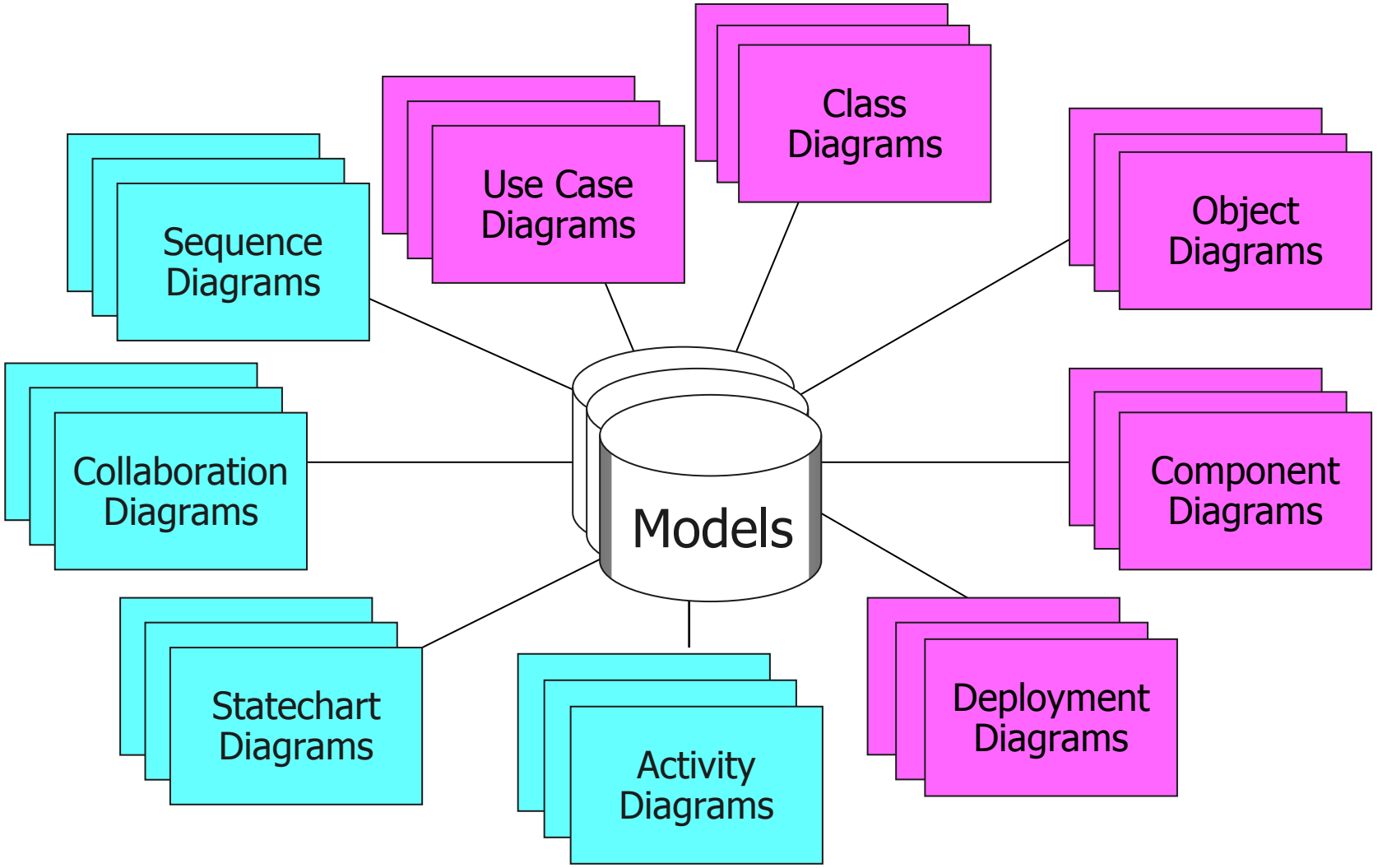


Cơ chế mở rộng

- Stereotype
- Tagged value
- Constraint



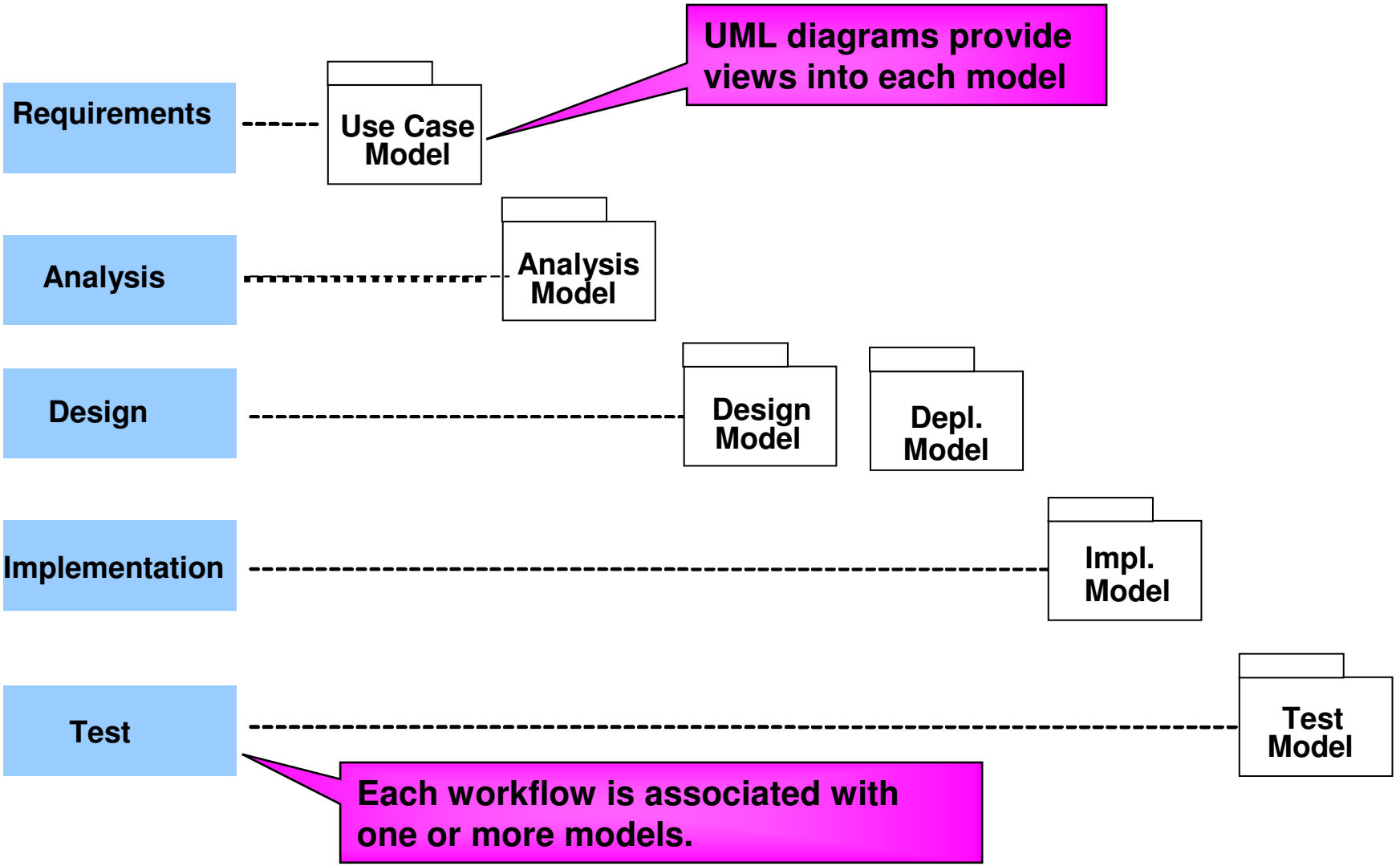
Models and Diagrams



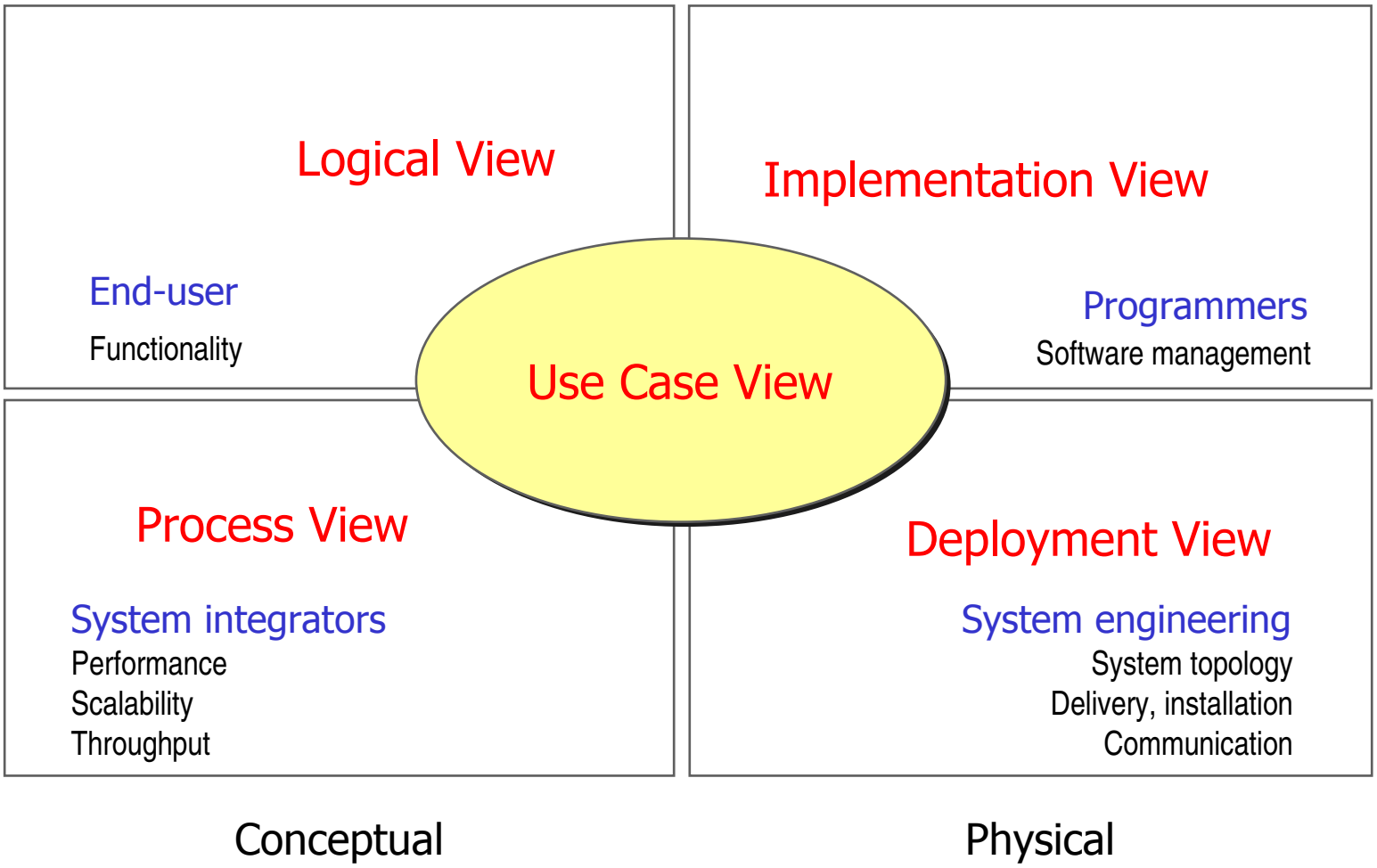
Diagrams

- A diagram is a view into a model
  - Presented from the aspect of a particular stakeholder
  - Provides a partial representation of the system
  - Is semantically consistent with other views
- In the UML, there are nine standard diagrams
  - Static views: use case, class, object, component, deployment
  - Dynamic views: sequence, collaboration, statechart, activity

Workflows and Models



Representing System Architecture



Cần bao nhiêu khung nhìn?

- Mô hình phù hợp với ngữ cảnh phát triển hệ thống
- Không phải tất cả các mô hình đòi hỏi đầy đủ khung nhìn
  - Đơn xử lý: Bỏ qua khung nhìn triển khai
  - Đơn tiến trình: Bỏ qua khung nhìn tiến trình
  - Chương trình rất nhỏ: Bỏ qua khung nhìn cài đặt
- Bổ sung các khung nhìn
  - Data view
  - Security view

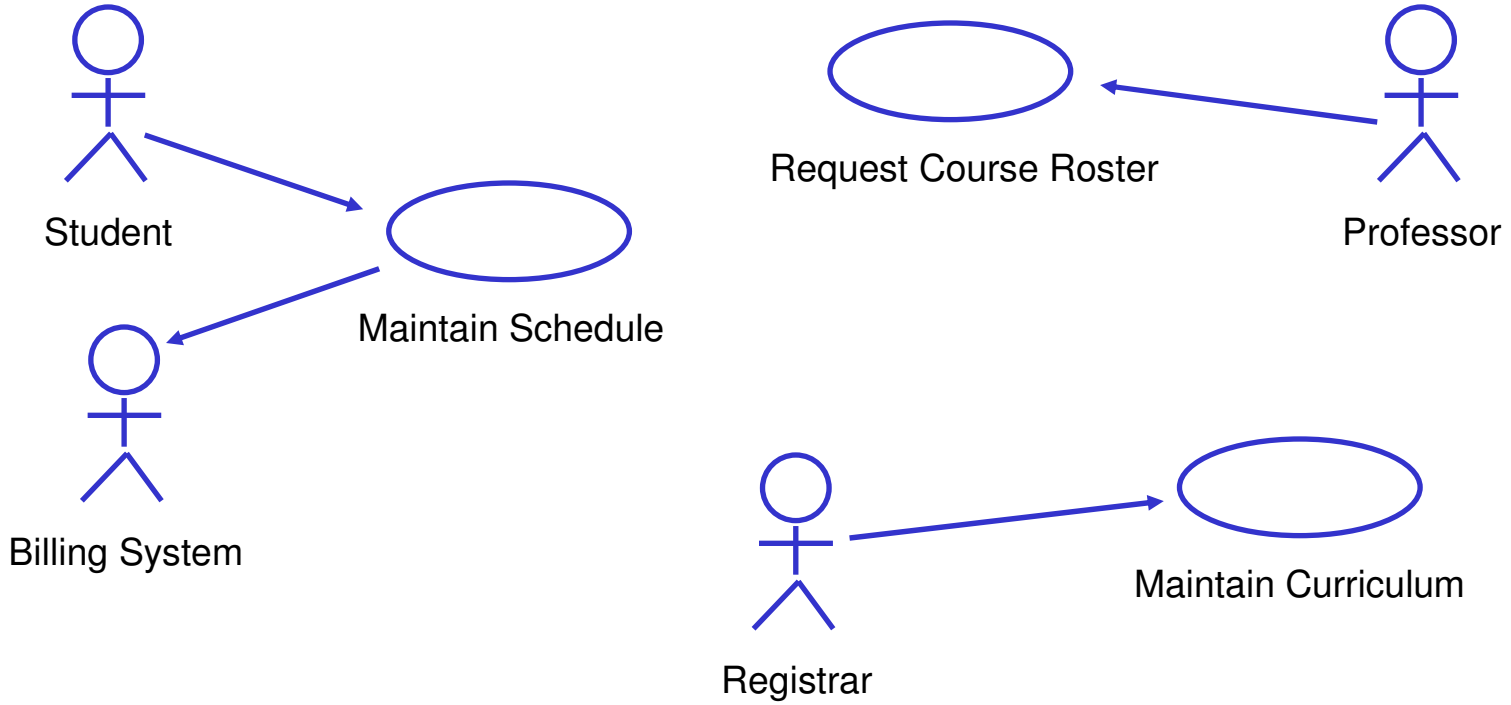
UML Concepts

- UML được sử dụng để:
  - Hiển thị biên hệ thống và các chức năng chính của nó bằng use cases và actors
  - Mô tả hiện thực use case bằng interaction diagrams
  - Biểu diễn các cấu trúc tĩnh của hệ thống bằng class diagrams
  - Mô hình hóa hành vi đối tượng bằng state transition diagrams
  - Biểu thị kiến trúc cài đặt vật lý bằng component & deployment diagrams
  - Mở rộng các chức năng bằng stereotypes

Thí dụ ứng dụng UML

- Một trường đại học thực hiện tin học hóa hệ thống đăng ký học và dạy học:
  - Giáo vụ (Registrar) lập chương trình giảng dạy (curriculum) cho một học kỳ
  - Sinh viên (Student) chọn 4 môn học chính và 2 môn dự bị
  - Khi sinh viên đăng ký học thì hệ thống thanh toán (billing system) in hóa đơn học phí cho sinh viên
  - Sinh viên có thể sử dụng hệ thống để bổ sung/loại bỏ môn học sau khi đã đăng ký (trong khoảng thời gian cố định)
  - Giáo sư (Professors) sử dụng hệ thống để xem bảng phân công dạy học (course rosters)
  - Người sử dụng hệ thống đăng ký được cấp passwords để vào máy

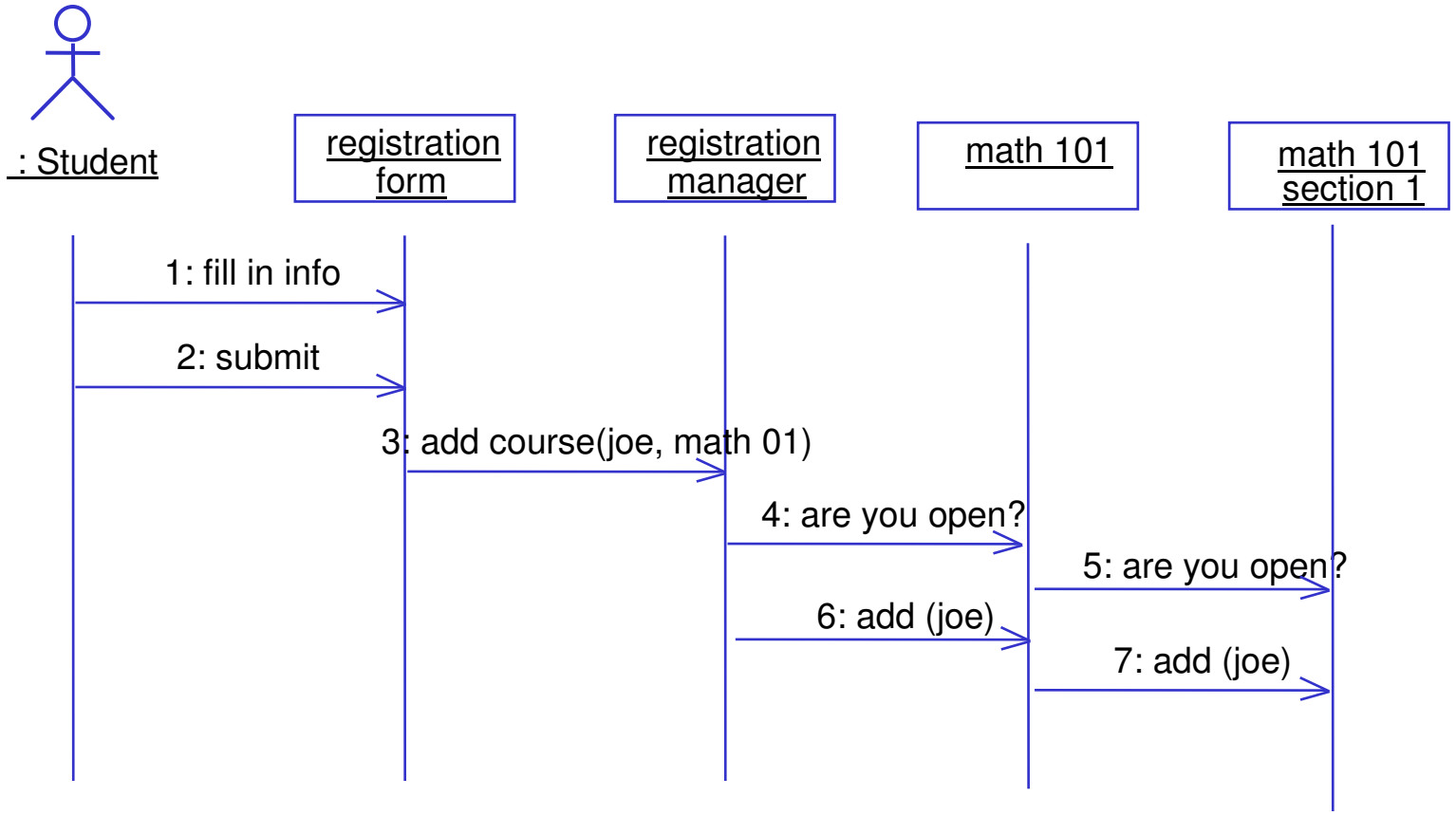
Use case Diagram



Use case Diagram

- Use case diagrams are created to visualize the relationships between actors and use cases
- Captures system functionality as seen by users
- Built in early stages of development
- Purpose
  - Specify the context of a system
  - Capture the requirements of a system
  - Validate a system's architecture
  - Drive implementation and generate test cases
- Developed by analysts and domain experts

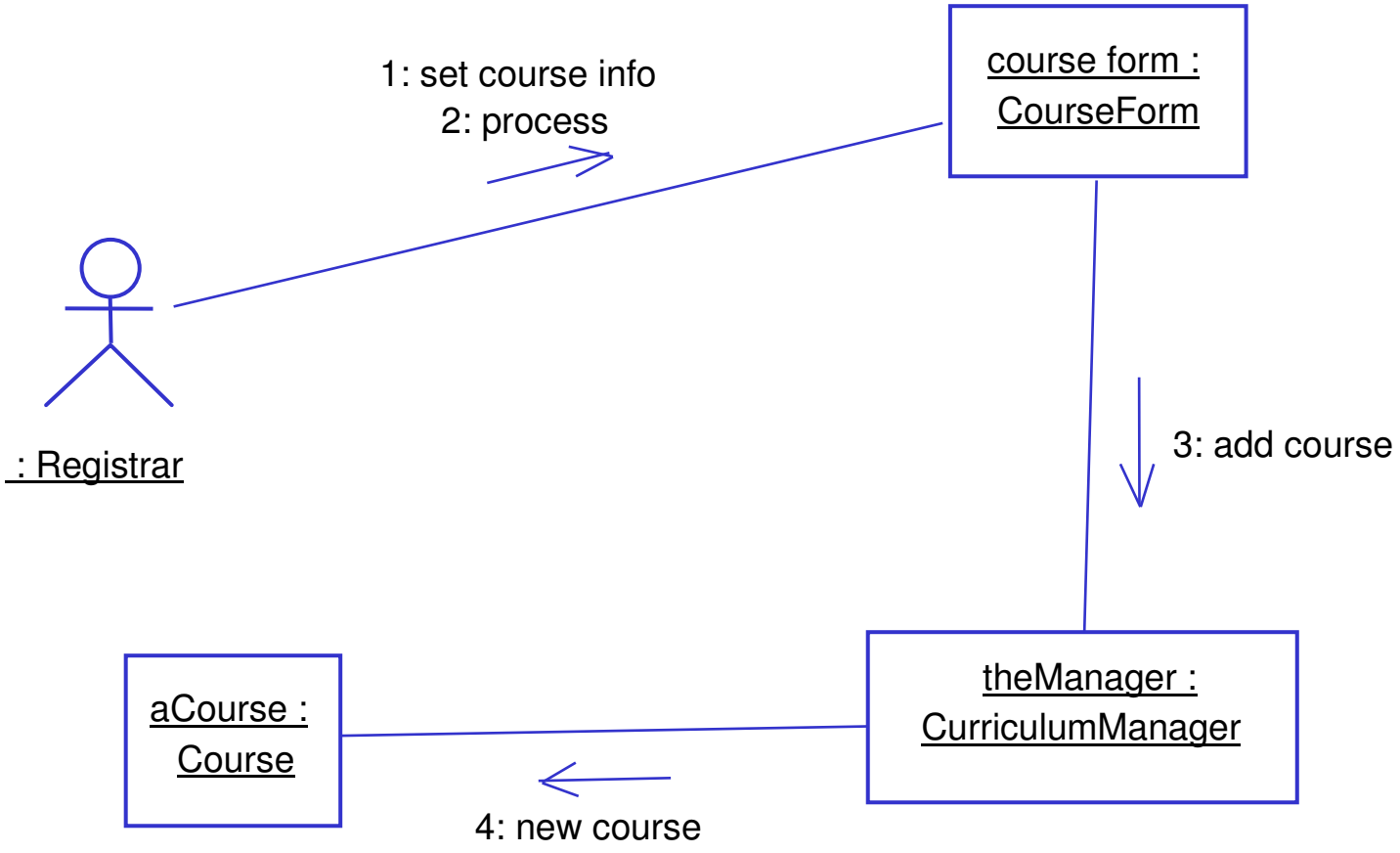
Sequence Diagram



Sequence Diagram

- A sequence diagram displays object interactions arranged in a time sequence
- Captures dynamic behavior (time-oriented)
- Purpose
  - Model flow of control
  - Illustrate typical scenarios

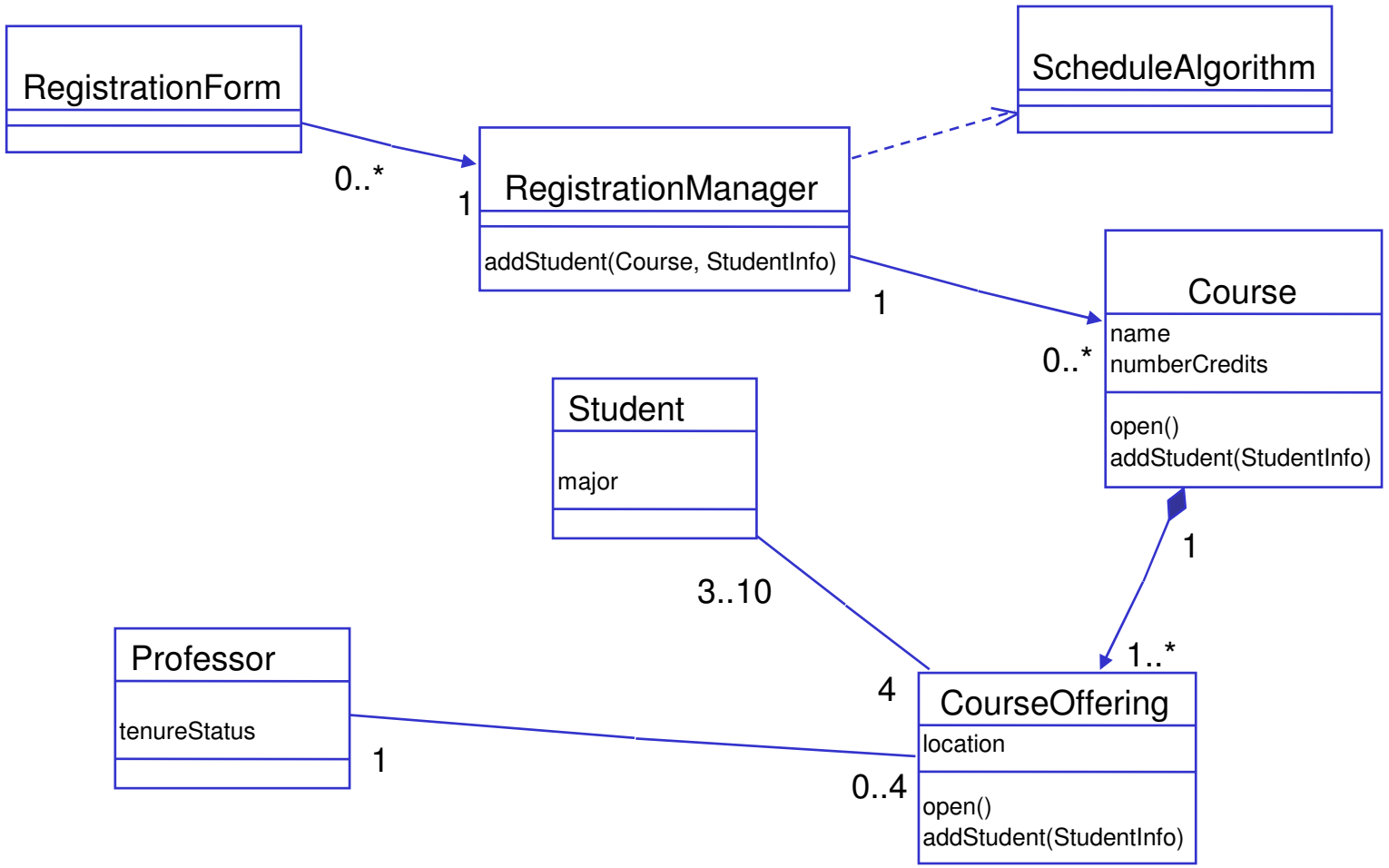
Collaboration Diagram



Collaboration Diagram

- A collaboration diagram displays object interactions organized around objects and their links to one another
- Captures dynamic behavior (message-oriented)
- Purpose
  - Model flow of control
  - Illustrate coordination of object structure and control

Class Diagram



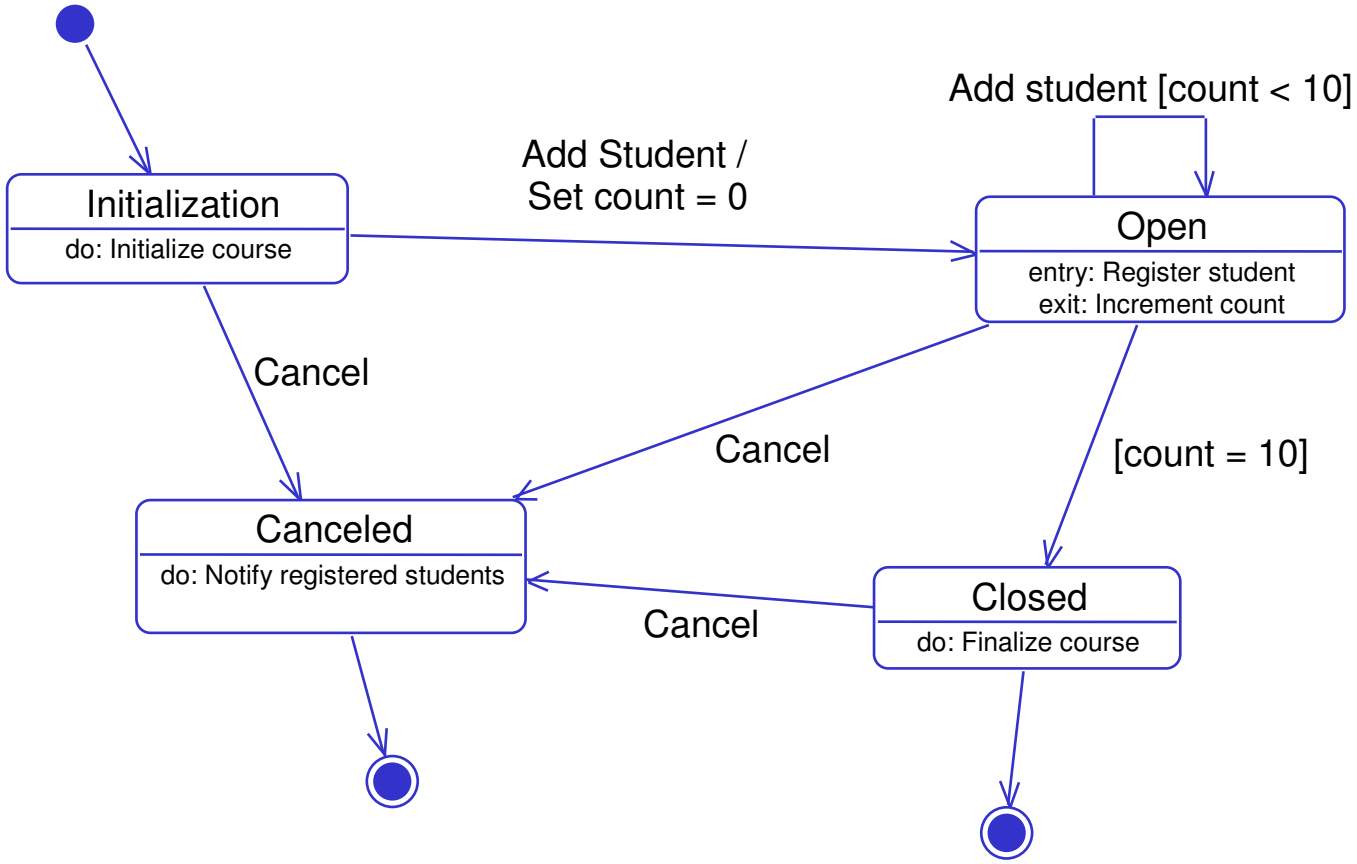
Class Diagram

- A class diagram shows the existence of classes and their relationships in the logical view of a system
- Captures the vocabulary of a system
- Built and refined throughout development
- Purpose
  - Name and model concepts in the system
  - Specify collaborations
  - Specify logical database schemas
- Developed by analysts, designers, and implementers

Object Diagram

- Shows instances and links
- Built during analysis and design
- Purpose
  - Illustrate data/object structures
  - Specify snapshots
- Developed by analysts, designers, and implementers

State Transition Diagram

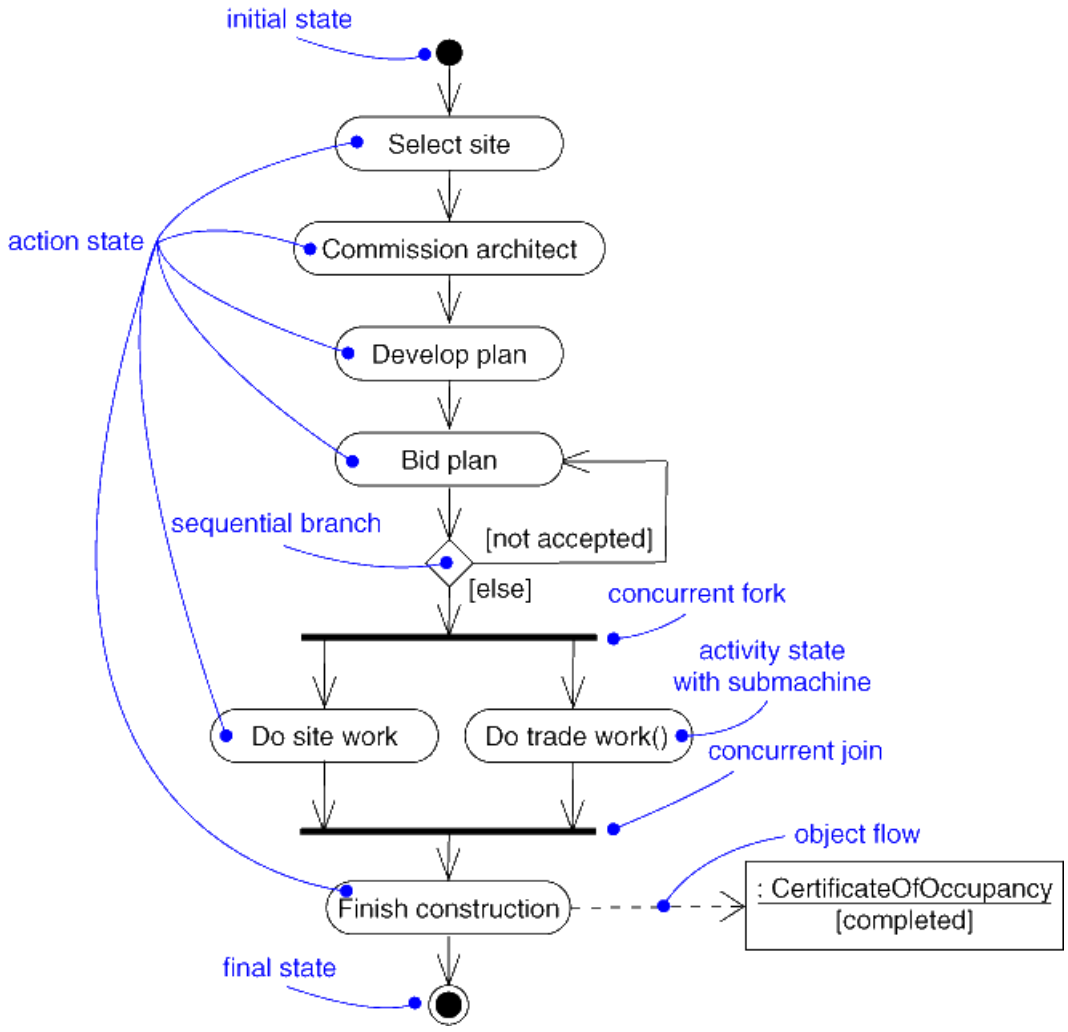


State Transition Diagram

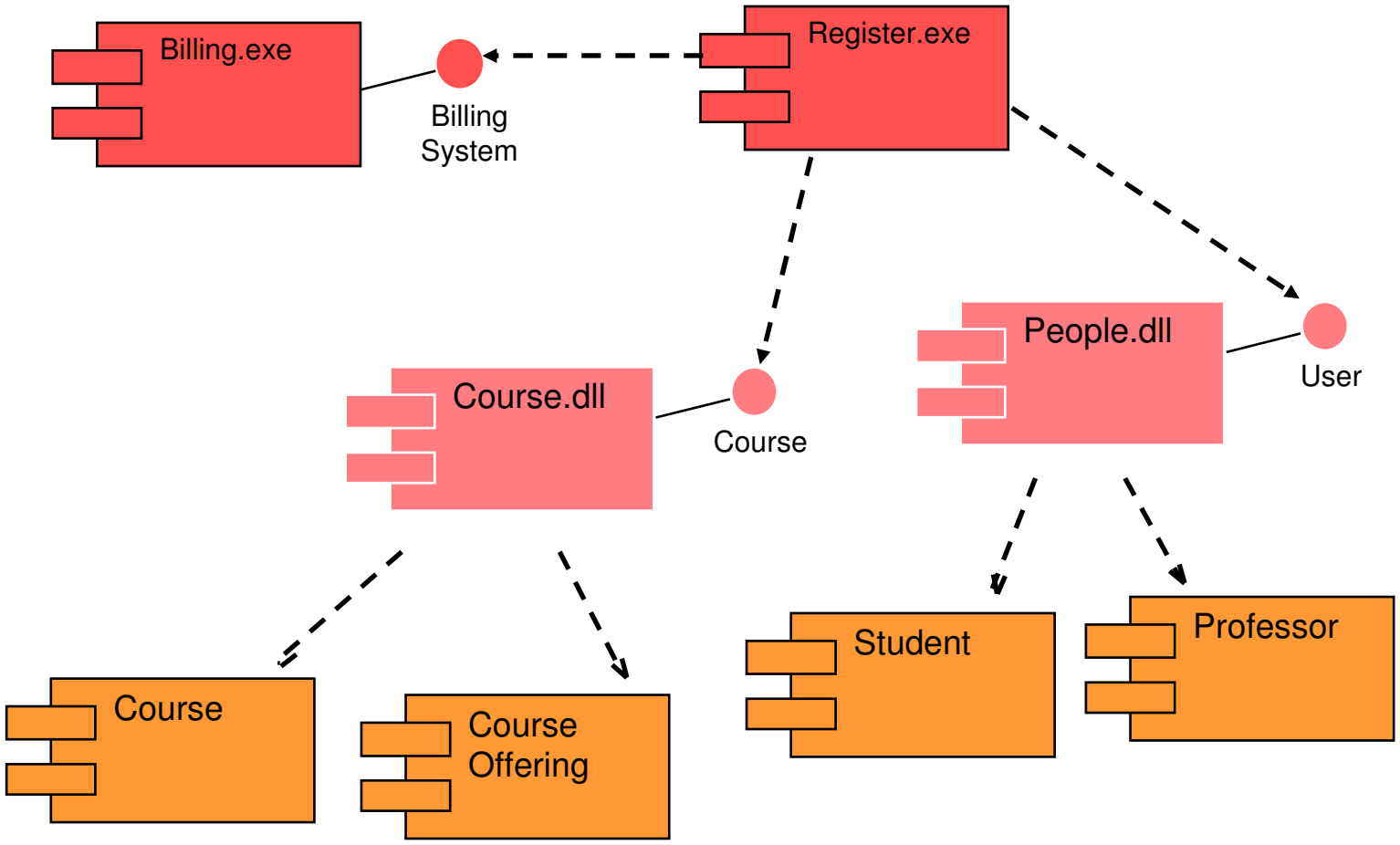
- State transition diagrams are created for objects with significant dynamic behavior
- Captures dynamic behavior (event-oriented)
- Purpose
  - Model object lifecycle
  - Model reactive objects (user interfaces, devices, etc.)

Activity Diagram

- Captures dynamic behavior (activity-oriented)
- Purpose
  - Model business workflows
  - Model operations



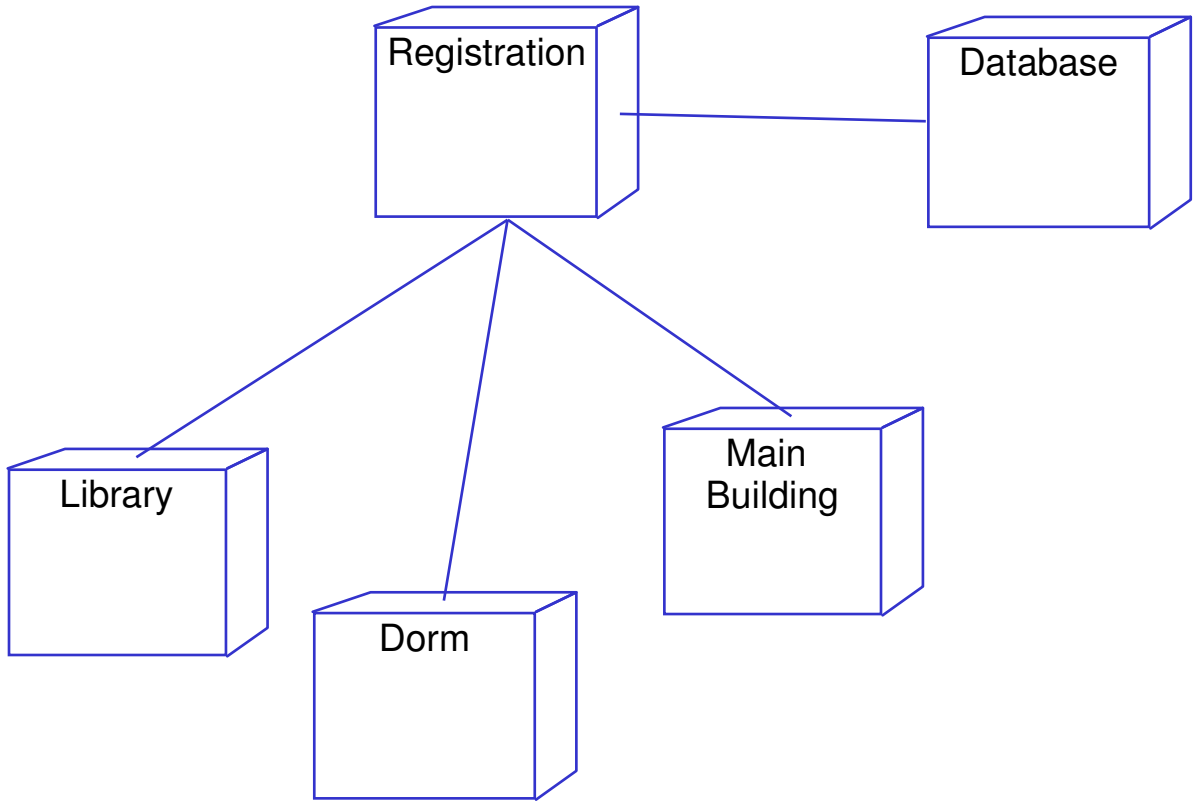
Component Diagram



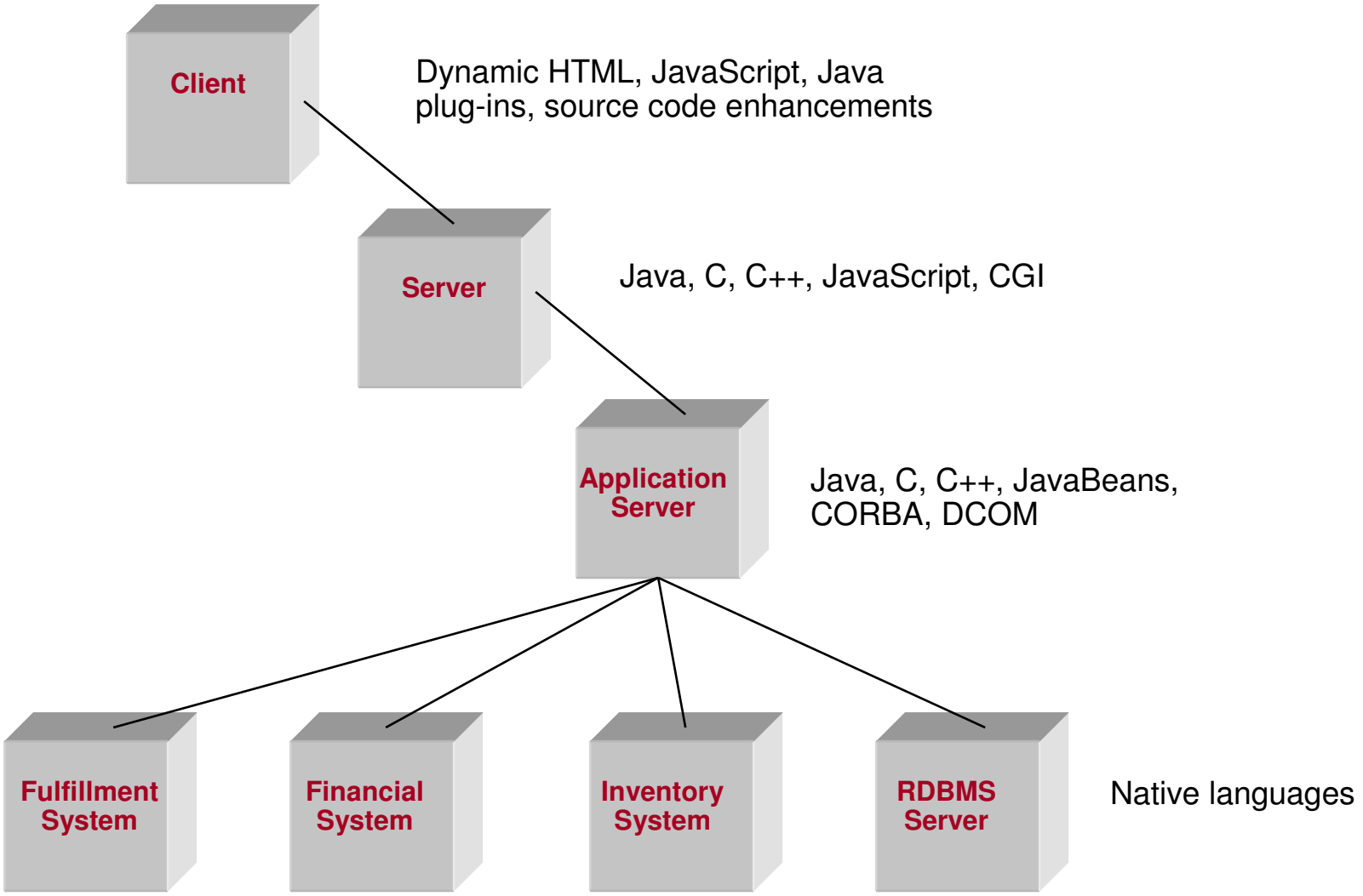
Component Diagram

- Component diagrams illustrate the organizations and dependencies among software components
- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database
- Developed by architects and programmers

Deployment Diagram



Deployment Diagram



Deployment Diagram

- The deployment diagram shows the configuration of run-time processing elements and the software processes living on them
- Captures the topology of a system's hardware
- Built as part of architectural specification
- Purpose
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers

Rational Rose

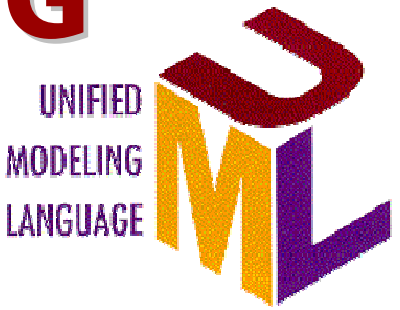
- Rose is available in three editions:
  - Rose Modeler – no language support
  - Rose Professional – support for 1 language
  - Rose Enterprise – supports multiple languages including (VC++, VB, Java, CORBA and XML)
- Why should we use Rational Rose?
  - Common standard language--the Unified Modeling Language (UML) --results in improved team communication
  - Reverse-engineering capabilities allow you to integrate with legacy OO systems
  - Models and code remain synchronized through the development cycle
  - ...

Demo Rose

Tóm tắt

- Các vấn đề đã nghiên cứu
  - Khái niệm mô hình, mô hình hóa trực quan
  - Khái quát về Ngôn ngữ mô hình hóa thống nhất
  - Thí dụ sử dụng các biểu đồ của UML

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. ✓ **Mô hình hóa nghiệp vụ**
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Bài 3

Mô hình hóa nghiệp vụ

Giới thiệu mô hình hóa nghiệp vụ

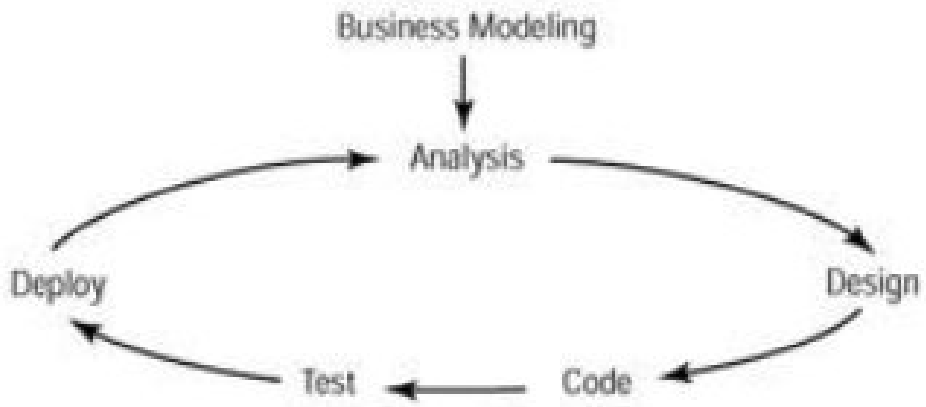
- Mô hình hóa nghiệp vụ (Business Modeling)
  - Là kỹ thuật mô hình hóa tiến trình nghiệp vụ
  - Mô hình hóa các chức năng của tổ chức
  - Quan tâm đến góc nhìn chức năng. Không phân biệt các tiến trình nghiệp vụ sẽ được tự động hóa hay thực hiện thủ công
- Biểu diễn mô hình nghiệp vụ bằng biểu đồ nghiệp vụ
  - Chỉ ra tương tác giữa các tiến trình nghiệp vụ với các vai trò (roles) thực hiện nghiệp vụ như customers hay vendors
  - Biểu diễn vai trò bên ngoài nghiệp vụ
- Hai lĩnh vực của mô hình hóa nghiệp vụ
  - Biên của tổ chức và nó cần giao tiếp với ai?
  - Luồng công việc bên trong tổ chức và tối ưu nó như thế nào?

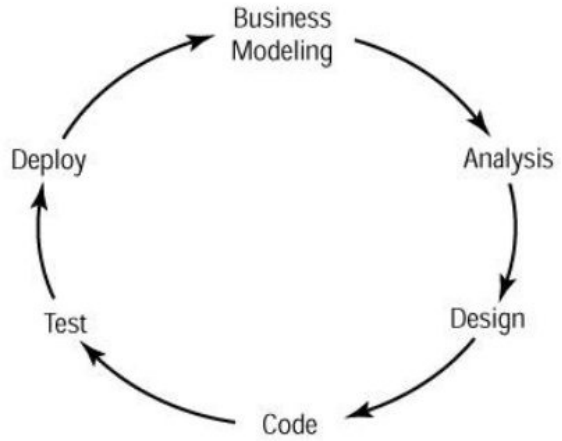
Giới thiệu mô hình hóa nghiệp vụ

- Không tập trung vào mô hình hóa hệ thống sẽ xây dựng
- Tập trung vào nghiệp vụ trên hệ thống
  - Mục tiêu là để hiểu rõ môi trường nghiệp vụ trước khi xây dựng hệ thống
- Mô hình hóa nghiệp vụ
  - Nghiên cứu về tổ chức
  - Khảo sát cấu trúc tổ chức, quan sát các vai trò trong tổ chức và quan hệ của chúng với nhau như thế nào.
  - Khảo sát luồng công việc trong tổ chức
    - Tiến trình chính, họ làm việc thế nào
    - Tính hiệu quả
    - Các hạn chế
  - Nghiên cứu các tổ chức bên ngoài và quan hệ với chúng?
  - Làm tài liệu về các thông tin bằng mô hình nghiệp vụ của UML

Giới thiệu mô hình hóa nghiệp vụ

- Khi nào không cần mô hình hóa nghiệp vụ?
  - Khi đã hiểu biết rõ ràng cấu trúc, mục đích tác nghiệp, stackeholders của tổ chức
  - Khi xây dựng phần mềm sử dụng cho một phần nhỏ của tổ chức, không ảnh hưởng đến nghiệp vụ khác
  - Luồng công việc khá rõ ràng và có tài liệu đầy đủ
  - Khi không có đủ thời gian!!!!
- Mô hình hóa nghiệp vụ trong tiến trình lặp



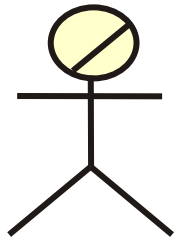


Các khái niệm cơ bản của BM

- Các khái niệm cơ bản bao gồm
  - Business actors
  - Business workers
  - Business use case
  - Biểu đồ Business use case
  - Quan hệ giao tiếp giữa Business use case và Business actor
  - Thực thể Business
  - Các biểu đồ hoạt động

Tác nhân nghiệp vụ

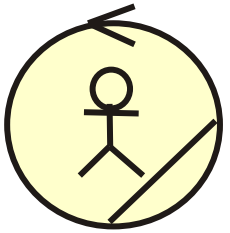
- Ai đó, cái gì đó bên ngoài tổ chức nhưng tương tác với nó
  - Customers, Investors, Suppliers...
  - Có thể là người hay nhóm người
- Tìm kiếm tác nhân nghiệp vụ?
  - Quan sát phạm vi dự án để tìm ra những gì nằm ngoài dự án
  - Những gì (ai, cái gì) nằm ngoài dự án có liên quan đến nghiệp vụ
  - Nghiên cứu tài liệu mô tả dự án, thị trường tổ chức, mục tiêu nghiệp vụ... để xác định thực thể bên ngoài liên quan
    - Thí dụ: Hãng hàng không liên quan đến nhà sản xuất máy bay, nhà sản xuất đồ ăn uống cho khách, khách hàng, hiệp hội hàng không...



Customer

Worker nghiệp vụ

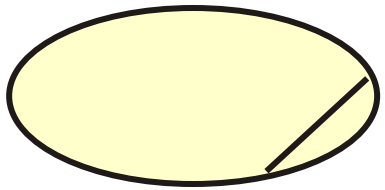
- Là vai trò (role) trong tổ chức
  - Một người có thể có nhiều vai trò
  - không phải là chức vụ
- Mô tả worker
  - Có trách nhiệm gì?
  - Kỹ năng cần có để thực hiện trách nhiệm?
  - Tương tác với worker nào?
  - Tham gia vào luồng công việc nào?
  - Trách nhiệm của worker trong luồng công việc
- Tìm kiếm worker nghiệp vụ
  - Quan sát phạm vi dự án – bắt đầu từ biểu đồ tổ chức
  - Khi đã có danh sách worker thì làm tài liệu cho chúng
- Thí dụ worker nghiệp vụ trong công ty hàng không
  - Phi công, người dẫn đường, thợ máy, tiếp viên, nhân viên an ninh...



Pilot

Ca nghiệp vụ

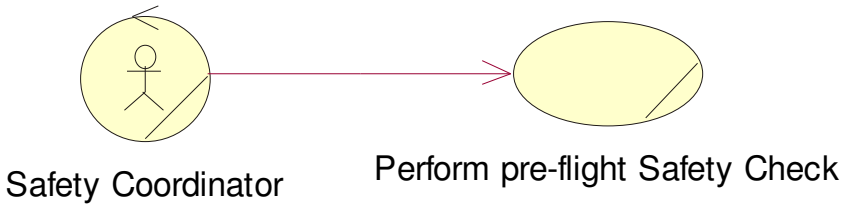
- Business use case là nhóm các luồng công việc liên quan có ý nghĩa với tác nhân nghiệp vụ
  - Cho biết tổ chức làm gì
  - Tập các ca nghiệp vụ mô tả đầy đủ nghiệp vụ của tổ chức
- Đặt tên
  - Theo hình thức "<động từ><danh từ>": "Price Products"
- Làm tài liệu luồng công việc
  - Thí dụ với UC nghiệp vụ Price Products
    - Nhân viên yêu người cầu quản lý cung cấp danh sách các mặt hàng mới cần định giá
    - Nhân viên kiểm tra hóa đơn kho để biết phải trả cho kho bao nhiêu kho hàng bán
    - Nhân viên cộng thêm 10% để có giá bán
    - Nhân viên trình giá để người quản lý phê duyệt
    - Nhân viên làm các thẻ sản phẩm
    - Gắn thẻ giá sản phẩm vào từng sản phẩm

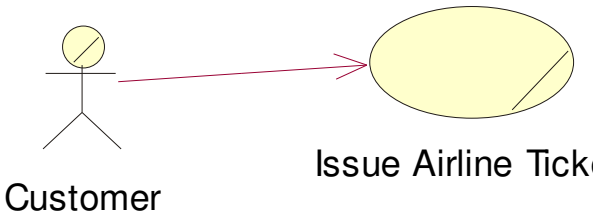


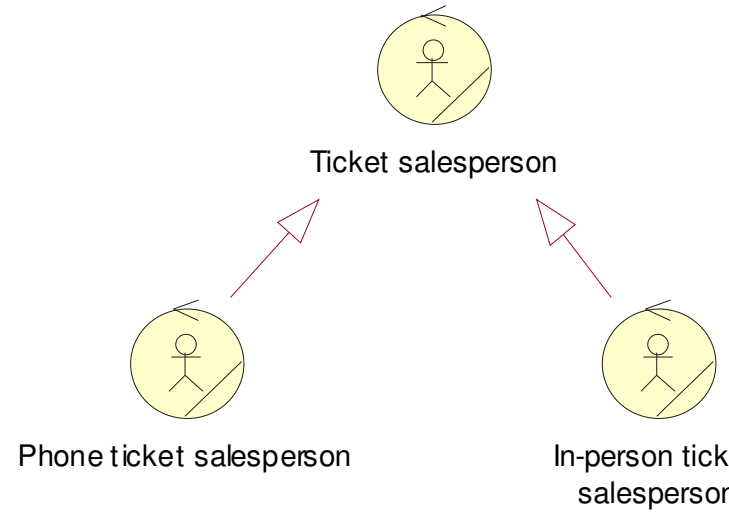
Price Products

Tương tác giữa các phần tử

- Biểu diễn tương tác
  - Quan hệ association
    - giữa tác nhân nghiệp vụ, worker nghiệp vụ với UC nghiệp vụ
    - mũi tên cho biết ai khởi xướng tiến trình
  - Quan hệ generalization
    - chỉ ra cấu trúc kế thừa giữa các phần tử mô hình nghiệp vụ
    - áp dụng cho hai hay nhiều phần tử tương tự nhau

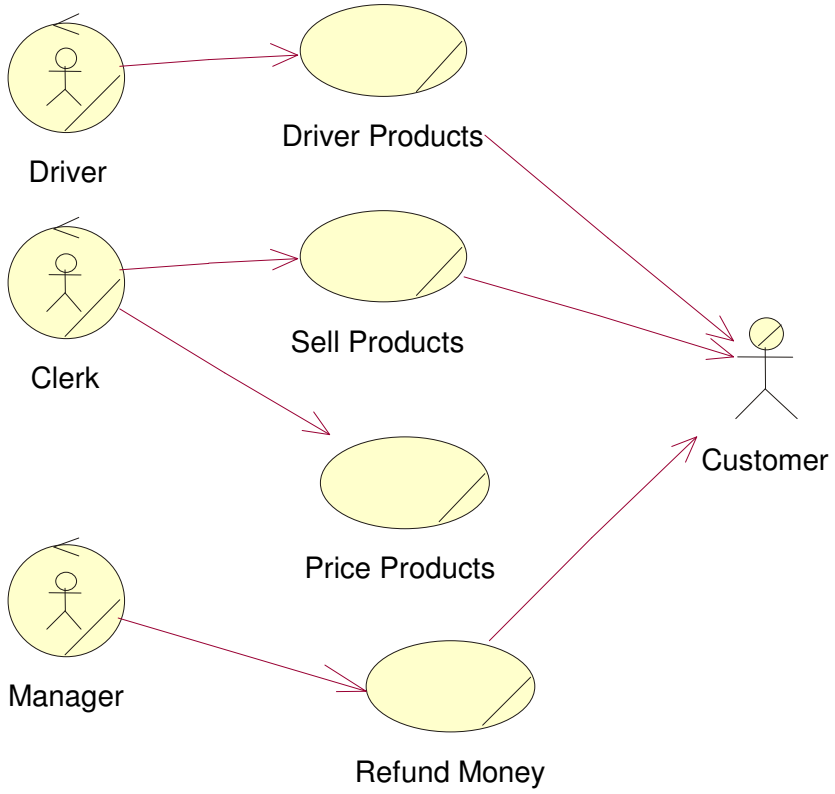






Biểu đồ UC nghiệp vụ

- Chỉ ra mô hình đầy đủ
  - cái công ty làm
  - ai ở trong công ty
  - ai ở ngoài công ty
- Cho biết phạm vi của tổ chức
- Nếu có nhiều UC nghiệp vụ
  - có thể tạo nhiều biểu đồ UC nghiệp vụ và mỗi biểu đồ chứa tập các UC nghiệp vụ
- Mũi tên đi từ tác nhân nghiệp vụ và worker nghiệp vụ đến UC nghiệp vụ cho thấy ai khởi động tiến trình nghiệp vụ.

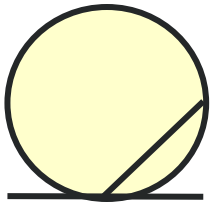


Thực thể nghiệp vụ

- Business entity là đối tượng mà tổ chức sử dụng để điều hành tác nghiệp hay sản xuất.
- Thực thể bao gồm tất cả những gì mà worker nghiệp vụ có liên quan hàng ngày
  - Thí dụ: Sales Order, Account, Shiping Box, Contract, Ghim giấy...
- Cái gì là thực thể nghiệp vụ, hãy trả lời:
  - Sản phẩm của công ty?
  - Công ty có các dịch vụ?
  - Công ty phải mua vật liệu gì để sản xuất?
  - Khách hàng cung cấp/nhận gì từ công ty?
  - Các worker nghiệp vụ trao đổi nhau cái gì khi sản xuất?
- Tìm kiếm thực thể nghiệp vụ ở nơi khác
  - Các danh từ trong UC nghiệp vụ

Thực thể nghiệp vụ

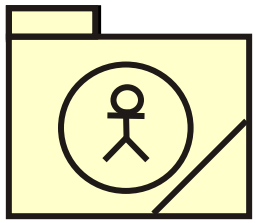
- Biểu tượng



- Bổ sung các thuộc tính cho thực thể nghiệp vụ
  - Thí dụ, thực thể nghiệp vụ Account có các thuộc tính account number, account type, balance, date opened, status...
  - Chú ý rằng chưa có thiết kế CSDL ở đây
  - Chỉ bổ sung các thuộc tính để dễ hiểu nghiệp vụ

Đơn vị tổ chức

- Đơn vị tổ chức (Organization Unit) là tập hợp các worker nghiệp vụ, thực thể nghiệp vụ và các phần tử mô hình nghiệp vụ khác
- Là cơ chế được sử dụng để tổ chức mô hình nghiệp vụ
- Nhiều công ty tổ chức theo phòng, ban, đơn vị...
  - Mỗi chúng được mô hình hóa như đơn vị tổ chức
  - Mỗi đơn vị tổ chức sẽ bao gồm các worker nghiệp vụ bên trong phòng, ban, đơn vị đó
- Biểu tượng



Marketing

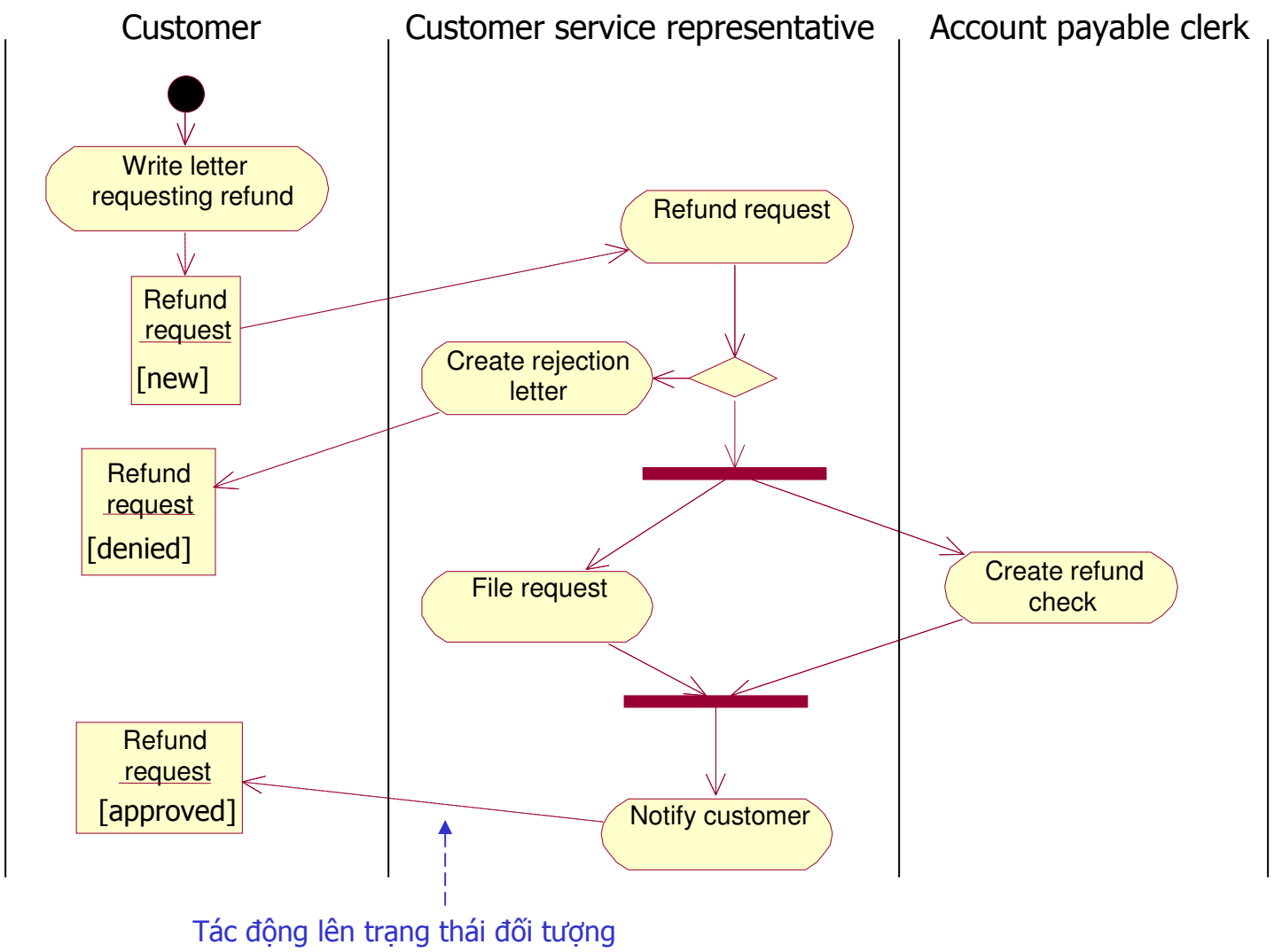
Biểu đồ UC nghiệp vụ

- Thực tế: luồng công việc (Workflow) không đơn giản mà có nhiều logíc điều kiện
  - worker nghiệp vụ có thể thực hiện một vài actions khi điều kiện A xảy ra và thực hiện một vài actions khác khi điều kiện B xảy ra...
  - hãy sử dụng biểu đồ hoạt động (Activity Diagram) để mô hình hóa các luồng công việc
- Nếu trong biểu đồ UC nghiệp vụ có nhiều UC nghiệp vụ, tác nhân nghiệp vụ và worker nghiệp vụ thì có thể nhóm chúng thành các đơn vị tổ chức (Organizational Units)
  - tổ chức lại mô hình để dễ đọc và dễ hiểu
  - sau đó xây dựng biểu đồ UC nghiệp vụ chi từng đơn vị tổ chức

Biểu đồ hoạt động

- Biểu đồ Activity được sử dụng để mô hình hóa luồng công việc của UC bằng các phần tử đồ họa
- Nó chỉ ra
  - các bước trong luồng công việc
  - các điểm quyết định
  - ai có trách nhiệm thực hiện từng bước
  - các đối tượng ảnh hưởng đến luồng công việc
- Thí dụ
  - Khách hàng nhận được sản phẩm lỗi, yêu cầu trả lại hàng
    - Customer viết thư yêu cầu bồi thường. Customer service representative nghiên cứu thư. Nếu thiếu tài liệu yêu cầu thì họ viết thư từ chối bồi thường. Nếu đầy đủ tài liệu thì họ lưu trữ thư và đồng thời Account payable clerk viết séc. Khi xong hai viêc này, Customer service representative thông báo cho khách hàng và yêu cầu của họ được chấp nhận.

Thí dụ biểu đồ hoạt động



Biểu đồ hoạt động

- Các phần tử chính của biểu đồ hoạt động
  - Swimlines: chỉ ra ai có trách nhiệm thực hiện các nhiệm vụ trong biểu đồ
  - Activities: các bước trong luồng công việc
  - Actions: các bước trong activity
    - action xảy ra khi vào (entry), ra (exit), đang ở (do) trong activity hay phụ thuộc vào sự kiện (event) nào đó xảy ra.
  - Business objects: thực thể bị luồng công việc tác động
  - Transitions: chỉ ra luồng công việc chuyển từ activity này đến activity khác
  - Decision points: chỉ ra nơi lập quyết định rẽ nhánh trong luồng công việc
  - Synchronizations: chỉ ra hai hay nhiều bước trong luồng công việc xảy ra đồng thời
  - Start state: chỉ ra nơi luồng công việc bắt đầu
  - End state: chỉ ra nơi luồng công việc kết thúc

Làm tài liệu chi tiết

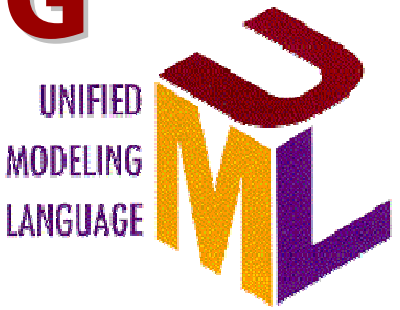
- Tiến trình cho biết cái nhìn mức cao những gì bên ngoài và bên trong tổ chức
  - Chi tiết luồng công việc sẽ được thực hiện trong các bước sau
- Phải làm tài liệu cho luồng công việc trong từng UC nghiệp vụ
  - có thể là tài liệu mô tả từng bước, flowchart hay biểu đồ hoạt động
  - tiến trình phức tạp có nhiều luồng chính luồng rẽ nhánh

    -> sử dụng biểu đồ hoạt động
- Kết quả là bức tranh tổng thể về tổ chức
  - UC cho biết tổ chức làm gì
  - Workflow cho biết mỗi UC được thực hiện chi tiết như thế nào
  - Actor cho biết cài gì bên ngoài tổ chức và tương tác với nó
  - Business worker cho biết các vai trò trong tổ chức
  - Units tổ chức cho biết cấu trúc tổ chức
  - Biểu đồ UC nghiệp vụ cho biết quan hệ các phần tử này

Tóm tắt

- Mô hình hóa nghiệp vụ
  - Thực tế, không phải tất cả các dự án đều thực hiện mô hình hóa nghiệp vụ
  - Mô hình hóa nghiệp vụ không liên quan đến cái sẽ được tự động hóa bằng hệ thống cụ thể
  - Giúp ta hình thành ngữ cảnh cho mô hình hóa hệ thống
- Mô hình hóa hệ thống
  - Tập trung vào cài đặt một hệ thống phần mềm cụ thể

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. ✓ **Mô hình hóa trường hợp sử dụng**
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Bài 4

Mô hình hóa trường hợp sử dụng

Giới thiệu mô hình hóa UC

- Trong pha thu thập yêu cầu và phân tích hệ thống thường phải xây dựng các biểu đồ cho
  - Mô hình nghiệp vụ
  - Mô hình trường hợp sử dụng
  - Mô hình giao diện người sử dụng
- Mô hình trường hợp sử dụng (Use case model) mô tả hệ thống được sử dụng như thế nào
  - Use case (UC) hệ thống và tác nhân hệ thống xác định phạm vi hệ thống
    - UC là những gì bên trong hệ thống
    - Actor là những gì bên ngoài hệ thống
  - Biểu đồ UC mô tả tương tác giữa các UC và tác nhân để hình thành chức năng hệ thống
- Sự khác nhau giữa mô hình hóa nghiệp vụ và mô hình hóa trường hợp sử dụng
  - Mô hình hóa nghiệp vụ tập trung vào tổ chức của cơ quan
  - Mô hình hóa hệ thống tập trung vào hệ thống đang xây dựng

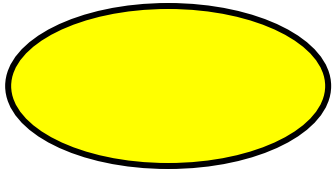
Các khái niệm mô hình hóa UC

	Mô hình hóa nghiệp vụ	Mô hình hóa hệ thống
Use case	Mô tả cái nghiệp vụ làm	Mô tả cái mà hệ thống bên trong nghiệp vụ làm
Actor	Bên ngoài tổ chức	Bên ngoài hệ thống (có thể bên trong tổ chức)
Business worker	Bên trong tổ chức	Không sử dụng

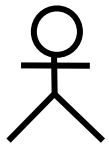
- Các khái niệm cơ bản
  - Trường hợp sử dụng (Use case-UC)
  - Tác nhân (Actor)
  - Quan hệ (Relationship)
  - Biểu đồ hoạt động (Activity Diagram)
  - Biểu đồ trường hợp sử dụng (Use case Diagram)

Use case, tác nhân là gì?

- 1994: Ivar Jacobson đề xuất sử dụng UC
- Use case?
  - UC là chức năng mức cao do hệ thống cung cấp, cái nhìn tổng thể về hệ thống
  - Không cho biết hệ thống làm việc bên trong?
  - Không phải là thiết kế, cài đặt mà là một phần của vấn đề cần giải quyết
  - Mô tả bất kỳ cái gì bên trong phạm vi hệ thống
- Tác nhân?
  - Mô tả ai, cái gì tương tác với hệ thống
  - Ba loại:
    - Ai: con người sử dụng trực tiếp hệ thống
    - Cái gì: hệ thống khác tương tác với hệ thống đang xây dựng
    - Thời gian: khi đồng hồ khởi sự sự kiện của hệ thống
  - Đặt tên: theo vai trò, không theo tên cụ thể vì nó là lớp



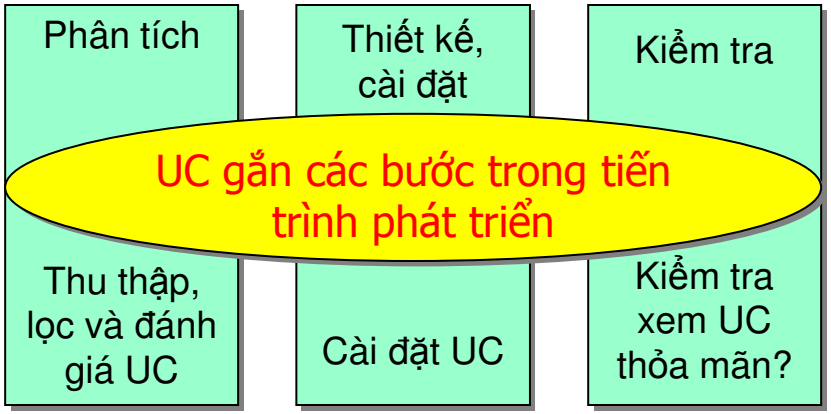
Purchase Ticket



Customer

Xây dựng UC để làm gì?

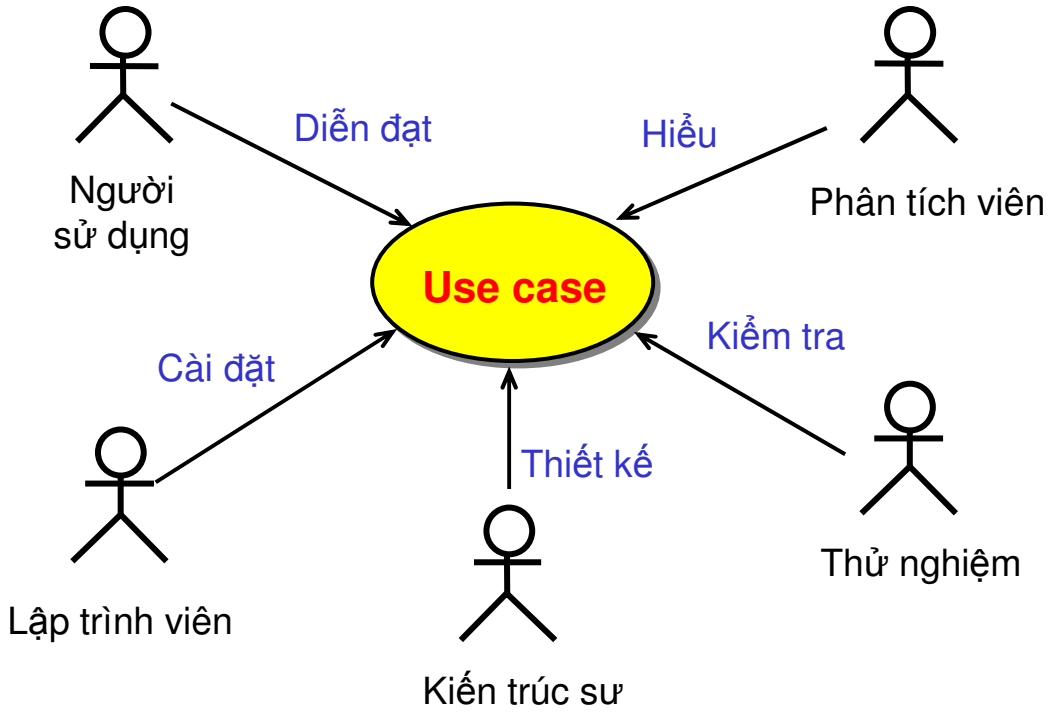
- Hình thành và mô tả yêu cầu chức năng hệ thống
  - Là kết quả thỏa thuận giữa khách hàng và người phát triển hệ thống phần mềm
- Cho phép mô tả rõ ràng và nhất quán cái hệ thống sẽ làm
  - Mô hình có khả năng được sử dụng xuyên suốt quá trình phát triển
- Cung cấp cơ sở để kiểm tra, thử nghiệm hệ thống
- Cho khả năng dễ thay đổi hay mở rộng yêu cầu hệ thống



UC và tiến trình phát triển

Xây dựng UC để làm gì?

- Ai quan tâm đến UC?



Tìm kiếm tác nhân như thế nào?

- Hãy trả lời các câu hỏi sau để tìm ra tác nhân hệ thống
  - Ai sẽ sử dụng chức năng chính của hệ thống?
  - Ai giúp hệ thống làm việc hàng ngày?
  - Ai quản trị, bảo dưỡng để hệ thống làm việc liên tục?
  - Hệ thống quản lý thiết bị phần cứng nào?
  - Hệ thống đang xây dựng tương tác với hệ thống khác nào?
  - Ai hay cái gì quan tâm đến kết quả hệ thống cho lại?

Tìm kiếm UC như thế nào?

- Với mỗi tác nhân đã tìm ra, hãy trả lời các câu hỏi sau để tìm ra các Use case hệ thống
  - Tác nhân yêu cầu hệ thống thực hiện chức năng nào?
  - Tác nhân cần đọc, tạo lập, bãi bỏ, lưu trữ, sửa đổi các thông tin nào trong hệ thống?
  - Tác nhân cần thông báo cho hệ thống sự kiện xảy ra trong nó?
  - Hệ thống cần thông báo cái gì đó cho tác nhân?
  - Hệ thống cần vào/ra nào? Vào/ra đi đến đâu hay từ đâu?
- Đặt tên UC hệ thống
  - Theo khái niệm nghiệp vụ của tổ chức
  - Không sử dụng từ kỹ thuật, chuyên môn
  - Sử dụng các động từ, cụm từ ngắn gọn
- Tùy theo tầm cỡ dự án mà mỗi hệ thống có từ 20-70 UC

Làm tài liệu UC

- Mô tả UC bao gồm các thông tin sau
  - Khởi đầu UC - sự kiện khởi động UC
    - "UC bắt đầu khi X xảy ra"
  - Kết thúc UC - sự kiện dừng UC
    - "Khi Y xảy ra thì UC kết thúc"
  - Tương tác giữa UC và tác nhân
  - Trao đổi thông tin
    - "Người sử dụng làm việc với hệ thống và nhập tên, mật khẩu"
  - Niên đại và nguồn gốc của thông tin
    - khi nào hệ thống đòi hỏi thông tin và khi nào hệ thống lưu trữ chúng
  - Lặp hành vi trong UC
    - có thể được mô tả bằng pseudo-code, biểu đồ activity
  - Tình thế phụ

Đã tìm đầy đủ UC cho hệ thống?

- Các câu hỏi sau giúp xác định đã tìm đầy đủ UC?
  - Mỗi yêu cầu chức năng ở trong ít nhất một UC?
    - Nếu yêu cầu chức năng không ở trong UC nào thì nó sẽ không được cài đặt sau này.
  - Đã khảo sát mọi tác nhân tương tác với hệ thống?
  - Tác nhân cung cấp cho hệ thống thông tin nào?
  - Tác nhân nhận thông tin nào từ hệ thống?
  - Đã nhận biết mọi hệ thống bên ngoài tương tác với hệ thống đang xây dựng?
  - Thông tin nào hệ thống bên ngoài nhận và gửi cho hệ thống đang xây dựng?

Khả năng truy nguyên

- Mỗi UC hệ thống phải có khả năng truy nguyên (traceability) đến UC nghiệp vụ
  - UC hệ thống cài đặt phần chức năng trong UC nghiệp vụ
- Truy nguyên không phải là ánh xạ 1-1
  - UC nghiệp vụ ở mức rất cao
    - nhiều UC hệ thống hỗ trợ 1 UC nghiệp vụ
- Thí dụ hệ thống quản lý hàng không

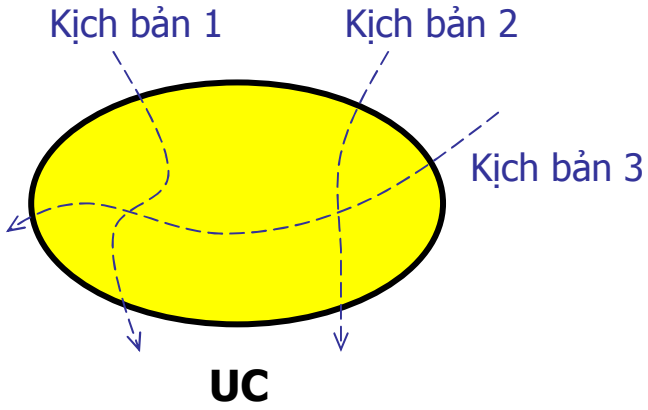
UC nghiệp vụ	UC hệ thống
Repair plane	Enter problem; Check inventory for parts; Receive part from inventory; Order part; Schedule maintenance
Load supplies on plane	Determine needed suplies; Check suply availability; Reserve supplies; Receive supplies
Perform pre-flight safety check	Confirm luggages inspection; Confirm passenger check-in; Inspect plane exterior; Check status of emergency equipment

Khả năng truy nguyên

- Không phải mọi UC nghiệp vụ đều được UC hệ thống hỗ trợ
  - Với các UC nghiệp vụ là tiến trình thủ công
    - Unload Passengers and Luggage,...
- Có thể sử dụng phần mềm Rational Requisite Pro để ánh xạ trực tiếp các UC hệ thống vào UC nghiệp vụ
- Mục đích của truy nguyên
  - Đảm bảo rằng hệ thống được xây dựng và cài đặt thì mọi mã trình phù hợp với yêu cầu của hệ thống
- Sau khi truy nguyên UC hệ thống vào UC nghiệp vụ phải truy nguyên các yêu cầu chức năng vào UC hệ thống
  - UC hệ thống mô tả chức năng mà hệ thống cung cấp
  - UC hệ thống điều khiển toàn bộ quá trình thiết kế
    - Nếu yêu cầu chức năng không truy nguyên vào UC hệ thống thì chúng sẽ không có trong thiết kế
- Không cần truy nguyên các yêu cầu phi chức năng vào UC hệ thống

Luồng sự kiện trong UC

- Tài liệu luồng sự kiện (flow of events) mô tả hành vi của UC
  - mô tả luồng logíc đi qua UC
  - mô tả người sử dụng làm gì, hệ thống làm gì
  - Trong một UC có nhiều luồng sự kiện: luồng chính, luồng phụ
- Kịch bản (Scenario)
  - Một luồng sự kiện trong một hiện thực của UC
  - Là trình tự hành động cụ thể để mô tả hành vi
  - Kịch bản đi xuyên suốt UC theo nhánh chính, nhánh phụ, nhánh đặc biệt



Tài liệu luồng sự kiện

- Tài liệu luồng sự kiện bao gồm
  - Mô tả vắn tắt UC
    - Mô tả ngắn gọn UC làm gì?
    - Những ai sử dụng UC?
    - Nó cho lại kết quả gì?
  - Tiền điều kiện (pre-condition)
    - Điều kiện cần thực hiện trước khi UC khởi động
    - Không phải UC nào cũng có tiền điều kiện
  - Luồng sự kiện chính và luồng sự kiện rẽ nhánh
  - Hậu điều kiện (post-condition)

Tài liệu luồng sự kiện

- Tài liệu luồng sự kiện bao gồm
  - Mô tả vắn tắt UC
  - Tiền điều kiện (pre-condition)
  - Luồng sự kiện chính và luồng sự kiện rẽ nhánh
    - chi tiết về UC được mô tả trong hai luồng sự kiện này
    - mô tả cái gì sẽ xảy ra để thực hiện chức năng của UC
    - Nội dung tài liệu
      - UC khởi động như thế nào?
      - Các đường đi xuyên qua các UC
      - Luồng chính thông qua UC
      - Luồng rẽ nhánh thông qua UC
      - Các luồng lỗi
      - UC kết thúc thế nào.
  - Hậu điều kiện (post-condition)
    - Là điều kiện được thực hiện ngay sau khi kết thúc UC

Thí dụ tài liệu luồng sự kiện

- Làm tài liệu các luồng sự kiện cho UC "Purchase Ticket"
  - Các bước trong luồng sự kiện chính
    1. UC bắt đầu khi customer chọn chức năng xem thông tin chuyến bay
    2. Hệ thống hiển thị thành phố đến, đi và thời gian hạ cánh, cất cánh
    3. User nhập nơi đến, đi, thời gian ngày tháng khởi hành và trở về
    4. Hệ thống hiển thị danh sách chuyến bay và giá vé

       A1. Không còn chuyến bay
    5. User chọn chuyến bay để đặt trước
    6. Hệ thống hiển thị các loại vé để user chọn
    7. User chọn giá vé

       A2. User chọn giá vé cho thành viên frequent-flyer
    8. Hệ thống hiển thị giá vé sẽ bán cho khách hàng
    9. User khẳng định giá vé
    10. Hệ thống hiển thị loại thẻ tín dụng, số thẻ, thời gian hết hạn
    11. User nhập loại thẻ tín dụng, số thẻ, thời gian hết hạn
    12. Hệ thống trình mua bằng thẻ

(còn nữa)

Thí dụ tài liệu luồng sự kiện

A6. Không thấy tài khoản

A7. Không đủ tiền

E1. Không xâm nhập được hệ thống tín dụng

13. Hệ thống dành chỗ cho user

14. Hệ thống phát sinh và hiển thị mã xác thực cho user

15. User khẳng định đã nhận mã

16. Use case kết thúc

- Luồng phụ

A1. Không có chuyến bay

1. Hệ thống hiển thị thông điệp thông báo không có chuyến bay
2. User khẳng định thông điệp
3. Trở lại luồng chính Bước 2.

A2. Vé dành cho thành viên frequent-flyer

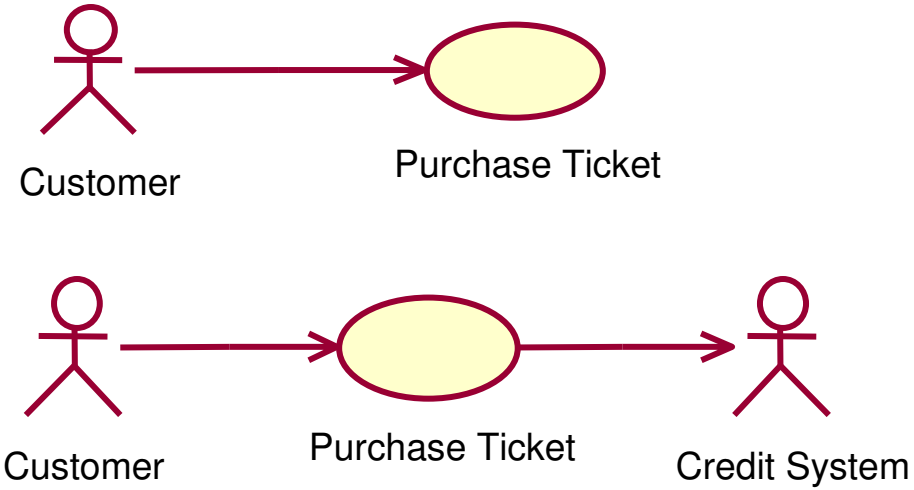
1. Hệ thống hiển thị số hiệu frequent-flayer
2. User nhập số
3. Hệ thống khẳng định tính hợp lệ của số

A3. Số không hợp lệ

...

Các quan hệ

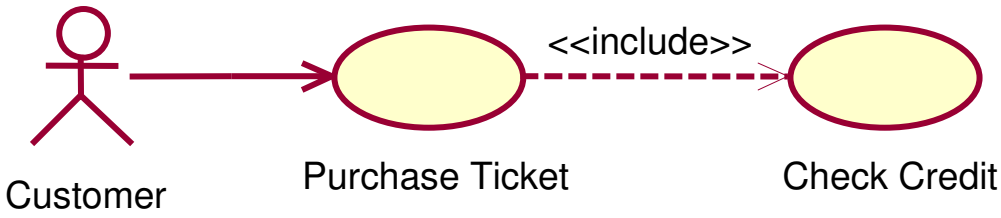
- Quan hệ kết hợp (Association)
  - Là loại quan hệ giữa tác nhân và UC
  - Mũi tên cho biết ai là người khởi xướng giao tiếp



- Quan hệ gộp (Includes)
- Quan hệ mở rộng (Extends)
- Quan hệ khái quát hóa (Generalization)

Các quan hệ

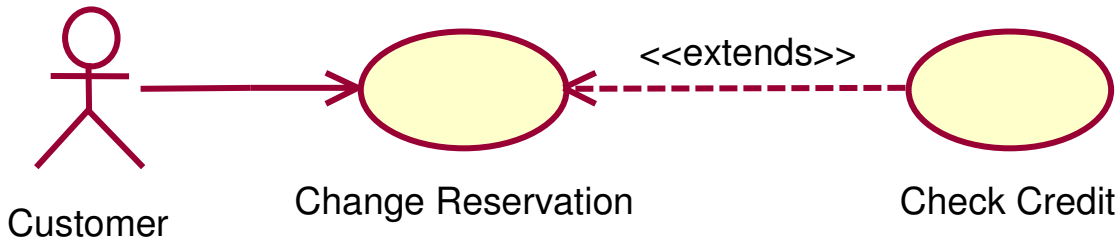
- Quan hệ kết hợp (Association)
- Quan hệ gộp (Includes)
  - Trước phiên bản UML 1.3 quan hệ <<includes>> có tên là <<uses>>
  - Thể hiện một UC luôn luôn sử dụng chức năng của UC khác
  - Sử dụng để mô hình hóa một vài chức năng dùng chung, sử dụng lại giữa hai hay nhiều UC



- Quan hệ mở rộng (Extends)
- Quan hệ khái quát hóa (Generalization)

Các quan hệ

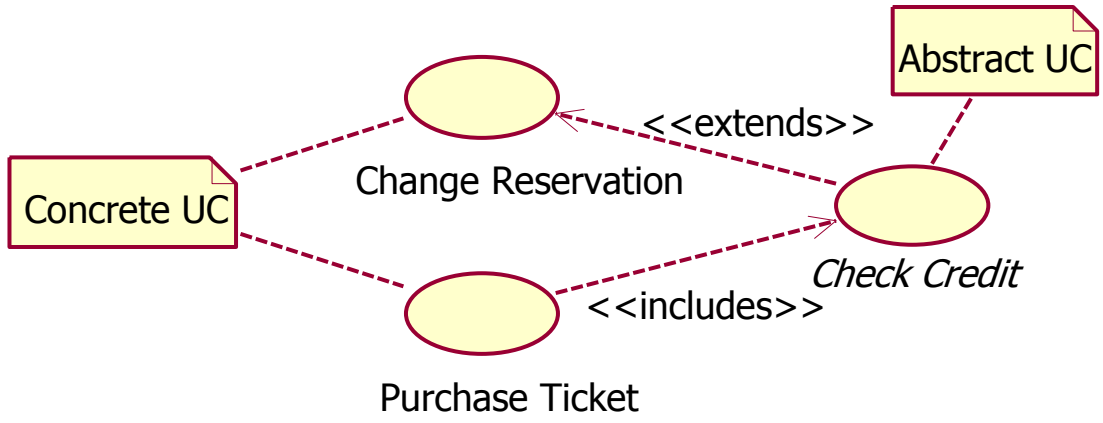
- Quan hệ kết hợp (Association)
- Quan hệ gộp (Includes)
- Quan hệ mở rộng (Extends)
  - Một UC tùy ý mở rộng chức năng do UC khác cung cấp
  - Mô tả một UC sử dụng chức năng của UC khác if and only if...
  - Sử dụng để mô hình hóa một vài chức năng dùng chung, sử dụng lại giữa hai hay nhiều UC



- Quan hệ khái quát hóa (Generalization)

Các quan hệ

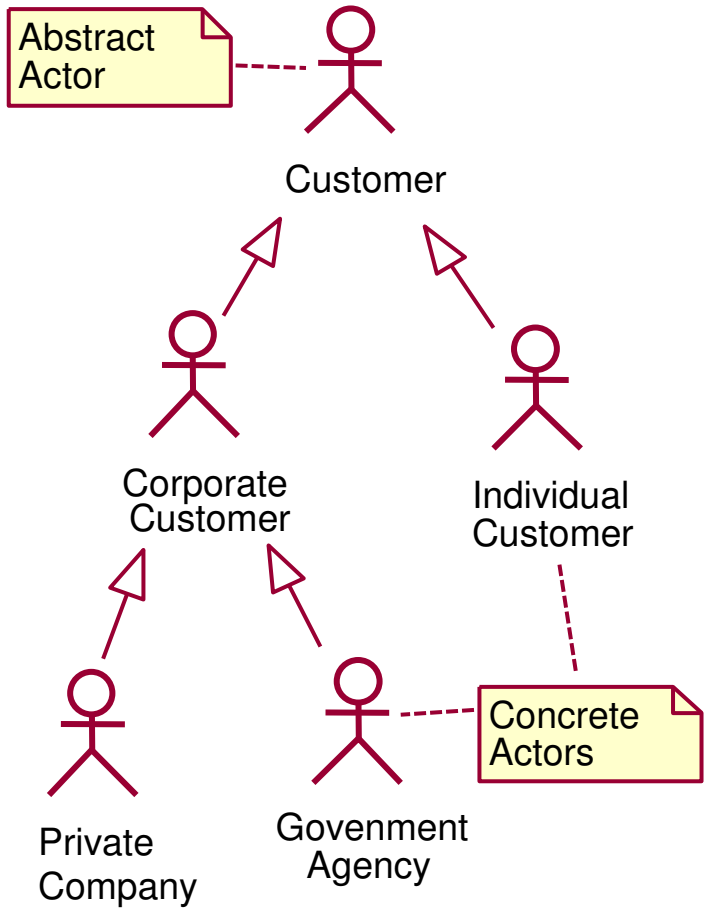
- Quan hệ kết hợp (Association)
- Quan hệ gộp (Includes)
- Quan hệ mở rộng (Extends)
- UC trừu tượng
  - Quan hệ includes và extends đều có tính chất chung là cùng sử dụng chức năng do UC khác cung cấp
  - Phần chức năng sử dụng chung có thể để trong UC mới – UC trừu tượng
    - UC trừu tượng không bị tác nhân kích hoạt giao tiếp



- Quan hệ khái quát hóa (Generalization)

Các quan hệ

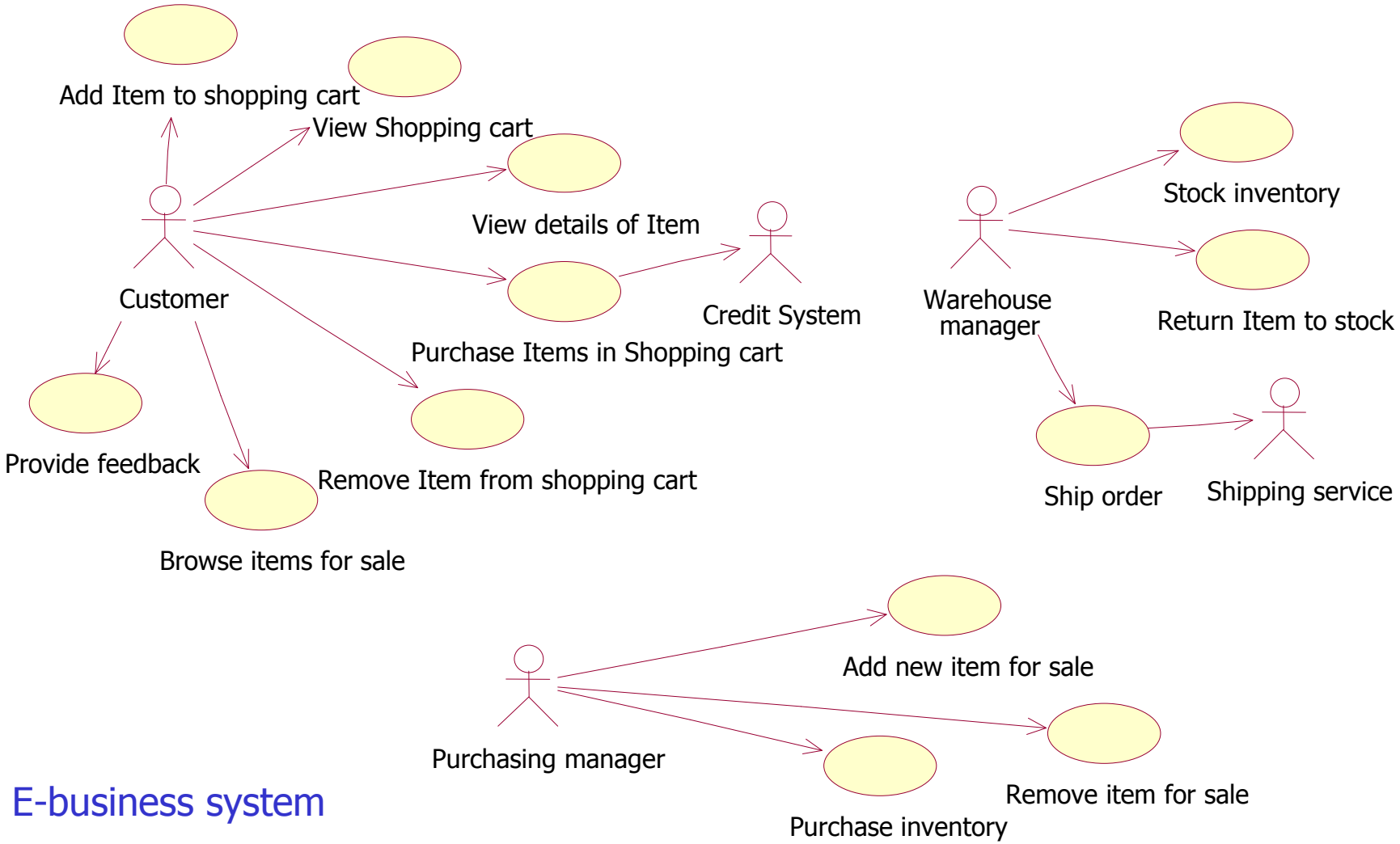
- Quan hệ kết hợp (Association)
- Quan hệ gộp (Includes)
- Quan hệ mở rộng (Extends)
- Quan hệ khái quát hóa (Generalization)
  - Chỉ ra một vài tác nhân hay UC có một số cái chung, giống nhau
  - Không nhất thiết hình thành quan hệ này cho các tác nhân
    - Khi một loại tác nhân kích hoạt một hay vài UC mà loại tác tác nhân khác không kích hoạt -> nên hình thành quan hệ khái quát hóa
    - Khi cả hai loại tác nhân cùng sử dụng các UC -> không cần mô hình hóa quan hệ khái quát hóa



Biểu đồ Use Case

- Mô hình UC được mô tả bởi một hay nhiều biểu đồ UC
  - Số lượng biểu đồ UC cho một dự án là tùy ý
    - Không quá nhiều làm rối loạn
    - Phải đảm bảo đầy đủ để biểu diễn đầy đủ thông tin của hệ thống
- Nó là công cụ mạnh giúp thu thập yêu cầu chức năng hệ thống
- Nó chỉ ra quan hệ giữa UC và tác nhân và giữa UC với nhau
- Sử dụng biểu đồ để làm tài liệu UC, tác nhân và các quan hệ giữa chúng
- Lợi ích chính của biểu đồ UC là làm giao tiếp
  - Khi quan sát các UC, customer biết hệ thống có các chức năng nào
  - Khi quan sát các tác nhân, customer biết ai giao tiếp với hệ thống
  - Khi quan sát cả UC và tác nhân, customer biết phạm vi dự án

Thí dụ biểu đồ Use Case



E-business system

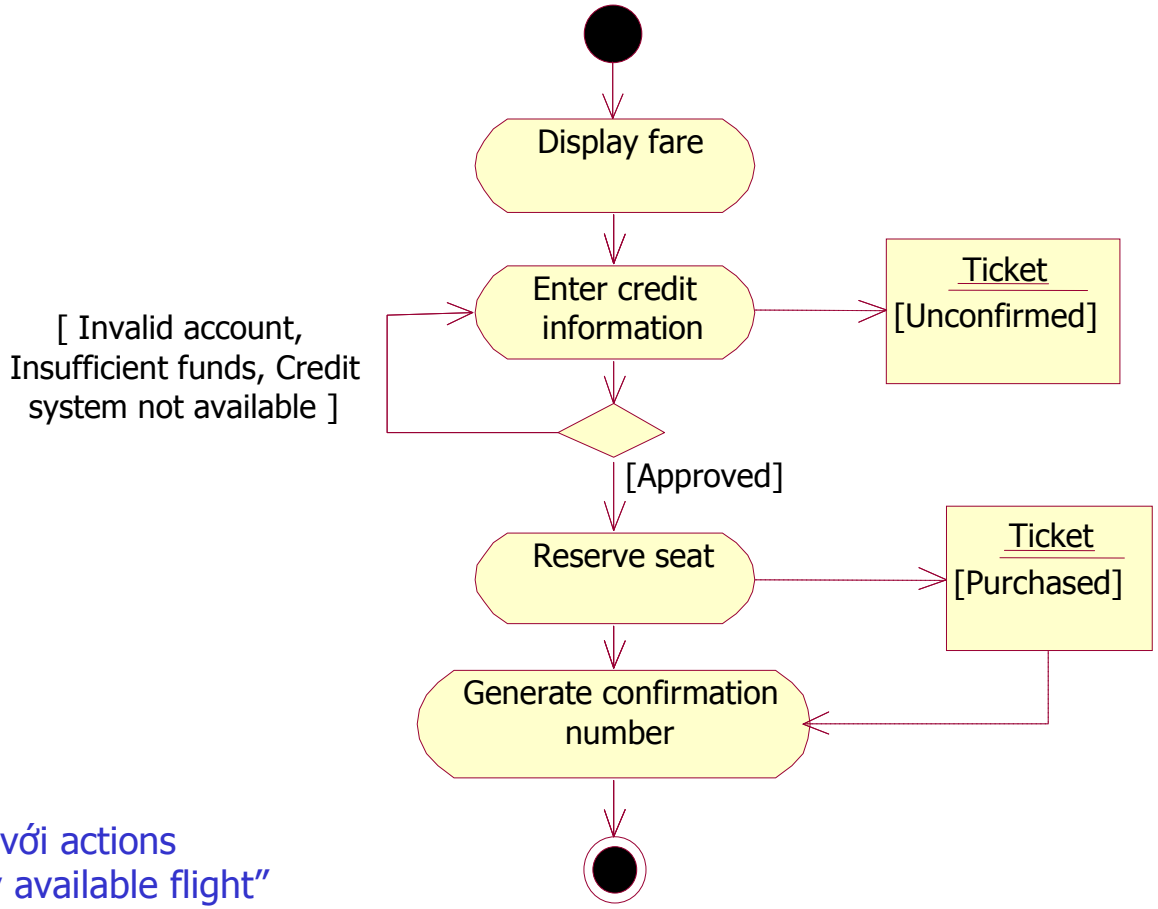
Biểu đồ Use Case

- Các chú ý khi xây dựng biểu đồ UC
  - Không nên mô hình hóa quan hệ kết hợp giữa tác nhân với tác nhân -> vì giao tiếp giữa các tác nhân là ở bên ngoài hệ thống
    - Hãy sử dụng biểu đồ luồng công việc để khảo sát quan hệ giữa các tác nhân
  - Không hình thành quan hệ Association giữa các UC
    - Biểu đồ chỉ ra có các UC nào nhưng không chỉ ra trật tự thực hiện chúng
  - Mỗi UC phải có tác nhân kích hoạt (trừ UC trong quan hệ extends và quan hệ includes)
    - Nên vẽ mũi tên thể hiện association đi từ tác nhân đến UC
  - Có thể xem CSDL là lớp ở dưới biểu đồ UC
    - Có thể nhập tin vào CSDL ở UC này và xâm nhập dữ liệu trong CSDL ở UC khác
    - Không vẽ association giữa các UC để chỉ ra luồng thông tin

Biểu đồ hoạt động

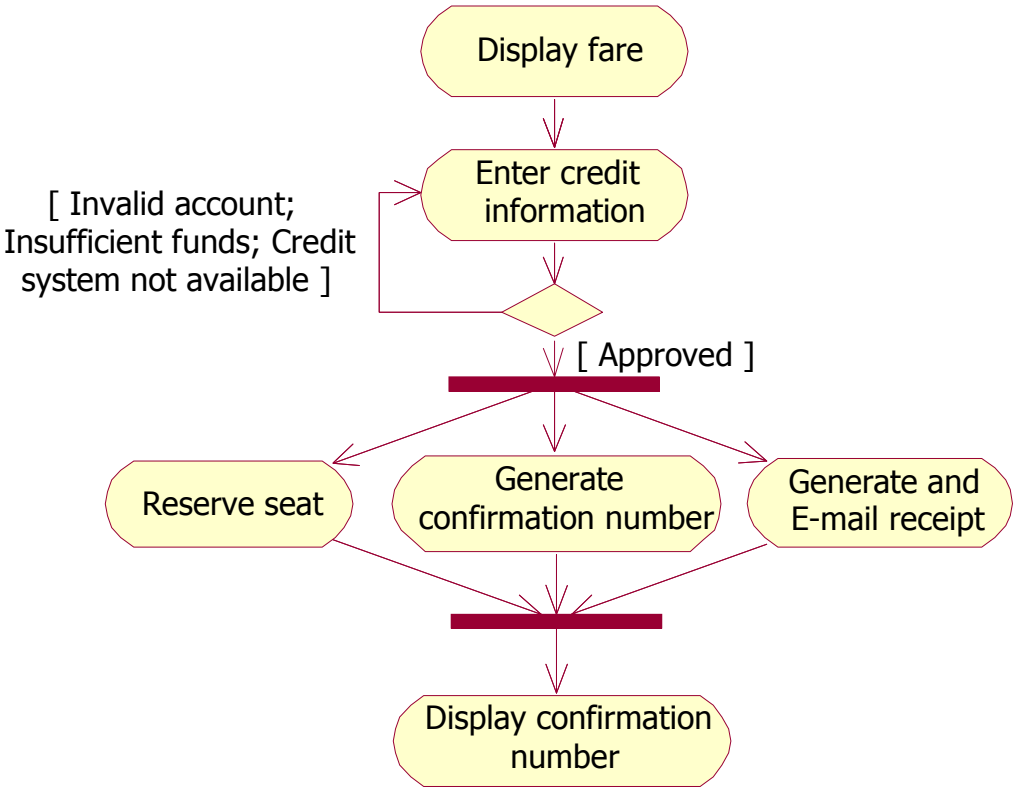
- Biểu đồ hoạt động (Activity diagram) do Odell đề xuất cho UML để
  - mô tả luồng công việc trong tiến trình nghiệp vụ trong mô hình hóa nghiệp vụ
  - mô tả luồng sự kiện trong mô hình hóa hệ thống
    - Sử dụng text như trước đây sẽ khó đọc khi logíc phức tạp, có nhiều rẽ nhánh
- Biểu đồ hoạt động sử dụng để mô hình hóa
  - khía cạnh động của hệ thống
  - các bước trình tự hay tương tranh trong quá trình tính toán

Biểu đồ hoạt động



Activity với actions
"Display available flight"

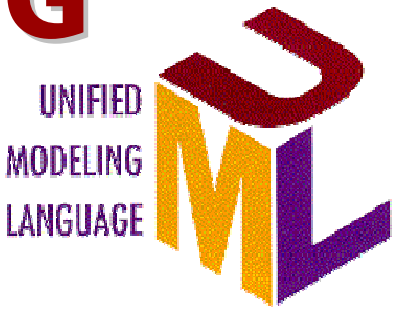
Biểu đồ hoạt động



Tóm tắt

- Bài này đã xem xét các vấn đề sau
  - Biểu đồ UC là gì?
  - Quan hệ giữa biểu đồ UC và biểu đồ nghiệp vụ
  - Các khái niệm của mô hình UC
  - Cách tìm kiếm UC, tác nhân, quan hệ trong mô hình UC
  - Cách mô tả luồng sự kiện
    - văn bản
    - biểu đồ hoạt động
  - Các phần tử đồ họa xây dựng biểu đồ UC

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. ✓ **Mô hình hóa tương tác đối tượng**
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Bài 5

Mô hình hóa tương tác đối tượng

Mô hình hóa đối tượng

- Mô hình hóa tương tác giữa các đối tượng trong hệ thống
- Hai loại biểu đồ được sử dụng để mô hình hóa đối tượng
  - Biểu đồ trình tự (Sequence diagram)
    - Tập trung vào mô tả điều khiển
  - Biểu đồ cộng tác (Colaboration diagram)
    - Tập trung vào mô tả dữ liệu
  - Biểu đồ trình tự và biểu đồ cộng tác đều chỉ ra cùng loại thông tin. Gọi tên chung cho hai loại biểu đồ này là biểu đồ tương tác (Interaction diagram)
- Biểu đồ tương tác chỉ ra các đối tượng tham gia vào luồng xuyên qua UC và các thông điệp gửi giữa chúng?
  - Biểu đồ tương tác giúp xác định hệ thống làm việc như thế nào?
  - Luồng sự kiện tập trung vào cái hệ thống làm.

Đối tượng?

- Ta nhìn thấy đối tượng xung quanh ta
  - Bàn, ghế, quyển sách...
- Đối tượng là cái gì đó gói thông tin và hành vi
- Nó là khái niệm biểu diễn cái cụ thể trong thế giới thực
- Thí dụ Máy bay VN358 là đối tượng
  - Có các thông tin:

    Ngày bay 10 April, giờ bay 8h30, số hiệu máy bay VN358, bay từ Hà Nội
  - Có các hành vi
    - Nó biết đón khách vào máy bay, biết đưa khách ra khỏi máy bay, xác định khi nào máy bay đầy khách
- Thông tin được lưu trữ bởi thuộc tính (Attribute)
- Hành vi của đối tượng được gọi là thao tác (Operation)

Lớp là gì?

- Lớp (class) là cái gì đó cung cấp kế hoạch (blueprint) cho đối tượng
  - Lớp cung cấp thông tin nào đối tượng lưu trữ và hành vi nào đối tượng có
  - Cung cấp mẫu (template) cho đối tượng
- Định nghĩa lớp của UML
  - Là mô tả tập đối tượng chia sẻ cùng thuộc tính, thao tác, phương pháp, quan hệ và ngữ nghĩa.
- Thí dụ
  - Đối tượng: Sinh viên A, Sinh viên B...
  - Lớp: Sinh viên

Xây dựng biểu đồ tương tác

- Để xây dựng biểu đồ tương tác ta bắt đầu từ luồng sự kiện
- Xây dựng từng biểu đồ cho
  - luồng chính, luồng thay thế, luồng lỗi
  - Nếu hai luồng thay thế và luồng lỗi tương tự nhau thì gộp chúng lại
- Sử dụng mẫu (Pattern) xây dựng biểu đồ tương tác để giảm thời gian
  - Xây dựng các mẫu cho các logíc chung: Khai thác CSDL, quản lý lỗi, giao tiếp giữa các tiến trình...
- Các bước xây dựng biểu đồ tương tác
  - Tìm kiếm đối tượng
  - Tìm kiếm tác nhân
  - Bổ sung thông điệp vào biểu đồ

Tìm kiếm đối tượng

- Khảo sát các danh từ trong luồng sự kiện
- Tìm đối tượng trong tài liệu kịch bản
  - Kịch bản (Scenario) là một hiện thực của luồng sự kiện
  - Mỗi luồng sự kiện có nhiều kịch bản
  - Mỗi UC có thể có nhiều biểu đồ tương tác
    - Mỗi biểu đồ được xây dựng cho một kịch bản thông qua luồng sự kiện
- Tìm đối tượng không được mô tả trong luồng sự kiện
  - Các đối tượng cho phép tác nhân nhập và quan sát thông tin
  - Các đối tượng tham gia điều khiển trình tự luồng xuyên qua UC
- Tìm đối tượng tương ứng với khái niệm trừu tượng khi phân tích
  - Thí dụ Tên sách, Tên tạp chí là trừu tượng không tương ứng với đối tượng nào trong thế giới thực

Tìm kiếm đối tượng

- Có thể hình thành các biểu đồ tương tác
  - Ở mức cao: để chỉ ra hệ thống giao tiếp như thế nào
  - Ở mức rất thấp: để chỉ ra lớp nào cần tham gia vào kịch bản
- Nên xem xét các nhóm đối tượng sau khi tìm kiếm chúng
  - Đối tượng thực thể (Entity)
    - Lưu trữ thông tin, có thể ánh xạ sang bảng, trường của CSDL
    - Nhiều danh từ trong luồng sự kiện thuộc loại này
      - Thí dụ: Chuyến bay VN358, Hành khách John, Vé số #1347A...
  - Đối tượng biên (Boundary)
    - Là đối tượng tại biên hệ thống và thế giới bên ngoài
      - Là các Forms, cửa sổ của ứng dụng và giao diện với các ứng dụng khác
  - Đối tượng điều khiển (Control)
    - Là các đối tượng bổ sung, không thực hiện chức năng nghiệp vụ nào
    - Nó điều phối các đối tượng khác và điều khiển toàn bộ luồng logíc

Tìm kiếm tác nhân

- Sau khi xác định đối tượng là tìm kiếm tác nhân cho biểu đồ tương tác
- Tác nhân trong biểu đồ tương tác là sự kích hoạt từ ngoài để khởi động luồng công việc của luồng sự kiện
- Tìm kiếm tác nhân trong luống sự kiện
  - Ai hay cái gì khởi xướng tiến trình?
- Có thể có nhiều tác nhân cho một biểu đồ tương tác
- Nếu tác nhân nhận hay gửi thông điệp cho hệ thống theo kịch bản nào đó thì chúng phải có mặt trong biểu đồ tương tác của kịch bản đó

Sử dụng biểu đồ tương tác

- Từ biểu đồ tương tác người thiết kế và người phát triển xác định các
  - lớp sẽ xây dựng
  - quan hệ giữa các lớp
  - thao tác và các trách nhiệm của lớp
- Biểu đồ trình tự theo thứ tự thời gian
  - Giúp người sử dụng quan sát luồng logíc thông qua kịch bản
- Biểu đồ cộng tác tập trung vào tổ chức cấu trúc của các đối tượng
  - Giúp dễ dàng quan sát đối tượng nào giao tiếp với các đối tượng nào
  - Khi thay đổi đối tượng, dễ dàng nhận thấy tác động trên các đối tượng khác?

Xây dựng biểu đồ tương tác

- Biểu đồ tương tác bao gồm các thành phần sau
  - Đối tượng (Objects)
    - Biểu đồ tương tác sử dụng tên đối tượng, tên lớp hay cả hai
  - Thông điệp (Messages)
    - Thông qua thông điệp, một đối tượng hay lớp có thể yêu cầu lớp hay đối tượng khác thực hiện vài chức năng cụ thể
      - Thí dụ: Form yêu cầu đối tượng Report tự in
  - Liên kết (Links)
    - Là hiện thực của quan hệ kết hợp giữa các đối tượng
  - Chú thích (Notes) và ràng buộc

Xây dựng biểu đồ tương tác

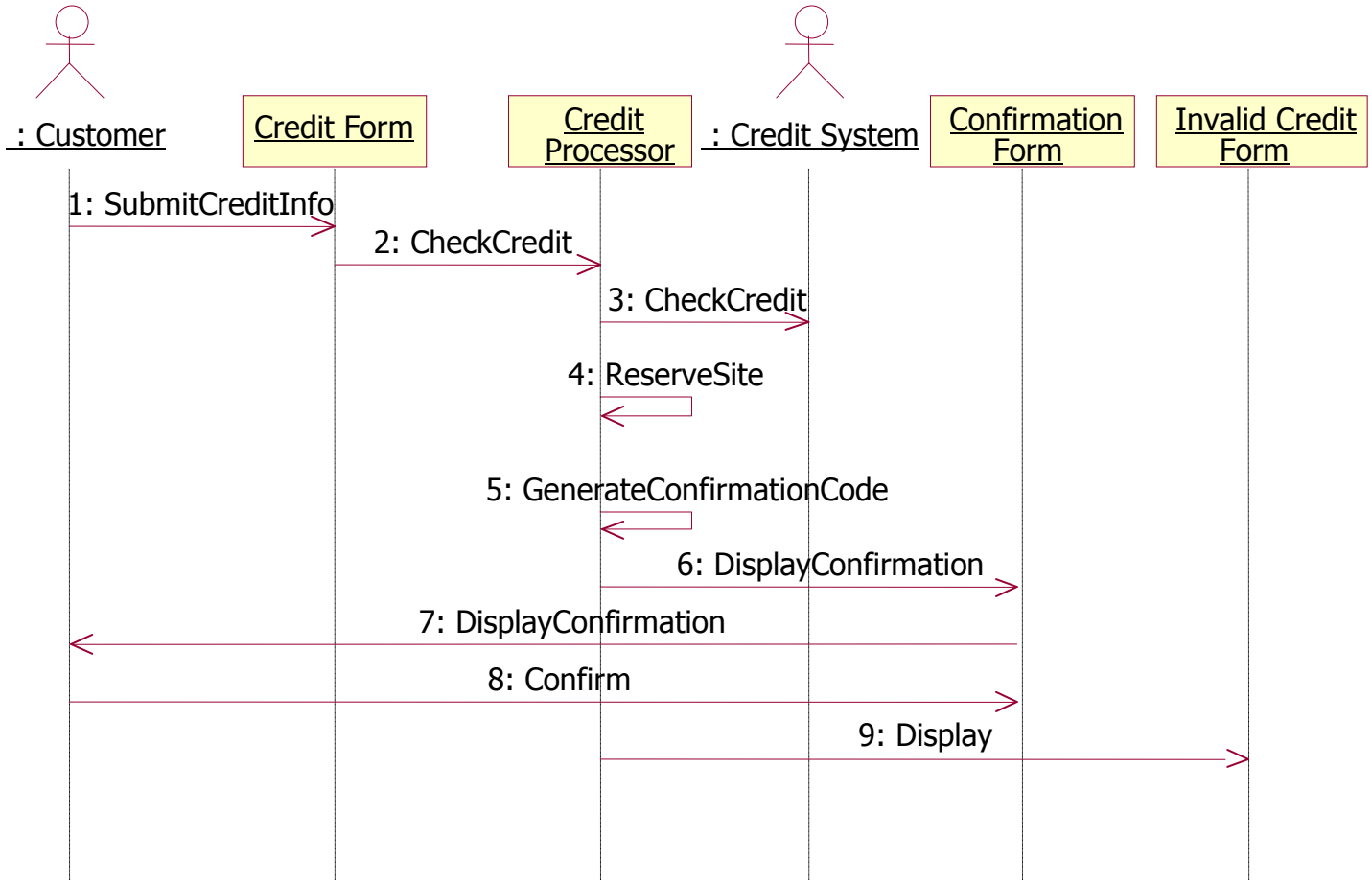
- Khi tạo lập biểu đồ tương tác có nghĩa là gán trách nhiệm cho đối tượng
  - Gán trách nhiệm cho đối tượng nhận thông điệp
  - Phải gán trách nhiệm cho đối tượng một cách phù hợp
    - Thí dụ không gán trách nhiệm nghiệp vụ cho đối tượng Form, Screen
- Dựa trên các loại lớp để cân nhắc hình thành trách nhiệm cho chúng
  - Lớp Entity: lưu trữ thông tin và thực hiện các chức năng nghiệp vụ
  - Lớp Boundary
    - form và windows: hiển thị và nhận thông tin. Có thể xử lý vài nghiệp vụ rất nhỏ
    - interfaces: trao đổi thông tin với hệ thống khác. Xử lý vài nghiệp vụ rất nhỏ
  - Lớp Control: theo dõi trình tự thực hiện

Biểu đồ trình tự

- Biểu đồ trình tự là biểu đồ theo thứ tự thời gian
  - Đọc biểu đồ từ đỉnh xuống đáy
  - Đọc biểu đồ bằng quan sát các đối tượng và thông điệp
  - Mỗi đối tượng có vòng đời (Lifeline)
    - Bắt đầu khi hình thành đối tượng, kết thúc khi phá hủy đối tượng
    - Thông điệp được vẽ giữa hai đối tượng – thể hiện đối tượng gọi hàm đối tượng khác
    - Thông điệp phản thân

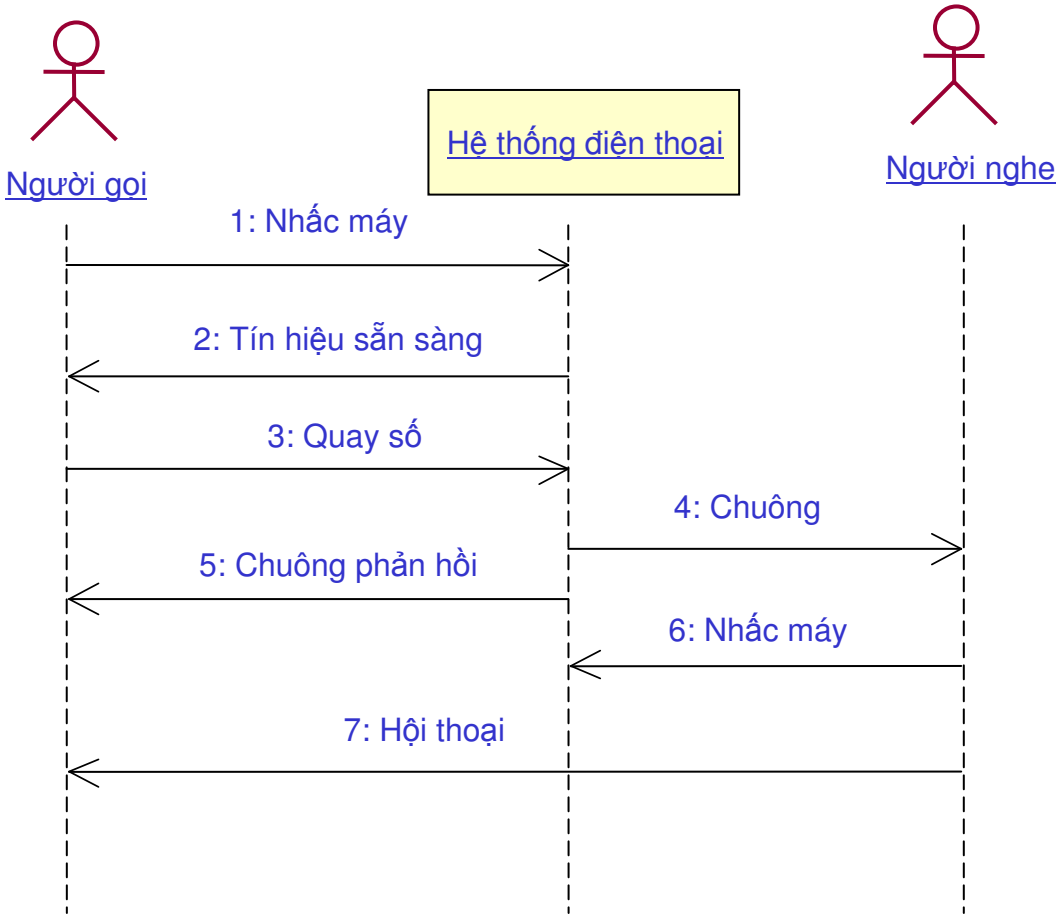
Biểu đồ trình tự

- Thí dụ: Luồng sự kiện Khách hàng đặt chỗ cho chuyến bay



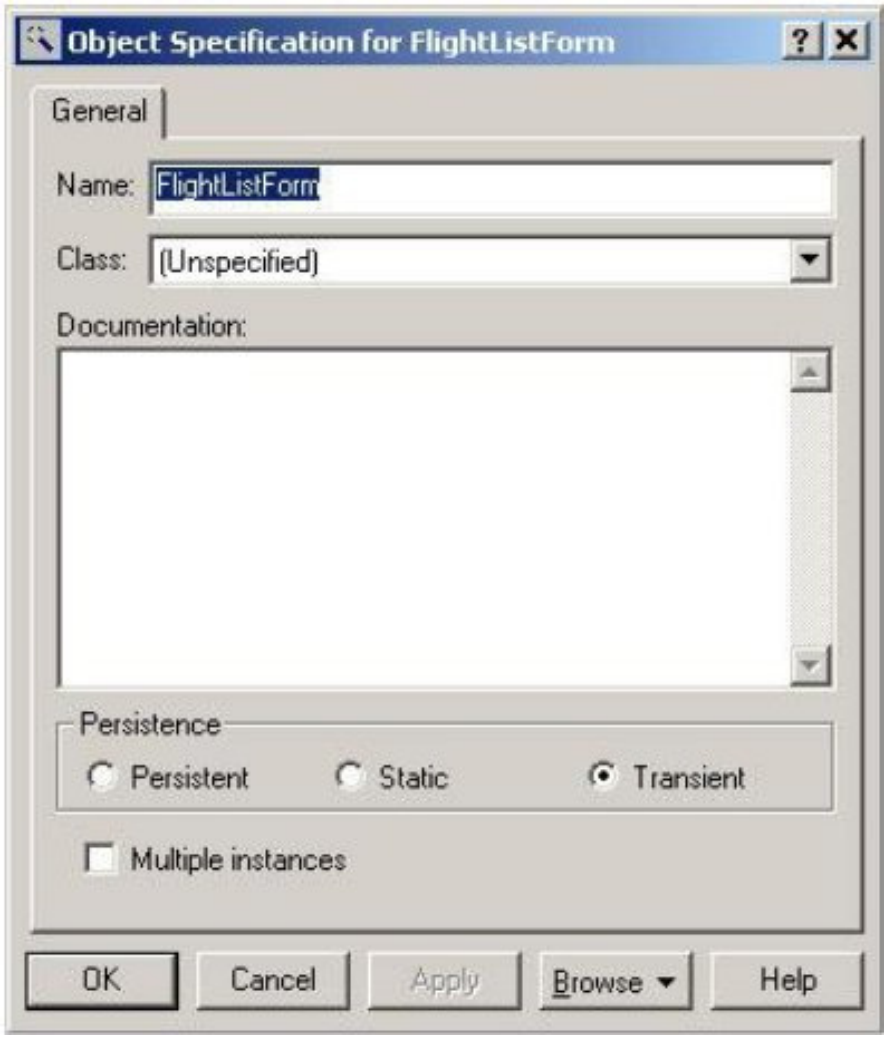
Biểu đồ trình tự

- Thí dụ: Gọi điện thoại



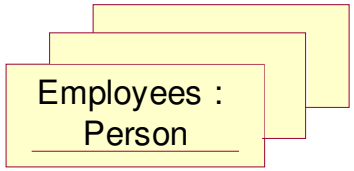
Mô tả đối tượng trong biểu đồ tương tác

- Đặc tả đối tượng
- Đặt tên
- Ánh xạ sang lớp
- Duy trì
- Đa hiện thực



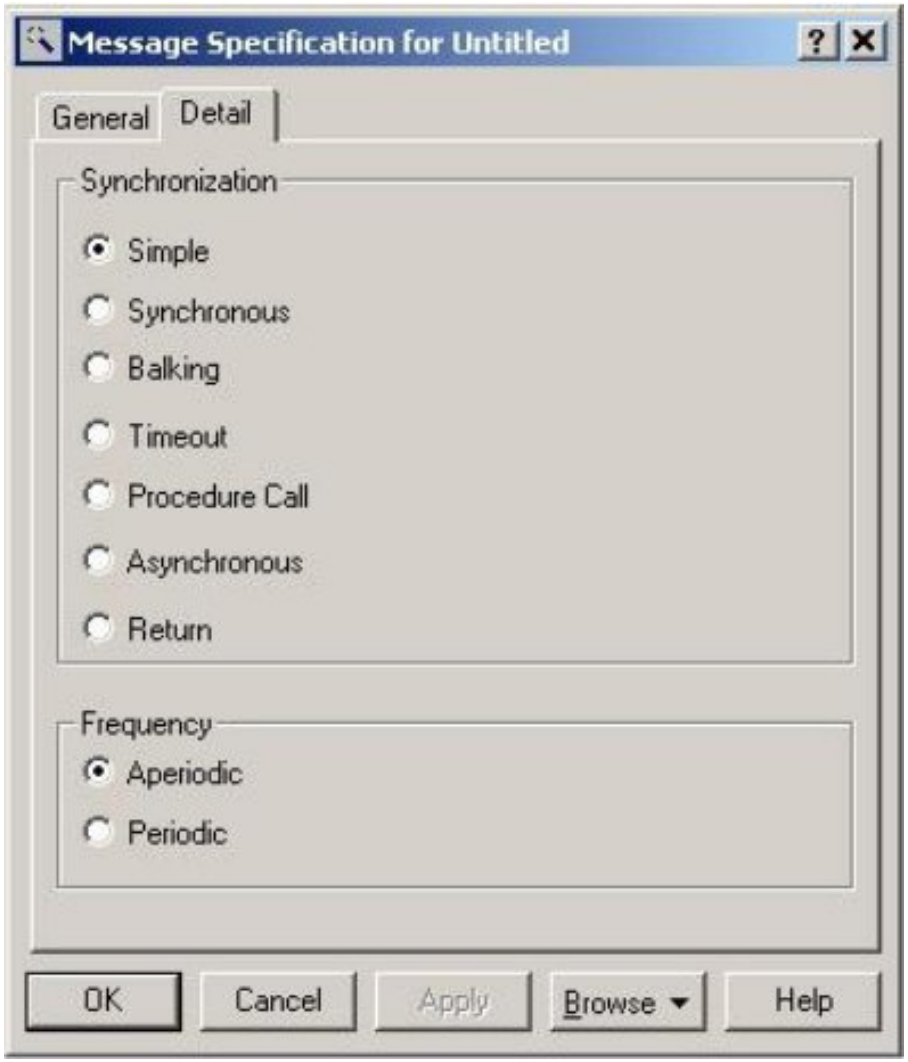
Mô tả đối tượng trong biểu đồ tương tác

- Đặt tên đối tượng
  - Tên đối tượng là cụ thể, tên lớp là tên khái quát
- Ánh xạ đối tượng sang lớp
  - Mỗi đối tượng trong biểu đồ tương tác được ánh xạ sang lớp
  - Thí dụ, Máy bay số VN358 ánh xạ sang lớp Máy bay
  - Có thể ánh xạ vào lớp mới hay lớp có sẵn trong biểu đồ
- Lựa chọn duy trì cho đối tượng
  - Persistent: có thể lưu trữ vào CSDL hay theo khuôn dạng khác
  - Static: tồn tại trong bộ nhớ cho đến khi chương trình kết thúc
  - Transient: tồn tại trong bộ nhớ với khoảng thời gian ngắn
- Đa hiện thực đối tượng (Multiple Instance)
  - UML có ký pháp dành cho đa hiện thực lớp
  - Thể hiện danh sách (nhân viên) trong biểu đồ
  - Ký pháp đồ họa đa hiện thực đối tượng trong biểu đồ cộng tác



Xây dựng biểu đồ trình tự

- Sau khi vẽ đối tượng trong biểu đồ, cần
  - vẽ liên kết các đối tượng
  - bổ sung thông điệp cho chúng
- Đặc tả thông điệp
  - Đặt tên thông điệp
  - Ánh xạ thông điệp vào thao tác
  - Đặt đặc tính đồng bộ cho thông điệp
  - Đặt tần số cho thông điệp



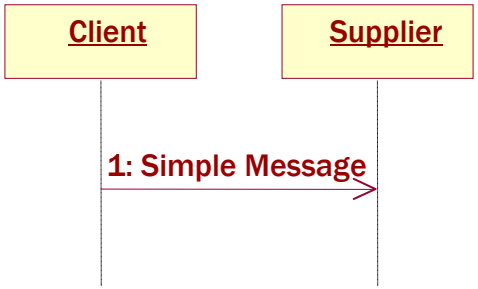
Xây dựng biểu đồ trình tự

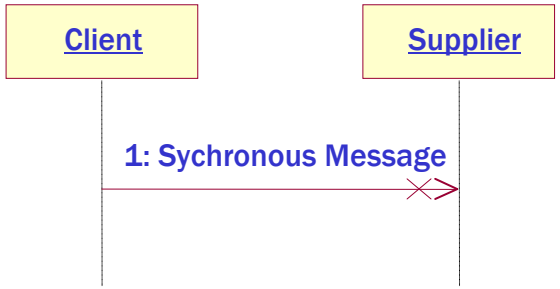
- Đặc tả thông điệp
  - Đặt tên thông điệp
    - Tên chỉ ra mục tiêu của thông điệp
  - Ánh xạ thông điệp vào thao tác
    - Trước khi phát sinh mã trình phải ánh xạ mọi thông điệp thành thao tác
  - Đặt tần số cho thông điệp
    - Đánh dấu thông điệp sẽ được gửi đều đặn, thí dụ mỗi 30s
    - Hai loại
      - Periodic: cho biết thông điệp được gửi đều đặn theo chu kỳ
      - Aperiodic: cho biết thông điệp không được gửi đều đặn mà được gửi một lần hay theo thời điểm không đều
  - Đặc tả đặc tính tương tranh cho thông điệp

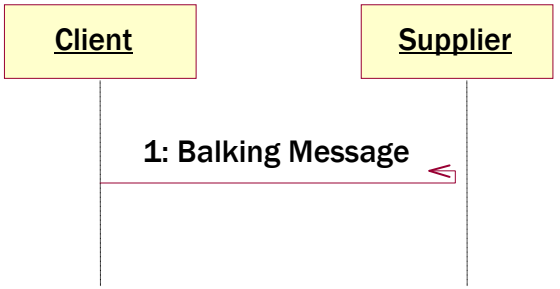
Xây dựng biểu đồ trình tự

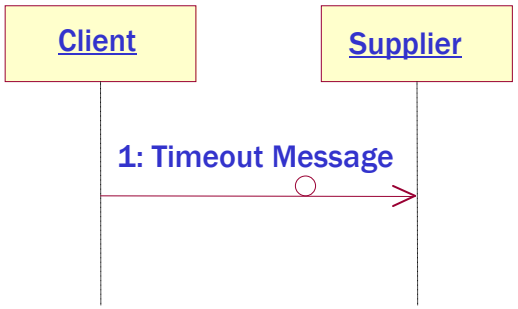
- Đặc tả thông điệp
  - Đặt tên thông điệp
  - Ánh xạ thông điệp vào thao tác
  - Đặt tần số cho thông điệp
  - Đặt đặc tính tương tranh cho thông điệp
    - Đơn (Simple): Giá trị mặc định của thông điệp, cho biết thông điệp chạy trong đơn tiến trình
    - Đồng bộ (Synchronous): Client gửi thông điệp, chờ đến khi Supplier xử lý xong thông điệp
    - Cản trở (Balking / Rendez-vous): Client gửi thông điệp, nếu Supplier không sẵn sàng xử lý ngay thông điệp hủy bỏ
    - Hết hạn (Timeout): Client gửi thông điệp chờ đến Supplier xử lý trong khoảng thời gian nhất định
    - Dị bộ (Asynchronous): Client gửi thông điệp không chờ để Supplier xử lý xong mà tiếp tục làm công việc khác
    - Lời gọi thủ tục (Procedure Call): Client gửi thông điệp đến Supplier, chờ đến khi mọi trình tự lặp của các thông điệp được xử lý xong (mũi tên đặc)
    - Trở về (Return): Mô tả trở về từ lời gọi thủ tục (mũi tên nét đứt)

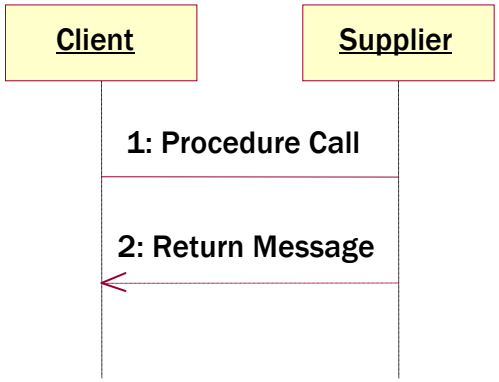
Xây dựng biểu đồ trình tự

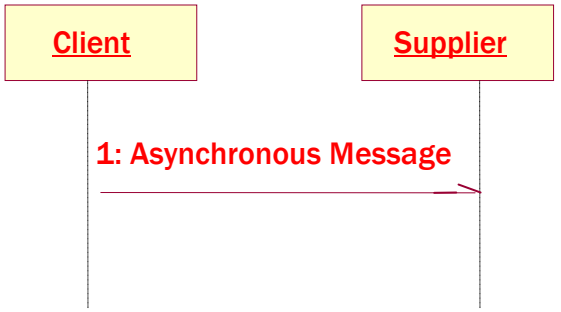


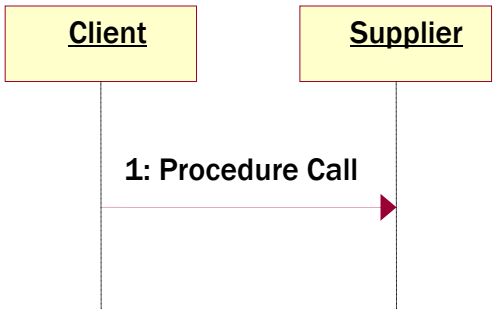








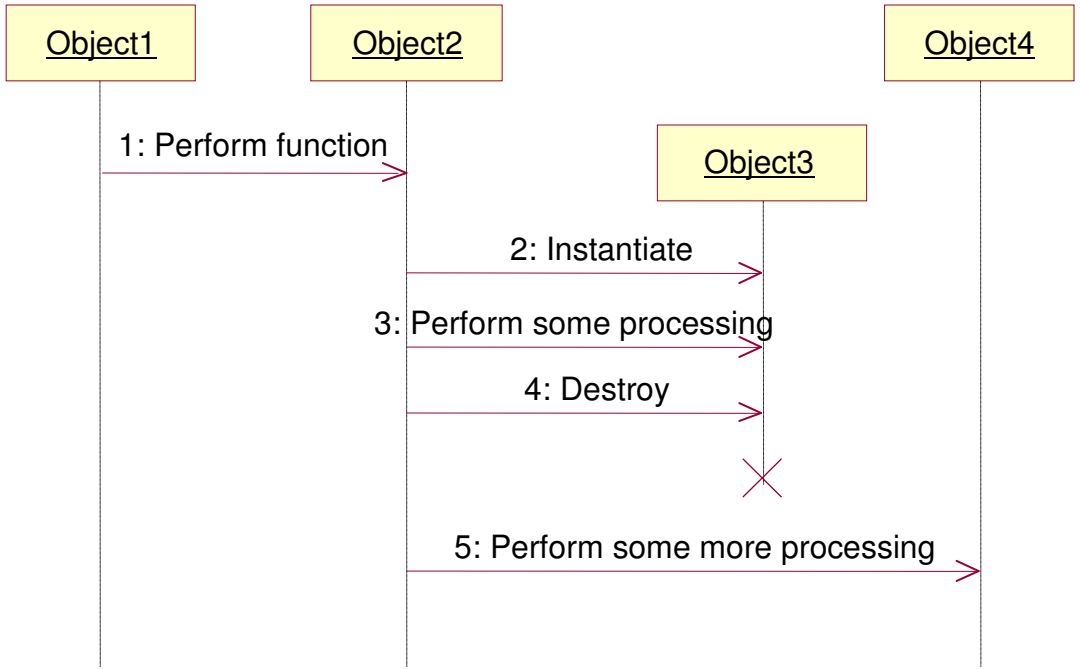




Các loại đồng bộ thông điệp trong biểu đồ trình tự

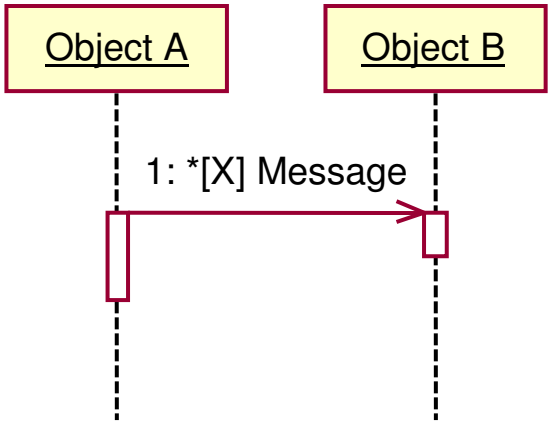
Lifeline trong biểu đồ trình tự

- Từ phiên bản Rose 2001A trở đi có khả năng
  - Biểu diễn đối tượng vào thời điểm nó được tạo lập ra
  - Đánh dấu kết thúc lifeline nơi nó bị phá hủy



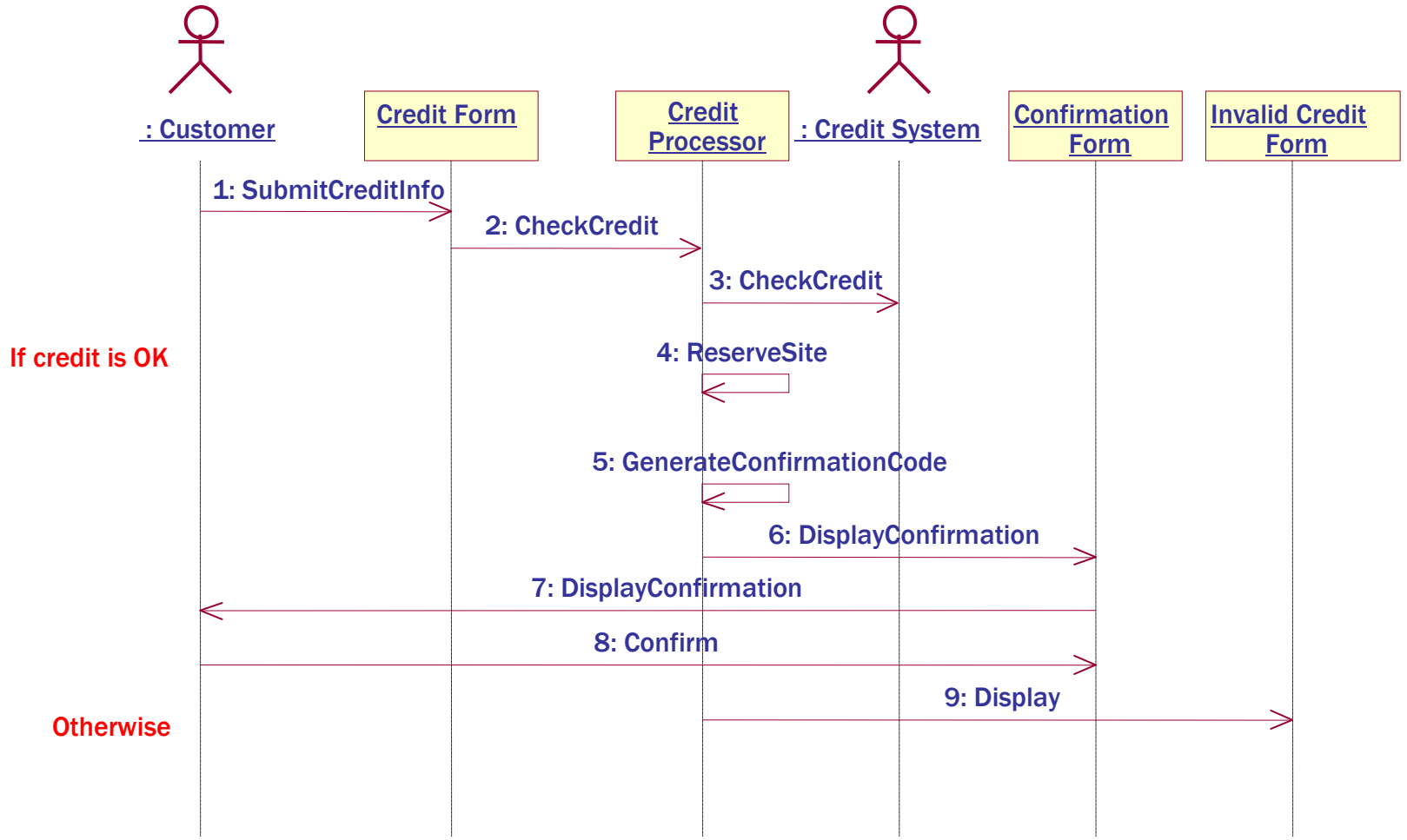
Scripts trong biểu đồ trình tự

- 1996 Buschman bổ sung scripts cho biểu đồ trình tự của UML
  - Diễn tả chú thích làm rõ các thông điệp
  - Diễn tả điều kiện logíc trong biểu đồ
- Biểu diễn tổng quát của scripts



* - Ký hiệu lặp
[] – Ký hiệu điều kiện lặp

Scripts trong biểu đồ trình tự

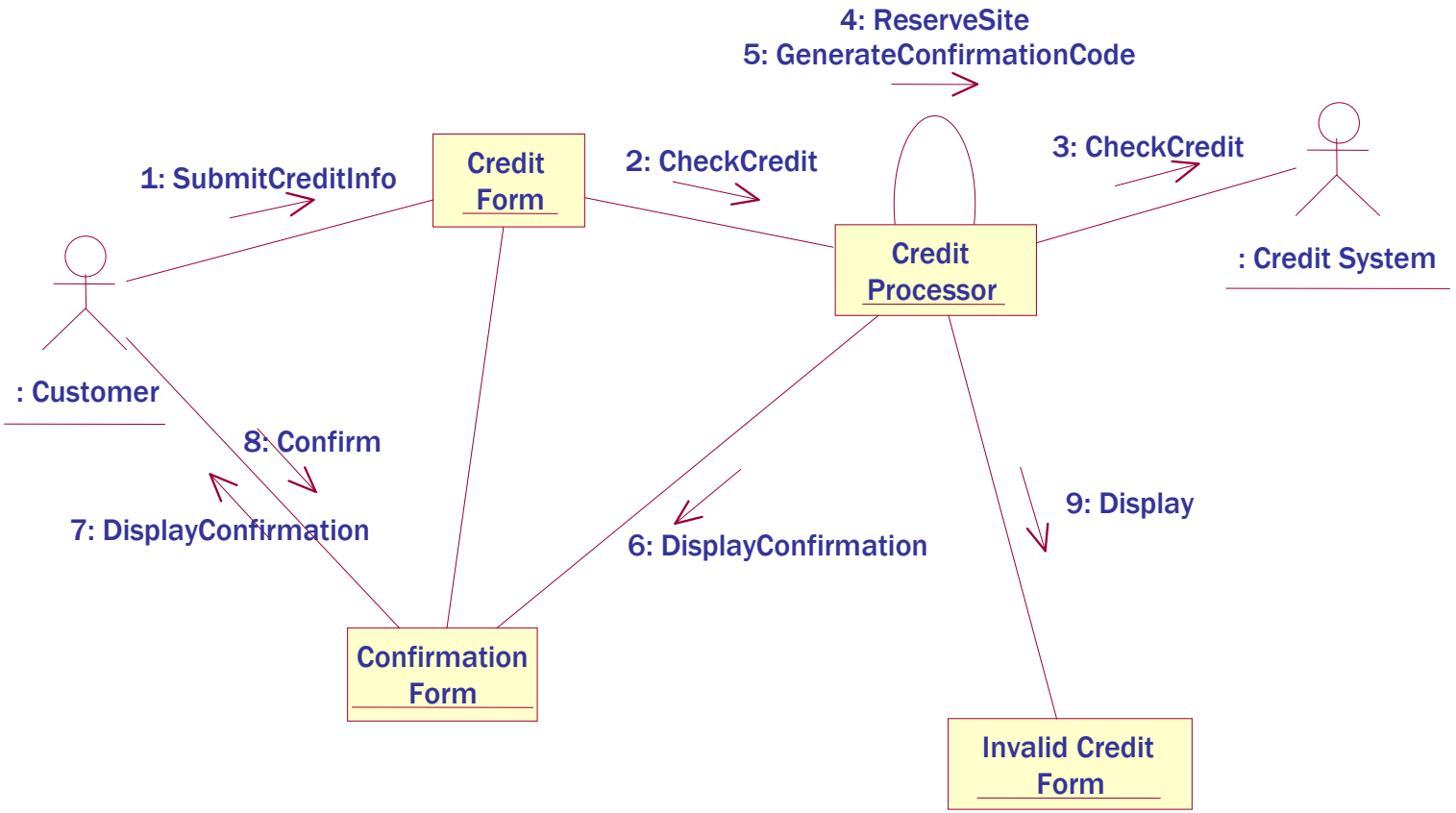


Biểu đồ cộng tác

- Tương tự biểu đồ trình tự, biểu đồ cộng tác (Collaboration diagram) chỉ ra luồng thực hiện trong kịch bản của UC
- Biểu đồ cộng tác tập trung vào
  - quan hệ giữa các đối tượng
  - cấu trúc tổ chức của các đối tượng
  - luồng dữ liệu trong kịch bản
- Tương đối khó quan sát trình tự các thông điệp trong biểu đồ cộng tác
  - Do vậy, các dự án thường hay xây dựng cả hai loại biểu đồ này
  - Trong Rose: Có thể tự động chuyển đổi qua lại giữa biểu đồ trình tự và biểu đồ cộng tác (nhấn phím F5)

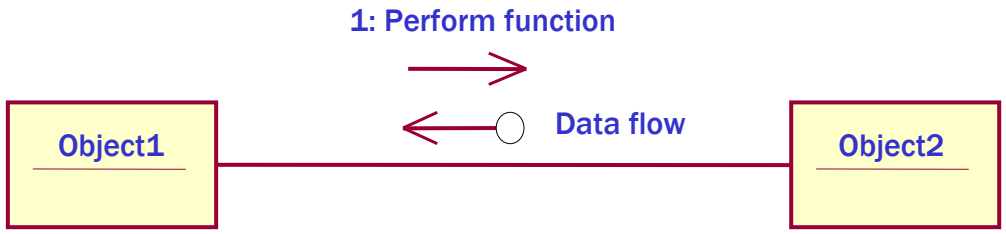
Thí dụ biểu đồ cộng tác

- Thí dụ: Luồng sự kiện Khách hàng đặt chỗ cho chuyến bay



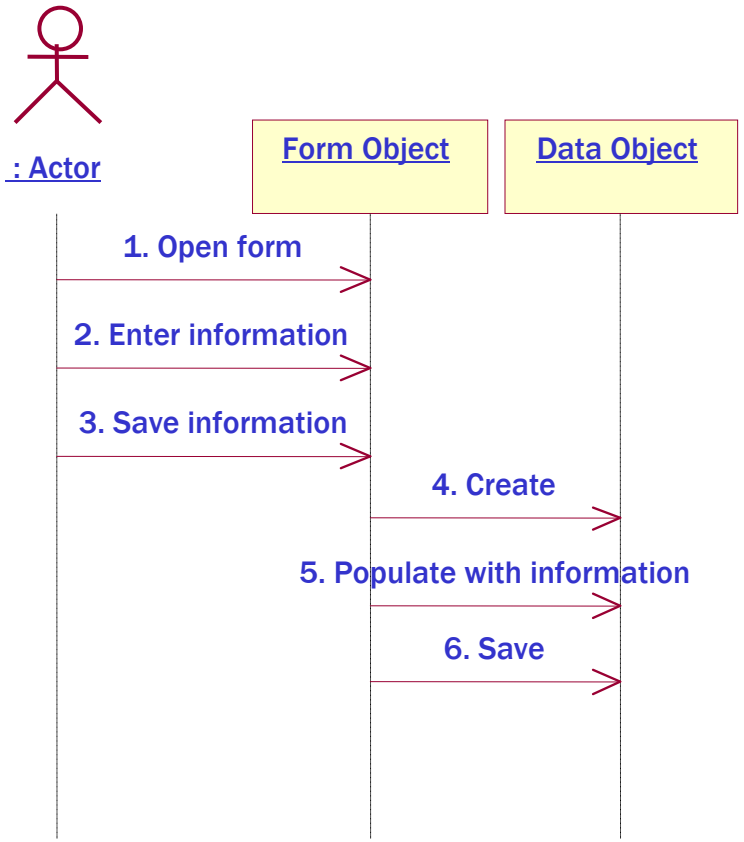
Thí dụ biểu đồ cộng tác

- Cả biểu đồ trình tự và biểu đồ cộng tác đều mô tả luồng điều khiển trong kịch bản
- Khác biệt giữa biểu đồ trình tự và biểu đồ cộng tác
  - biểu đồ cộng tác mô tả luồng dữ liệu
  - biểu đồ trình tự không mô tả luồng dữ liệu
- Luồng dữ liệu được sử dụng để mô tả thông tin trả lại khi một đối tượng gửi thông điệp đến đối tượng kia
  - Không nên bổ sung mọi luồng dữ liệu bào biểu đồ vì nó sẽ làm rối
  - Sử dụng nó khi thấy cần thiết



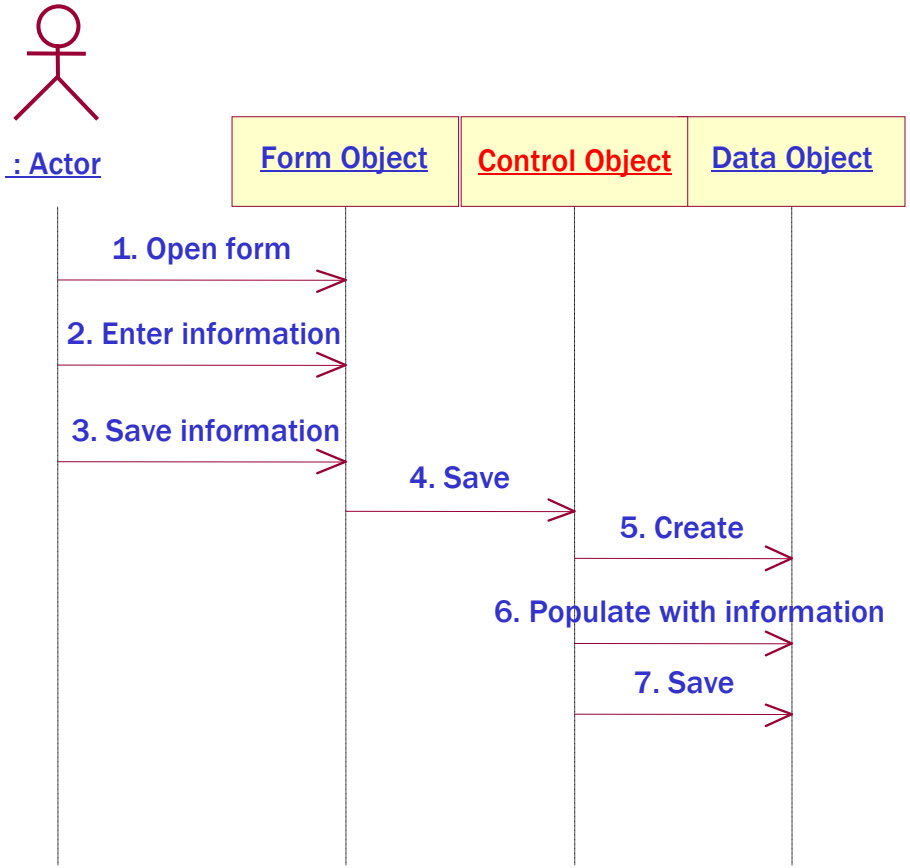
Kỹ thuật xây dựng biểu đồ tương tác

- Xây dựng biểu đồ tương tác theo tiệm cận 2 bước [Boggs]
  - Bước thứ nhất tập trung vào thông tin mức cao mà khách hàng quan tâm
    - Chưa ánh xạ thông điệp vào thao tác
    - Chưa ánh xạ đối tượng vào lớp
    - Dành cho phân tích viên, khách hàng và những ai quan tâm đến luồng nghiệp vụ -> thấy được luồng logíc trong hệ thống
  - Bước thứ 2 bổ sung chi tiết hơn vào biểu đồ tạo ra từ bước 1



Kỹ thuật xây dựng biểu đồ tương tác

- Xây dựng biểu đồ tương tác theo tiệm cận 2 bước [Boggs]
  - Bước thứ nhất tập trung vào thông tin mức cao
  - Bước thứ 2 bổ sung chi tiết hơn vào biểu đồ tạo ra từ bước 1
    - Không hiệu quả cho khách hàng
    - Rất hữu ích cho người phát triển, kiểm thử và các thành viên khác...
    - Bổ sung vào biểu đồ đối tượng điều khiển (đối tượng quản lý): Có trách nhiệm điều khiển trình tự đi qua kịch bản.
    - Mọi biểu đồ trình tự trong UC có thể cùng chia sẻ một đối tượng điều khiển.

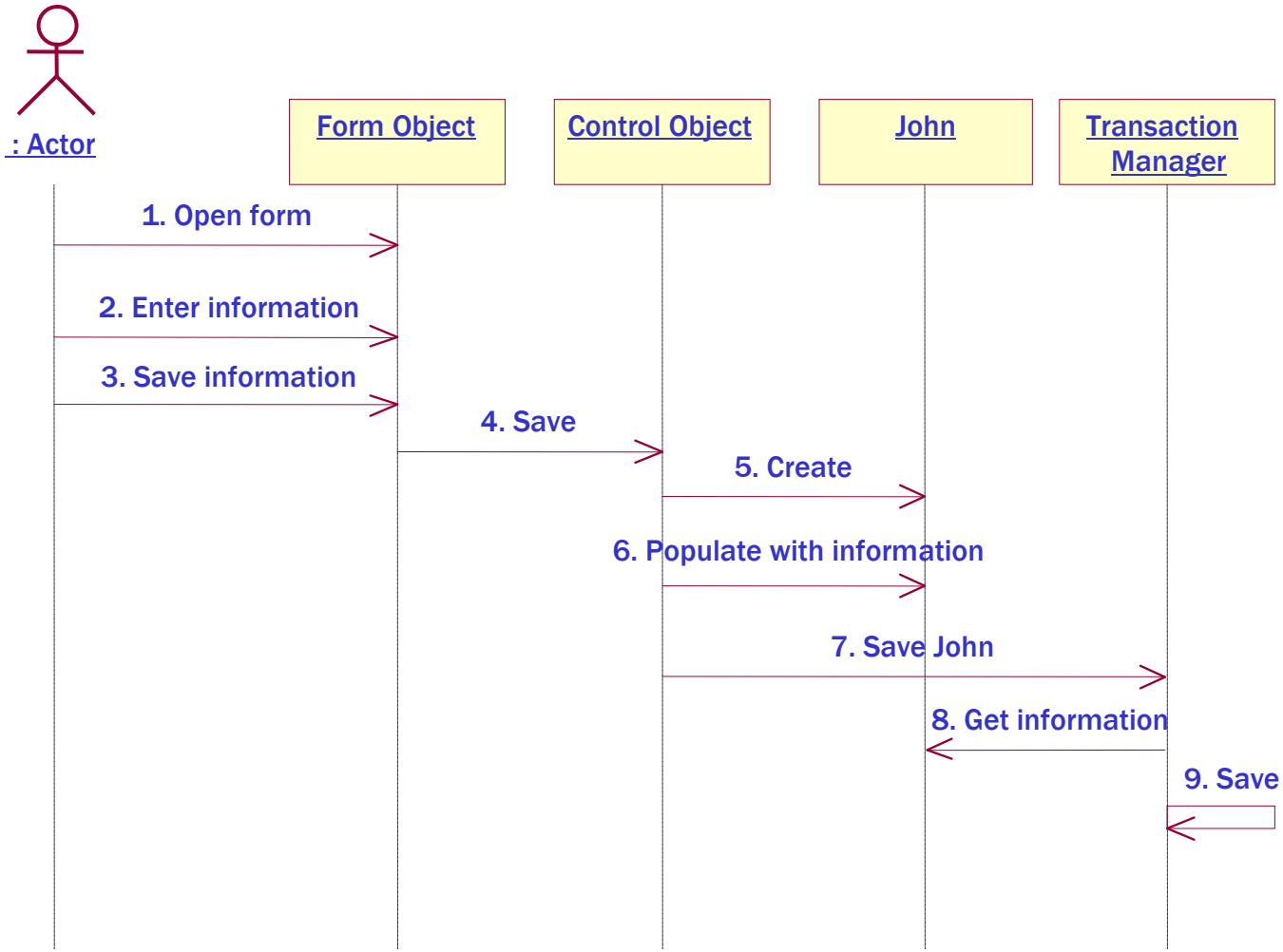


Kỹ thuật xây dựng biểu đồ tương tác

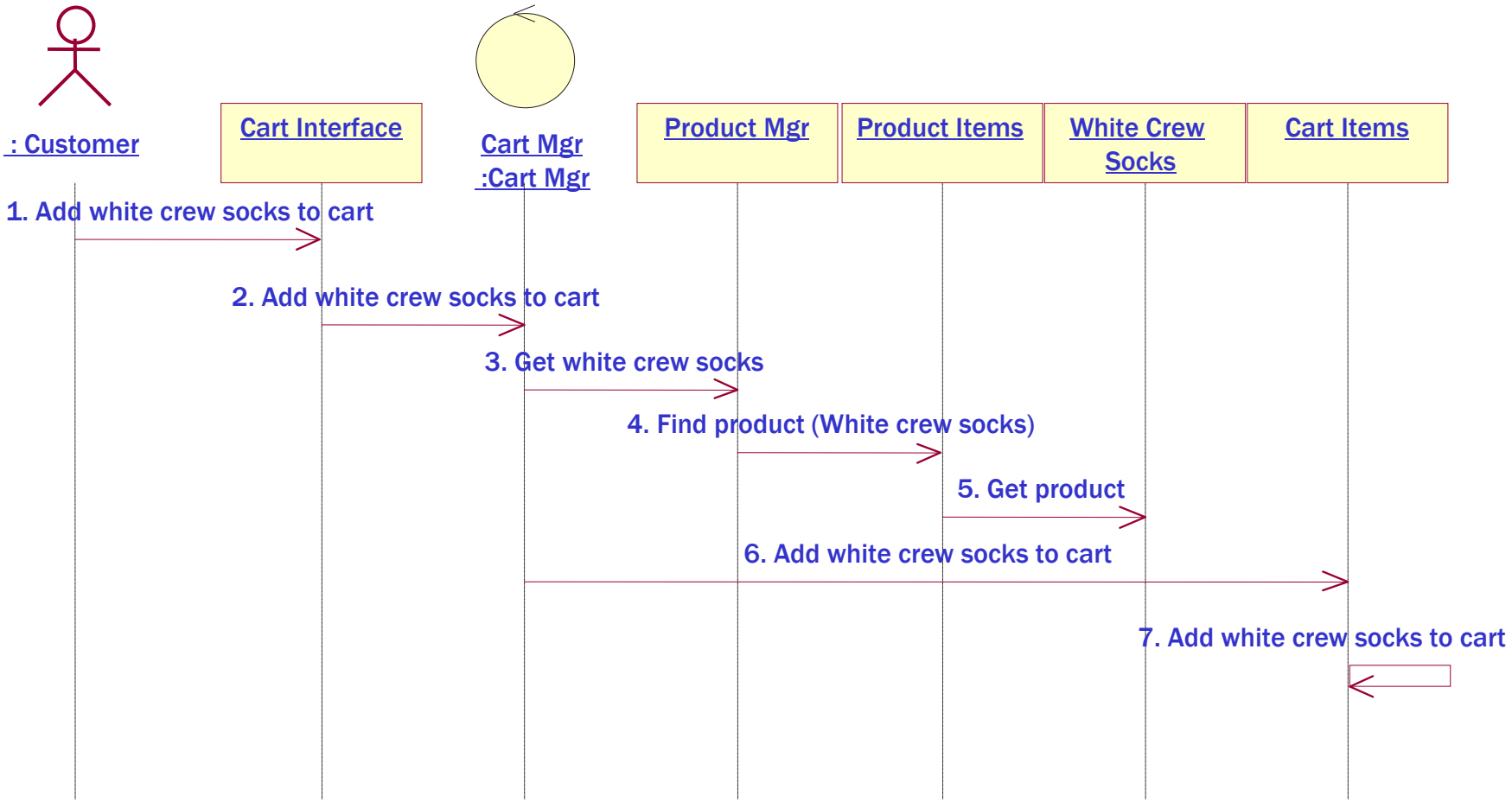
- Chi tiết hơn trong bước 2
  - Control object không thực tiến trình hiện nghiệp vụ, nó chỉ gửi thông điệp đến các đối tượng khác
    - Control object cho khả năng tách logíc nghiệp vụ khỏi logíc trình tự
    - Nếu logíc trình tự thay đổi thì chỉ ảnh hưởng đến đối tượng điều khiển
  - Có thể bổ sung các đối tượng quản lý an toàn, lỗi hay kết nối CSDL
  - Thí dụ lưu trữ hay truy vấn thông tin CSDL cho đối tượng John: có hai khả năng
    - Đối tượng John biết về CSDL và tự lưu trữ
      - Bất lợi: khi thay đổi CSDL thì sẽ phải thay đổi rất nhiều đối tượng trong hệ thống
      - Lợi thế: dễ mô hình hóa, dễ cài đặt
    - Đối tượng John tách khỏi logíc CSDL -> cần đối tượng khác thực hiện lưu trữ John vào CSDL
      - Cần tạo ra đối tượng mới để quản lý logíc CSDL gọi là Transaction Manager
      - Đối tượng Transaction Manager biết lưu trữ và truy vấn thông tin trong CSDL cho đối tượng John
      - Lợi ích: dễ sử dụng lại đối tượng John
      - Bất lợi: thời gian mô hình hóa

Kỹ thuật xây dựng biểu đồ tương tác

- Thí dụ biểu đồ trình tự sau bước hai



Thí dụ xây dựng biểu đồ tương tác

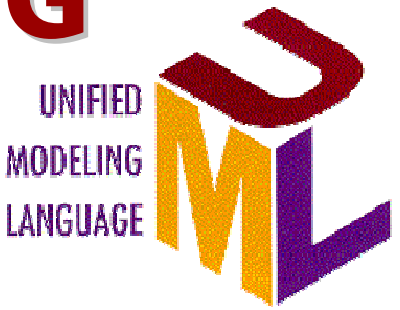


Biểu đồ trình tự cho UC "Add Item to Shopping Cart"

Tóm tắt

- Bài này đã xem xét các vấn đề sau
  - Các loại biểu đồ tương tác
    - Biểu đồ trình tự
    - Biểu đồ cộng tác
  - Tìm kiếm đối tượng, thông điệp của luồng sự kiện trong UC
  - Đặc tả các phần tử mô hình xây dựng biểu đồ tương tác
  - Kỹ thuật xây dựng biểu đồ trình tự

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

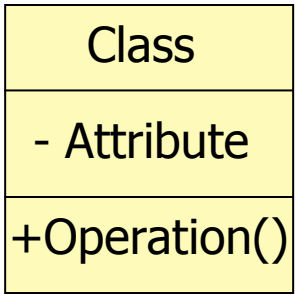
1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. ✓ **Biểu đồ lớp và gói**
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Bài 6

Biểu đồ lớp và gói

Lớp là gì?

- Đối tượng là cái gì đó tồn tại trong thế giới thực
- Lớp là mô tả thuộc tính, hành vi, ngữ nghĩa của một nhóm đối tượng
  - Lớp xác định thông tin nào được lưu trữ trong đối tượng và hành vi nào đối tượng có
- Thí dụ về lớp: Lớp Employee
  - Đối tượng của lớp có các attribute: Name, Address, Salary
  - Các operation: Thuê mướn, Đuổi việc và Đề bạt nhân viên?
- Ký pháp đồ họa của lớp trong biểu đồ
  - Tên lớp
  - Thuộc tính
  - Thao tác
    - Private
    + Public



Tìm kiếm lớp như thế nào?

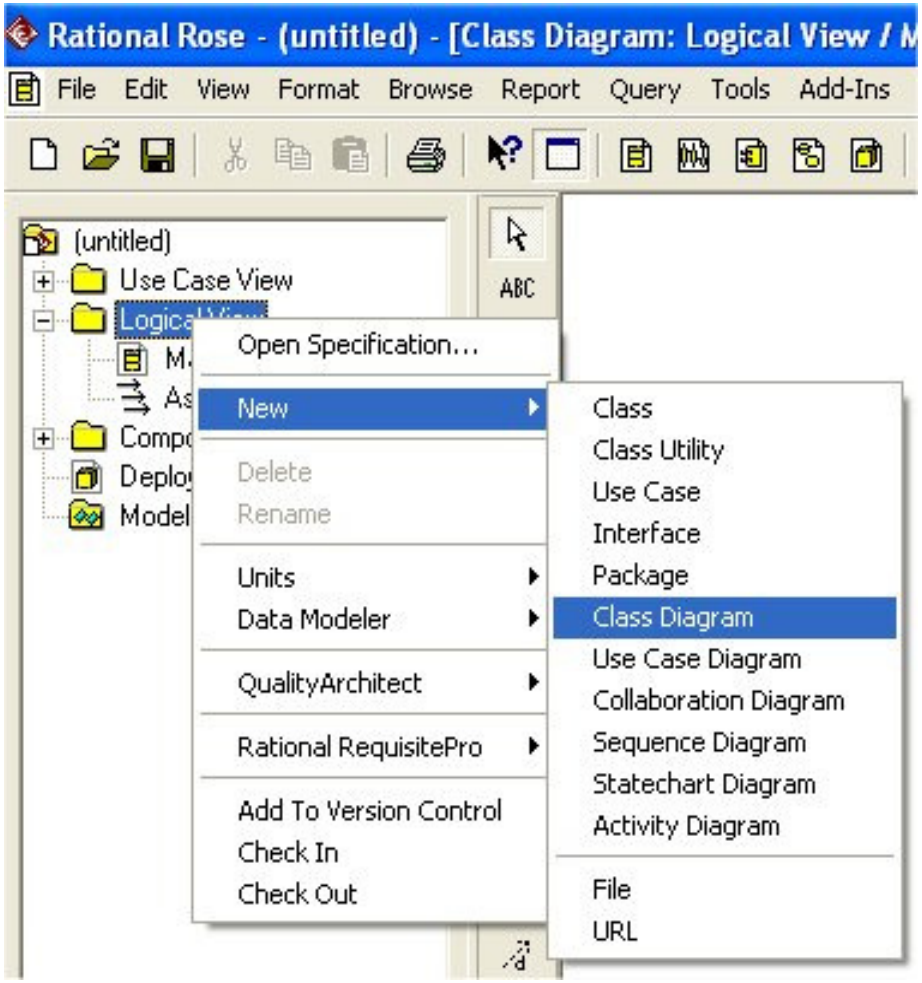
- Việc tìm kiếm đầy đủ lớp là khó khăn
- Khuyến cáo
  - Tìm lớp từ các danh từ trong luồng sự kiện
    - Chú ý rằng danh từ có thể là tác nhân, lớp, thuộc tính và biểu thức không phải loại trên
  - Tìm lớp từ biểu đồ tương tác
    - Những cái chung của đối tượng tạo thành lớp
  - Tìm lớp ở các nơi khác
    - Các báo cáo tìm ra trong pha phân tích yêu cầu hình thành lớp giao diện
    - Các thiết bị phần cứng được biểu diễn bởi lớp khác nhau

Tìm kiếm lớp như thế nào?

- Cùng với chuyên gia lĩnh vực vấn đề trả lời các câu hỏi sau đây để tìm ra lớp
  - Có thông tin nào cần lưu trữ hay phân tích? Nếu có, nó là lớp
  - Có hệ thống ngoài không? Nếu có thì nó được xem như những lớp chứa trong hệ thống của ta hay hệ thống của ta tương tác với chúng
  - Có mẫu, thư viện lớp, thành phần...? Nếu có, thông thường chúng chứa các ứng viên lớp
  - Hệ thống cần quản lý các thiết bị ngoại vi nào? Mọi thiết bị kỹ thuật nối với hệ thống đều là ứng viên lớp.
  - Tác nhân đóng vai trò tác nghiệp nào? Các nhiệm vụ này có thể là lớp; thí dụ người sử dụng, thao tác viên hệ thống, khách hàng...

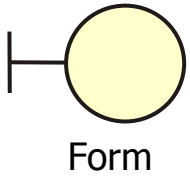
Lập biểu đồ lớp

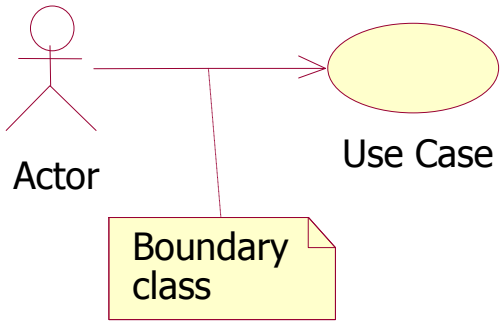
- Biểu đồ lớp cho biết hình ảnh tĩnh của bộ phận hệ thống
- Biểu đồ lớp bao gồm các lớp và quan hệ giữa chúng
- Thông thường mỗi hệ thống có vài biểu đồ lớp
  - Xây dựng vài biểu đồ lớp để mô tả đầy đủ hệ thống
- Biểu đồ lớp giúp người phát triển quan sát, lập kế hoạch cấu trúc hệ thống trước khi viết mã trình
- Rose
  - Biểu đồ lớp được hình thành trong Logical View

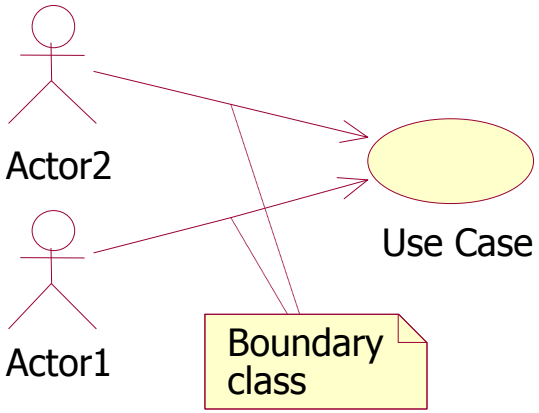


Stereotype của lớp

- Trong biểu đồ lớp, stereotype là cơ chế để phân nhóm lớp
- UML có sẵn nhiều stereotype để sử dụng
- Ba stereotype lớp cơ sở sử dụng trong pha phân tích là
  - Boundary
    - Dành cho lớp nằm trên biên hệ thống với thế giới còn lại
    - Chúng có thể là form, report, giao diện với phần cứng như máy in, scanner...
    - Khảo sát biểu đồ UC để tìm kiếm lớp biên



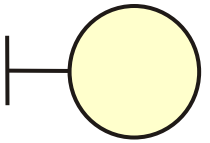




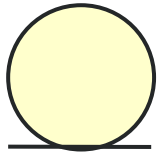
  - Entity
  - Control

Stereotype của lớp

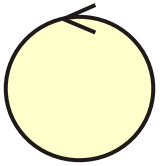
- Ba stereotype lớp cơ sở sử dụng trong pha phân tích là
  - Boundary
  - Entity
    - Lớp thực thể là lớp lưu trữ thông tin sẽ ghi vào bộ nhớ ngoài
    - Tìm chúng trong luồng sự kiện và biểu đồ tương tác
    - Thông thường phải tạo ra bảng CSDL cho lớp loại này
      - Mỗi thuộc tính của lớp thực thể sẽ là trường trong bảng CSDL
  - Control
    - Có trách nhiệm điều phối hoạt động của các lớp khác
    - Thông thường mỗi UC có một lớp điều khiển
    - Nó không thực hiện chức năng nghiệp vụ nào
    - Các lớp điều khiển khác: điều khiển sự kiện liên quan đến an ninh và liên quan đến giao dịch CSDL
- Người sử dụng tự tạo ra stereotype mới



BoundaryClass



EntityClass



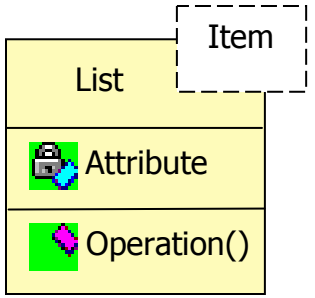
ControlClass

Các loại lớp trong biểu đồ

- Phân loại lớp theo các khái niệm của ngôn ngữ lập trình cụ thể: C++, Java, Web, Visual Basic, CORBA, Oracle...
- Rose hỗ trợ nhiều stereotype cho các nhóm lớp, thí dụ
  - Lớp thông thường
  - Lớp tham số (Parameterized class)
  - Lớp hiện thực (Instantiated class)
  - Lớp tiện ích (Class utility)
  - Lớp tiện ích tham số (Parameterized class utility)
  - Lớp tiện ích hiện thực (Instantiated class utility)
  - Metaclass
  - Giao diện (Interfaces)

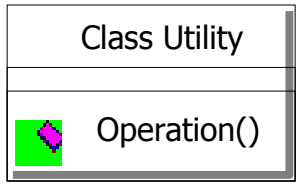
Các loại lớp trong biểu đồ

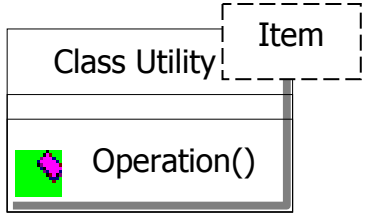
- Lớp tham số (Parameterized class)
  - Sử dụng để tạo ra họ các lớp khác
  - Có tên khác là template
  - Sử dụng cho C++, Ada
  - Thí dụ với lớp tham số List, có thể tạo ra các lớp EmployeeList, OrderList...
  - Đặt đối số cho lớp tham số
    - Các đối được hiển thị trong hộp nét đứt
- Lớp hiện thực (Instantiated class)
  - Là lớp hiện thực mà đối của chúng có giá trị
  - Trong UML, ký pháp lớp hiện thực là lớp có tên đối số trong angle brackets <>

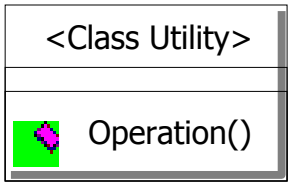


Các loại lớp trong biểu đồ

- Lớp tiện ích (Class utility)
  - Là tập hợp các thao tác
  - Thí dụ chúng có thể là các hàm toán học để các lớp khác sử dụng
  - Ký pháp đồ họa: Hình chữ nhật bóng
- Lớp tiện ích tham số (Parameterized class utility)
  - Là lớp tham số chứa tập các thao tác
  - Là template để tạo ra các lớp tiện ích
- Lớp tiện ích hiện thực (Instantiated class utility)
  - Là lớp tiện ích tham số mà đối số của chúng có giá trị

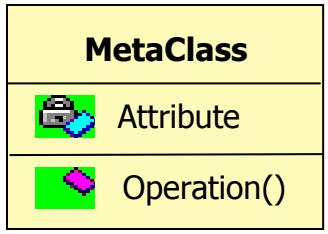




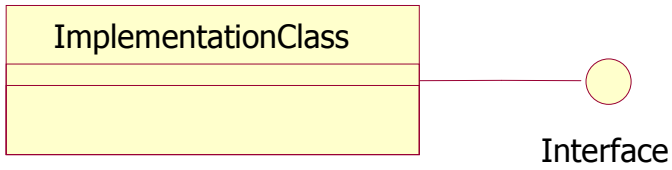


Các loại lớp trong biểu đồ

- Metaclass
  - Là lớp mà hiện thực của nó là lớp chứ không phải đối tượng

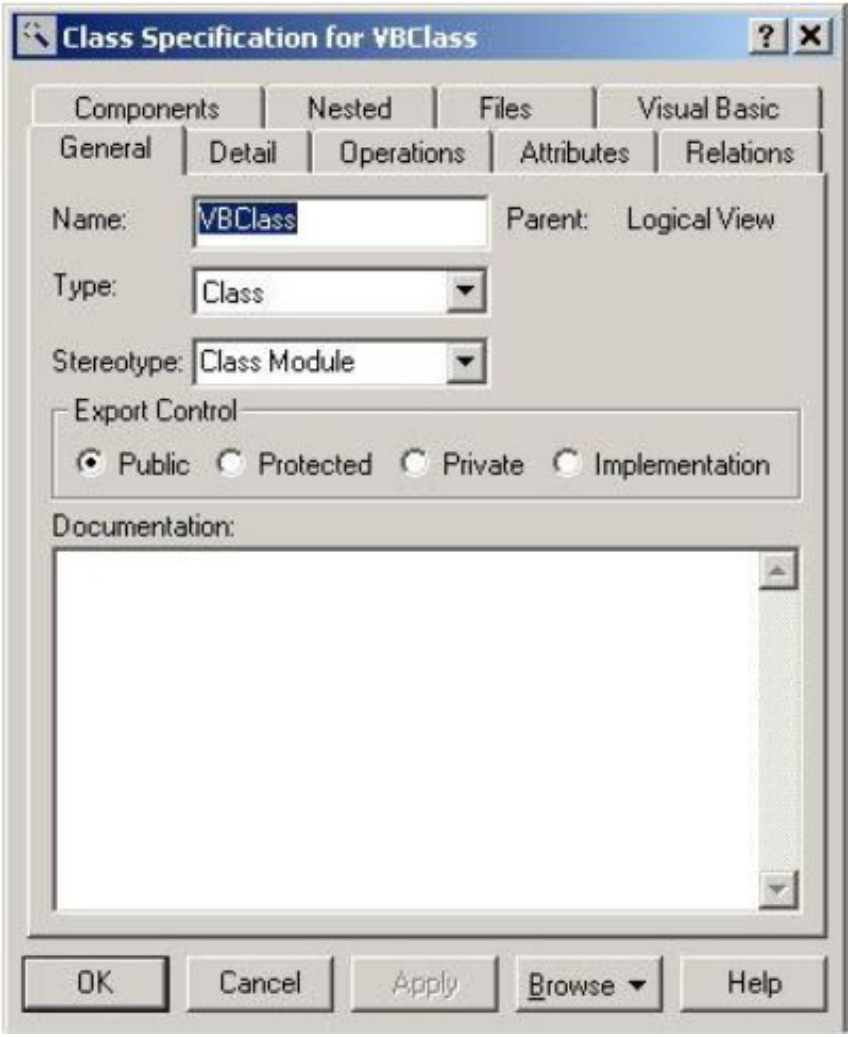


- Giao diện (Interfaces)
  - Nhiều ngôn ngữ hướng đối tượng hỗ trợ khái niệm giao diện để tách cài đặt lớp khỏi giao diện
  - Giao diện chỉ chứa signatures của phương pháp cho lớp chứ không chứa cài đặt
  - Cách tiếp cận này là cơ sở của ngôn ngữ định nghĩa giao diện (Interface Definition Language – IDL)
    - Cho phép định nghĩa giao diện độc lập ngôn ngữ



Đặc tả lớp trong biểu đồ

- Trong Rose:
  - Sử dụng cửa sổ đặc tả lớp để gán các thuộc tính cho lớp như stereotype, persistent, visibility...
  - Cửa sổ đặc tả khác nhau với các ngôn ngữ khác nhau khi chọn để cài đặt mô hình sau này
    - Các lớp của Java, XML, CORBA



Đặc tả lớp trong biểu đồ

- Đặc tả lớp bao gồm
  - Tên lớp
    - Mỗi lớp trong mô hình có tên duy nhất
    - Thông thường sử dụng danh từ đơn, không nên có dấu cách
    - Thí dụ: Flight, Airplane
  - Phạm vi (Visibility)
    - Xác định khả năng nhìn thấy lớp từ ngoài gói
    - Các loại
      - Public: mọi lớp trong hệ thống có thể nhìn thấy
      - Private hay Protected: có thể nhìn thấy từ bên trong lớp hay từ lớp friend
      - Package hay Implementation: chỉ các lớp trong cùng gói mới nhìn thấy
  - Tính nhiều (Multiplicity)
  - Yêu cầu lưu trữ
  - Duy trì (Persistent)
  - Tương tranh (Concurrency)
  - Trừu tượng (Abstract)

Đặc tả lớp trong biểu đồ

- Đặc tả lớp bao gồm
  - ...
  - Tính nhiều của lớp (Multiplicity)
    - Là số hiện thực mong đợi của lớp
    - Thí dụ: tính nhiều của lớp Employee là n, của lớp điều khiển và lớp Security Manager là 1...

Multiplicity	Ý nghĩa
n (Mặc định)	Nhiều
0..0	Không
0..1	Không hoặc 1
0..n	Không hoặc nhiều
1..1	Chính xác 1
1..n	Một hoặc nhiều

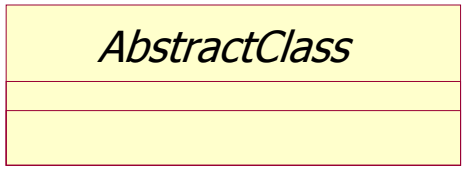
  - Yêu cầu lưu trữ cho lớp
  - ...

Đặc tả lớp trong biểu đồ

- Đặc tả lớp bao gồm
  - ...
  - Yêu cầu lưu trữ cho lớp
    - Đặt kích thước bộ nhớ mong đợi để lưu trữ đối tượng của lớp
  - Duy trì (Persistent)
    - Rose: có khả năng sinh ngôn ngữ định nghĩa dữ liệu (Data Definition Language – DDL) để định nghĩa cấu trúc của CSDL
    - Khi phát sinh DDL, Rose tìm kiếm các lớp có đánh dấu một trong hai loại đặc tính Persistence:
      - Persistent: Thông tin trong đối tượng của lớp sẽ lưu trữ vào CSDL hay tệp có khuôn mẫu khác
      - Transient: Thông tin trong đối tượng của lớp sẽ không lưu trữ lâu dài
    - Không sử dụng tính chất persistence cho lớp công cụ, lớp công cụ tham số và lớp công cụ hiện thực tham số.
  - Tương tranh (Concurrency)
  - ...

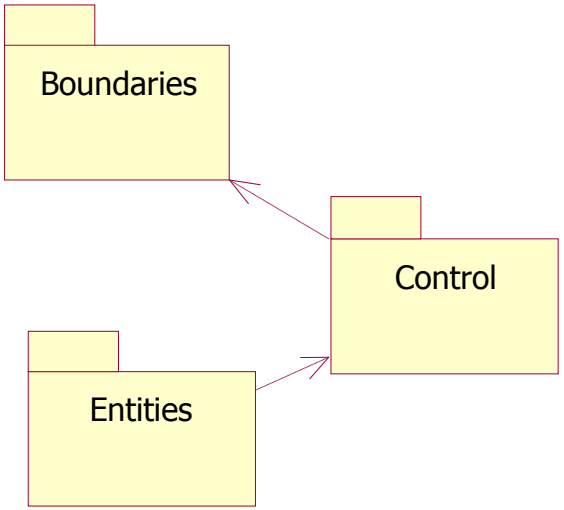
Đặc tả lớp trong biểu đồ

- Đặc tả lớp bao gồm
  - ...
  - Tương tranh (Concurrency)
    - Tương tranh mô tả ứng xử của lớp trong đa luồng điều khiển
    - Bốn loại tương tranh
      - Sequential: (trạng thái mặc định) lớp ứng xử như hoạt động chỉ trong một luồng điều khiển
      - Guarded: Lớp ứng xử như trong đa luồng điều khiển, các lớp trong các luồng khác nhau cộng tác với nhau để không làm ảnh hưởng đến các lớp khác
      - Active: Lớp có luồng điều khiển riêng
      - Synchronous: Lớp ứng xử như trong đa luồng điều khiển. Các lớp không cộng tác với nhau vì chúng hoạt động loại trừ tương hỗ.
  - Trừu tượng (Abstract)
    - Là lớp không được hiện thực hóa
    - Sử dụng trong cấu trúc kế thừa



Gói các lớp

- Gói (Packages) để nhóm các lớp có những cái chung
- Có nhiều quan điểm hình thành gói
  - Gói lớp theo prototype
    - Thí dụ có gói Boundaries, gói Control và gói Entities
  - Gói lớp theo chức năng
    - Thí dụ gói Security, gói Reporting, gói Error Handling...
  - Sử dụng tổ hợp hai loại tiếp cận trên để hình thành gói
- Có thể tổ chức gói bên trong gói khác
- Quan hệ giữa các gói hình thành trên cơ sở quan hệ giữa các lớp trong các gói.



Thuộc tính lớp

- Thuộc tính là nhóm thông tin liên kết với lớp
- Có thể gắn một hay nhiều thuộc tính vào lớp
- Tìm kiếm thuộc tính?
  - Tìm trong tài liệu UC
  - Tìm các danh từ trong luồng sự kiện
    - Thí dụ: "Người sử dụng nhập tên, địa chỉ ngày sinh của Nhân viên"

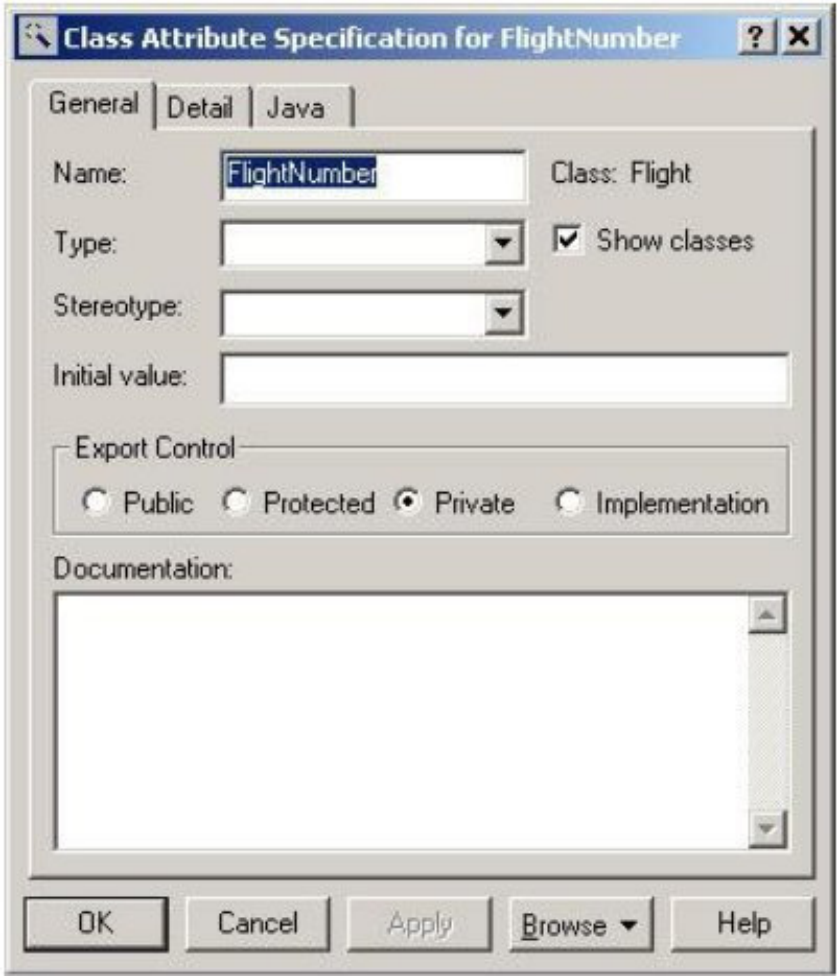
      -> Tên, địa chỉ, ngày sinh là danh từ và là thuộc tính của lớp Nhân viên
  - Tìm trong tài liệu yêu cầu hệ thống
    - Thí dụ tài liệu yêu cầu hệ thống mô tả các thông tin cần thu thập
  - Tìm thuộc tính trong cấu trúc CSDL
    - Nếu đã xác định cấu trúc CSDL thì các trường trong bảng là thuộc tính của lớp

Thuộc tính lớp

- Trong trường hợp khó khăn quyết định danh từ tìm ra là thuộc tính hay là lớp
  - Thí dụ: Tên công ty là thuộc tính hay lớp?
  - Loại ứng dụng cụ thể quyết định việc này
  - Mặt khác cần quan sát nhóm thông tin có hành vi hay không
- Khi kết thúc tìm kiếm thuộc tính
  - Đảm bảo rằng các thuộc tính tìm ra phải có ích cho yêu cầu hệ thống
  - Gán thận trọng thuộc tính cho các lớp
  - Không nên hình thành lớp có quá nhiều hay quá ít thuộc tính (tốt nhất nên có lớp ít hơn 10 thuộc tính)

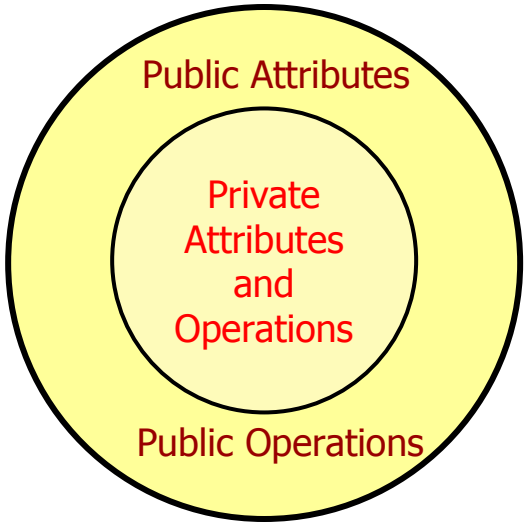
Đặc tả thuộc tính lớp

- Trong Rose: sử dụng cửa sổ đặc tả thuộc tính để gán đặc tính cho thuộc tính
- Với mỗi thuộc tính trong biểu đồ cần có
  - Tên thuộc tính
  - Kiểu dữ liệu thuộc tính lưu trữ. Phụ thuộc vào ngôn ngữ lập trình
    - Thí dụ, Add : String
  - Giá trị khởi đầu
    - Thí dụ, IDNumber: Interger=0
  - Stereotype
  - Phạm vi (visibility)
  - ....



Đặc tả thuộc tính lớp

- Với mỗi thuộc tính trong biểu đồ cần có
  - ...
  - Phạm vi (visibility)
    - Một tính chất quan trọng của lập trình hướng đối tượng là tính gói
    - Bốn lựa chọn phạm vi cho thuộc tính
      - Public: Mọi lớp đều nhìn thấy thuộc tính (+)
      - Private: Lớp khác không nhìn thấy thuộc tính (-)
      - Protected: Các lớp kế thừa có thể nhìn thấy (#)
      - Package và Implementation: Thuộc tính là public đối với các lớp trong cùng gói

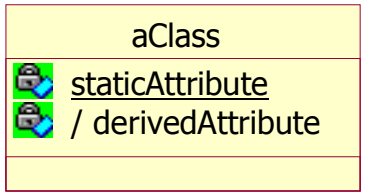


	Public	+	Public
	Private	-	Private
	Protected	#	Protected
	Package (Implementation)		

  - ...

Đặc tả thuộc tính lớp

- Với mỗi thuộc tính trong biểu đồ cần có
  - ...
  - Kiểu lưu trữ thuộc tính
    - By value: Lớp chứa thuộc tính
    - By reference: Thuộc tính đặt ngoài lớp, lớp có con trỏ đến thuộc tính
    - Unspecified: Không xác định
  - Thuộc tính tĩnh
    - Là thuộc tính chia sẻ cho mọi hiện thực lớp
    - Ký hiệu trong lớp là tên thuộc tính có gạch chân (phiên bản cũ: $)
  - Thuộc tính suy diễn
    - Là thuộc tính được tạo bởi 1 hay nhiều thuộc tính khác
    - Ký hiệu: dấu / trước tên thuộc tính
  - ...



Thao tác lớp

- Thao tác là hành vi kết hợp với lớp, nó xác định trách nhiệm của lớp
- Mô tả thao tác bao gồm
  - Tên thao tác
  - Tham số thao tác
  - Kiểu giá trị cho lại
- Ký pháp trong UML

  Operation Name (arg1: arg1 data type, arg2: arg2 data type...): return type
- Chú ý khi bổ sung thao tác trong lớp
  - Không nên để lớp chỉ có 1 hay 2 thao tác
  - Nếu lớp không có thao tác thì mô hình hóa nó như thuộc tính
  - Nếu lớp có quá nhiều thao tác thì khó quản lý, nên chia sẻ chúng ra các lớp khác

Các loại thao tác

- Thao tác cài đặt (Implementor)
  - Cài đặt một vài chức năng nghiệp vụ
  - Hầu như mọi thông điệp trong biểu đồ tương tác ánh xạ vào thao tác cài đặt
- Thao tác quản lý (Manager)
  - Quản lý việc lập và hủy bỏ đối tượng
    - Thí dụ: các cấu tử, hủy tử của lớp
- Thao tác xâm nhập (Access)
  - Thao tác xâm nhập vào các thuộc tính private và protected
    - Thí dụ: các thao tác Get và Set cho mỗi thuộc tính trong lớp
- Thao tác trợ giúp (Helper)
  - Là các thao tác private và protected của lớp
  - Các thông điệp phản thân trong biểu đồ tương tác ánh xạ đến thao tác này

Quan hệ giữa các lớp

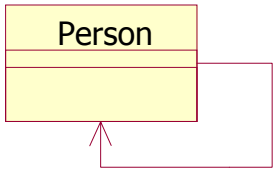
- Quan hệ là kết nối ngữ nghĩa giữa các lớp
  - Quan hệ cho một lớp biết thuộc tính, thao tác và quan hệ của lớp khác
- Các loại quan hệ chính
  - Kết hợp (Associations)
  - Phụ thuộc (Dependencies)
  - Tụ hợp (Aggregations)
  - Hiện thực quan hệ (Realizes Relationships)
  - Khái quát hóa (Generalizations)

Tìm kiếm quan hệ

- Khảo sát biểu đồ trình tự và biểu đồ cộng tác
  - Nếu lớp A gửi thông điệp đến lớp B thì giữa chúng có quan hệ
    - Thông thường là quan hệ kết hợp hay phụ thuộc
- Khảo sát các lớp để tìm ra các quan hệ
  - Quan hệ tổng thể - thành phần
    - Bất kỳ lớp nào được hình thành từ lớp khác thì chúng có quan hệ tập hợp
  - Quan hệ khái quát hóa
    - Nếu nhiều lớp kế thừa từ lớp thứ ba thì giữa chúng với lớp thứ ba có quan hệ khái quát hóa

Quan hệ kết hợp

- Association là kết nối ngữ nghĩa giữa các lớp
  - Kết hợp cho một lớp biết về thuộc tính và thao tác public của lớp khác
  - Quan hệ kết hợp hai chiều, một chiều
  - Quan hệ kết hợp phản thân




```
class Person
{
public:
Person();
~Person();
private:
House *the_House;
};
```

```
class House
{
public:
House();
~House();
private:
Person *the_Person;
};
```

```
class Person
{
public:
Person();
~Person();
private:
House *the_House;
};
```

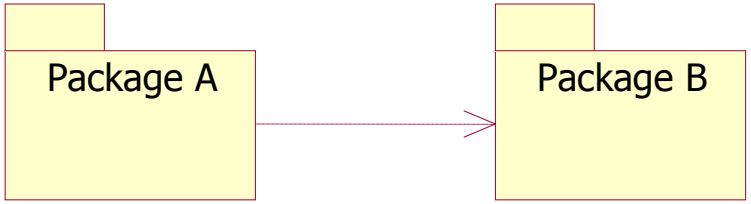
```
class House
{
public:
House();
~House();
private:
};
```

Quan hệ phụ thuộc

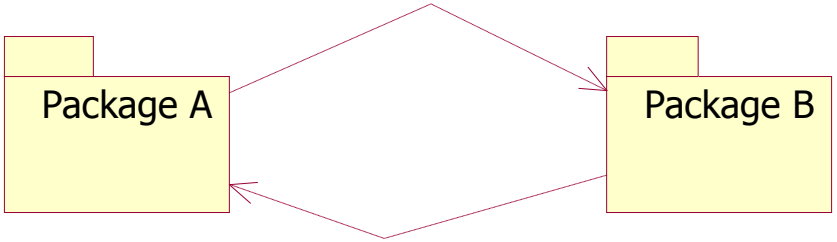
- Dependency là quan hệ chỉ ra một lớp tham chiếu lớp khác
  - Khi thay đổi đặc tả lớp tham chiếu thì lớp sử dụng nó bị ảnh hưởng
  - Rose không bổ sung thuộc tính cho hai lớp có quan hệ phụ thuộc
  - Các lệnh ngôn ngữ được phát sinh để hỗ trợ quan hệ phụ thuộc
    - Thí dụ: #include

Quan hệ phụ thuộc gói

- Có thể vẽ quan hệ phụ thuộc giữa các gói như giữa các lớp
- Phụ thuộc gói từ gói A đến gói B có nghĩa rằng vài gói trong lớp A có quan hệ một chiều với các lớp trong gói B

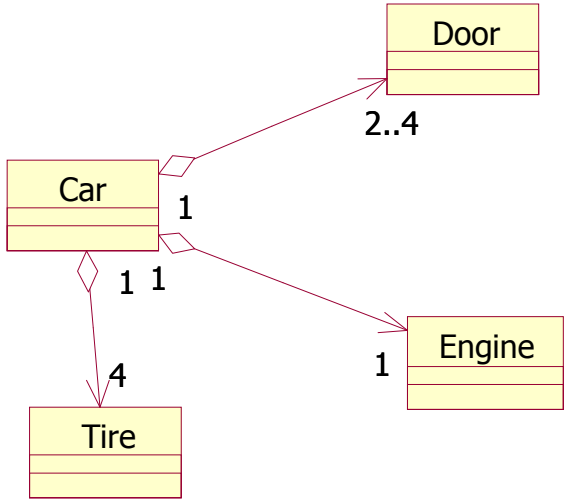


- Tránh phụ thuộc vòng giữa các gói



Phụ thuộc tụ hợp

- Aggregation là quan hệ giữa tổng thể và bộ phận (Whole-Parts)
  - Trong quan hệ này, một lớp biểu diễn cái lớn hơn còn lớp kia biểu diễn cái nhỏ hơn
  - Biểu diễn quan hệ has-a
    - Một đối tượng của lớp tổng thể có nhiều đối tượng của lớp thành phần
  - Tổng thể và bộ phận có thể hủy bỏ vào thời điểm khác nhau
  - Tên khác: quan hệ tụ hợp bởi tham chiếu (by reference)




```
#include "car.h"
class Door {
.....
private:
Car *the_car;
};

#include "door.h"
class Car {
...
private:
Door *the_Door;
};
```

Quan hệ gộp

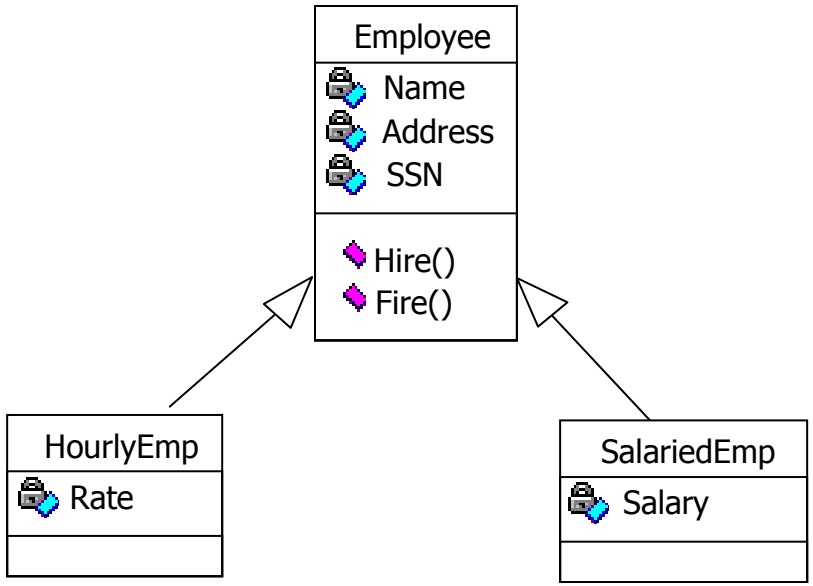
- Composition là dạng đặc biệt (mạnh hơn) của quan hệ tụ hợp
  - Tổng thể và thành phần được hình thành hay hủy bỏ vào cùng thời điểm
  - Tên khác: quan hệ tụ hợp bởi giá trị (by value)


```
#include "Window.h"
class Frame
{
...
private:
}
```

```
#include "Frame.h"
class Window
{
...
private:
Frame the_Frame;
}
```

Quan hệ khái quát hóa

- Generalization là quan hệ kế thừa của hai phần tử mô hình như lớp, tác nhân, Use case và gói
  - Cho phép một lớp kế thừa các thuộc tính, thao tác public và protected của lớp khác




```
#include "Employee.h"
class HourlyEmp: public Employee
{
private:
   float Rate;
.....
};
```

Đặc tả quan hệ giữa các lớp

- Đặc tả chi tiết quan hệ bao gồm
  - Multiplicity
  - Tên quan hệ
  - Tên nhiệm vụ
  - Export control
  - Quan hệ tĩnh
  - Quan hệ Friend
  - Phạm vi kết hợp (Qualifier)
  - Phần tử liên kết

Đặc tả quan hệ giữa các lớp

- Đặc tả chi tiết quan hệ bao gồm
  - Multiplicity
    - Cho biết bao nhiêu hiện thực của một lớp liên kết với một hiện thực của lớp khác vào cùng thời điểm

  - Tên quan hệ
    - Tên quan hệ là động từ mô tả tại sao lại tồn tại quan hệ

  - ...

Đặc tả quan hệ giữa các lớp

- Đặc tả chi tiết quan hệ bao gồm
  - .....
  - Tên nhiệm vụ
    - Sử dụng tên nhiệm vụ thay thế cho tên quan hệ trong quan hệ kết hợp hay tụ hợp để chỉ ra tại sao quan hệ tồn tại

  - Export control
    - Trong quan hệ kết hợp, thuộc tính được phát sinh trong mã trình
    - Phạm vi được gán cho thuộc tính bao gồm: Public, Private, Protected, Package hay Implementation
  - ...

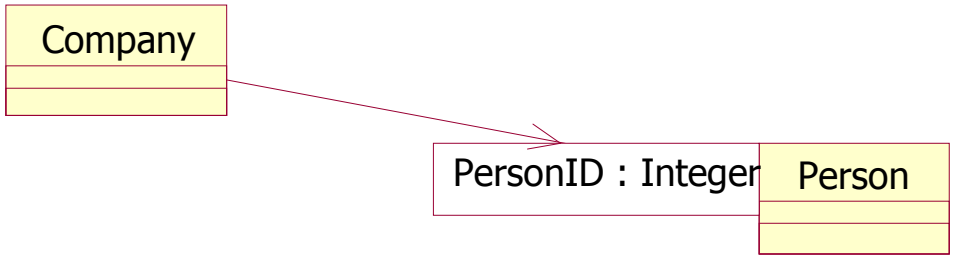
Đặc tả quan hệ giữa các lớp

- Đặc tả chi tiết quan hệ bao gồm
  - .....
  - Quan hệ tĩnh
    - Rose phát sinh thuộc tính cho quan hệ kết hợp và quan hệ tụ hợp
    - Có thể gán tính static cho thuộc tính để chia sẻ cho mọi hiện thực lớp

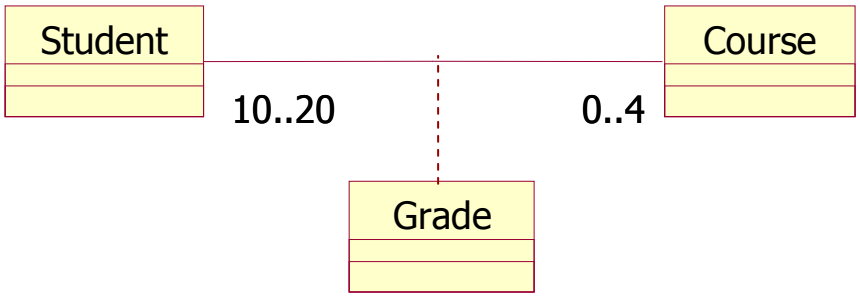
  - Quan hệ Friend
    - Quan hệ này chỉ ra rằng lớp Client có thể xâm nhập các thuộc tính và thao tác không phải public của lớp Supplier
    - Có thể gán Friend cho kết hợp, tụ hợp, phụ thuộc hay khái quát hóa
    - Mã nguồn của lớp Supplier sẽ bao gồm logic để lớp Client có phạm vi Friend
  - ...

Đặc tả quan hệ giữa các lớp

- Đặc tả chi tiết quan hệ bao gồm
  - .....
  - Phạm vi kết hợp (Qualifier)
    - Sử dụng qualifier để giảm phạm vi kết hợp



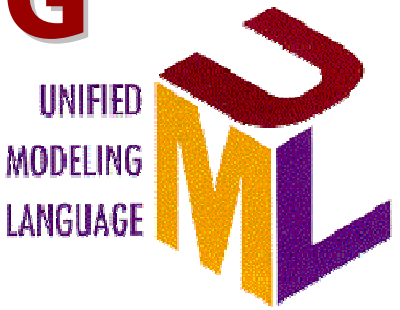
  - Phần tử liên kết
    - Còn gọi là lớp kết hợp, nơi lưu trữ thuộc tính liên quan đến kết hợp



Tóm tắt

- Bài này đã xem xét các vấn đề sau
  - Tìm kiếm lớp
  - Tìm kiếm thuộc tính, thao tác lớp
  - Tìm kiếm các loại quan hệ giữa các lớp
  - Biểu diễn biểu đồ lớp và gói
  - Biểu diễn đồ họa các thuộc tính của thuộc tính, thao tác trong lớp
  - Biểu diễn các thuộc tính cho quan hệ giữa các lớp

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói

✓ **Biểu đồ chuyển trạng thái và biểu đồ hoạt động**

8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Bài 7

Biểu đồ chuyển trạng thái và biểu đồ hoạt động

Biểu đồ chuyển trạng thái

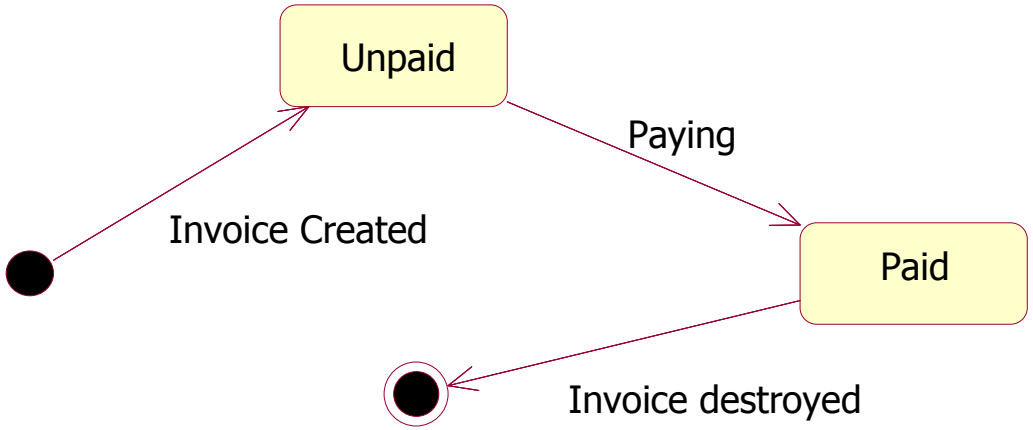
- Mô tả chu kỳ tồn tại của đối tượng từ khi nó sinh ra đến khi nó bị phá hủy
- Sử dụng để mô hình hóa khía cạnh động của lớp
- Biểu đồ bao gồm các thông tin sau
  - Các trạng thái của đối tượng
  - Hành vi của đối tượng
  - Sự kiện tác động làm thay đổi trạng thái
- Thông thường
  - Xây dựng biểu đồ chuyển trạng thái cho một vài đối tượng của lớp có nhiều hành vi động trong dự án
  - Không phải mọi dự án sử dụng biểu đồ loại này

Trạng thái đối tượng?

- Trạng thái đối tượng là kết quả của các hoạt động trước đó của đối tượng
- Đối tượng luôn ở trong một trạng thác xác định tại một thời điểm
  - Trạng thái được xác định bởi giá trị của thuộc tính và liên kết với đối tượng khác
- Thí dụ
  - Con người cụ thể của lớp Person có các trạng thái: Người lao động, Thất nghiệp, Về hưu
  - Hóa đơn mua hàng: Đã thanh toán, chưa thanh toán
  - Xe ô tô: Đang chạy, Đang đứng
- Thay đổi trạng thái đối tượng
  - Có sự kiện xảy ra
  - Thí dụ: ai đó thanh toán hóa đơn hàng

Biểu đồ trạng thái

- Thí dụ biểu đồ trạng thái



- Sử dụng biểu đồ trạng thái để làm gì?
  - Phân tích viên, người thiết kế và người sử dụng hiểu hành vi đối tượng
  - Người phát triển hiểu hành vi đối tượng để cài đặt nó

Biểu đồ trạng thái

- Các phần tử đồ họa
  - Trạng thái khởi đầu: Khi đối tượng được tạo ra

  - Trạng thái dừng: Khi đối tượng bị phá hủy

  - Trạng thái (State)
    - Hoạt động
    - Hành động vào
    - Hành động ra
  - Quá độ (Transition)
    - Sự kiện
    - Điều kiện canh
    - Hành động
  - Trạng thái ẩn

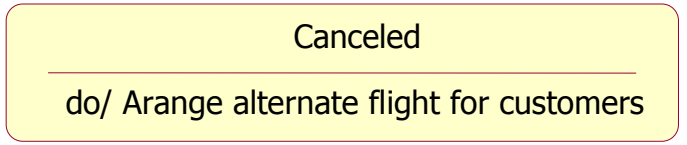
Biểu đồ trạng thái

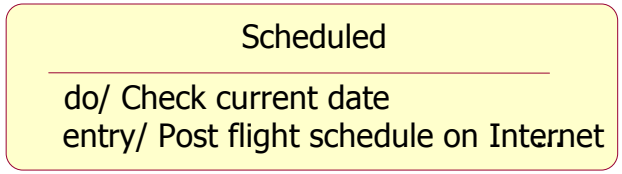
- Các phần tử đồ họa
  - ...
  - Trạng thái (State)
    - Trạng thái được xác định từ khảo sát thuộc tính lớp và quan hệ giữa các lớp
    - Ký pháp đồ họa

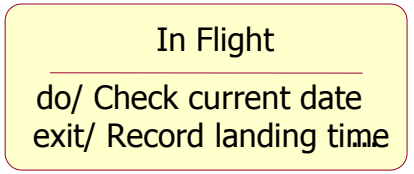
    - Khi đối tượng trong trạng thái nào đó nó thực hiện vài hoạt động (Activity)
      - Phát sinh báo cáo, Thực hiện tính toán và Gửi thông điệp đến đối tượng khác
    - Có năm loại thông tin có thể gộp trong trạng thái
      - Hoạt động, Hành động vào, Hành động ra, Sự kiện, Lịch sử trạng thái.
  - ...

Biểu đồ trạng thái

- Các phần tử đồ họa
  - ...
  - Trạng thái (State)
    - Hoạt động (Activity)
      - Là hành vi mà đối tượng thực hiện khi nó đang ở trạng thái cụ thể
      - Nó là hành vi có thể ngắt được
      - Biểu diễn trong phần tử biểu đồ: do hay /
    - Hành động vào (Entry Action)
      - Là hành vi xảy ra khi đối tượng đang chuyển vào trạng thái
      - Nó là hành vi không thể ngắt được
      - Biểu diễn trong phần tử biểu đồ: Entry
    - Hành động ra (Exit Action)
      - Là hành vi xảy ra khi đối tượng đang chuyển ra trạng thái
      - Nó là hành vi không thể ngắt được
      - Biểu diễn trong phần tử biểu đồ: Exit
  - ...



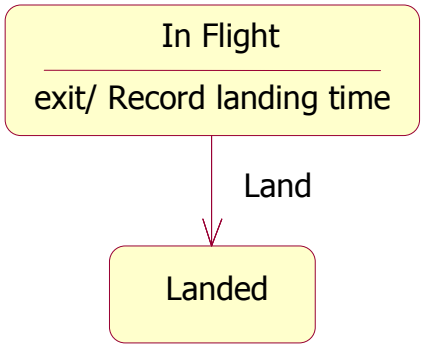


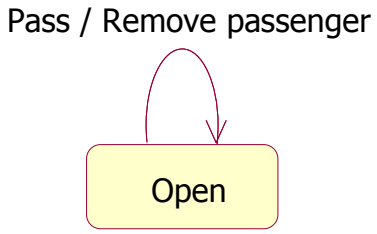


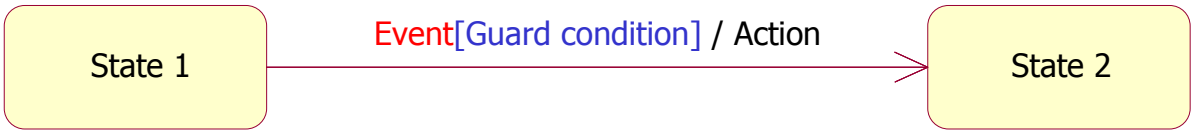
Biểu đồ trạng thái

- Các phần tử đồ họa
  - ...
  - Quá độ (Transition)
    - Quá độ là chuyển động từ trạng thái này sang trạng thái khác
    - Quá độ phản thân
    - Đặc tả quá độ
      - Sự kiện (Event): cái gì đó là nguyên nhân chuyển từ trạng thái này sang trạng thái khác

        Hầu hết quá độ đều có sự kiện. Sự kiện có thể có đối số, thí dụ, Remove passenger(name)
      - Điều kiện canh (Guard): xác định khi nào sự kiện xảy ra, thí dụ, Trạng thái máy bay từ Open sang Full khi chỗ cuối cùng đã có người mua vé
      - Hành động (Action): hành vi không ngắt được, xảy ra như một phần của chuyển tiếp.
  - ...



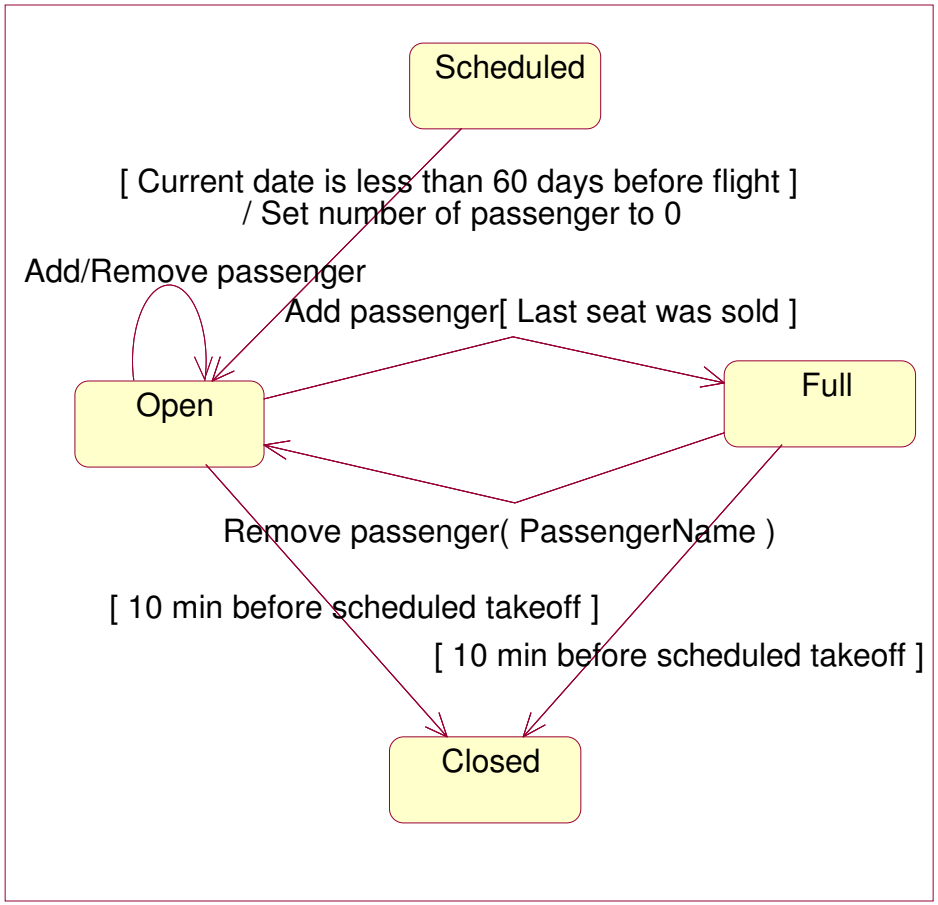


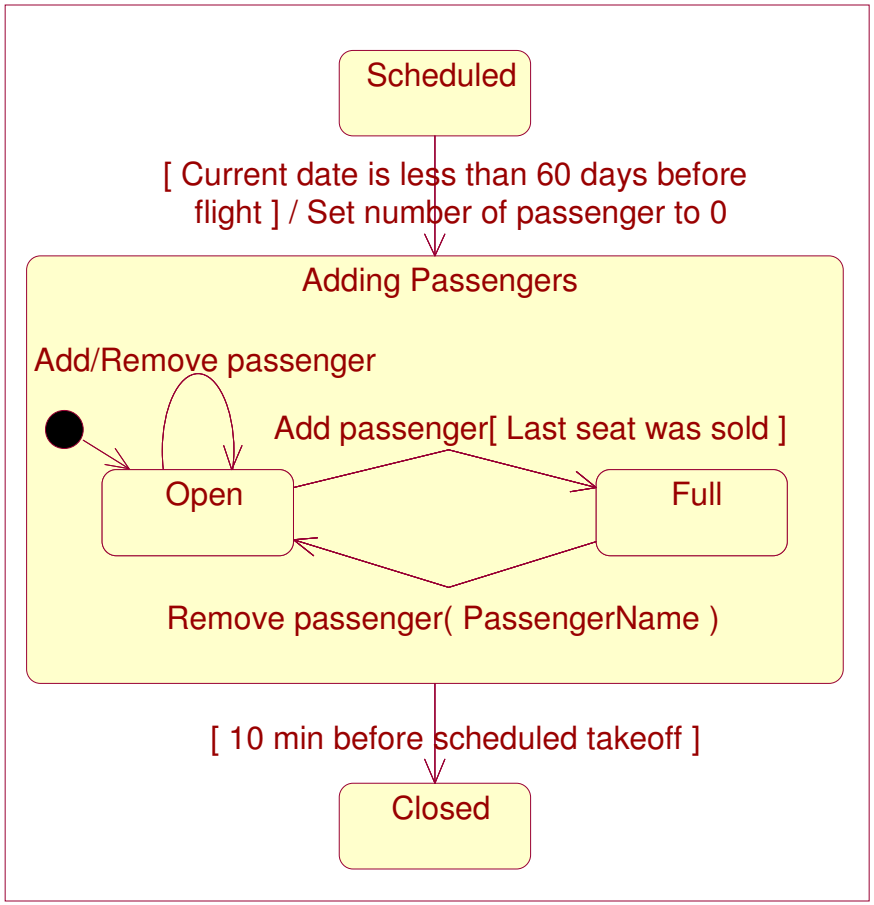


Biểu đồ trạng thái

- Các phần tử đồ họa
  - ...
  - Trạng thái ẩn (Nested state)
    - Để giảm quá nhiều trạng thái trong biểu đồ ta có thể lồng trạng thái vào trong trạng thái khác: Substate, Superstate
    - Nếu hai hay nhiều trạng thái có cùng quá độ -> nhóm chúng thành superstate
    - Thí dụ
      - Biểu đồ biểu đồ biến đổi trạng thái của lớp Flight có các trạng thái Scheduled, Open, Full và Closed
      - Chuyến bay chuyển vào trạng thái Closed 10 phút trước khi cất cánh, không quan tâm đến trạng thái trước đó của nó là Open hay Full.
  - Lịch sử trạng thái (Superstate history)
    - Nhiều khi có nhu cầu nhớ lại trạng thái vừa trước đó của đối tượng
      - Bổ sung trạng thái khởi đầu trong siêu trạng thái
      - Sử dụng chỉ báo lịch sử lịch sử trạng thái (H) nơi đối tượng vừa đi qua

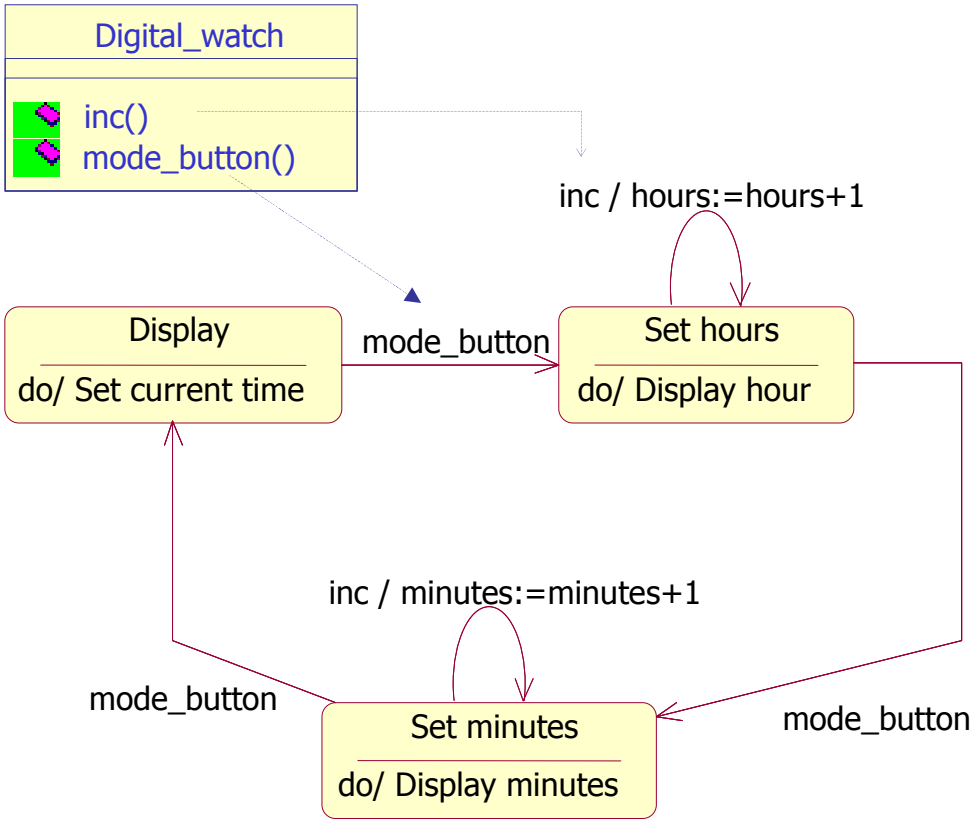
Biểu đồ trạng thái với trạng thái ẩn





Biểu đồ trạng thái và lớp

- Mô tả quan hệ giữa biểu đồ trạng thái và lớp
  - Thí dụ Biểu đồ trạng thái của lớp Digital watch



Biểu đồ hoạt động

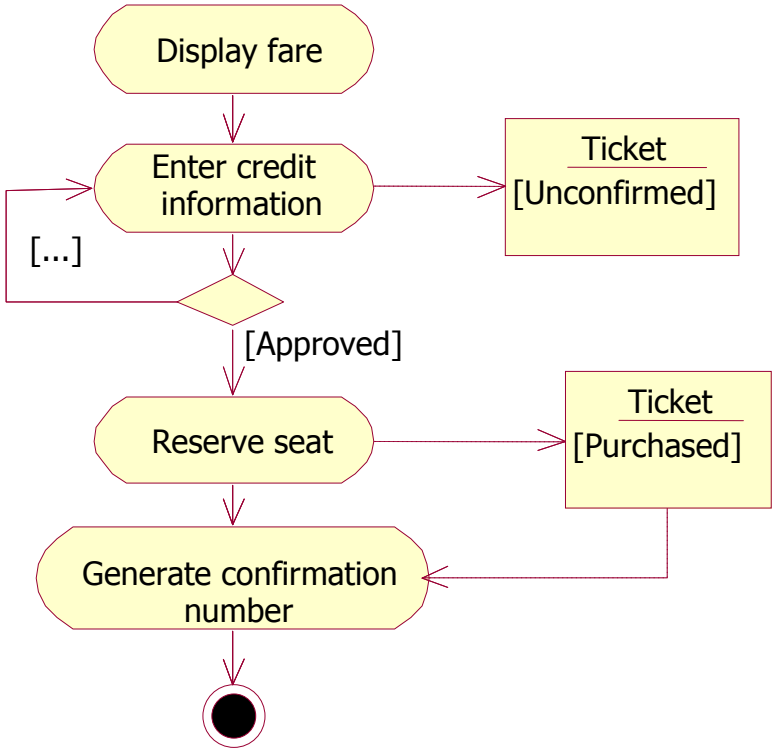
- Biểu đồ hoạt động (Activity diagram) do Odell đề xuất cho UML để
  - mô tả luồng công việc trong tiến trình nghiệp vụ trong mô hình hóa nghiệp vụ
  - mô tả luồng sự kiện trong mô hình hóa hệ thống
    - Sử dụng text như trước đây sẽ khó đọc khi logíc phức tạp, có nhiều rẽ nhánh
- Biểu đồ hoạt động sử dụng để mô hình hóa
  - khía cạnh động của hệ thống
  - các bước trình tự hay tương tranh trong quá trình tính toán

Biểu đồ hoạt động

- Hoạt động (Activity)
  - Là một bước trong tiến trình
- Hành động (Actions)
  - Là các bước nhỏ hơn trong Activity
  - Action có thể xảy ra khi
    - Đang vào activity
      - Hành động vào xảy ra khi activity bắt đầu, đánh dấu bằng "Entry"
    - Khi đang đi ra khỏi activity
      - Hành động ra xảy ra khi rời bỏ activity, đánh dấu bằng "Exit"
    - Khi thực hiện activity
      - Hành động xảy ra khi đang trong activity, đánh dấu bằng "do"
    - Khi có sự kiện đặc biệt xảy ra
      - Hành động xảy ra khi và chỉ khi có sự kiện cụ thể xảy ra, đánh dấu bằng "event" tiếp theo là tên sự kiện

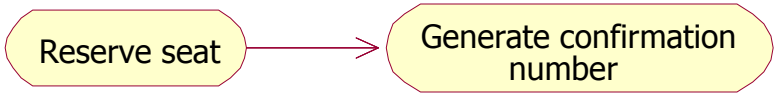
Biểu đồ hoạt động

- Đối tượng và luồng đối tượng (Object flow)
  - Đối tượng là thực thể, có thể bị các hoạt động trong luồng sử dụng và thay đổi
  - Trong biểu đồ hoạt động:
    - hiển thị đối tượng với trạng thái của nó
  - Liên kết đối tượng với các hoạt động thông qua luồng đối tượng
  - Một đối tượng có thể là đầu vào cho hoạt động.

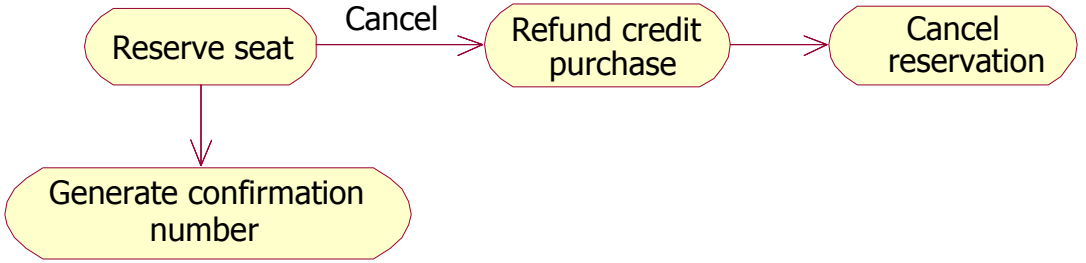


Biểu đồ hoạt động

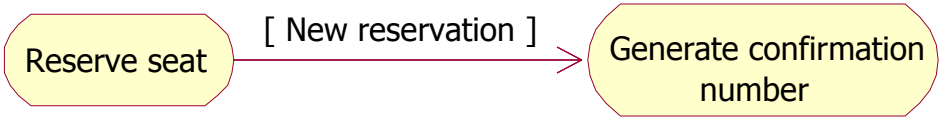
- Quá độ (Transition)
  - Chỉ ra luồng điều khiển từ hoạt động này đến hoạt động khác
    - Trường hợp đơn giản



    - Đặt giới hạn trên quá độ để điều khiển: event hay guard condition

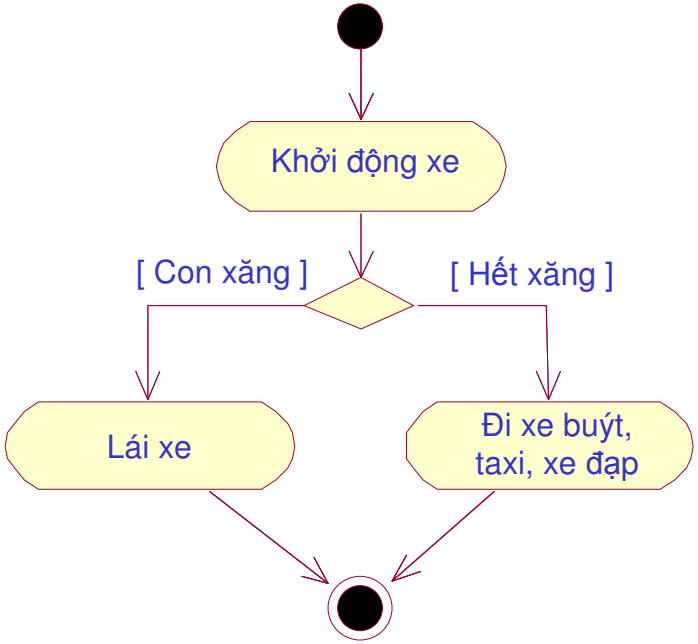


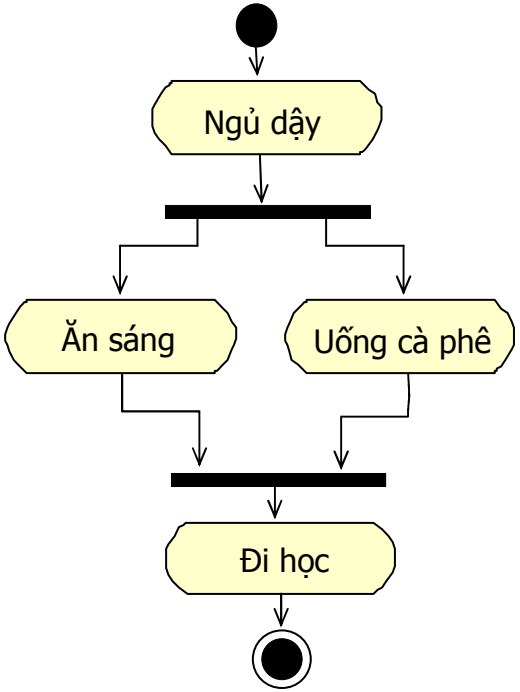
    - Khi có sự kiện, điều kiện canh điều khiển để quá độ có thể xảy ra?

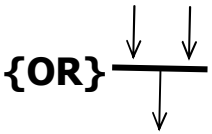


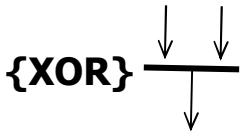
Biểu đồ hoạt động

- Rẽ nhánh và đồng bộ (Synchonization)
  - Đồng bộ là cách mô tả hai hay nhiều nhánh flows xảy ra đồng thời





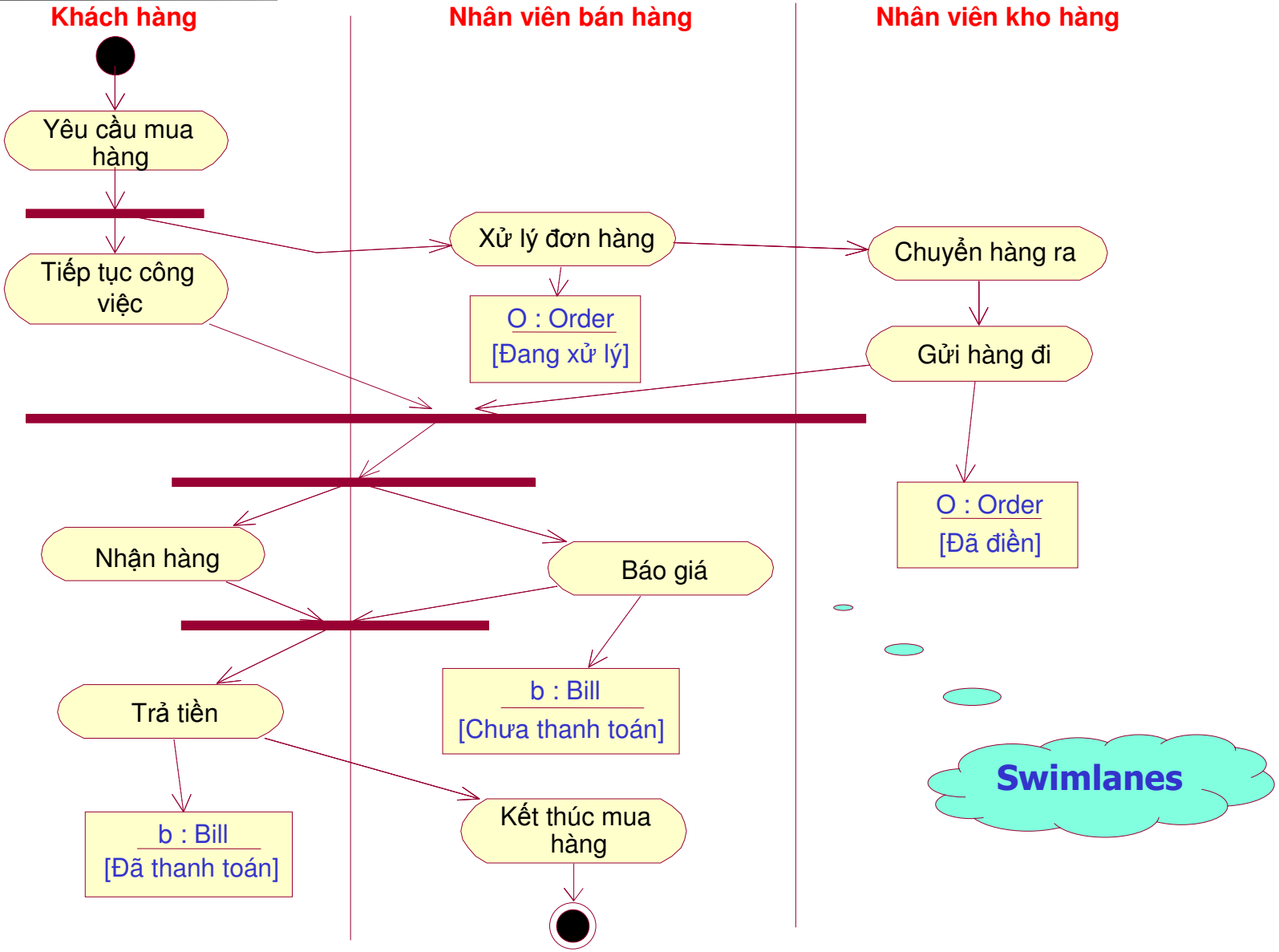




Biểu đồ hoạt động

- Làn bơi (Swimlanes)
  - Sử dụng để mô hình hóa luồng công việc trong tiến trình nghiệp vụ
  - Chỉ ra ai có trách nhiệm thực hiện từng hoạt động
  - Để phân hoạch các trạng thái hoạt động vào nhóm
  - Phân tách nhóm trên biểu đồ bằng các làn bơi
  - Mỗi hoạt động thuộc về một làn bơi
  - Quá độ có thể được vẽ từ làn bơi này đến làn bơi khác
  - Mỗi làn bơi có thể được cài đặt bởi một hay nhiều lớp

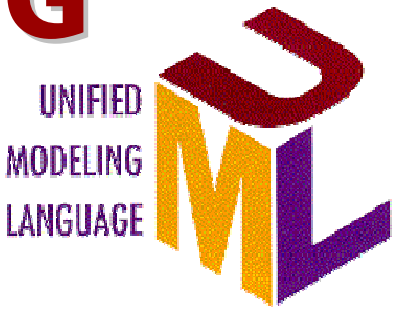
Biểu đồ hoạt động



Tóm tắt

- Bài này đã xem xét các vấn đề sau
  - Biểu đồ chuyển trạng thái
    - Trạng thái của đối tượng
    - Các phần tử đồ họa xây dựng biểu đồ
    - Kỹ thuật xây dựng biểu đồ trạng thái
  - Biểu đồ hoạt động
    - Ứng dụng của biểu đồ hoạt động
    - Các phần tử đồ họa xây dựng biểu đồ hoạt động

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động

✓ **Biểu đồ kiến trúc vật lý và phát sinh mã trình**

9. Mô hình hóa dữ liệu
10. Bài học thực nghiệm

Bài 8

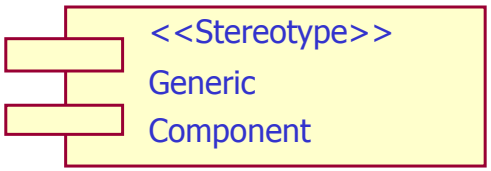
Biểu đồ kiến trúc vật lý và phát sinh mã trình

Kiến trúc phần mềm?

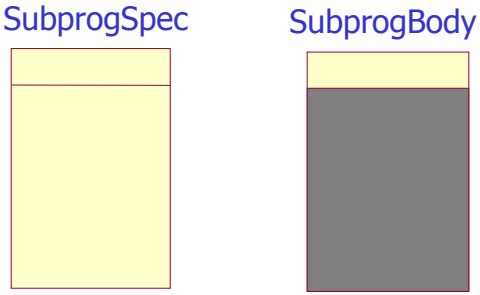
- Kiến trúc hệ thống là kế hoạch chi tiết của các bộ phận hình thành hệ thống
- UML định nghĩa:
  - Kiến trúc là cấu trúc tổ chức của hệ thống
  - Kiến trúc bao gồm các bộ phận tương tác thông qua giao diện
- Theo Buschman:
  - Kiến trúc phần mềm là mô tả các phân hệ, các thành phần của hệ thống phần mềm và các quan hệ giữa chúng
- Hai loại kiến trúc hệ thống
  - Kiến trúc logíc
    - Chỉ ra các lớp đối tượng và các quan hệ giữa chúng để hình thành chức năng hệ thống
    - Nó được thể hiện bằng các biểu đồ UC, biểu đồ lớp, trạng thái, hoạt động...
  - Kiến trúc vật lý
    - Là mô tả từ khía cạnh phần cứng và các mođun phần mềm trên đó
    - Nó được mô tả bằng các biểu đồ cài đặt: biểu đồ thành phần và biểu đồ triển khai

Các thành phần

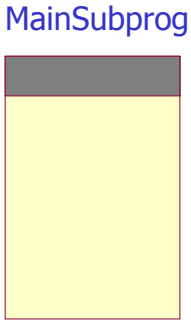
- Thành phần?
  - Là mô đun vật lý mã trình: thư viện mã nguồn, mã khả thực.
- Các loại thành phần



Đặc tả thành phần
bằng Stereotype



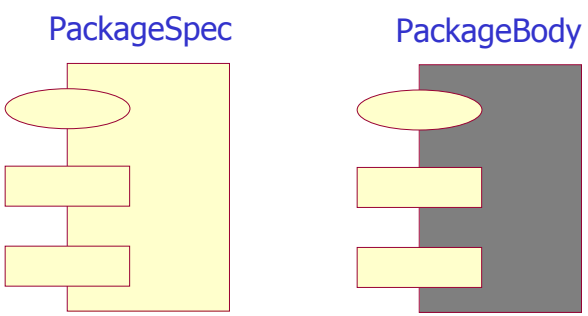
Đặc tả và thân chương trình con
Tập hợp các hàm
Không chứa định nghĩa lớp



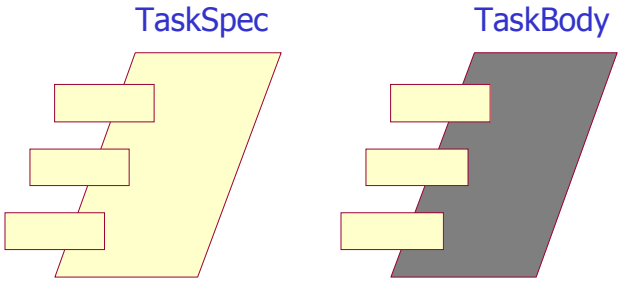
Chương trình chính
Chứa đầu vào chương trình

Các thành phần

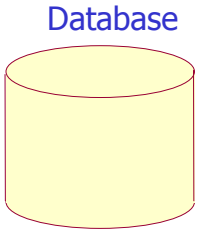
- Các loại thành phần



Đặc tả và thân gói
Gói là cài đặt lớp
Đặc tả gói là tệp header



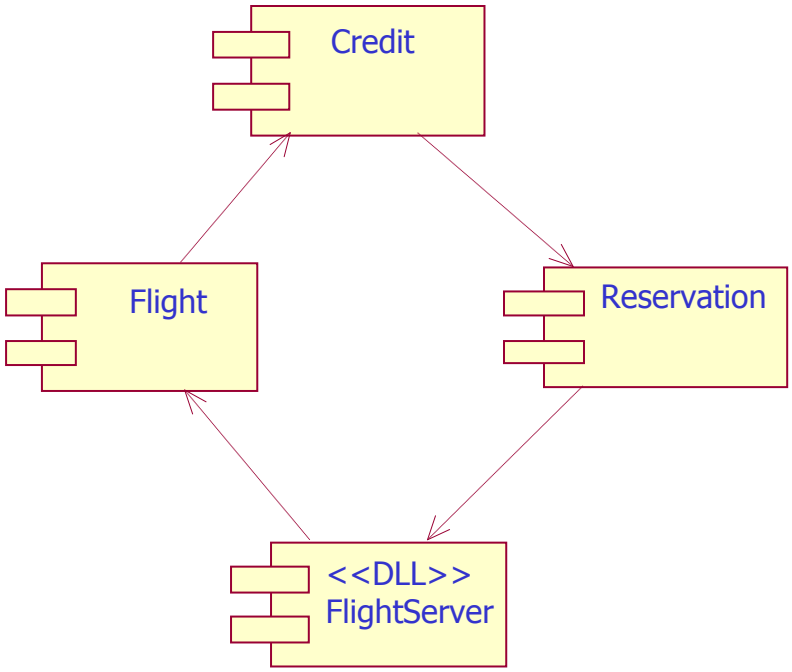
Đặc tả và thân nhiệm vụ
Là các thành phần Run-time
Biểu diễn các gói có thread độc lập



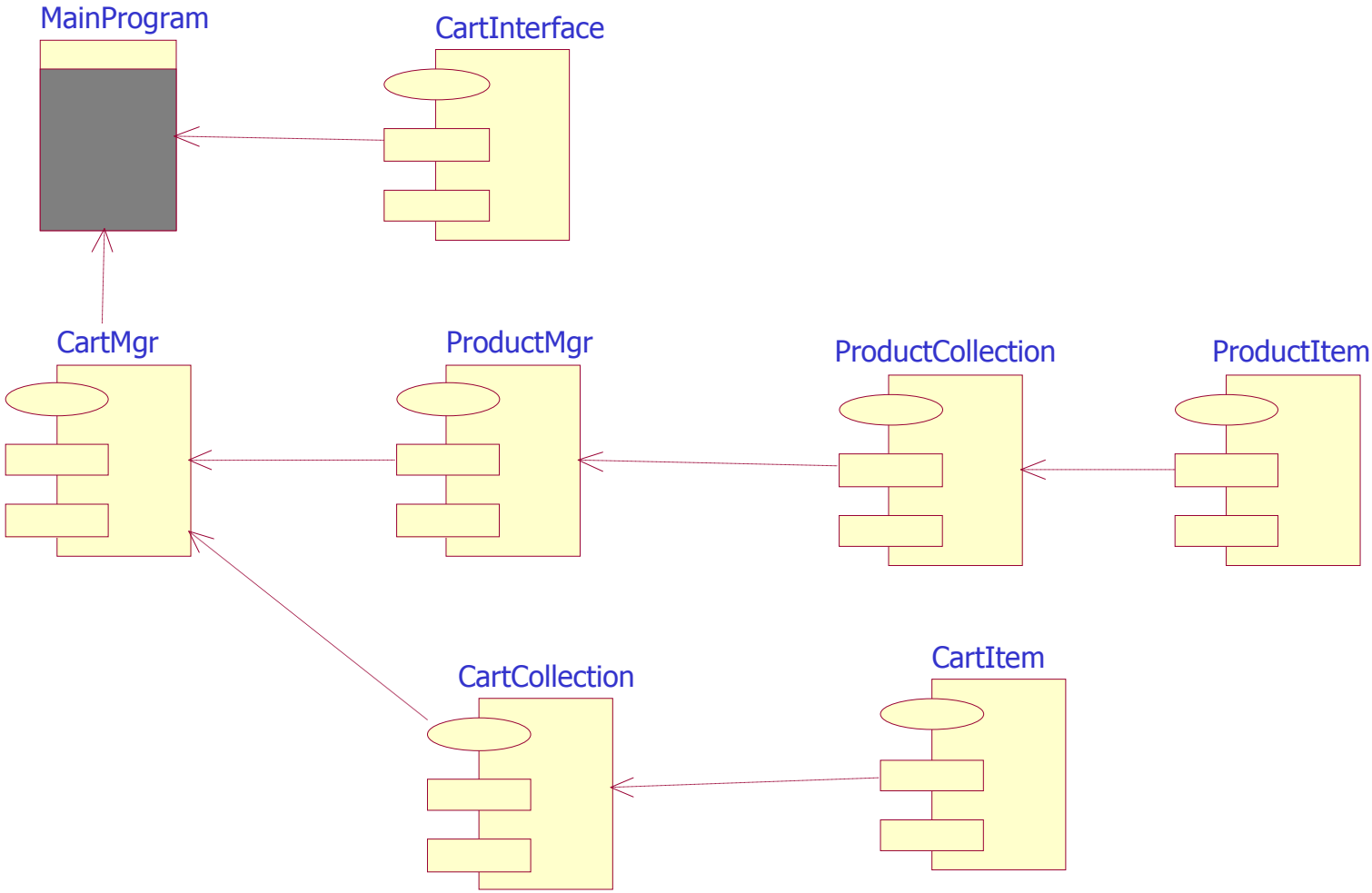
Biểu diễn CSDL
Chứa một hay nhiều lược đồ

Biểu đồ thành phần

- Biểu đồ thành phần là biểu đồ hiển thị các thành phần trong hệ thống và phụ thuộc giữa chúng
  - Thành phần A phụ thuộc vào thành phần B khi vài lớp trong A phụ thuộc vào vài lớp trong B
- Biểu đồ cho biết
  - Thư viện nào được sử dụng, tệp khả thực (.exe) nào được tạo ra khi dịch chương trình
  - Các quan hệ giữa các thư viện mã trình
- Có khả năng tổ chức các thành phần vào các gói



Thí dụ Biểu đồ thành phần



Bổ sung chi tiết cho thành phần

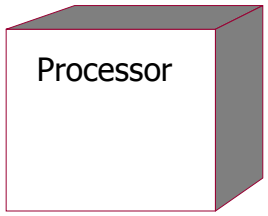
- Stereotype
  - Lựa chọn biểu tượng để biểu diễn thành phần
  - Có thể là:
    - <none>, ActiveX, Applet, Subroutine Spec, dll... tự định nghĩa
- Language
  - Trong Rose có thể gán ngôn ngữ cho thành phần
  - Cho khả năng phat sinh các ngôn ngữ khác nhau cho mỗi thành phần
- Declaration
  - Gán các khai báo vào mã trình của từng thành phần
- Class
  - Gán lớp vào thành phần trước khi phát sinh mã trình
  - Có thể ánh xạ một hay nhiều lớp vào một thành phần
- Dependency
  - Thành phần chỉ có một loại quan hệ: quan hệ phụ thuộc
  - Tránh hình thành quan hệ vòng

Biểu đồ triển khai

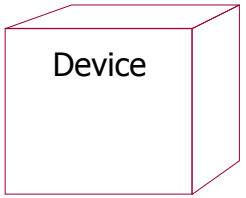
- Biểu đồ triển khai mô tả kiến trúc phần cứng (các nút) có phần mềm chạy trên chúng, bao gồm các bộ xử lý, các tiến trình, các thiết bị và các kết nối giữa chúng
  - Mô tả tôpô của hệ thống
  - Chỉ ra toàn bộ các nút trên mạng, kết nối giữa chúng và các phần mềm chạy trên chúng
- Nút là đối tượng vật lý có tài nguyên tính toán
  - Máy tính, máy in, thiết bị đọc thẻ từ và truyền tin
- Giữa các nút là kết nối giao tiếp, kiểu kết nối được thể hiện bằng stereotype

Các phần tử của biểu đồ triển khai

- Bộ xử lý
  - Là máy xử lý: máy chủ, máy trạm
  - Bổ sung thuộc tính:
    - Stereotype
    - Mô tả vật lý của bộ xử lý: tốc độ, dung lượng nhớ
    - Lập lịch xử lý: Preemptive, Non-preemptive, Cyclic, Executive, Manual

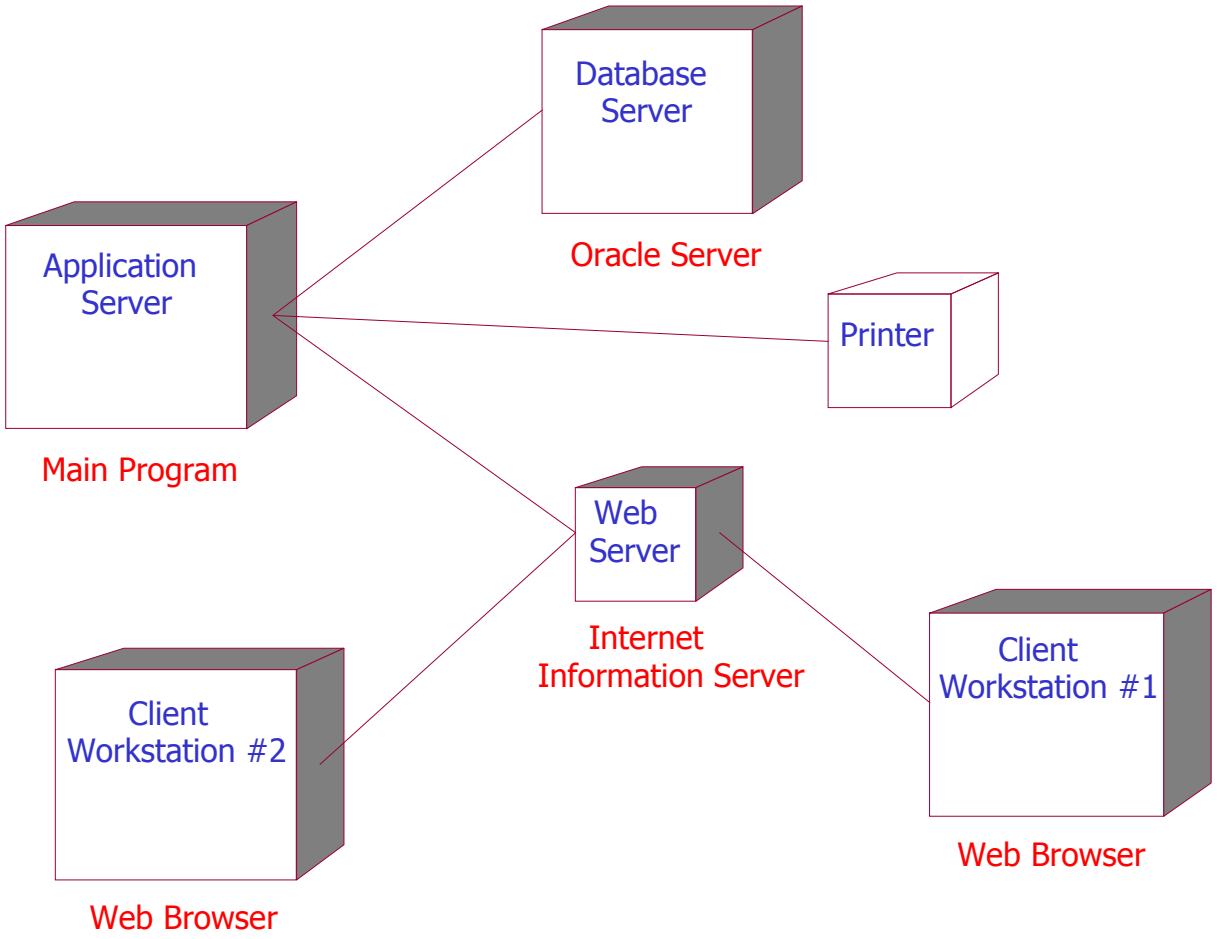


- Thiết bị
  - Là phần cứng chỉ có một mục đích: máy in, scanner...
  - Bổ sung thuộc tính:
    - Stereotype
    - Mô tả vật lý của thiết bị



- Kết nối
  - Là liên kết vật lý giữa các thiết bị và bộ xử lý
  - Bổ sung stereotype và đặc tính vật lý cho kết nối: T1
- Bổ sung tiến trình cho bộ xử lý

Thí dụ biểu đồ triển khai



Phát sinh mã trình

- Sáu bước cơ bản để phát sinh mã trình
  - Kiểm tra mô hình
  - Tạo lập thành phần
  - Ánh xạ lớp vào thành phần
  - Gán thuộc tính phát sinh mã trình
  - Chọn lớp, thành phần hay gói để phát sinh mã
  - Phát sinh mã trình

Phát sinh mã trình

- Bước 1: Kiểm tra mô hình
  - Rose có chức năng kiểm tra mô hình độc lập ngôn ngữ để đảm bảo tính nhất quán trong mô hình
  - Khi kiểm tra có thể phát hiện các lỗi sau
    - Ánh xạ không đầy đủ: Các đối tượng hay thông điệp trong biểu đồ trình tự chưa ánh xạ vào thao tác hay lớp trong biểu đồ lớp
    - Vi phạm xâm nhập: Thí dụ, hai lớp trong hai gói có quan hệ nhưng vẽ thiếu quan hệ giữa hai gói
    - Kiểm tra phụ thuộc ngôn ngữ: Sẽ phát hiện, thí dụ, nhiều lớp cùng tên khai báo public trong một mođun chương trình
- Bước 2: Tạo lập thành phần
  - Tạo lập thành phần để chứa lớp
  - Trước khi phát sinh mã trình phải ánh xạ các lớp vào thành phần tương ứng
  - Bổ sung quan hệ thành phần trên Biểu đồ thành phần

Phát sinh mã trình

- Bước 3: Ánh xạ lớp vào thành phần
  - Mỗi thành phần mã nguồn biểu diễn tệp mã nguồn cho một hoặc vài lớp
    - Thí dụ C++: Mỗi lớp ánh xạ đến hai thành phần – Các tệp Header và Body
  - Bước này yêu cầu ánh xạ lớp vào thành phần tương ứng
- Bước 4: Đặt đặc tính cho phát sinh mã trình
  - Nhiều đặc tính có thể gán cho lớp, thuộc tính, thành phần của mô hình để điều khiển mã được phát sinh như thế nào.
    - Thí dụ C++: Đặc tính GenerateGetOperation điều khiển việc có phát sinh hàm Get() hay không.
    - Thí dụ khác: GenerateDefaultConstructor
  - Đặt tập đặc tính tạm thời
    - Thay vì thay đổi trực tiếp tập đặc tính ta có thể tạo ra tập đặc tính tạm thời để sử dụng, không ảnh hưởng đến tập đặc tính mặc định
  - Hủy bỏ tập đặc tính tạm thời

Phát sinh mã trình

- Bước 5: Chọn lớp, thành phần hay gói
  - Có thể chọn lớp, thành phần hay gói để phát sinh mã trình vào các thời điểm khác nhau
  - Phát sinh mã từ biểu đồ hay Browser
  - Có thể phát sinh mã trình cho một vài lớp, thành phần hay gói đồng thời
- Bước 6: Phát sinh mã trình
  - Lựa chọn ngôn ngữ theo yêu cầu để phát sinh mã từ mô hình

Phát sinh mã trình

- Cái gì đã được phát sinh từ mô hình?
  - Thực tế
    - Không có công cụ mô hình hóa nào phát sinh mã trình đầy đủ
    - Rose cũng chỉ phát sinh khung chương trình
  - Các phần tử được phát sinh
    - Lớp: Mọi lớp trong mô hình được sinh mã
    - Thuộc tính: Mã trình sẽ chứa các thuộc tính lớp bao gồm phạm vi, kiểu dữ liệu và giá trị mặc định, các hàm Get(), Set().
    - Signature: Các thao tác được khai báo trong mã trình cùng với danh sách tham số, kiểu dữ liệu của tham số và kiểu giá trị cho lại của thao tác
    - Quan hệ: Một số quan hệ trong mô hình được chuyển sang thuộc tính
    - Thành phần: Mỗi thành phần được hiện thực trong tệp tương ứng
    - Tài liệu: Tài liệu trong mô hình được chèn vào nơi thích ứng trong mã trình

Phát sinh mã trình

- Nhiệm vụ của người phát triển sau khi Rose sinh mã trình
  - Thu thập các tệp mã trình, viết mã trình cho các thao tác lớp
  - Thiết kế giao diện đồ họa
- Thí dụ đoạn mã trình do Rose phát sinh

```
#include "stdafx.h"
#include "Order.h"
//##ModelId=3A77E3CD0280
Boolean Order::Create()
{
  // TODO: Add your specialized code here.
  // NOTE: Requires a correct return value to compile.
}
//##ModelId=3A77E3E60316
Boolean Order::SetInfo(Integer OrderNum, String Customer, Date OrderDate, Date FillDate)
{
  // TODO: Add your specialized code here.
  // NOTE: Requires a correct return value to compile.
}
//##ModelId=3A77E40E0230
String Order::GetInfo()
{
  // TODO: Add your specialized code here.
  // NOTE: Requires a correct return value to compile.
}
```

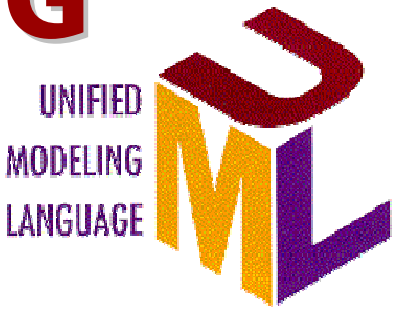
Phát sinh mã trình

```
class Order
{
public:
  //##ModelId=3A7F695F019A
  OrderItem* theOrderItem;
  //##ModelId=3A77E3CD0280
  Boolean Create();
  //##ModelId=3A77E3E60316
Boolean SetInfo(Integer OrderNum, String Customer, Date OrderDate, Date
FillDate);
  //##ModelId=3A77E40E0230
  String GetInfo();
private:
  //##ModelId=3A7E13F9038E
  Integer OrderNumber;
  //##ModelId=3A7E14260122
  String CustomerName;
  //##ModelId=3A7E14470208
  Date OrderDate;
  //##ModelId=3A7E145303D4
  Date OrderFillDate;
};
```

Tóm tắt

- Bài này đã xem xét các vấn đề sau
  - Kiến trúc vật lý của hệ thống
  - Xây dựng biểu đồ thành phần
    - Các thành phần phần mềm và quan hệ giữa chúng
    - Các phần tử đồ họa vẽ biểu đồ thành phần
  - Xây dựng biểu đồ triển khai
    - Các phần tử đồ họa vẽ biểu đồ triển khai
  - Các bước chuyển đổi mô hình thành phần mềm

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. ✓ **Mô hình hóa dữ liệu**
10. Bài học thực nghiệm

Bài 9

Mô hình hóa dữ liệu

Mô hình đối tượng - mô hình dữ liệu

- Rose 2001 và các phiên bản sau đó: Hỗ trợ mô hình hóa dữ liệu
- Mô hình đối tượng
  - Tập trung vào dữ liệu và hành vi
  - Sử dụng cho mọi thành phần của ứng dụng: lớp, thuộc tính, thao tác, quan hệ... như đã được xem xét trong các bài trước đây
  - Quan tâm trước hết của mô hình dữ liệu là mô hình trong bộ nhớ:
    - Việc tạo lập đối tượng, quan hệ giữa chúng và trách nhiệm của chúng
- Mô hình dữ liệu
  - Tập trung vào dữ liệu
  - Tập trung vào CSDL hơn là tập trung vào ứng dụng

Mô hình đối tượng - mô hình dữ liệu

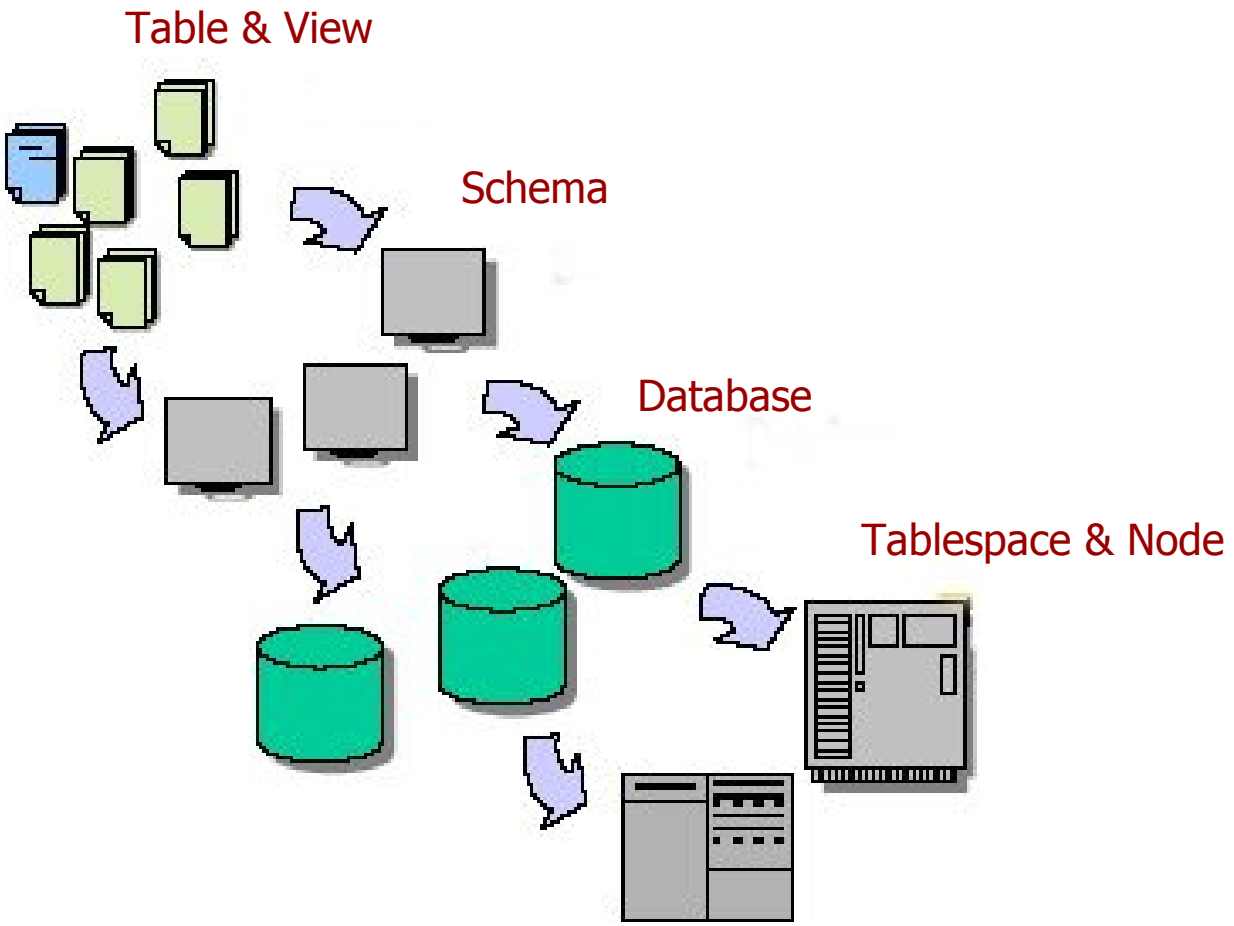
- Các quan tâm khác nhau giữa mô hình đối tượng và mô hình dữ liệu

Mô hình đối tượng	Mô hình dữ liệu
Thiết kế lớp như thế nào để sử dụng hiệu quả bộ nhớ?	Thiết kế CSDL như thế nào để lưu trữ hiệu quả
Các đối tượng nào cần quan hệ trong mô hình?	Bảng nào cần quan hệ trong mô hình dữ liệu?
Cấu trúc dữ liệu tại giao diện như thế nào để thỏa mãn người sử dụng cuối cùng?	Cấu trúc dữ liệu như thế nào để tăng tốc độ xâm nhập?
Gói dữ liệu với hành vi như thế nào để tạo ra lớp?	Chuẩn hóa dữ liệu?
Dữ liệu nào được sử dụng xuyên suốt ứng dụng? Loại dữ liệu nào chỉ được sử dụng trong một vùng?	Dữ liệu nào được truy vấn thường xuyên?
Có thể sử dụng khái quát hóa hay các chiến lược thiết kế khác để có mã trình sử dụng lại?	Có thể tích hợp khái niệm kế thùa vào mô hình dữ liệu ngay cả khi CSDL không hỗ trợ trực tiếp kế thừa?

Mô hình đối tượng - mô hình dữ liệu

- Sự khác biệt giữa hai loại mô hình
  - Hỗ trợ kế thừa
  - Quan hệ:
    - Giữa các lớp (lớp này biết về lớp kia)
    - Giữa các bảng (kết nối logíc)
- Có thể xây dựng tách biệt mô hình dữ liệu và mô hình đối tượng, nhưng cũng có thể xây dựng chúng đồng thời
- Trong Rose:
  - Nếu dự án đã có mô hình dữ liệu -> chuyển ngược lại để hình thành mô hình đối tượng
  - Với dự án mới, ta có thể phát sinh mô hình dữ liệu từ mô hình đối tượng

Tạo lập mô hình dữ liệu



Tạo lập mô hình dữ liệu

- Trong Logical View
  - Lược đồ chứa các thủ tục lưu trữ
  - Bảng với các trường, ràng buộc, triggers, khóa chính, chỉ số và quan hệ
- Trong Component View
  - Mô hình hóa CSDL
    - Mỗi thành phần được gán Stereotype
    - Rose 2001A trở đi hỗ trợ DB2, Oracle, Sybase, SQL Server, ANSI Server

Tạo lập mô hình dữ liệu

- Các bước chính tạo lập mô hình dữ liệu (nhưng không nhất thiết phải theo trình tự này)
  - Tạo lập CSDL
  - Bổ sung lược đồ để chứa mô hình dữ liệu và gán lược đồ vào CSDL
  - Tạo lập gói lĩnh vực và các lĩnh vực
  - Bổ sung các bảng vào từng lược đồ
  - Bổ sung chi tiết vào từng bảng
    - Trường, ràng buộc, trigger, chỉ số và khóa chính
  - Bổ sung quan hệ giữa các bảng và khóa ngoài
  - Tạo lập các khung nhìn
  - Tạo lập mô hình đối tượng từ mô hình dữ liệu
  - Phát sinh CSDL
  - Đồng bộ CSDL với mô hình khi cập nhật

Logíc trong mô hình dữ liệu

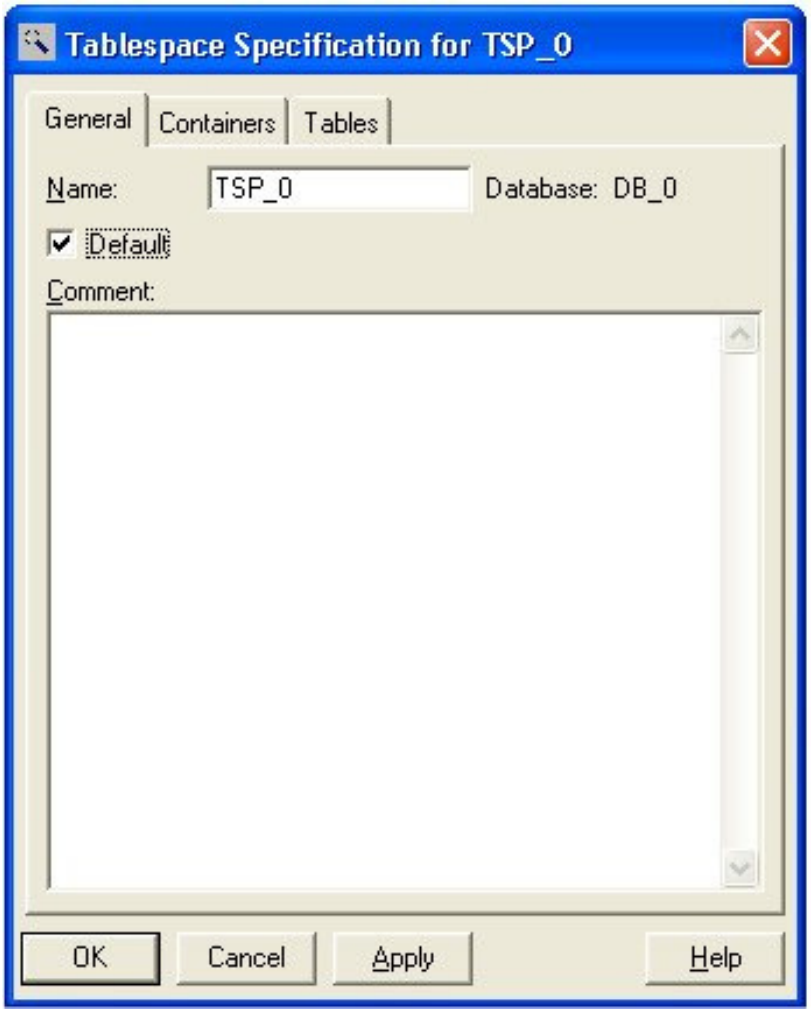
- Rất khó xác định logíc nào để ở đâu: tầng CSDL hay tầng ứng dụng?
  - Một số logíc nghiệp vụ nên để tại tầng ứng dụng thay cho tầng CSDL
  - Tổng thể thì chỉ logíc nào liên quan đến dữ liệu mới để trên tầng CSDL.
    - Thí dụ: Các field, giá trị hợp lệ của field và độ dài của field
  - Có thể gắn các qui tắc nghiệp vụ vào CSDL thông qua sử dụng ràng buộc.
    - Nếu để logíc nghiệp vụ trong CSDL
      - Ứng dụng phải thu thập dữ liệu từ người sử dụng cuối cùng. Chuyển nó đến tầng nghiệp vụ: truyền qua kết nối mạng (có thể chậm). Cuối cùng là đánh giá tính đúng đắn.
    - Do vậy, nên để logíc nghiệp vụ tại tầng nghiệp vụ để làm giảm lưu lượng truyền trên mạng.
  - Một vài logíc hệ thống có thể thực hiện bên trong CSDL thông qua sử dụng các thủ tục lưu trữ
    - Lợi thế:
      - Thực hiện nhanh khi các chức năng phải xử lý khối dữ liệu lớn
    - Bất lợi:
      - Nếu sử dụng các thủ tục lưu trữ cài đặt logíc nghiệp vụ thì khi nó thay đổi đòi hỏi thay đổi cả tầng nghiệp vụ và tầng CSDL.
      - Một bất lợi khác là các thủ tục trong DBMS khác nhau có cú pháp khác nhau, do vậy khi chuyển đổi DBMS phải viết lại các thủ tục lưu trữ.

Bổ sung CSDL

- CSDL được mô hình hóa trong Rose như thành phần với stereotype
  - CSDL là hệ thống lưu trữ dữ liệu vật lý và điều khiển xâm nhập dữ liệu
    - Có tên duy nhất
    - Được gán cho DBMS cụ thể (ANSI SQL, SQL Server, DB2, Oracle...)
- Bổ sung không gian bảng (Tablespaces)
  - Node là thực thể vật lý (máy tính) nơi lưu trữ CSDL
  - Không gian bảng là đơn vị lưu trữ lôgíc của bảng (SQL Server, DB2, Oracle)
    - Là kết nối cấu trúc vật lý (CSDL) và nút.
  - Mỗi không gian bảng có một hay nhiều container
    - Container là thiết bị lưu trữ vật lý (ổ đĩa)
    - Mỗi container được chia thành đơn vị nhỏ hơn – extents
  - SQL Server: Không gian bảng là filegroups, containers là files
  - Mỗi không gian bảng có kích thước khởi đầu (KB)
    - DBMS có khả năng tăng tự động kích thước không gian bảng
    - Không có khả năng tăng không gian bảng quá mức giới hạn bằng Rose
  - Gán các bảng cho không gian bảng

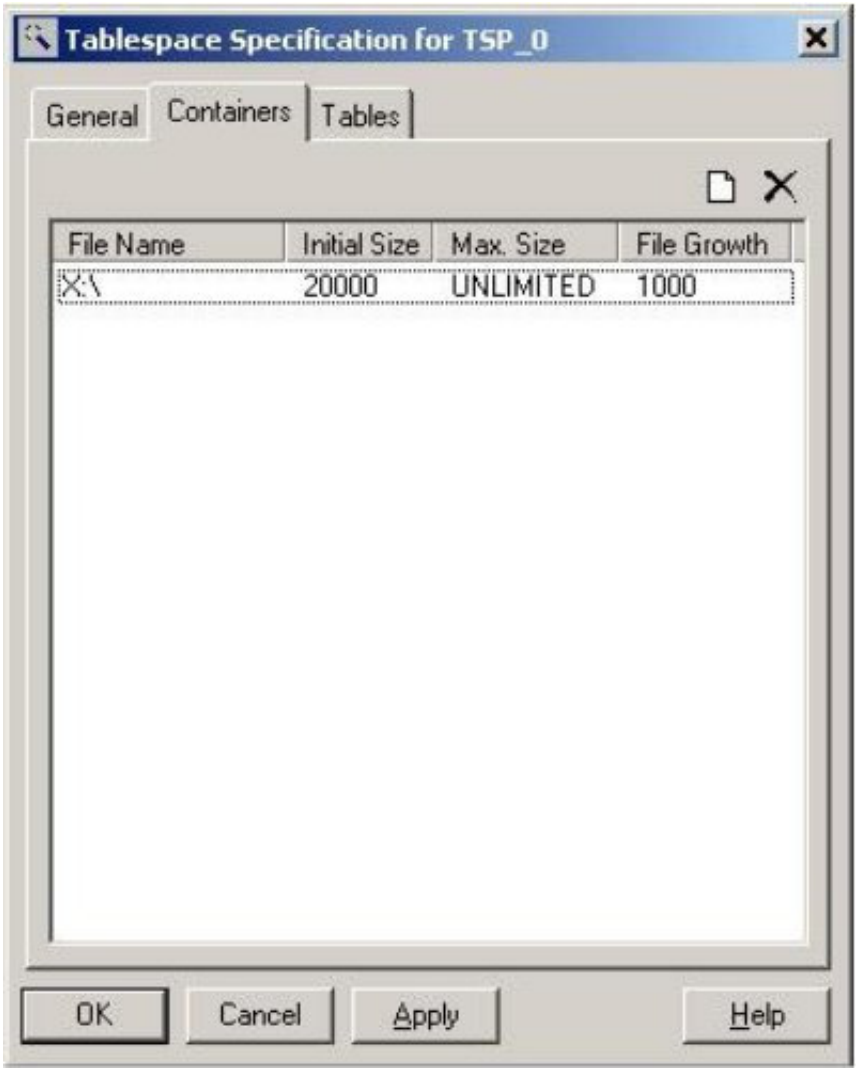
Bổ sung Không gian bảng

- Thí dụ với SQL server
  - Nhấn phím phải chuột trên CSDL trong Browser
  - Chọn Data Modeler->New-> Tablespace
  - Đặt tên cho Tablespace
  - Nhấn phím phải chuột trên Tablespace mới để chọn Open Specification
  - Đánh dấu Default nếu muốn đây là Tablespace mặc định
    - Mọi bảng chưa gán vào Tablespace nào thì được gán vào Tablespace mặc định



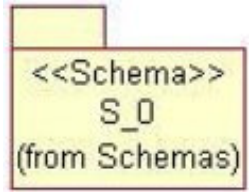
Bổ sung Không gian bảng

- Thí dụ với SQL server
  - Đặt Container trong Tablespace
    - Nhấn phím phải trên Tablespace trong Browser, chọn Open Specification
    - Chọn Container Tab
    - Nhấn phím phải trong vùng trắng, chọn New
    - Nhập tên tệp tablespace, kích thước khởi đầu, kích thước cực đại và kích thước tăng (file Growth)



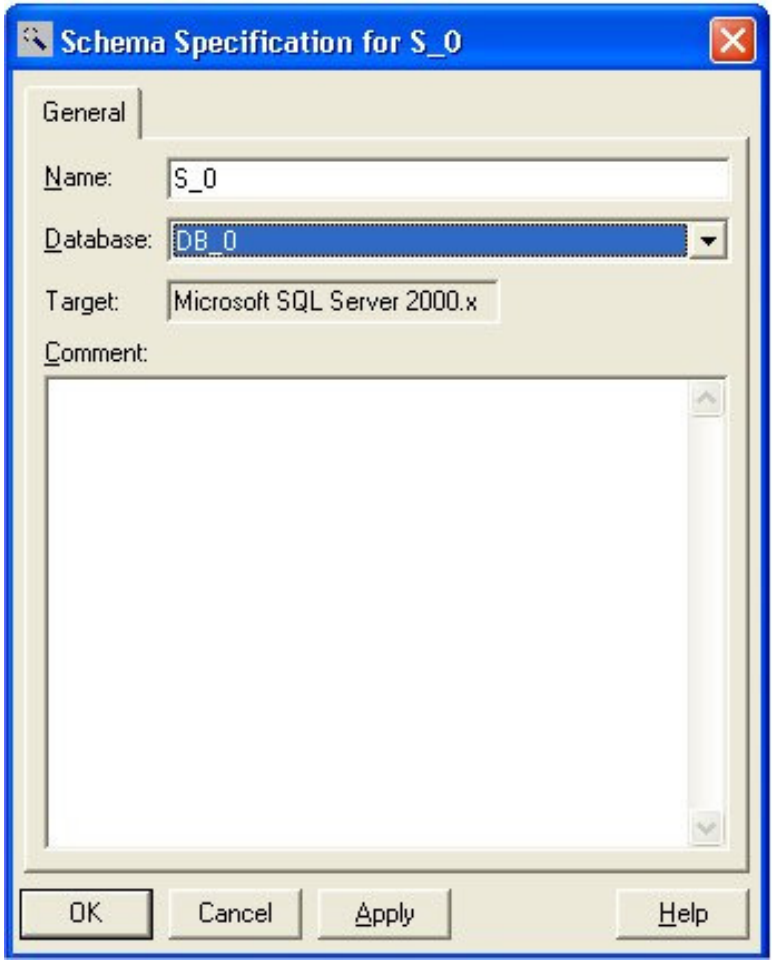
Bổ sung lược đồ

- Lược đồ (schema) là đơn vị cơ sở của tổ chức các bảng
  - Là container trong mô hình dữ liệu
  - Nó còn là cơ chế an toàn
- Lược đồ bao gồm
  - Bảng, trường, trigger, ràng buộc và các phần tử mô hình dữ liệu khác
- Trong khung nhìn logíc có gói Schemas
  - Mọi lược đồ ta tạo ra đều chứa trong gói này
- Mỗi lược đồ được ánh xạ vào CSDL
- Mỗi CSDL có thể chứa một hay nhiều lược đồ



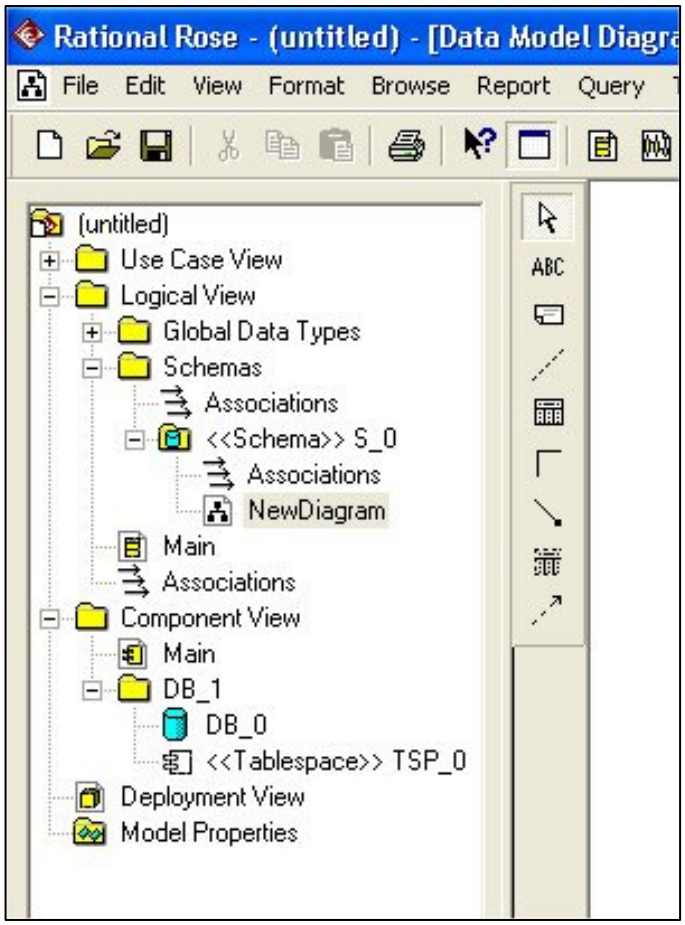
Bổ sung lược đồ

- Tạo lập lược đồ trong Rose
  - Nhấn phím phải chuột trên Schema trong Logical View
  - Chọn Modeler-> New-> Schema
  - Nhấn phím phải trên lược đồ mới để chọn Open Specification
  - Chọn bảng phù hợp trong hộp thoại Database



Tạo lập biểu đồ mô hình dữ liệu

- Biểu đồ mô hình dữ liệu được tạo lập trong lược đồ
- Biểu đồ mô hình dữ liệu được sử dụng để add, edit và quan sát các bảng và các phần tử khác trong CSDL
  - Tương tự biểu đồ lớp trong mô hình dữ liệu
- Có thể tạo ra rất nhiều mô hình dữ liệu trong mỗi lược đồ
- Tạo lập biểu đồ dữ liệu trong Rose
  - Nhấn phím phải trên lược đồ trong Browser
  - Chọn Data Modeler-> New -> Data Model Diagram
  - Nhập tên cho biểu đồ mới: NewDiagram
  - Nhấn đúp trên biểu đồ để mở nó

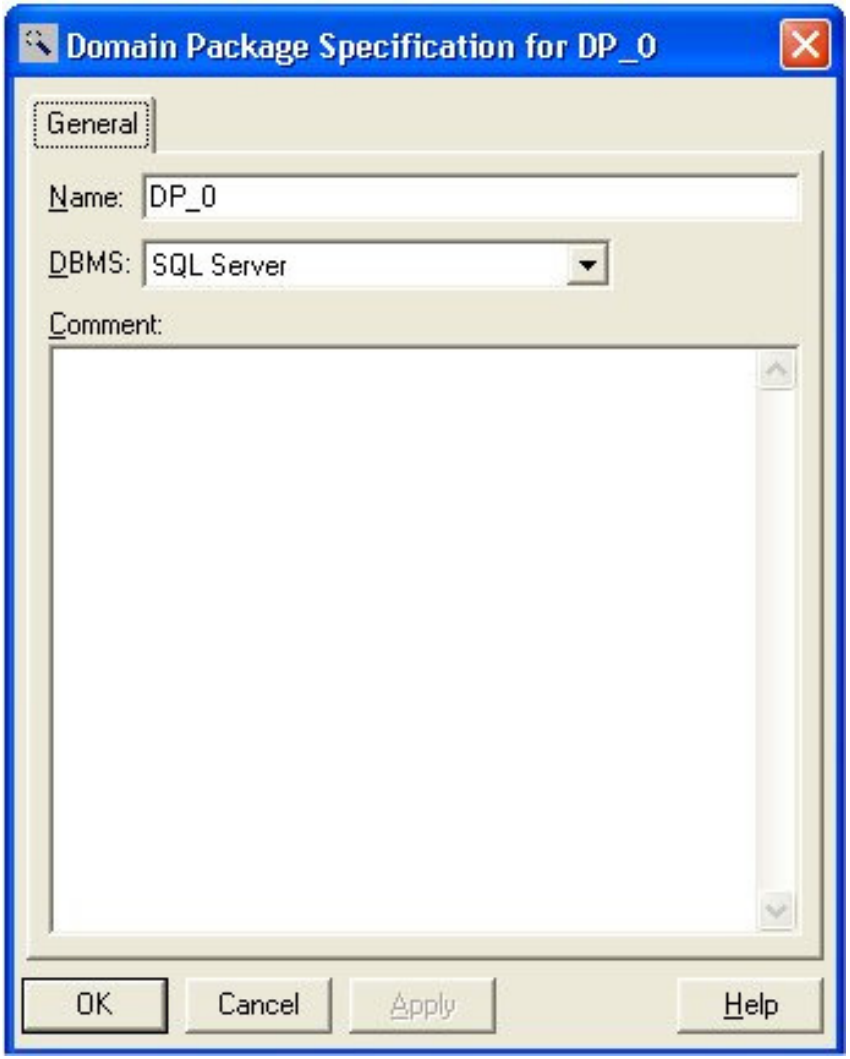


Tạo lập gói lĩnh vực

- Lĩnh vực (Domain) được sử dụng để áp dụng qui tắc nghiệp vụ (giá trị hợp lệ, giá trị mặc định của fields...) cho fields
  - Là mẫu cho một hoặc nhiều fields trong CSDL
  - Thí dụ:
    - Lập domain Phone (có kiểu dữ liệu Long, giá trị mặc định 0...) để áp dụng cho HomePhone, WorkPhone, FaxPhone...
- Việc sử dụng domain là tùy ý
- Trong Rose:
  - Các domains đặt trong gói domain
  - Mỗi gói domain được gán duy nhất cho DBMS
  - Có thể áp dụng một domain cho nhiều schema

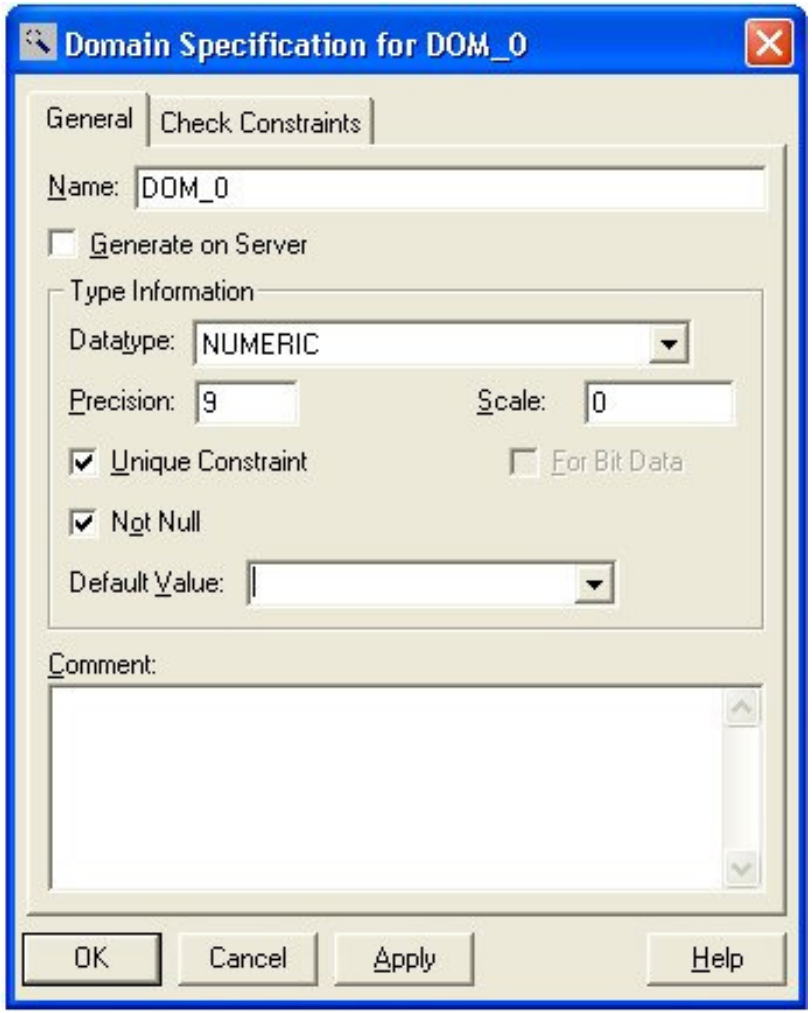
Tạo lập gói lĩnh vực

- Tạo lập gói lĩnh vực trong Rose:
  - Nhấn phím chuột phải trên Logical View trong Browser
  - Chọn Data Modeler-> New -> Domain Package
  - Nhấn phím phải trên gói mới và chọn Open Specification
  - Chọn DBMS sẽ sử dụng cho gói domain



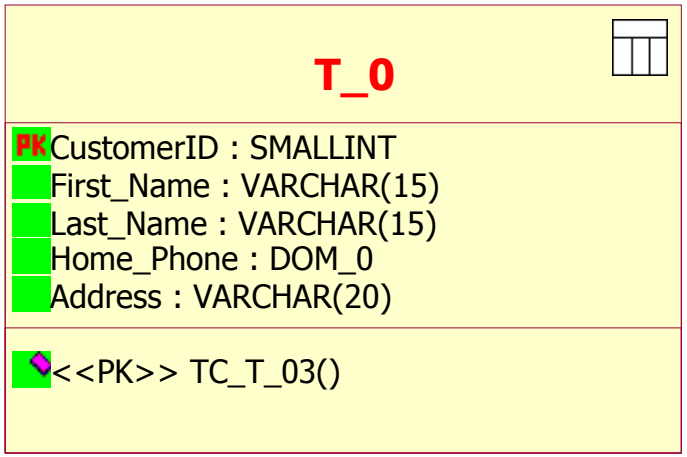
Tạo lập lĩnh vực

- Tạo lập lĩnh vực trong Rose:
  - Nhấn phím chuột phải trên gói lĩnh vực trong Browser
  - Chọn Data Modeler-> New -> Domain
  - Nhấn phím phải trên domain mới và chọn Open Specification
  - Nhập tên domain trong General Tab
  - Nhập các tham số khác trong Tab
    - Scale: Tổng chữ số sau dấu thập phân
    - Unique Constraint: Các fields sử dụng domain này sẽ có giá trị duy nhất
  - Bảng Check Constraints:
    - Là biểu thức cần có giá trị True trước khi thay thế dữ liệu trong CSDL



Bổ sung bảng

- Bảng là cấu trúc mô hình cơ sở của CSDL quan hệ
  - Biểu diễn tập các bản ghi có cùng cấu trúc (cột)
  - Mỗi bản ghi chứa dữ liệu, thông tin về bản ghi được lưu trữ ngay trong CSDL
- Khi đã có lược đồ, ta có thể tạo bảng trong nó
- Mỗi bảng trong CSDL được mô hình hóa như lớp persistent với stereotype Table
- Các bảng trong lược đồ có tên duy nhất



Bổ sung các chi tiết trong bảng

- Bổ sung cột
  - Cột dữ liệu
    - Chứa dữ liệu không phải tính từ các cột khác
  - Cột tính toán
    - Sử dụng các lệnh SQL để tính dữ liệu từ các cột khác
  - SQL Server hỗ trợ khái niệm cột đồng nhất
    - Là cột có giá trị kiểu Interger
    - Các giá trị của cột được SQL server tự động gán 1,2,3...
- Đặt khóa chính
  - Nếu cột đánh dấu là primary key thì giá trị của chúng là duy nhất để phân biệt các hàng
- Bổ sung ràng buộc
  - Là lệnh điều kiện cần được thỏa mãn để có thể cập nhật bảng
  - Là cách để áp dụng quy tắc nghiệp vụ
    - Thí dụ trường Gender phải là M hoặc F

Bổ sung các chi tiết trong bảng

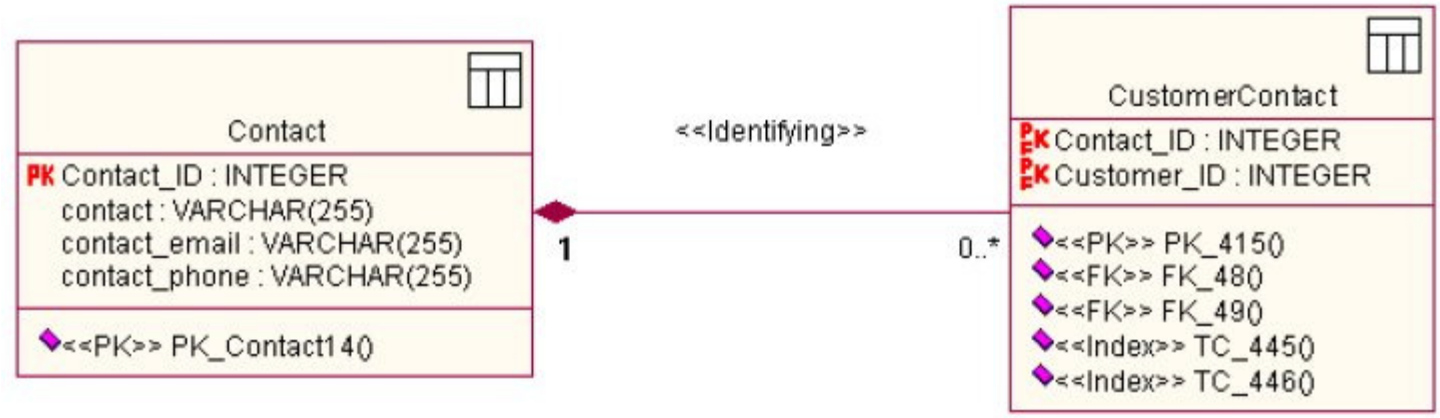
- Ràng buộc khóa
  - Ràng buộc khóa chính
    - Đảm bảo rằng dữ liệu nhập vào trường khóa là khác null và duy nhất
    - Rose tự động tạo ràng buộc khóa chính khi khóa chính của bảng được xác định
  - Ràng buộc duy nhất
    - Đảm bảo rằng giá trị nhập vào trường là duy nhất
    - Rose tự động tạo ràng buộc này khi ta chọn đặc tả Unique Constraint
  - Chỉ số
    - Cho khả năng xâm nhập nhanh bản ghi thông qua danh sách các cột khóa khi tìm kiếm các bản ghi trong bảng
- Ràng buộc kiểm tra
  - Là ràng buộc không thuộc loại ràng buộc khóa
- Bổ sung Trigger
  - Trigger là thủ tục SQL chạy khi có sự kiện xảy ra
    - Thí dụ: Đặt Trigger khi chèn, thay đổi hay hủy hàng trong bảng.
  - Đặc tả Trigger khác nhau trong các DBMS khác nhau
  - Được mô hình hóa trong Logical View

Bổ sung các chi tiết trong bảng

- Bổ sung index
  - Chỉ số được mô hình hóa như ràng buộc khóa trong bảng
  - Là cấu trúc cho phép tìm kiếm nhanh trong bảng
  - Có thể sử dụng một hay nhiều cột làm index
    - Khi tìm kiếm thì chỉ tìm kiếm trên cột này.
- Bổ sung thủ tục lưu trữ
  - Tương tự Trigger, Stored procedure là một đoạn chức năng trong CSDL
  - Nó là đoạn trình nhỏ được chương trình hay trigger kích hoạt
  - Nó chấp nhận tham số đầu vào và cho lại một hay nhiều giá trị (tham số đầu ra)
  - Trong Rose, thủ tục lưu trữ được mô hình hóa như thao tác với stereotype <<SP>>

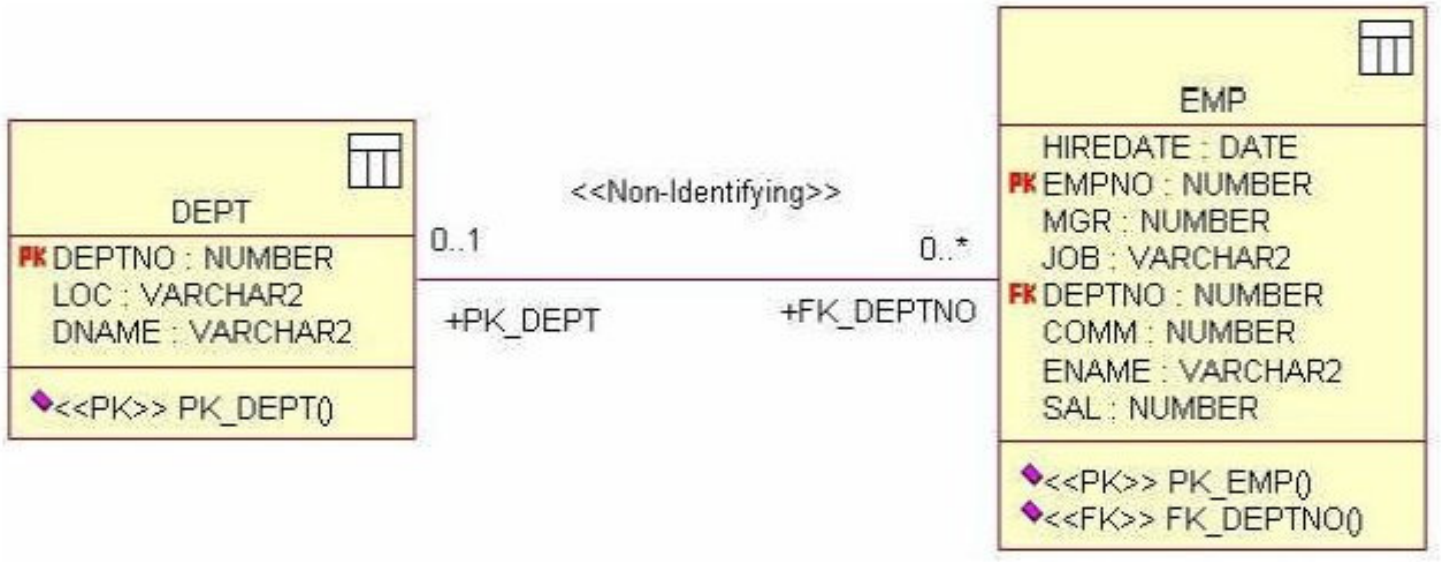
Bổ sung các chi tiết trong bảng

- Bổ sung quan hệ
  - Quan hệ trong mô hình dữ liệu tương tự quan hệ trong mô hình đối tượng
    - Quan hệ trong mô hình dữ liệu kết nối hai bảng
  - Rose có hai loại quan hệ chính
    - indentifying relationship và
    - non- indentifying relationship
  - Khóa ngoài được bổ sung vào bảng con để hỗ trợ quan hệ
  - Trong identifying relationship
    - Khóa ngoài trở thành một phần khóa chính của bảng con
    - Identifying relationship được mô hình hóa như composite aggregation.



Bổ sung các chi tiết trong bảng

- Bổ sung quan hệ
  - Trong non-identifying relationship
    - Khóa ngoài được tạo lập trong bảng con, nhưng nó không phải là một phần khóa chính của bảng con
    - Cardinality điều khiển bản ghi trong bảng con có thể tồn tại mà không cần liên kết với bản ghi trong bảng cha nó?
    - Thí dụ, Cardinality bằng 1 thì bản ghi của bảng cha phải tồn tại, nếu bằng 0..1 thì không cần.

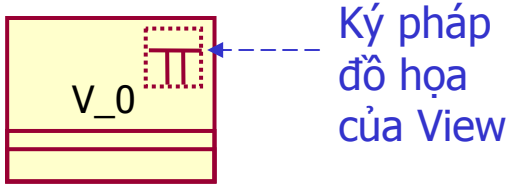


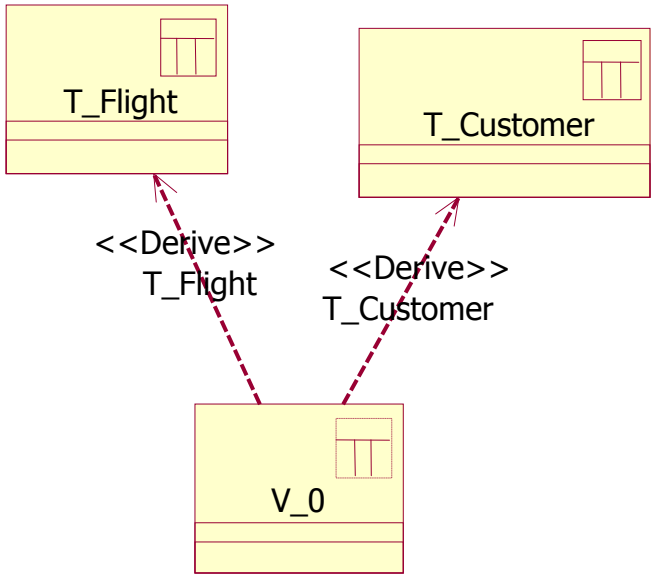
Bổ sung các chi tiết trong bảng

- Bổ sung qui tắc toàn vẹn tham chiếu (Referential Integrity)
  - Toàn vẹn tham chiếu hình thành tập các qui tắc giúp đảm bảo tính nhất quán
  - Thí dụ,
    - Worker A có bản gi trong bảng Employee và hai bảng ghi trong bảng Address. Nếu bản ghi trong bảng Employee bị xóa thì mất tính nhất quán
  - Toàn vẹn tham chiếu tránh tình huống này bằng cách xác định cái gì có thể xảy ra khi cập nhật hoặc hủy bỏ, các lựa chọn là
    - Bản ghi con tự động cập nhật, hủy bỏ
    - Tránh cập nhật hủy bỏ bản ghi cha
  - Trong Rose: Thông tin giải pháp lựa chọn được mô tả trong đặc tả quan hệ
  - Hai toàn vẹn tham số cơ bản
    - Trigger: Thực hiện Trigger khi cập nhật, hủy bỏ bản ghi cha
    - Declarative: Bao gồm các ràng buộc thuộc một phần của khóa ngoài

Các khung nhìn

- Khung nhìn (view) là cách quan sát dữ liệu dưới khuôn mẫu khác với cấu trúc lưu trữ của chúng
- Có thể tạo lập bảng virtual nhờ khung nhìn để chứa dữ liệu từ một hay nhiều bảng trong CSDL
- Khung nhìn đảm bảo an toàn cơ sở dữ liệu
  - Ta có thể lập nhóm người sử dụng chỉ có thể đọc dữ liệu thông qua khung nhìn trong CSDL để tránh tự do sửa đổi dữ liệu
- Biểu diễn khung nhìn bằng quan hệ vẽ giữa view và các bảng nguồn





Phát sinh Object model từ Data model

- Rose cho khả năng tự động phát sinh object model từ data model
- Khả năng đặc biệt hữu ích khi ta đã có ứng dụng và CSDL
- Không phải mọi kiến trúc trong mô hình dữ liệu đều chuyển đổi sang mô hình đối tượng

Phần tử mô hình dữ liệu	Phần tử mô hình đối tượng
Schema	Package
Table	Class
Column	Attribute
Trigger, Stored procedure	None
Intersection table with primary/secondary key columns	Many-to-many association
Intersection table with columns other than primary/secondary key	Many-to-many association with association class
Identifying relationship	Composite aggregation
Non- identifying relationship	Association
Cardinality	Cardinality
Index, Database, Constraint, Domain	None

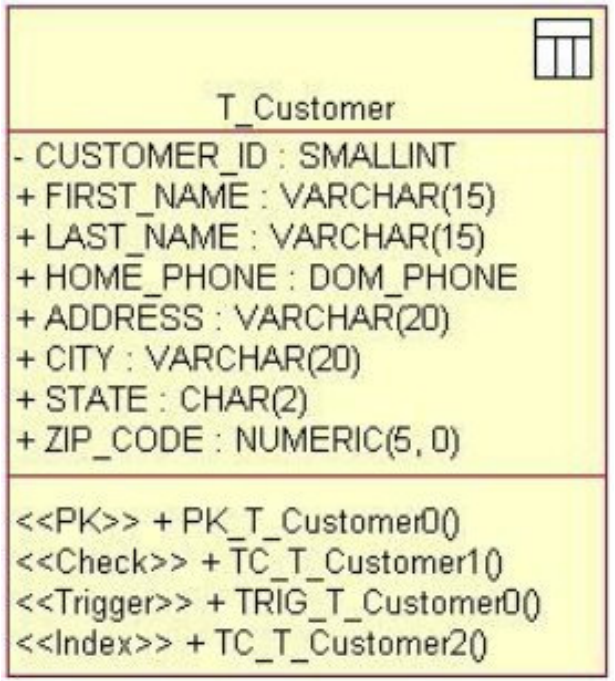
Phát sinh Data model từ Object model

- Khi yêu cầu phát sinh data model từ mô hình, Rose tìm kiếm các lớp có thuộc tính đánh dấu persistent là True (trong cửa sổ đặc tả lớp).

Phần tử mô hình đối tượng	Phần tử mô hình dữ liệu
Package	Schema
Persistent class	Table
Attribute	Column
Operation	None
Many-to-many association	Intersection table
Composite aggregation	Identifying relationship
Association	Non- identifying relationship
Cardinality	Cardinality
Association class	Intersection table

Phát sinh CSDL từ Data model

- Vào bất cứ thời điểm nào ta đều có thể phát sinh CSDL hay DDL script từ mô hình dữ liệu
- Rose cho hai khả năng
  - Phát sinh đơn thuần DDL
  - Chạy DDL để phát sinh CSDL
- Cái gì được phát sinh?
  - Bảng, cột và quan hệ trong lược đồ được phát sinh trong DDL hay CSDL



Table

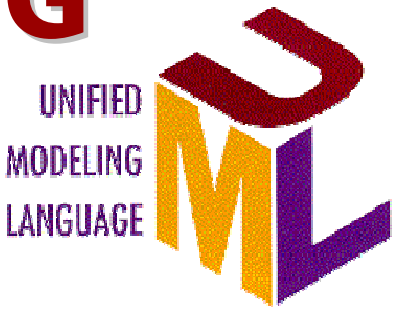
DDL

```
CREATE TABLE T_Customer (
CUSTOMER_ID SMALLINT IDENTITY NOT NULL,
FIRST_NAME VARCHAR(15) NOT NULL
LAST_NAME VARCHAR(15) NOT NULL
CONSTRAINT PK_T_Customer0 PRIMARY KEY NONCLUSTERED (CUSTOMER_ID)
CONSTRAINT PC_T_Customer1 CHECK(CUSTOMER_ID>1000)
) ON STP0
GO
CREATE INDEX TC_T_Customer2 ON T_Customer(ZIP_CODE)
GO
```

Tóm tắt

- Bài này đã xem xét các vấn đề sau
  - Mô hình đối tượng và mô hình dữ liệu
  - Tạo lập mô hình dữ liệu
  - Bổ sung CSDL, lược đồ, gói lĩnh vực, bảng, thủ tục lưu trữ và các quan hệ
  - Ứng dụng Rose vào
    - Mô hình hóa dữ liệu
    - Chuyển đổi qua lại giữa mô hình dữ liệu và mô hình đối tượng

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG



Nội dung

1. Tiến trình phát triển phần mềm theo hướng đối tượng
2. Giới thiệu Ngôn ngữ mô hình hóa thống nhất UML
3. Mô hình hóa nghiệp vụ
4. Mô hình hóa trường hợp sử dụng
5. Mô hình hóa tương tác đối tượng
6. Biểu đồ lớp và gói
7. Biểu đồ chuyển trạng thái và biểu đồ hoạt động
8. Biểu đồ kiến trúc vật lý và phát sinh mã trình
9. Mô hình hóa dữ liệu
- ✓ **Bài học thực nghiệm**

Bài 10

Bài học thực nghiệm
Hệ thống quản lý thư viện

Các bước xây dựng ứng dụng

- Mục đích:
  - Xây dựng hệ thống phần mềm với khả năng quản lý việc mượn sách và tạp chí trong thư viện.
  - Hệ thống sẽ xây dựng còn đơn giản nhưng dễ dàng mở rộng sau này
- Các bước chính
  - Hình thành mô hình phân tích là kết quả phân tích và mô tả ứng dụng
    - Phân tích lĩnh vực vấn đề
  - Mở rộng mô hình phân tích thành mô hình thiết kế
  - Lập trình để có chương trình chạy được

Đặc tả yêu cầu hệ thống

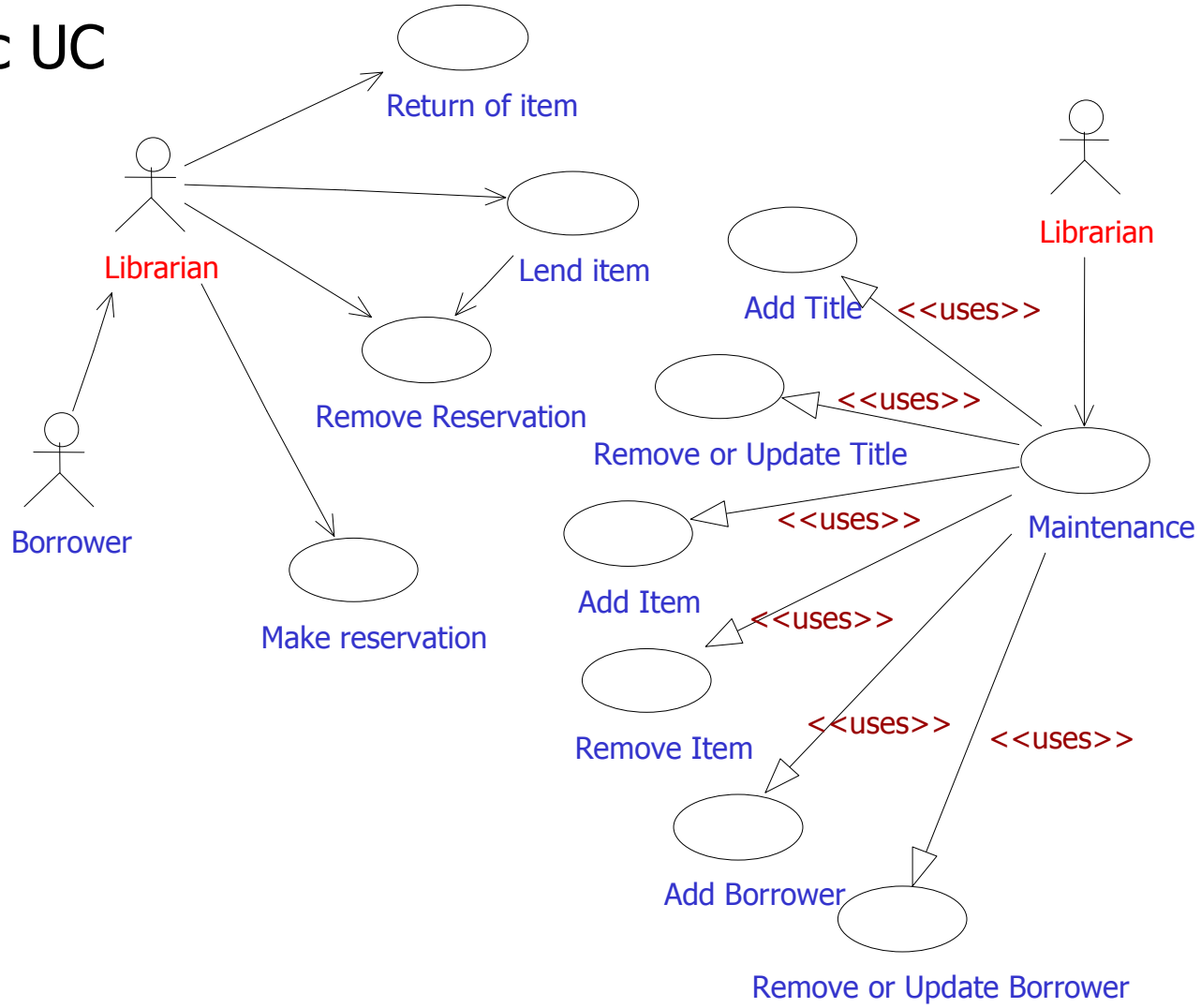
- Đây là hệ thống hỗ trợ quản lý thư viện
- Thư viện cho đọc giả mượn sách, tạp chí
  - Đọc giả, sách và tạp chí được đăng ký trước trong hệ thống
- Thư viện mua sách, tạp chí mới.
  - Sách, tạp chí thông dụng được mua nhiều bản.
  - Huỷ bỏ sách và tạp chí cũ khi quá hạn hay rách nát
- Thủ thư là nhân viên của thư viện giao tiếp với đọc giả và hệ thống sẽ hỗ trợ công việc của nó.
- Độc giả có thể đặt trước sách hay tạp chí mà nó chưa có trong thư viện.
  - Khi người khác trả hay được mua mới về thì độc giả được thông báo.
  - Độc giả hay thủ thư có khả năng huỷ bỏ đặt trước.
- Có khả năng tạo lập, cập nhật, huỷ bỏ thông tin về đầu sách, độc giả, việc cho mượn (*loan*) và đặt trước trong hệ thống.
- Hệ thống có thể chạy trên *Unix, Windows*... và có giao diện đồ họa dễ sử dụng.
- Dễ dàng mở rộng các chức năng mới cho hệ thống trong tương lai.

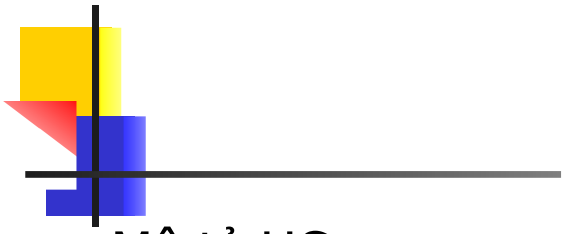
Phân tích hệ thống

- Nhiệm vụ của phân tích
  - Thu thập mô tả toàn bộ yêu cầu để hình thành mô hình phân tích
  - Xác định các lớp chính trong lĩnh vực vấn đề (các khái niệm)
    - Xác định cái sẽ được hệ thống quản lý
- Các bước trong phân tích hệ thống
  - Phân tích yêu cầu
    - Xác định các UC để mô tả cái mà hệ thống thư viện cần có về mặt chức năng -> yêu cầu chức năng hệ thống
    - Tìm kiếm các tác nhân: Thủ thư và Độc giả
    - Giả sử: Độc giả không trực tiếp sử dụng hệ thống.
    - Mô tả tác nhân
      - Độc giả (Borrower): là người có thể mượn, đặt trước sách hay tạp chí. Tác nhân này có thể là con người hay thư viện khác.
      - Thủ thư (Librarian): là người thực sự sử dụng hệ thống. Họ duy trì hệ thống, thực hiện các chức năng cơ bản như cho mượn, đặt trước và được thông báo về các thông tin khác của thư viện

Phân tích hệ thống

- Xác định các UC





Phân tích hệ thống

- Mô tả UC
  - UC Cho mượn tài liệu (*Lend Item*)
    - Nếu độc giả không đặt trước để mượn
      1. Tìm tên sách
      2. Tìm thấy còn sách trong thư viện
      3. Nhận danh độc giả
      4. Thư viện cho mượn sách
      5. Đăng ký mượn
    - Nếu độc giả đã đặt trước
      1. Nhận danh độc giả
      2. Nhận danh sách mượn
      3. Nhận thấy còn sách trong thư viện
      4. Thư viện cho mượn
      5. Đăng ký mượn
      6. Huỷ bỏ đặt trước
  - UC Trả tài liệu (*Return Item*)
    1. Người mượn trả tài liệu
    2. Nhận danh độc giả trả sách
    3. Nhận danh tài liệu mượn
    4. Nhận biết việc cho mượn thông qua tên tài liệu và tài liệu
    5. Huỷ bỏ việc cho mượn

Phân tích hệ thống

- Mô tả UC
  - UC Đặt trước (*Make Reservation*)
    1. Nhận biết tên tài liệu
    2. Nhận biết tên người mượn
    3. Đăng ký mượn theo tên tài liệu với số lượng và tên người mượn
  - UC Huỷ đặt trước (*Remove Reservation*)
  - UC Bổ sung tên tài liệu (*Add Title*)
    - Bổ sung vào hệ thống tên tài liệu mới bao gồm tên tài liệu, tác giả và số ISBN
  - UC Cập nhật hay huỷ bỏ tên tài liệu (*Update or Remove Title*)
    - Cập nhật tên tài liệu:
      1. Xác định tên tài liệu sẽ cập nhật
      2. Hiển thị các thông tin như tên tài liệu, tên tác giả, ISBN... để cập nhật
      3. Cập nhật thông tin về tài liệu
    - Huỷ bỏ tên tài liệu:
      1. Xác định tên tài liệu sẽ huỷ
      2. Huỷ toàn bộ tài liệu có tên đó kèm theo mọi tài liệu cho mượn liên quan
      3. Huỷ bỏ tên tài liệu

Phân tích hệ thống

- Mô tả UC
  - UC Bổ sung tài liệu (*Add Item*)
    1. Nhận biết tên của tài liệu
    2. Kiểm tra xem tên tài liệu đã được đăng ký trong hệ thống? Nếu chưa cần đăng ký trước khi bổ sung tài liệu.
    3. Đòi hỏi thông tin về tài liệu (gán *id* duy nhất cho tài liệu, thông thường là mã vạch hay số dán sau bìa sách)
    4. Bổ sung tài liệu vào hệ thống
  - UC Huỷ bỏ tài liệu (*Remove Item*)
    1. Nhận biết tên tài liệu
    2. Đòi hỏi *id* của tài liệu
    3. Nếu tìm thấy tài liệu thì huỷ bỏ khỏi hệ thống, sau đó huỷ bỏ mọi đăng ký về mượn tài liệu này.

Phân tích hệ thống

- Mô tả UC
  - UC Bổ sung người mượn (*Add Borrower*)
    1. Đòi hỏi thông tin về người mượn như tên, địa chỉ, mã zip...
    2. Lưu trữ thông tin về người mượn vào máy.
  - UC Cập nhật, huỷ bỏ người mượn (*Update or Remove Borrower*)
    - Cập nhật người mượn
      1. Xác định người mượn sẽ cập nhật
      2. Hiển thị thông tin người mượn, thông tin có thể thay đổi
      3. Cập nhật thông tin người mượn
    - Huỷ người mượn
      1. Yêu cầu tên người mượn
      2. Huỷ bỏ thông tin người mượn khỏi hệ thống, kèm theo huỷ bỏ đặt trước, hủy bỏ đăng ký mượn của người mượn đó.

Phân tích hệ thống

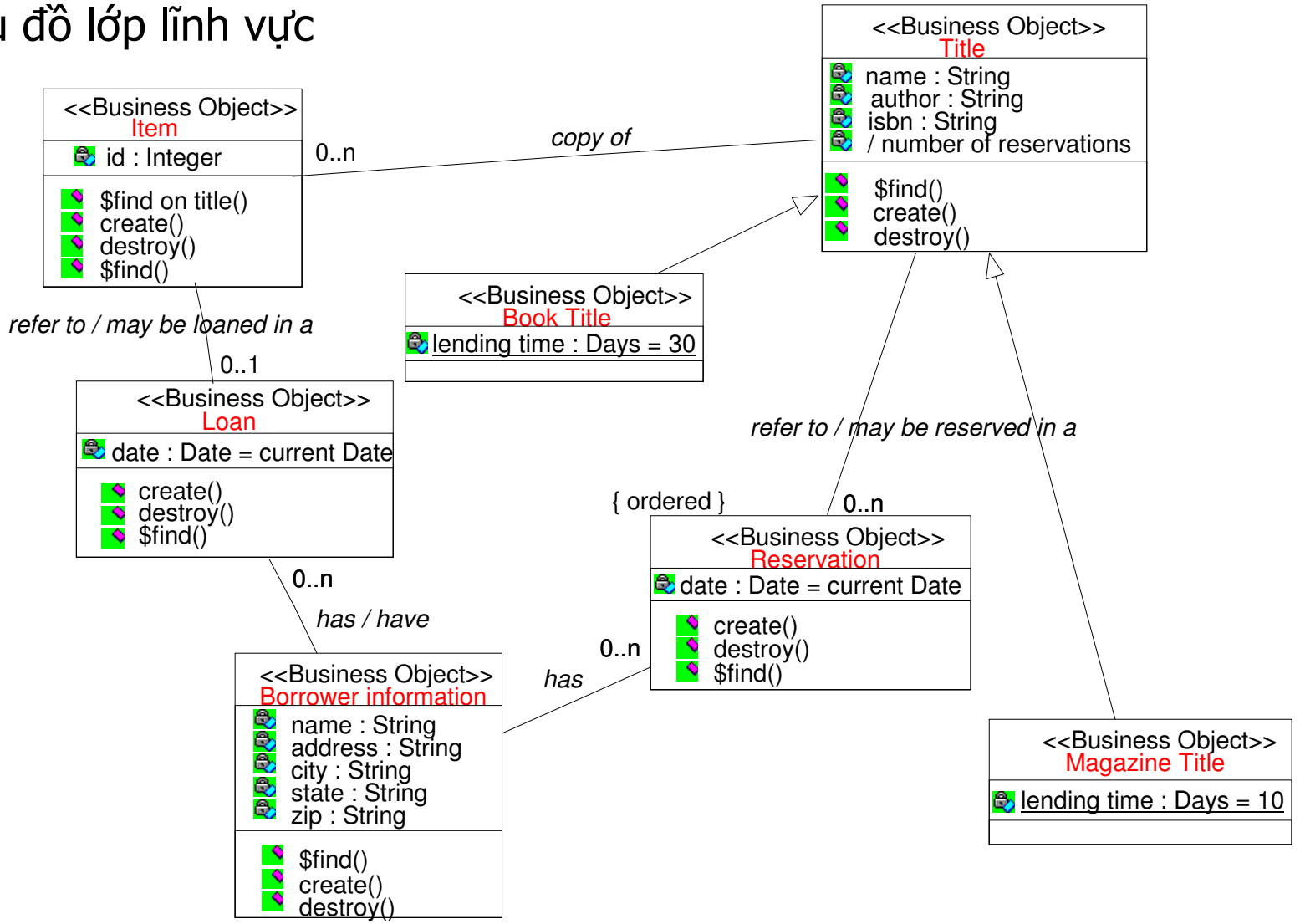
- Lập UC mới:
  - UC Bảo trì (*Maintenance*)
  - Là UC tổng quát và sử dụng một số UC khác
  - Để tách các chức năng nghiệp vụ khỏi chức năng bảo trì hệ thống
- Phân biệt hai khái niệm: Có nhiều tài liệu cùng tên
  - Tiêu đề (*Title*)
  - Tài liệu (*Item*)
- Xây dựng biểu đồ UC

Phân tích lĩnh vực vấn đề

- Tìm các khái niệm (lớp) trong đặc tả yêu cầu hệ thống và các UC
- Xác định các quan hệ giữa các lớp trong lĩnh vực vấn đề
  - Trong hệ thống thư viện có các khái niệm sau
    - BorrowerInformation (không đặt tên Borrower vì nó đã được chọn làm tên tác nhân)
    - Title
    - Book Title
    - Magazine Title
    - Item
    - Reservation
    - Loan
- Gán stereotype <<Bussines object>> cho các chúng

Phân tích lĩnh vực vấn đề

- Biểu đồ lớp lĩnh vực



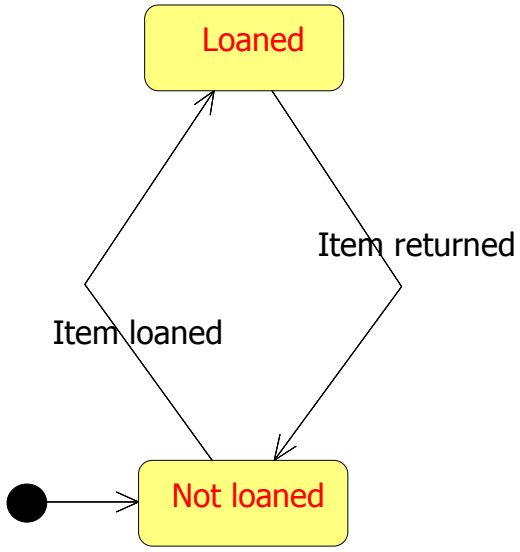
Phân tích lĩnh vực vấn đề

- Mô tả lớp lĩnh vực
  - Lớp Item
    - Biểu diễn tài liệu vật lý
    - Có hai trạng thái: Chưa cho mượn và Đã cho mượn
    - Có lớp Tên tài liệu tương ứng
  - Lớp Title
    - Biểu diễn tên sách hay tạp chí
    - Có tên, tác giả và ISBN...
    - Trạng thái: Đặt trước hay Không đặt trước để mượn
  - Lớp Book title:
    - Là đặc biệt hóa của lớp Title
    - Biểu diễn tên sách
  - Lớp Magazine title:
    - Là đặc biệt hóa của lớp Title
    - Biểu diễn tên tạp chí

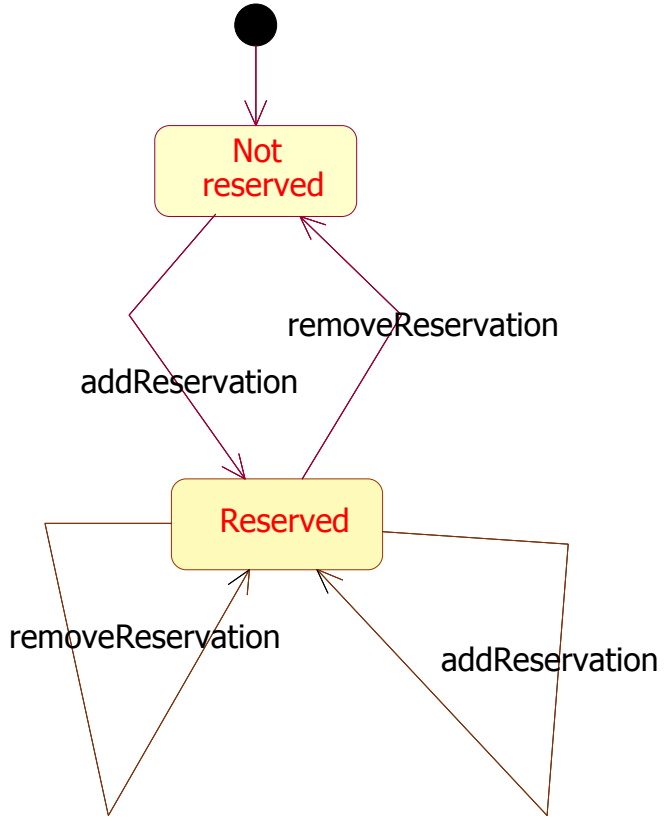
Phân tích lĩnh vực vấn đề

- Mô tả lớp lĩnh vực
  - Lớp Đặt trước (Reservation)
    - Có thể đặt trước Tên tài liệu để mượn Tài liệu
  - Lớp Cho mượn (Loan)
    - Là giao kèo giữa Độc giả và Thủ thư
    - Đối tượng này cho biết độc giả nào đó đã mượn tài liệu
    - Khi độc giả trả tài liệu thì đối tượng này mất đi
  - Lớp Thông tin độc giả (BorrowerInformation)
    - Là thông tin về con người cụ thể hay thư viện khác
    - Thông tin bao gồm tên, địa chỉ...
- Chú ý
  - Chỉ mới xem xét các lớp lĩnh vực
    - Chưa có đầy đủ thao tác và thuộc tính cho các lớp này
  - Có thể biểu diễn biểu đồ trạng thái cho một số lớp
    - Lớp Item và lớp Title

Phân tích lĩnh vực vấn đề



Lớp Item

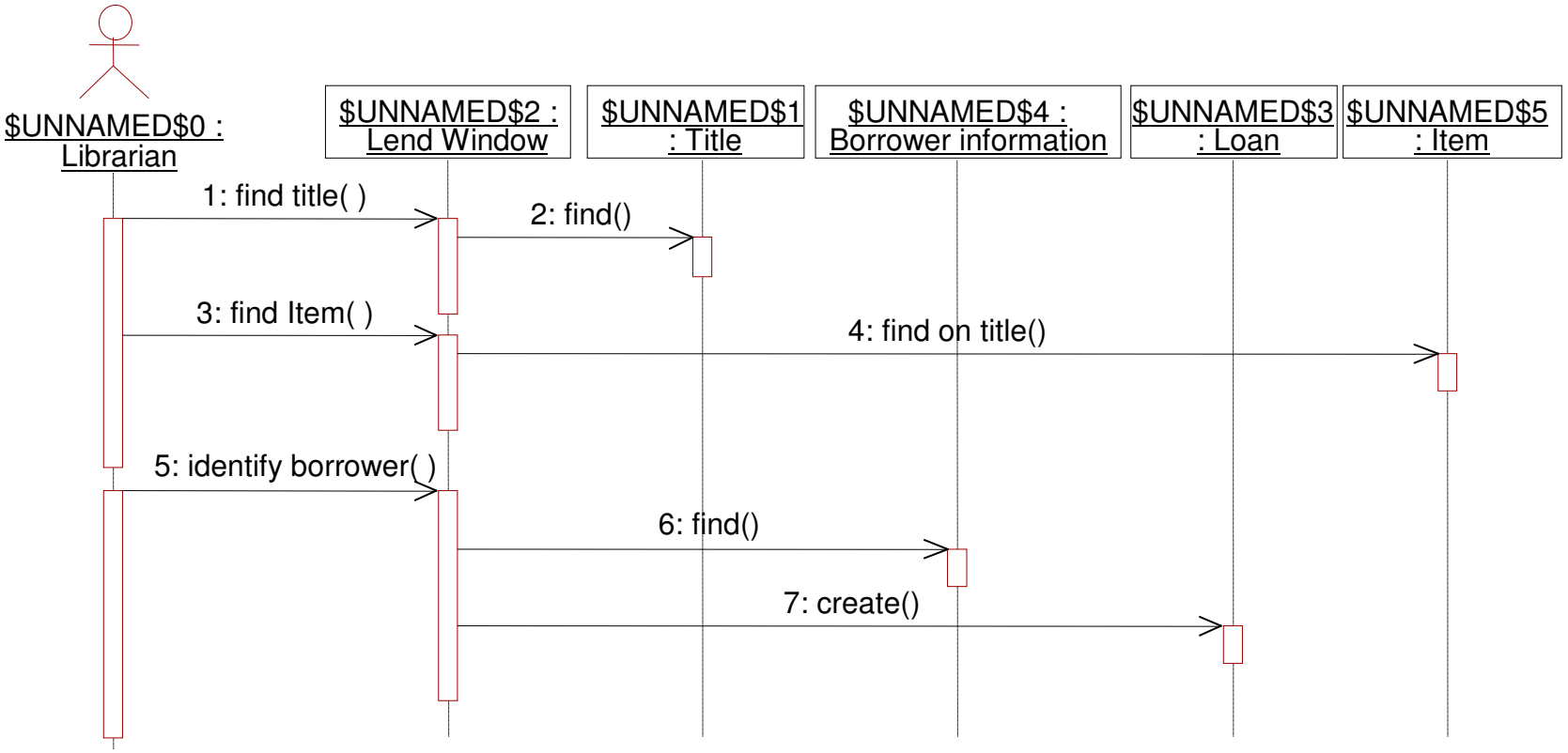


Lớp Title

Trạng thái lớp

Phân tích lĩnh vực vấn đề

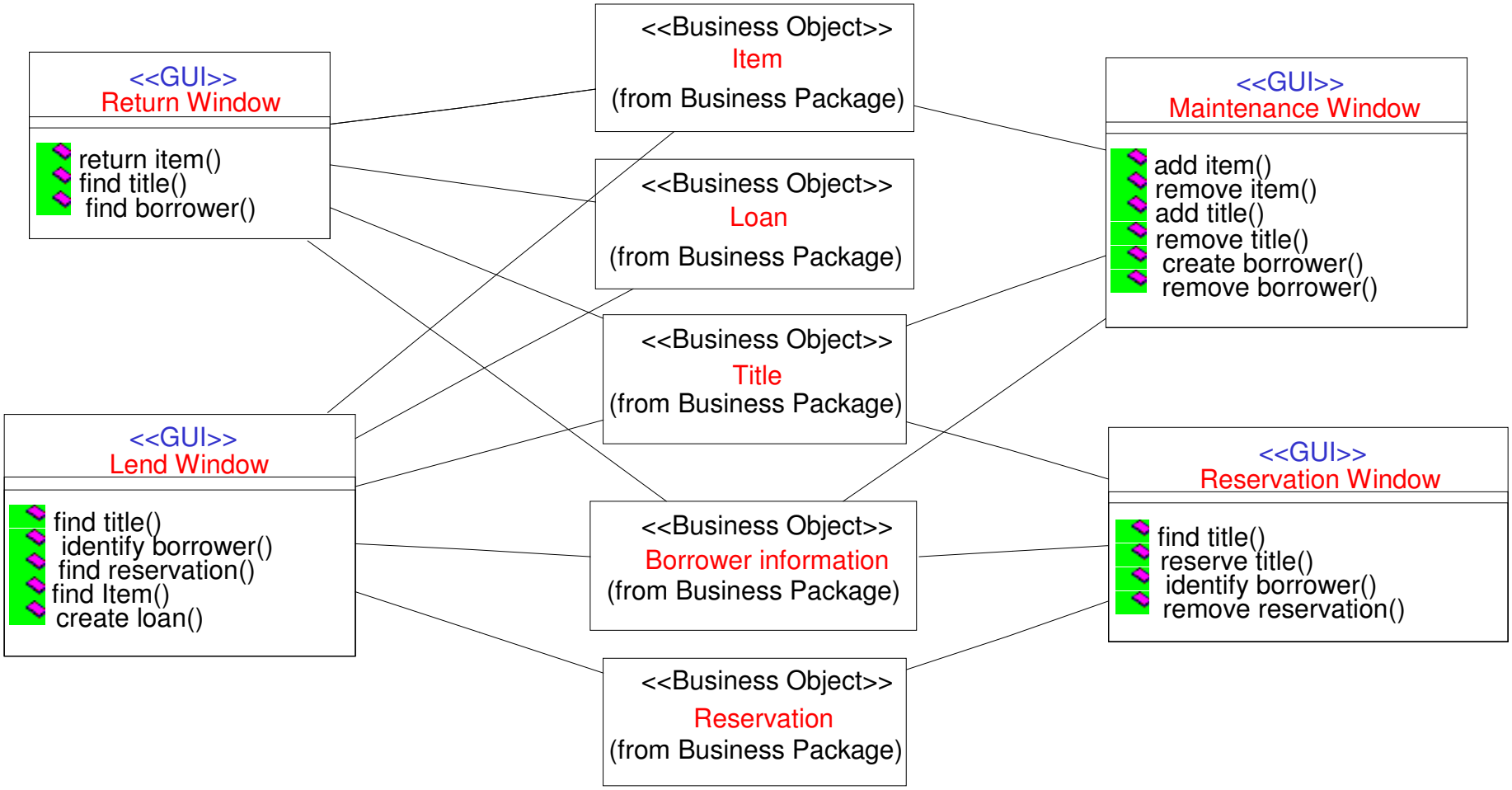
- Mô tả hành vi động các lớp lĩnh vực để thực hiện các UC
  - Sử dụng các Biểu đồ trình tự
  - Chỉ ra các thao tác của lớp nhưng chưa chi tiết tham số (signature)



Lend Item

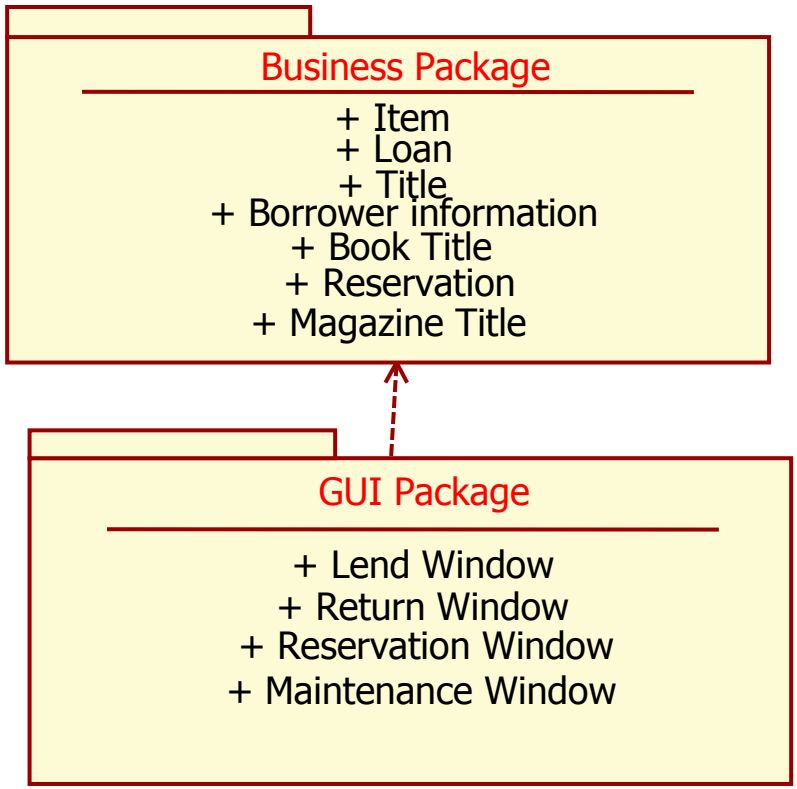
Phân tích lĩnh vực vấn đề

- Khi mô hình hóa biểu đồ trình tự ta nhận thấy cần có cửa sổ và hộp thoại để nhập liệu -> Bổ sung các lớp giao diện



Phân tích lĩnh vực vấn đề

- Phân gói để tách các lớp thực hiện logíc nghiệp vụ khỏi lớp giao diện
  - GUI Package
  - Business Package

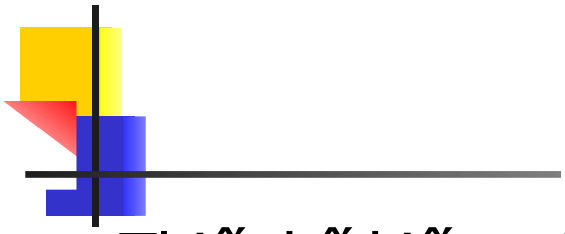


Thiết kế

- Trong pha này sẽ mở rộng và chi tiết hóa mô hình phân tích
  - Xem xét mọi vấn đề liên quan và phạm vị kỹ thuật
  - Mục tiêu là xác định giải pháp làm việc để dễ dàng chuyển sang mã trình
  - Chi tiết các lớp trong mô hình phân tích và bổ sung các lớp mới
- Hai loại thiết kế
  - Thiết kế kiến trúc
    - Thiết kế ở mức cao
    - Xác định các gói và phụ thuộc giữa chúng
    - Thiết kế kiến trúc tốt cho khả năng dễ dàng mở rộng và thay đổi hệ thống
  - Thiết kế chi tiết
    - Chi tiết nội dung trong các gói
    - Sử dụng các mô hình động của UML để mô tả ứng xử của các đối tượng lớp

Thiết kế

- Thiết kế kiến trúc
  - Mục tiêu: Tách logíc ứng dụng (các lớp nghiệp vụ) khỏi logíc kỹ thuật sao cho khi thay đổi nhóm này không ảnh hưởng đến nhóm khác
  - Mỗi gói giải quyết một vấn đề chức năng hay kỹ thuật cụ thể
  - Giải pháp
    - Nhận biết quy luật phụ thuộc giữa các gói (phân hệ)
      - Tránh phụ thuộc hai chiều (các gói không nên quá gắn chặt vào nhau)
    - Nhận biết các thư viện chuẩn sẽ sử dụng
      - Các thư viện CSDL, Giao diện, Truyền tin...

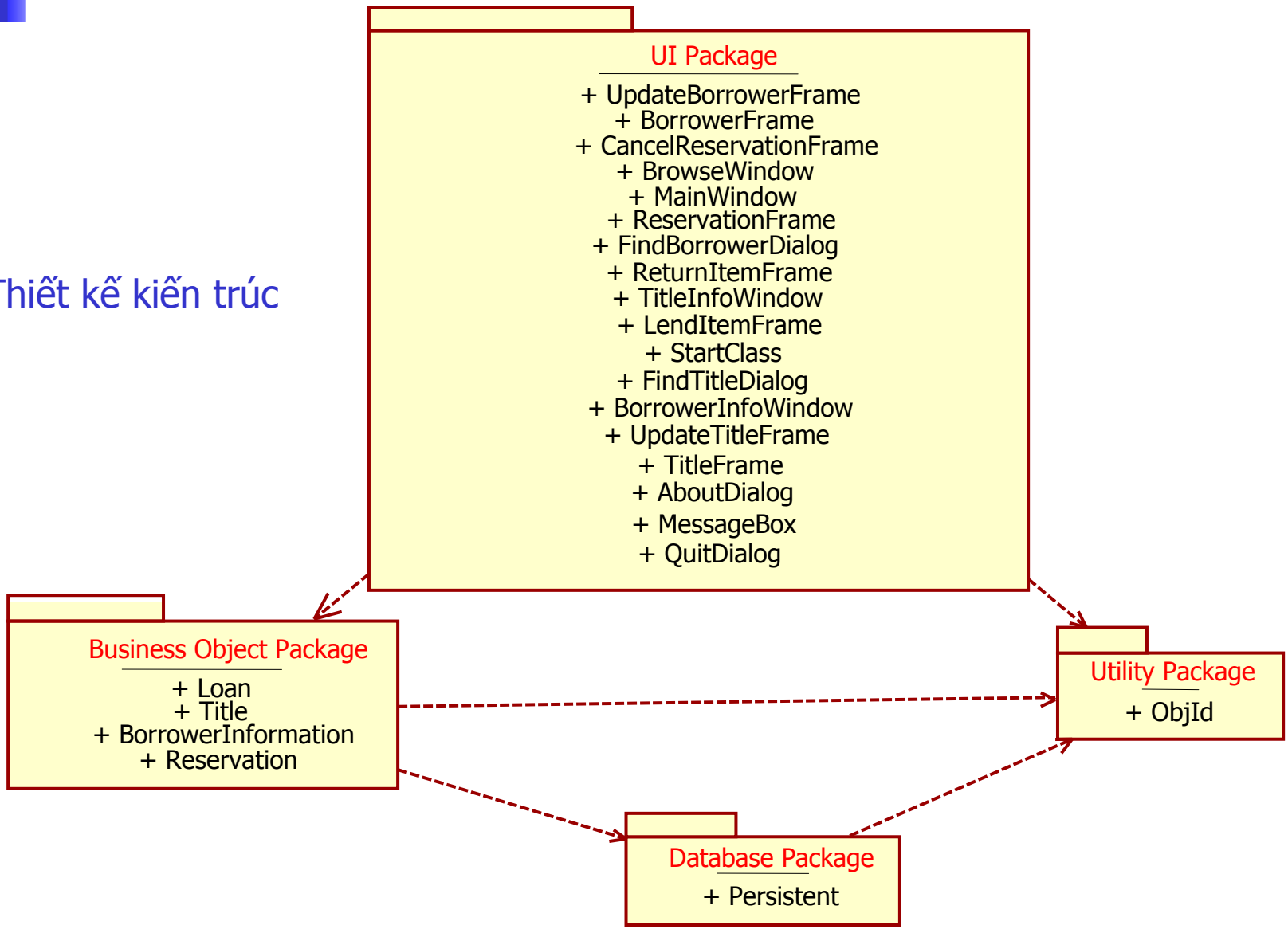


Thiết kế

- Thiết kế kiến trúc
  - Hình thành bốn gói trong ứng dụng Quản lý thư viện
    - Gói giao diện người sử dụng (User interface package)
      - Cho khả năng quan sát dữ liệu và nhập dữ liệu
      - Các lớp này hình thành trên cơ sở thư viện giao diện chuẩn hay của ngôn ngữ lập trình (MFC, Java AWT package...)
      - Chúng có quan hệ với gói đối tượng nghiệp vụ
    - Gói các đối tượng nghiệp vụ (Business objects package)
      - Bao gồm các lớp nghiệp vụ (Title, Item, Loan, BorrowerInformation...) từ mô hình phân tích
      - Các lớp được chi tiết hóa trong pha thiết kế
      - Nó có quan hệ với gói CSDL để lưu trữ dữ liệu trên tệp
    - Gói CSDL (Database package)
      - Cung cấp dịch vụ cho các lớp trong gói đối tượng nghiệp vụ để có thể lưu trữ nó trên tệp
      - Thí dụ gói này chứa lớp Persistent lưu trữ đối tượng lên tệp
    - Gói tiện ích (Utility package)
      - Chứa các dịch vụ cho mọi gói khác sử dụng
      - Thí dụ gói này chứa lớp ObjID sử dụng cho mọi đối tượng lưu trữ trên tệp

Thiết kế

Thiết kế kiến trúc

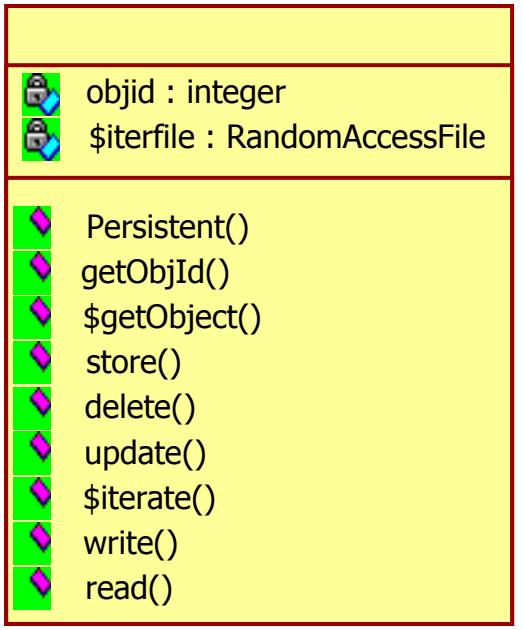


Thiết kế

- Thiết kế chi tiết
  - Chi tiết nội dung trong các gói
    - Các lớp được mô tả chi tiết đủ cho lập trình
  - Mô tả các lớp kỹ thuật mới bổ sung (trong gói UI và gói CSDL)
    - Để mở rộng, chi tiết hóa các lớp nghiệp vụ được phác họa trong pha phân tích
    - Thực hiện bằng cách tạo ra các các phần tử mới trong các biểu đồ bao gồm biểu đồ lớp, biểu đồ trạng thái và biểu đồ hoạt động

Thiết kế

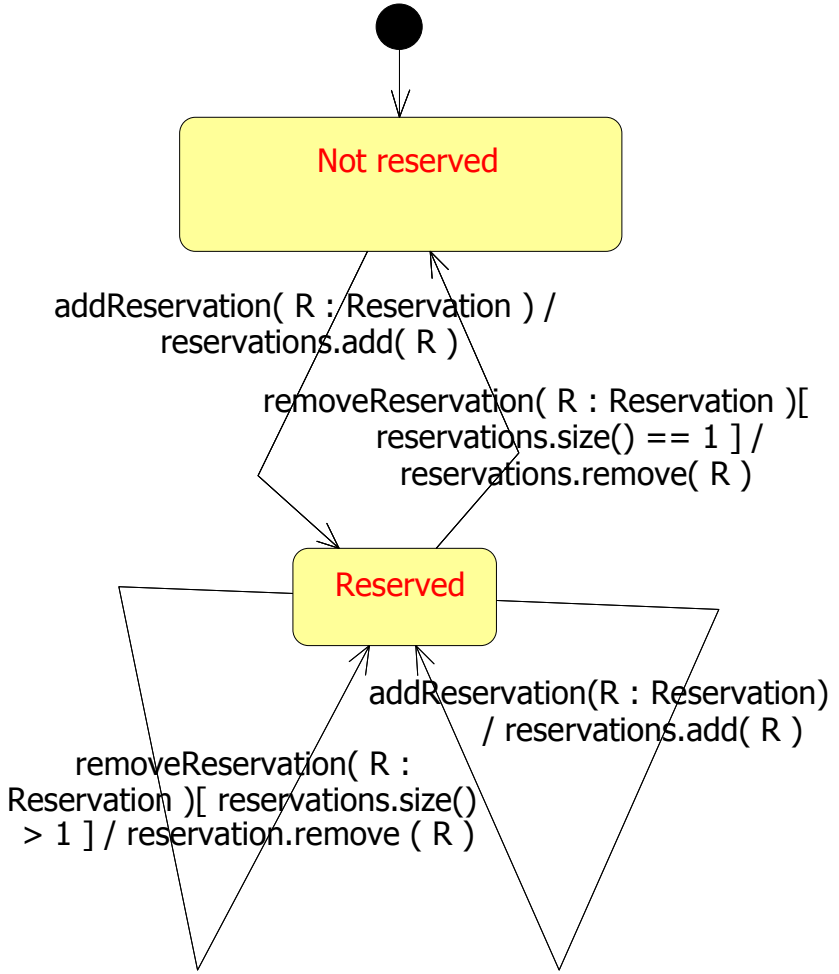
- Thiết kế chi tiết
  - Gói CSDL
    - Bổ sung dịch vụ lưu trữ đối tượng
      - Cho tệp phẳng, CSDL thương mại như CSDL quan hệ hay CSDL hướng đối tượng...
    - Thí dụ với giải pháp lưu trữ các đối tượng trên tệp
      - Chi tiết lưu trữ trên tệp là ẩn với các đối tượng nghiệp vụ
      - Các đối tượng nghiệp vụ chỉ việc gọi các thao tác chung của chúng như store(), update(), delete(), find()...
      - Hình thành lớp mới: Persistent
        - Là lớp cha của các lớp có nhu cầu lưu trữ đối tượng
        - Nó là lớp trừu tượng
        - Các lớp kế thừa từ Persistent cần cài đặt read(), write()
      - Sử dụng lớp ObjId của gói tiện ích



Khi thay đổi cách lưu trữu CSDL thì chỉ thay đổi lớp Persistent.

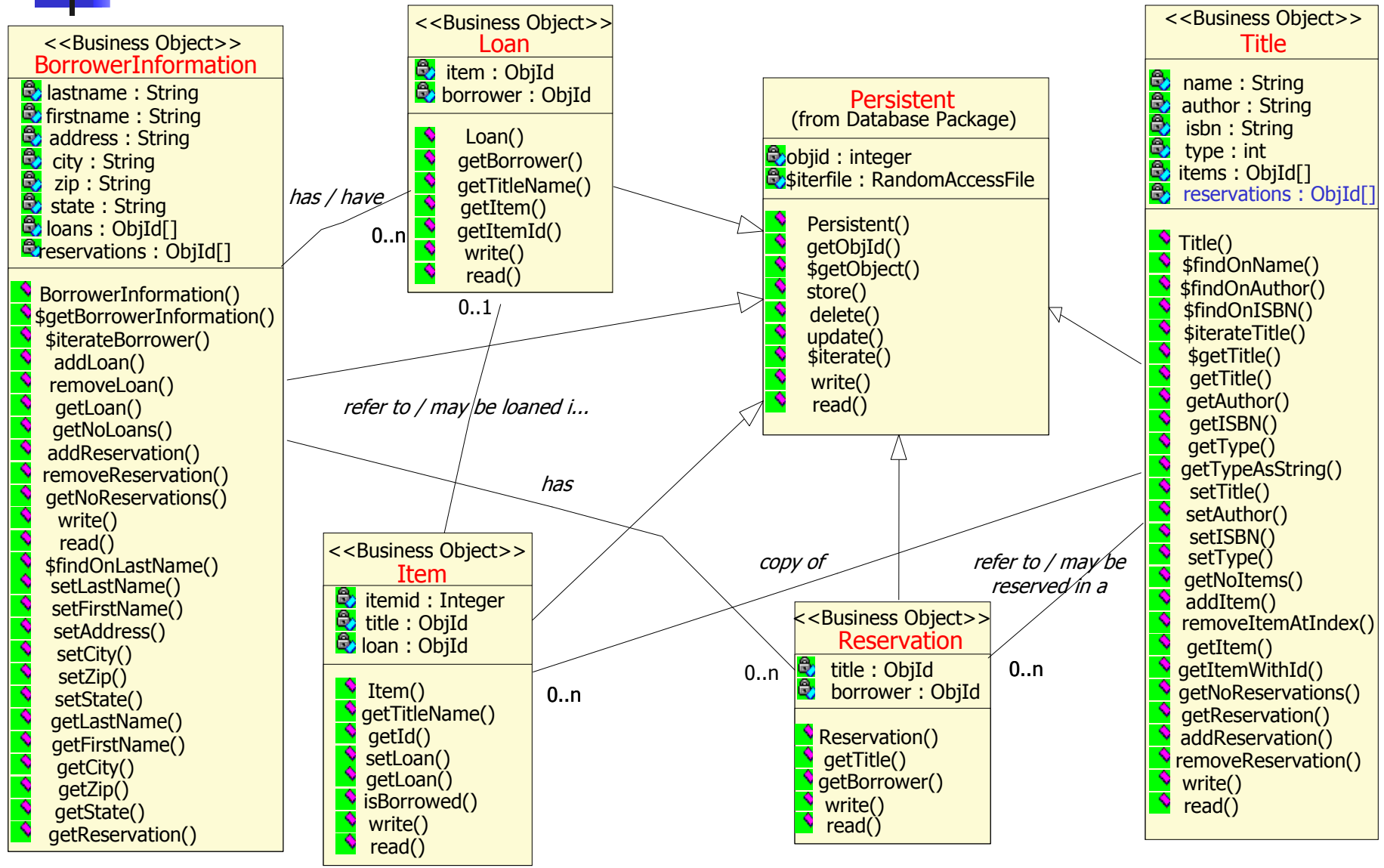
Thiết kế

- Thiết kế chi tiết
  - Gói Business objects package
    - Bổ sung chi tiết các lớp đối tượng nghiệp vụ trong biểu đồ lớp
      - Định nghĩa các đối số, giá trị cho lại của các hàm thành phần
      - Bổ sung hàm read(), write() cho các lớp kế thừa từ Persistent của gói CSDL
    - Bổ sung chi tiết trong biểu đồ trạng thái
      - Trong thiết kế lớp Title: trạng thái xác định nhờ véc tơ reservations
      - Các đối tượng khác làm thay đổi trạng thái Title bằng gọi các hàm addReservation() hay removeReservation()



Lớp Title

Thiết kế

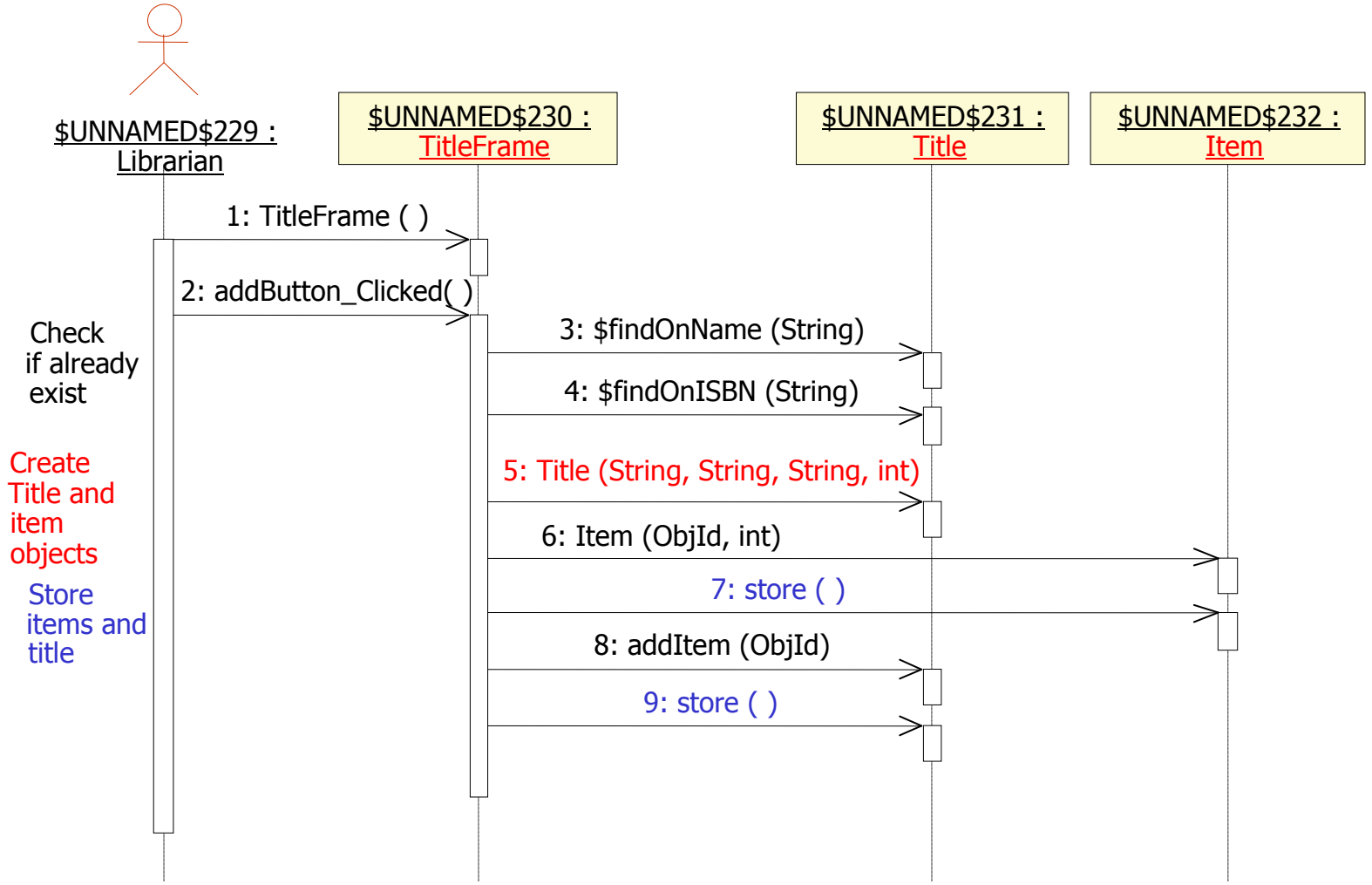

<<Business Object>>
BorrowerInformation
lastname : String
firstname : String
address : String
city : String
zip : String
state : String
loans : ObjId[]
reservations : ObjId[]
BorrowerInformation()
$getBorrowerInformation()
$iterateBorrower()
addLoan()
removeLoan()
getLoan()
getNoLoans()
addReservation()
removeReservation()
getNoReservations()
write()
read()
$findOnLastName()
setLastName()
setFirstName()
setAddress()
setCity()
setZip()
setState()
getLastName()
getFirstName()
getCity()
getZip()
getState()
getReservation()
has / have
0..n
<<Business Object>>
Loan
item : ObjId
borrower : ObjId
Loan()
getBorrower()
getTitleName()
getItem()
getItemId()
write()
read()
0..1
refer to / may be loaned i...
has
<<Business Object>>
Item
itemid : Integer
title : ObjId
loan : ObjId
Item()
getTitleName()
getId()
setLoan()
getLoan()
isBorrowed()
write()
read()
0..n
Persistent
(from Database Package)
objid : integer
$iterfile : RandomAccessFile
Persistent()
getObjId()
$getObject()
store()
delete()
update()
$iterate()
write()
read()
copy of
<<Business Object>>
Reservation
title : ObjId
borrower : ObjId
Reservation()
getTitle()
getBorrower()
write()
read()
0..n
refer to / may be
reserved in a
0..n
<<Business Object>>
Title
name : String
author : String
isbn : String
type : int
items : ObjId[]
reservations : ObjId[]
Title()
$findOnName()
$findOnAuthor()
$findOnISBN()
$iterateTitle()
$getTitle()
getTitle()
getAuthor()
getISBN()
getType()
getTypeAsString()
setTitle()
setAuthor()
setISBN()
setType()
getNoItems()
addItem()
removeItemAtIndex()
getItem()
getItemWithId()
getNoReservations()
getReservation()
addReservation()
removeReservation()
write()
read()

Thiết kế

- Thiết kế chi tiết
  - Gói UI
    - Là gói trên trên đỉnh các gói khác
    - Trình diễn thông tin cho người sử dụng và nhập dữ liệu cho hệ thống
    - Các lớp của gói này xây dựng trên cơ sở thư viện của ngôn ngữ lập trình bậc cao hay công cụ thương mại
    - Mô hình động (thí dụ biểu đồ trình tự) của thiết kế được đặt trong gói này
      - Trên cơ sở biểu đồ trình tự khi phân tích UC -> chi tiết hơn
    - Có thể sử dụng biểu đồ cộng tác để thiết kế giao diện

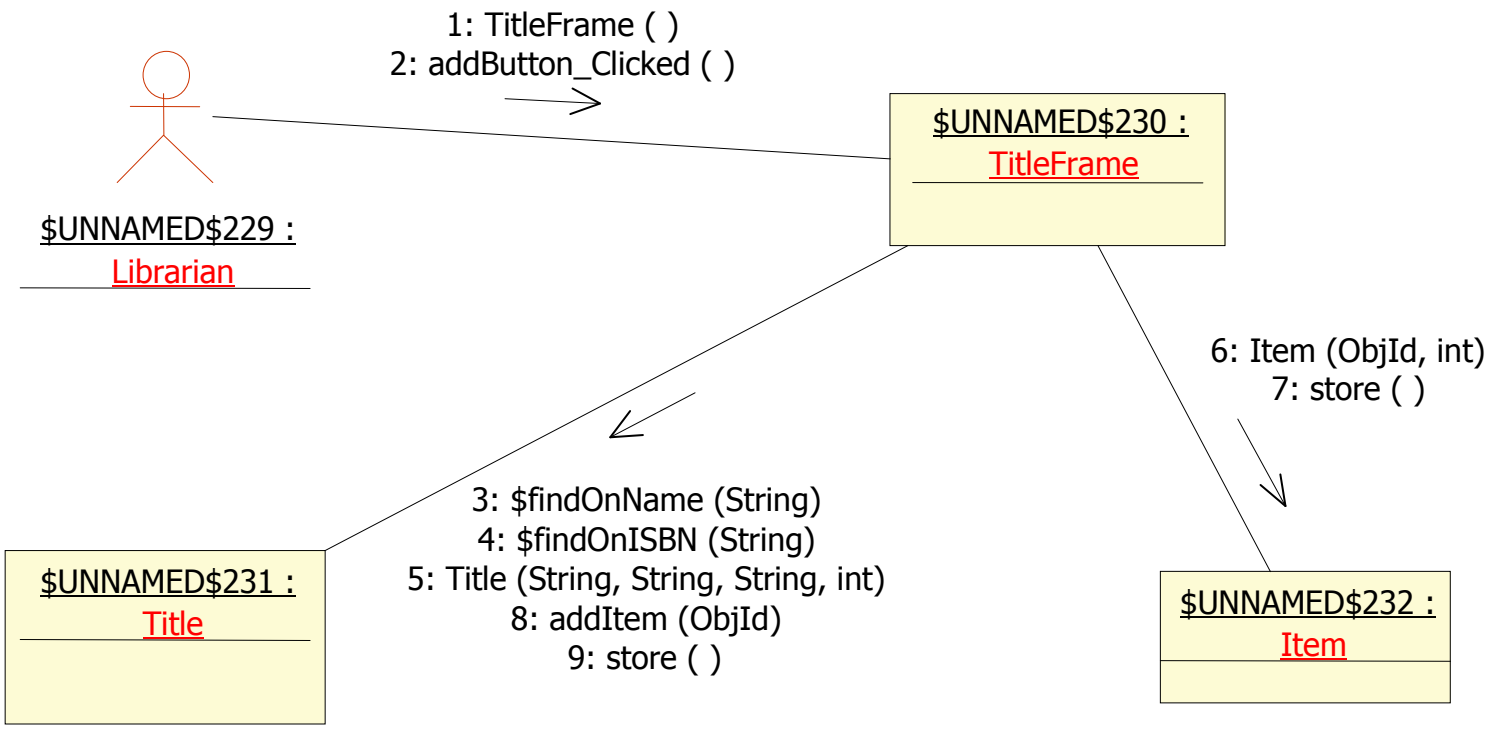
Thiết kế

- Thiết kế chi tiết- Gói UI – Biểu đồ trình tự Add Title



Thiết kế

- Thiết kế chi tiết- Gói UI – Biểu đồ cộng tác Add Title

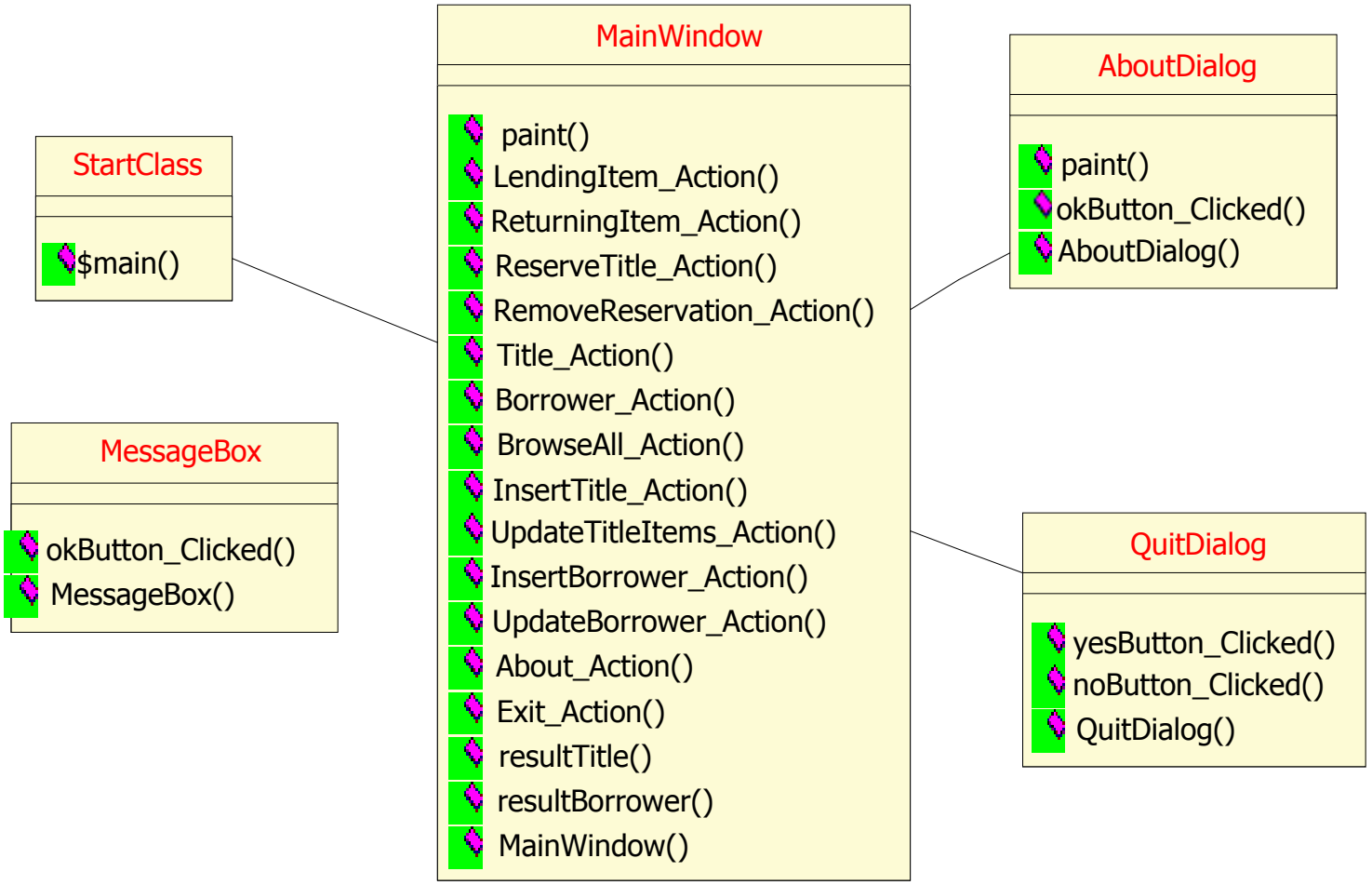


Thiết kế

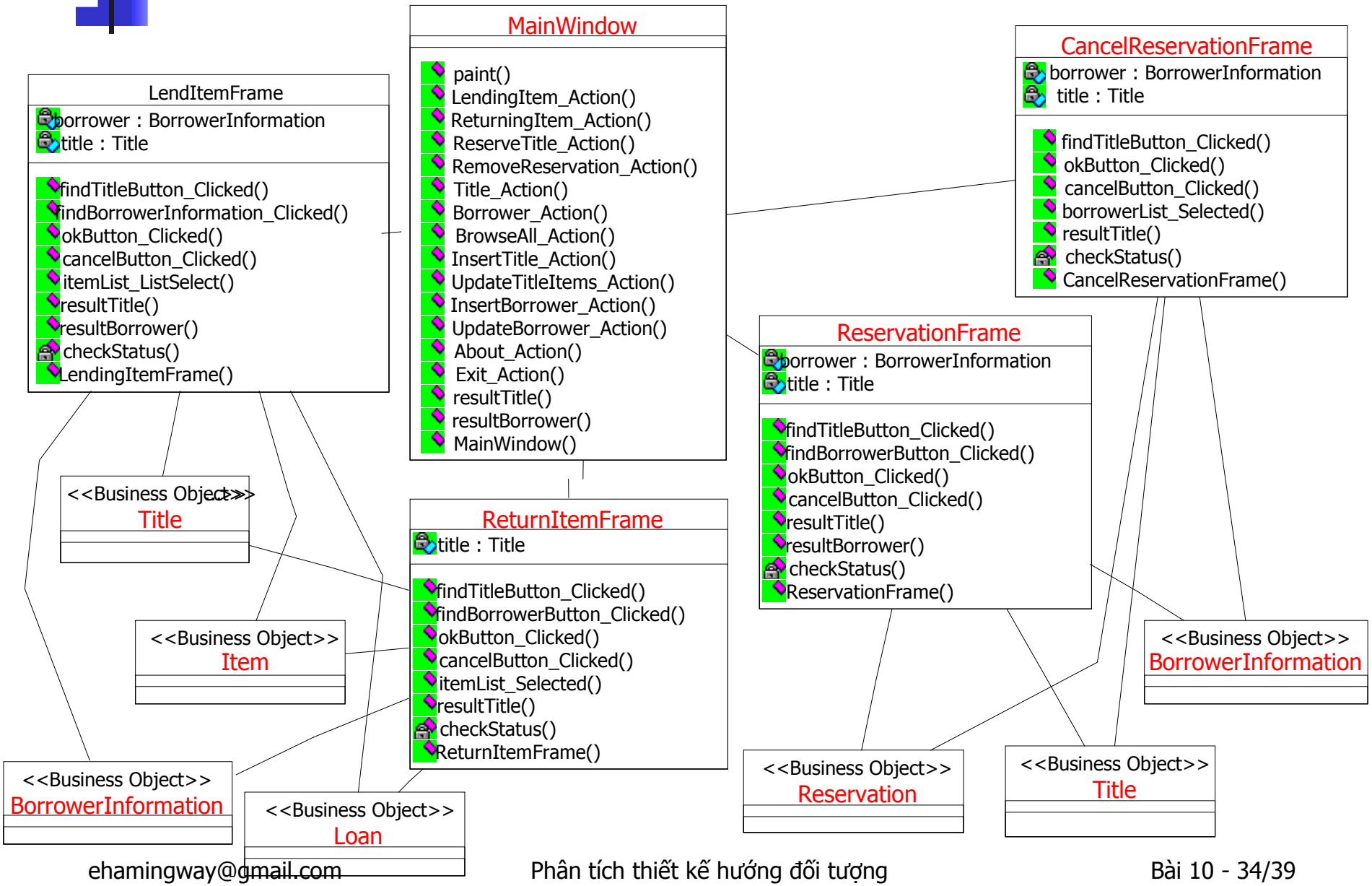
- Thiết kế giao diện người sử dụng
  - Có thể thực hiện riêng rẽ nhưng song song với các công việc khác trong pha phân tích
  - Dựa trên cơ sở các UC, hệ thống được chia thành các chức năng
    - Mỗi chúng có thực đơn riêng trong thực đơn chính
  - Thí dụ
    - Functions: Thiết kế các cửa sổ để thực hiện các chức năng chính của hệ thống như Cho mượn, Trả tài liệu, Đặt trước...
    - Informations: Các cửa sổ liên qaun đến trình diễn thông tin trong hệ thống như thông tin liên quan đến tên sách, độc giả
    - Maintenance: Các cửa sổ cho phép bảo trì hệ thống như cập nhật, hủy bỏ tài liệu, độc giả...
  - Có thể sử dụng IDE để thiết kế cửa sổ, bổ sung các controls...

Thiết kế UI

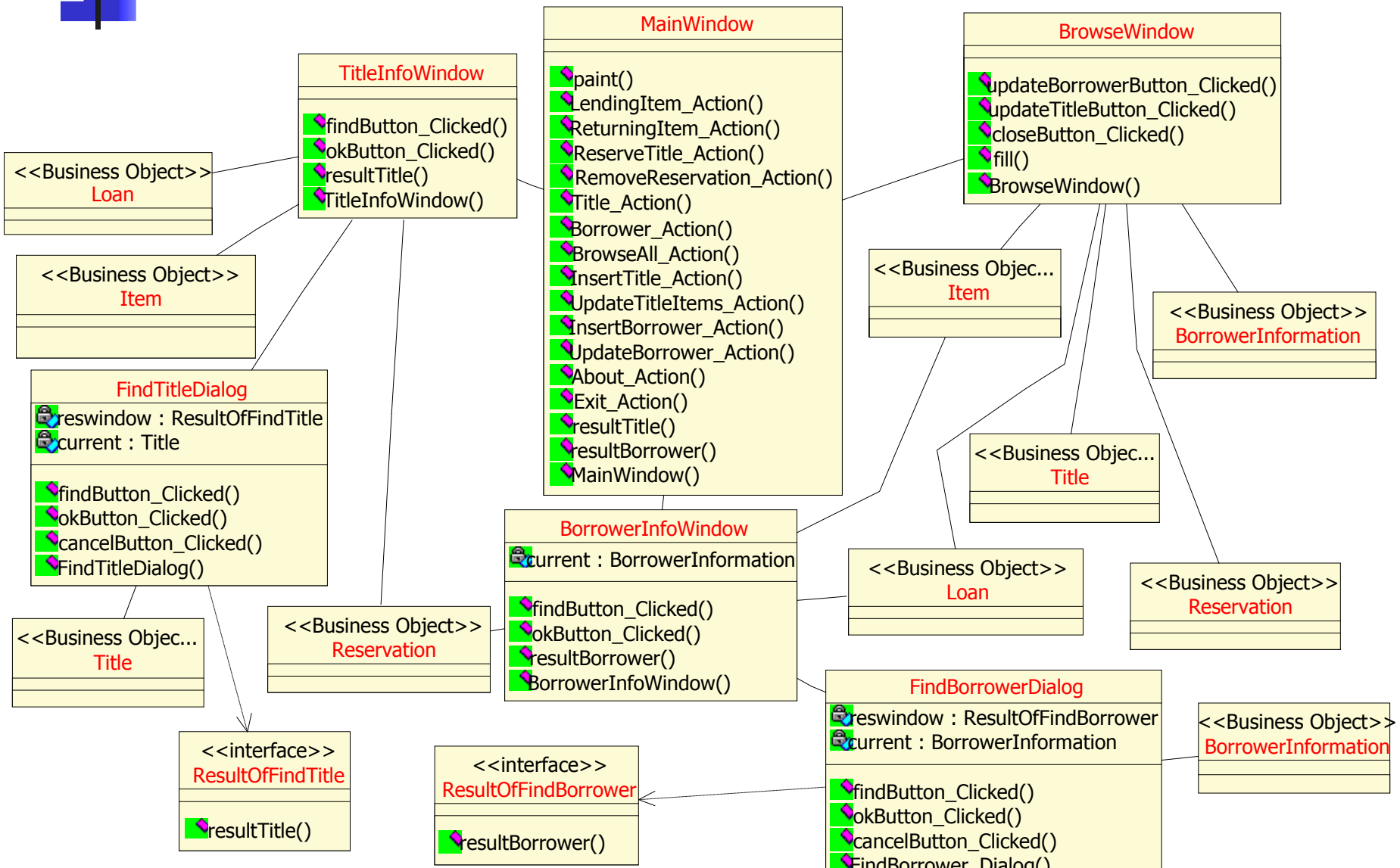
- Thiết kế giao diện người sử dụng



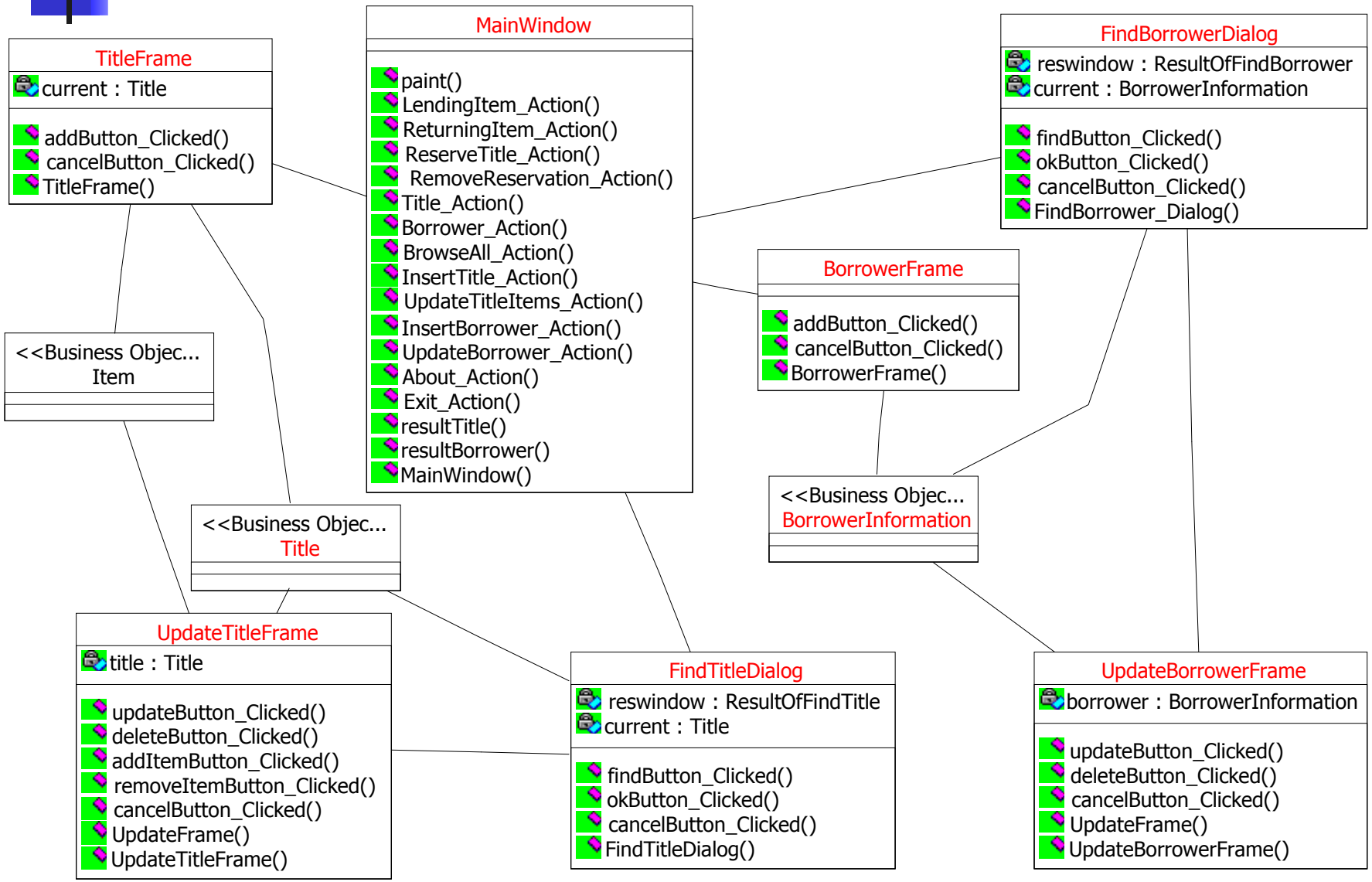
Thiết kế Gói UI/Functions



Thiết kế Gói UI/Information



Thiết kế Gói UI/Maintenance



Lập trình

- Pha xây dựng hay cài đặt là lập trình các lớp
- Ánh xạ các lớp trong mô hình thiết kế vào thành phần trong mô hình thành phần
- Ánh xạ các gói trong khung nhìn logíc vào các gói trong mô hình thành phần
  - Xâ dựng các gói trong khung nhìn thành phần
    - BO Package
    - DB Package
    - UI Package
    - Util Package
- Người phát triển có thể bổ sung các thuộc tính, thao tác mới
  - Nhưng phải giữ đồng bộ giữa mã trình và mô hình

Kiểm thử và triển khai

- Kiểm thử xem ứng dụng có hỗ trợ đầy đủ các UC?
- Kiểm thử theo nhiều cách
  - Thí dụ chuyển giao để sử dụng thử
  - Với ứng dụng lớn: kiểm thử theo đặc tả và có hệ thống thông báo lỗi đầy đủ
- Biểu đồ triển khai được hình thành trong khung nhìn triển khai

Tóm tắt

- Bài này đã xem xét một ví dụ cụ thể - Quản lý thư viện
- Trình bày các bước của một cách tiếp cận phát triển hệ thống sử dụng UML
  - Mô tả yêu cầu hệ thống
  - Phân tích
  - Thiết kế
  - Lập trình
  - Kiểm thử và triển khai