TRƯỜNG CAO ĐẲNG NGHỀ CÔNG NGHIỆP HÀ NỘI
Chủ biên: Vũ Thị Kim Phượng
Đồng tác giả: Nguyễn Thị Nhung

GIÁO TRÌNH
PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG
(Lưu hành nội bộ)

Hà Nội năm 2012

Mục lục

Chương 1. Mở đầu

1. Mô hình hướng đối tượng

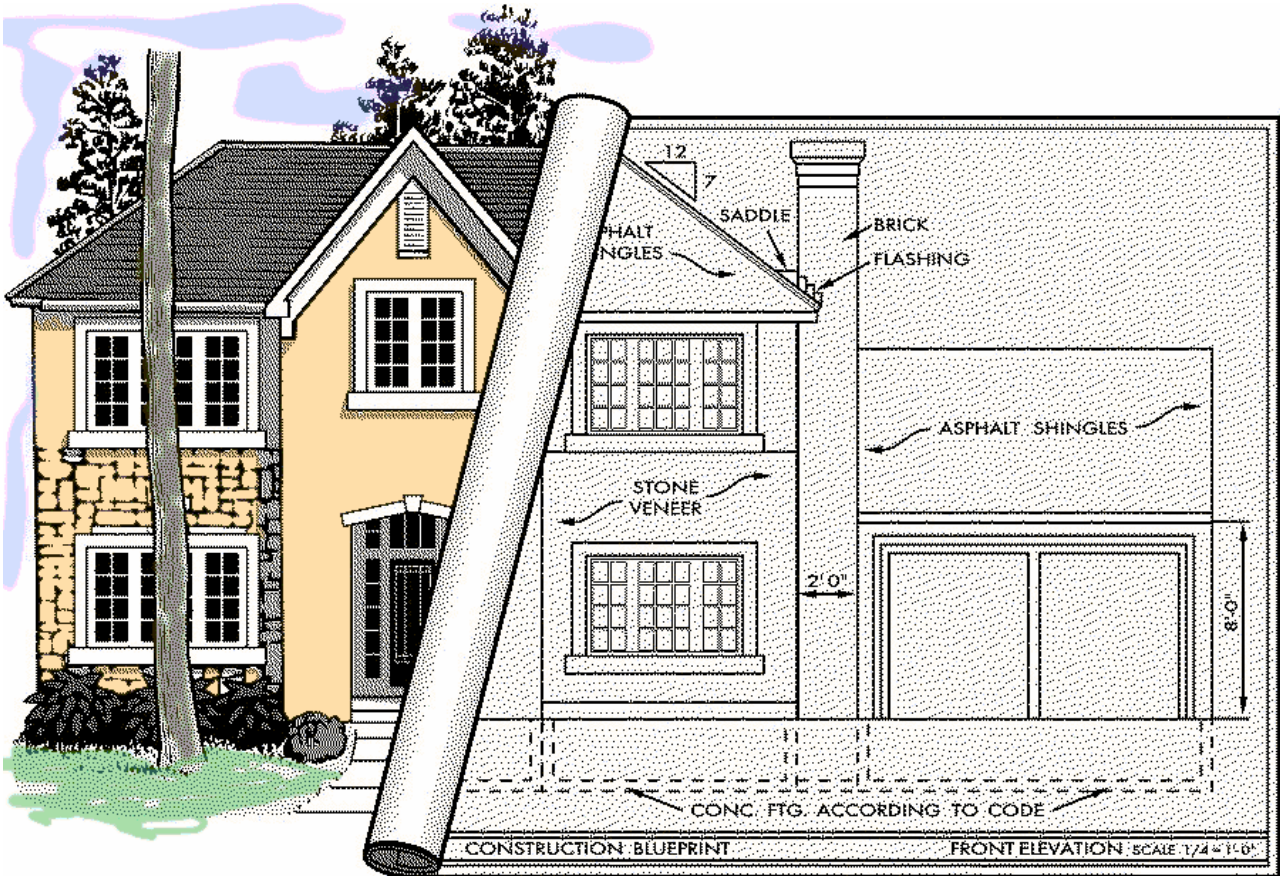
Một số ví dụ:

Ví dụ 1:

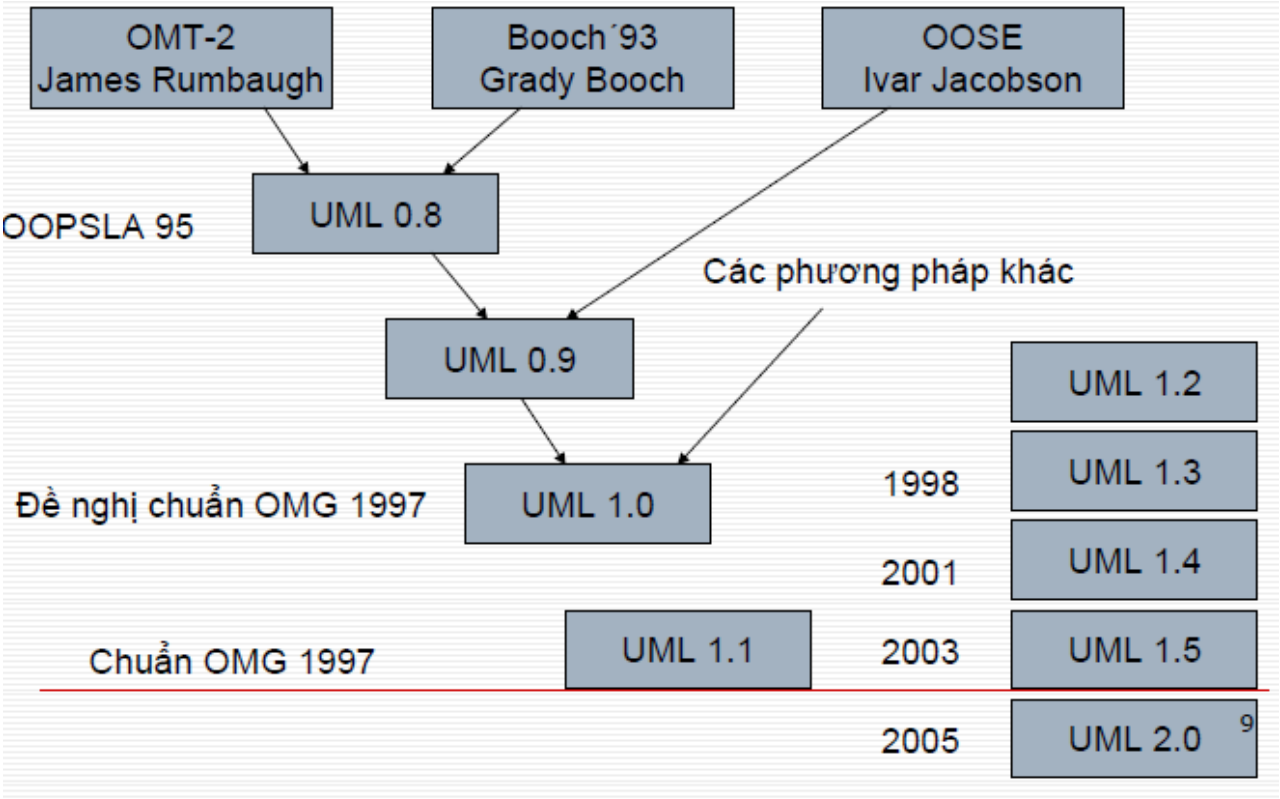
Hình ảnh thật

Mô hình: quả địa cầu sinh học

Ví dụ 2:



1.1. Lịch sử phát triển



1.2. Một số khái niệm cơ bản

Hướng đối tượng là thuật ngữ thông dụng hiện thời của ngành công nghiệp phần mềm. Các công ty đang nhanh chóng tìm cách áp dụng và tích hợp công nghệ mới này vào các ứng dụng của họ. Thật sự là đa phần các ứng dụng hiện thời đều mang tính hướng đối tượng. Nhưng hướng đối tượng có nghĩa là gì?

Lối tiếp cận hướng đối tượng là một lối tư duy về vấn đề theo lối ánh xạ các thành phần trong bài toán vào các đối tượng ngoài đời thực. Với lối tiếp cận này, chúng ta chia ứng dụng thành các thành phần nhỏ, gọi là các đối tượng, chúng tương đối độc lập với nhau. Sau đó ta có thể xây dựng ứng dụng bằng cách chắp các đối tượng đó lại với nhau. Hãy nghĩ đến trò chơi xây lâu đài bằng các mẫu gỗ. Bước đầu tiên là tạo hay mua một vài loại mẫu gỗ căn bản, từ đó tạo nên các khối xây dựng căn bản của mình. Một khi đã có các khối xây dựng đó, bạn có thể chắp ráp chúng lại với nhau để tạo lâu đài. Tương tự như vậy một khi đã xây dựng một số đối tượng căn bản trong thế giới máy tính, bạn có thể chắp chúng lại với nhau để tạo ứng dụng của mình.

Xin lấy một ví dụ đơn giản: vấn đề rút tiền mặt tại nhà băng. Các "mẫu gỗ" thành phần ở đây sẽ là ánh xạ của các đối tượng ngoài đời thực như tài khoản, nhân viên, khách hàng, …Và ứng dụng sẽ được sẽ được nhận diện cũng như giải đáp xoay quanh các đối tượng đó.

Phương pháp phân tích và thiết kế hướng đối tượng thực hiện theo các thuật ngữ và khái niệm của phạm vi lĩnh vực ứng dụng (tức là của doanh nghiệp hay đơn vị mà hệ thống tương lai cần phục vụ), nên nó tạo sự tiếp cận tương ứng giữa hệ thống và vấn đề thực ngoài đời. Trong ví dụ bán xe ô tô, mọi giai đoạn phân tích thiết kế và thực hiện đều xoay quanh các khái niệm như khách hàng, nhân viên bán hàng, xe ô tô, … Vì quá trình phát triển phần mềm đồng thời là quá trình cộng tác của khách hàng/người dùng, nhà phân tích, nhà thiết kế, nhà phát triển, chuyên gia lĩnh vực, chuyên gia kỹ thuật,... nên lối tiếp cận này khiến cho việc giao tiếp giữa họ với nhau được dễ dàng hơn.

Một trong những ưu điểm quan trọng bậc nhất của phương pháp phân tích và thiết kế hướng đối tượng là tính tái sử dụng: bạn có thể tạo các thành phần (đối tượng) một lần và dùng chúng nhiều lần sau đó. Giống như việc bạn có thể tái sử dụng các khối xây dựng (hay bản sao của nó) trong một toà lâu đài, một ngôi nhà ở, một con tàu vũ trụ, bạn cũng có thể tái sử dụng các thành phần (đối tượng) căn bản trong các thiết kế hướng đối tượng cũng như code của một hệ thống kế toán, hệ thống kiểm kê, hoặc một hệ thống đặt hàng.

Vì các đối tượng đã được thử nghiệm kỹ càng trong lần dùng trước đó, nên khả năng tái sử dụng đối tượng có tác dụng giảm thiểu lỗi và các khó khăn trong việc bảo trì, giúp tăng tốc độ thiết kế và phát triển phần mềm.

Phương pháp hướng đối tượng giúp chúng ta xử lý các vấn đề phức tạp trong phát triển phần mềm và tạo ra các thế hệ phần mềm có khả năng thích ứng và bền chắc.

1.3. Nguyên tắc mô hình hóa và quản lý độ phức tạp

Phương pháp hay phương thức (method) là một cách trực tiếp cấu trúc hoá sự suy nghĩ và hành động của con người. Phương pháp cho người sử dụng biết

phải làm gì, làm như thế nào, khi nào và tại sao (mục đích của hành động). Phương pháp chứa các mô hình (model), các mô hình được dùng để mô tả những gì sử dụng cho việc truyền đạt kết quả trong quá trình sử dụng phương pháp. Điểm khác nhau chính giữa một phương pháp và một ngôn ngữ mô hình hoá (modeling language) là ngôn ngữ mô hình hoá không có một tiến trình (process) hay các câu lệnh (instruction) mô tả những công việc người sử dụng cần làm.

Một mô hình được biểu diễn theo một ngôn ngữ mô hình hoá. Ngôn ngữ mô hình hoá bao gồm các ký hiệu – những biểu tượng được dùng trong mô hình – và một tập các quy tắc chỉ cách sử dụng chúng. Các quy tắc này bao gồm:

Syntactic (Cú pháp): cho biết hình dạng các biểu tượng và cách kết hợp chúng trong ngôn ngữ.

Semantic (Ngữ nghĩa): cho biết ý nghĩa của mỗi biểu tượng, chúng được hiểu thế nào khi nằm trong hoặc không nằm trong ngữ cảnh của các biểu tượng khác.

Pragmatic: định nghĩa ý nghĩa của biểu tượng để sao cho mục đích của mô hình được thể hiện và mọi người có thể hiểu được.

1.4. Mô hình hướng đối tượng với chu trình phát triển phần mềm

Kinh nghiệm của nhiều nhà thiết kế và phát triển cho thấy phát triển phần mềm là một bài toán phức tạp. Một số các lý do thường được kể đến:

Những người phát triển phần mềm rất khó hiểu cho đúng những gì người dùng cần.

Yêu cầu của người dùng thường thay đổi trong thời gian phát triển.

Yêu cầu thường được miêu tả bằng văn bản, dài dòng, khó hiểu, nhiều khi thậm chí mâu thuẫn.

Đội quân phát triển phần mềm, vốn là người "ngoài cuộc", rất khó nhận thức thấu đáo các mối quan hệ tiềm ẩn và phức tạp cần được thể hiện chính xác trong các ứng dụng lớn.

Khả năng nắm bắt các dữ liệu phức tạp của con người (tại cùng một thời điểm) là có hạn.

Khó định lượng chính xác hiệu suất của thành phẩm và thỏa mãn chính xác sự mong chờ từ phía người dùng.

Chọn lựa phần cứng và phần mềm thích hợp cho giải pháp là một trong những thách thức lớn đối với Designer.

Phần mềm ngoài ra cần có khả năng thích ứng và mở rộng. Phần mềm được thiết kế tốt là phần mềm đứng vững trước những biến đổi trong môi trường, dù từ phía cộng đồng người dùng hay từ phía công nghệ. Ví dụ phần mềm đã được phát triển cho một nhà băng cần có khả năng tái sử dụng cho một nhà băng khác với rất ít sửa đổi hoặc hoàn toàn không cần sửa đổi. Phần mềm thoả mãn các yêu cầu đó được coi là phần mềm có khả năng thích ứng.

Một phần mềm có khả năng mở rộng là phần mềm được thiết kế sao cho dễ phát triển theo yêu cầu của người dùng mà không cần sửa chữa nhiều.

Chính vì vậy, một số các khiếm khuyết thường gặp trong phát triển phần mềm là:

Hiểu không đúng những gì người dùng cần

Không thể thích ứng cho phù hợp với những thay đổi về yêu cầu đối với hệ thống

Các Module không khớp với nhau

Phần mềm khó bảo trì và nâng cấp, mở rộng

Phát hiện trễ các lỗ hổng của dự án

Chất lượng phần mềm kém

Hiệu năng của phần mềm thấp

Các thành viên trong nhóm không biết được ai đã thay đổi cái gì, khi nào, ở đâu, tại sao phải thay đổi.

2. Ngôn ngữ mô hình hóa thống nhất

2.1. Ngôn ngữ mô hình hóa thống nhất

Ngôn ngữ mô hình hóa thống nhất (Unifield Modeling Language – UML) là một ngôn ngữ để biểu diễn mô hình theo hướng đối tượng với chủ đích là:

Mô hình hoá các hệ thống sử dụng các khái niệm hướng đối tượng.

Thiết lập một kết nối từ nhận thức của con người đến các sự kiện cần mô hình hoá.

Giải quyết vấn đề về mức độ thừa kế trong các hệ thống phức tạp, có nhiều ràng buộc khác nhau.

Tạo một ngôn ngữ mô hình hoá có thể sử dụng được bởi người và máy.

Vì phát triển phần mềm là một bài toán khó, nên có lẽ trước hết ta cần điểm qua một số các công việc căn bản của quá trình này. Thường người ta hay tập hợp chúng theo tiến trình thời gian một cách tương đối, xoay quanh chu trình của một phần mềm, dẫn tới kết qủa khái niệm Chu Trình Phát Triển Phần Mềm (Software Development Life Cycle - SDLC) như sau:

Chu Trình Phát Triển Phần Mềm là một chuỗi các hoạt động của nhà phân tích (Analyst), nhà thiết kế (Designer), người phát triển (Developer) và người dùng (User) để phát triển và thực hiện một hệ thống thông tin. Những hoạt động này được thực hiện trong nhiều giai đọan khác nhau.

Nhà phân tích (Analyst): là người nghiên cứu yêu cầu của khách hàng/người dùng để định nghĩa một phạm vi bài toán, nhận dạng nhu cầu của một tổ chức, xác định xem nhân lực, phương pháp và công nghệ máy tính có thể làm sao để cải thiện một cách tốt nhất công tác của tổ chức này.

Nhà thiết kế (Designer): thiết kế hệ thống theo hướng cấu trúc của database, screens, forms và reports – quyết định các yêu cầu về phần cứng và phần mềm cho hệ thống cần được phát triển.

Chuyên gia lĩnh vực (Domain Experts): là những người hiểu thực chất vấn đề cùng tất cả những sự phức tạp của hệ thống cần tin học hoá. Họ không nhất thiết phải là nhà lập trình, nhưng họ có thể giúp nhà lập trình hiểu yêu cầu đặt ra đối với hệ thống cần phát triển. Quá trình phát triển phần mềm sẽ có rất nhiều thuận lợi nếu đội ngũ làm phần mềm có được sự trợ giúp của họ.

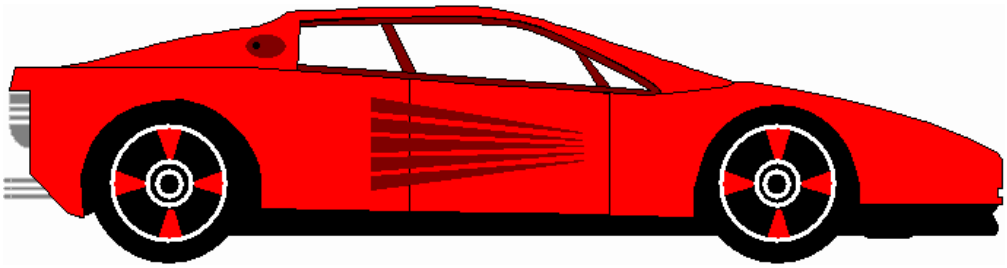
Lập trình viên (Programmer): là những người dựa trên các phân tích và thiết kế để viết chương trình (coding) cho hệ thống bằng ngôn ngữ lập trình đã được thống nhất.

Người dùng (User): là đối tượng phục vụ của hệ thống cần được phát triển.

Để cho rõ hơn, xin lấy ví dụ về một vấn đề đơn giản sau:

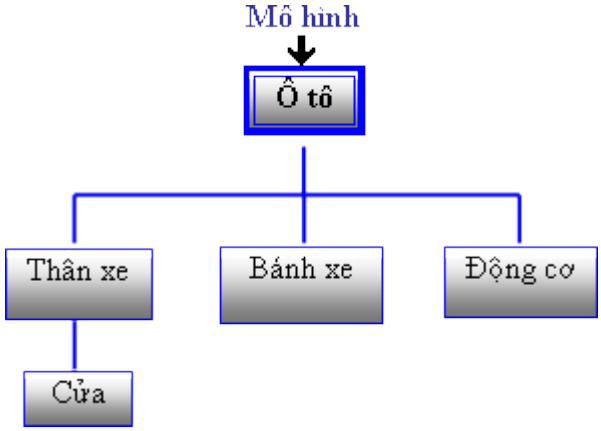
Người bình thường chúng ta khi nhìn một chiếc xe ô tô thường sẽ có một bức tranh từ bên ngoài như sau:

Vấn đề



Hình: Nhìn vấn đề ô tô của người bình thường

Chuyên gia lĩnh vực sẽ giúp nhà phân tích "trình bày lại" vấn đề như sau:



Hình: Nhìn vấn đề ô tô của chuyên gia phân tích

Chính vì sự trợ giúp của chuyên gia lĩnh vực có thể đóng vai trò rất quan trọng nên trong những giai đoạn đầu của quá trình phát triển phần mềm, kết quả phân tích nên được thể hiện sao cho dễ hiểu đối với các chuyên gia lĩnh vực. Đây cũng là môt trong rất nhiều lý do khiến cho phương pháp hướng đối tượng được nhiều người hưởng ứng.

2.2. UML và ứng dụng trong phân tích thiết kế hướng đối tượng

Phân tích hướng đối tượng (Object Oriented Analysis - OOA):

Là giai đoạn phát triển một mô hình chính xác và súc tích của vấn đề, có thành phần là các đối tượng và khái niệm đời thực, dễ hiểu đối với người sử dụng.

Trong giai đoạn OOA, vấn đề được trình bày bằng các thuật ngữ tương ứng với các đối tượng có thực. Thêm vào đó, hệ thống cần phải được định nghĩa sao cho người không chuyên Tin học có thể dễ dàng hiểu được.

Dựa trên một vấn đề có sẵn, nhà phân tích cần ánh xạ các đối tượng hay thực thể có thực như khách hàng, ô tô, người bán hàng, … vào thiết kế để tạo ra được bản thiết kế gần cận với tình huống thực. Mô hình thiết kế sẽ chứa các thực thể trong một vấn đề có thực và giữ nguyên các mẫu hình về cấu trúc, quan hệ cũng như hành vi của chúng. Nói một cách khác, sử dụng phương pháp hướng đối tượng chúng ta có thể mô hình hóa các thực thể thuộc một vấn đề có thực mà vẫn giữ được cấu trúc, quan hệ cũng như hành vi của chúng.

Đối với ví dụ một phòng bán ô tô, giai đoạn OOA sẽ nhận biết được các thực thể như:

Khách hàng

Người bán hàng

Phiếu đặt hàng

Phiếu (hoá đơn) thanh toán

Xe ô tô

Tương tác và quan hệ giữa các đối tượng trên là:

Người bán hàng dẫn khách hàng tham quan phòng trưng bày xe.

Khách hàng chọn một chiếc xe

Khách hàng viết phiếu đặt xe

Khách hàng trả tiền xe

Xe ô tô được giao đến cho khách hàng

Đối với ví dụ nhà băng lẻ, giai đoạn OOA sẽ nhận biết được các thực thể như:

Loại tài khoản: ATM (rút tiền tự động), Savings (tiết kiệm), Current (bình thường), Fixed (đầu tư),...

Khách hàng

Nhân viên

Phòng máy tính.

Tương tác và quan hệ giữa các đối tượng trên:

Một khách hàng mới mở một tài khoản tiết kiệm

Chuyển tiền từ tài khoản tiết kiệm sang tài khoản đầu tư

Chuyển tiền từ tài khoản tiết kiệm sang tài khoản ATM

Xin chú ý là ở đây, như đã nói, ta chú ý đến cả hai khía cạnh: thông tin và cách hoạt động của hệ thống (tức là những gì có thể xảy ra với những thông tin đó).

Lối phân tích bằng kiểu ánh xạ "đời thực" vào máy tính như thế thật sự là ưu điểm lớn của phương pháp hướng đối tượng.

Thiết kế hướng đối tượng (Object Oriented Design - OOD):

Là giai đoạn tổ chức chương trình thành các tập hợp đối tượng cộng tác, mỗi đối tượng trong đó là thực thể của một lớp. Các lớp là thành viên của một cây cấu trúc với mối quan hệ thừa kế.

Mục đích của giai đoạn OOD là tạo thiết kế dựa trên kết quả của giai đoạn OOA, dựa trên những quy định phi chức năng, những yêu cầu về môi trường, những yêu cầu về khả năng thực thi,.... OOD tập trung vào việc cải thiện kết quả của OOA, tối ưu hóa giải pháp đã được cung cấp trong khi vẫn đảm bảo thoả mãn tất cả các yêu cầu đã được xác lập.

Trong giai đoạn OOD, nhà thiết kế định nghĩa các chức năng, thủ tục (operations), thuộc tính (attributes) cũng như mối quan hệ của một hay nhiều lớp (class) và quyết định chúng cần phải được điều chỉnh sao cho phù hợp với môi trường phát triển. Đây cũng là giai đoạn để thiết kế ngân hàng dữ liệu và áp dụng các kỹ thuật tiêu chuẩn hóa.

Về cuối giai đoạn OOD, nhà thiết kế đưa ra một loạt các biểu đồ (diagram) khác nhau. Các biểu đồ này có thể được chia thành hai nhóm chính là Tĩnh và động. Các biểu đồ tĩnh biểu thị các lớp và đối tượng, trong khi biểu đồ động biểu thị tương tác giữa các lớp và phương thức hoạt động chính xác của chúng. Các lớp đó sau này có thể được nhóm thành các gói (Packages) tức là các đơn vị thành phần nhỏ hơn của ứng dụng.

3. Khái quát về UML

3.1. UML và các giai đoạn của chu trình phát triển phần mềm

Giai đoạn nghiên cứu sơ bộ:

UML đưa ra khái niệm **Use Case** để nắm bắt các yêu cầu của khách hàng (người sử dụng). UML sử dụng biểu đồ Use case (Use Case Diagram) để nêu bật mối quan hệ cũng như sự giao tiếp với hệ thống.

Qua phương pháp mô hình hóa Use case, các tác nhân (Actor) bên ngoài quan tâm đến hệ thống sẽ được mô hình hóa song song với chức năng mà họ đòi hỏi từ phía hệ thống (tức là Use case). Các tác nhân và các Use case được mô hình hóa cùng các mối quan hệ và được miêu tả trong biểu đồ Use case của UML. Mỗi một Use case được mô tả trong tài liệu, và nó sẽ đặc tả các yêu cầu của khách hàng: Anh ta hay chị ta chờ đợi điều gì ở phía hệ thống mà không hề để ý đến việc chức năng này sẽ được thực thi ra sao.

Giai đoạn phân tích:

Giai đoạn phân tích quan tâm đến quá trình trừu tượng hóa đầu tiên (các lớp và các đối tượng) cũng như cơ chế hiện hữu trong phạm vi vấn đề. Sau khi nhà phân tích đã nhận biết được các lớp thành phần của mô hình cũng như mối quan hệ giữa chúng với nhau, các lớp cùng các mối quan hệ đó sẽ được miêu tả bằng công cụ biểu đồ lớp (class diagram) của UML. Sự cộng tác giữa các lớp nhằm thực hiện các Use case cũng sẽ được miêu tả nhờ vào các mô hình động (dynamic models) của UML. Trong giai đoạn phân tích, chỉ duy nhất các lớp có tồn tại trong phạm vi vấn đề (các khái niệm đời thực) là được mô hình hóa. Các lớp kỹ thuật định nghĩa chi tiết cũng như giải pháp trong hệ thống phần mềm, ví dụ như các

lớp cho giao diện người dùng, cho ngân hàng dữ liệu, cho sự giao tiếp, trùng hợp, v.v..., chưa phải là mối quan tâm của giai đoạn này.

Giai đoạn thiết kế:

Trong giai đoạn này, kết quả của giai đoạn phân tích sẽ được mở rộng thành một giải pháp kỹ thuật. Các lớp mới sẽ được bổ sung để tạo thành một hạ tầng cơ sở kỹ thuật: Giao diện người dùng, các chức năng để lưu trữ các đối tượng trong ngân hàng dữ liệu, giao tiếp với các hệ thống khác, giao diện với các thiết bị ngoại vi và các máy móc khác trong hệ thống, ... Các lớp thuộc phạm vi vấn đề có từ giai đoạn phân tích sẽ được "nhúng" vào hạ tầng cơ sở kỹ thuật này, tạo ra khả năng thay đổi trong cả hai phương diện: Phạm vi vấn đề và hạ tầng cơ sở. Giai đoạn thiết kế sẽ đưa ra kết quả là bản đặc tả chi tiết cho giai đoạn xây dựng hệ thống.

Giai đoạn xây dựng:

Trong giai đoạn xây dựng (giai đoạn lập trình), các lớp của giai đoạn thiết kế sẽ được biến thành những dòng code cụ thể trong một ngôn ngữ lập trình hướng đối tượng cụ thể (không nên dùng một ngôn ngữ lập trình hướng chức năng!). Phụ thuộc vào khả năng của ngôn ngữ được sử dụng, đây có thể là một công việc khó khăn hay dễ dàng. Khi tạo ra các mô hình phân tích và thiết kế trong UML, tốt nhất nên cố gắng né tránh việc ngay lập tức biến đổi các mô hình này thành các dòng code. Trong những giai đoạn trước, mô hình được sử dụng để dễ hiểu, dễ giao tiếp và tạo nên cấu trúc của hệ thống; vì vậy, vội vàng đưa ra những kết luận về việc viết code có thể sẽ thành một trở ngại cho việc tạo ra các mô hình chính xác và đơn giản. Giai đoạn xây dựng là một giai đoạn riêng biệt, nơi các mô hình được chuyển thành code.

Thử nghiệm:

Như đã trình bày trong phần Chu Trình Phát Triển Phần Mềm, một hệ thống phần mềm thường được thử nghiệm qua nhiều giai đoạn và với nhiều nhóm thử nghiệm khác nhau. Các nhóm sử dụng nhiều loại biểu đồ UML khác nhau làm nền tảng cho công việc của mình: Thử nghiệm đơn vị sử dụng biểu đồ

lớp (class diagram) và đặc tả lớp, thử nghiệm tích hợp thường sử dụng biểu đồ thành phần (component diagram) và biểu đồ cộng tác (collaboration diagram), và giai đoạn thử nghiệm hệ thống sử dụng biểu đồ Use case (use case diagram) để đảm bảo hệ thống có phương thức hoạt động đúng như đã được định nghĩa từ ban đầu trong các biểu đồ này.

3.2. Các thành phần của UML (Hướng nhìn, Biểu đồ, Cơ chế khung, Thành phần mở rộng)

Ngôn ngữ UML bao gồm một loạt các phần tử đồ họa (graphic element) có thể được kếp hợp với nhau để tạo ra các biểu đồ. Bởi đây là một ngôn ngữ, nên UML cũng có các nguyên tắc để kết hợp các phần tử đó.

Một số những thành phần chủ yếu của ngôn ngữ UML:

Hướng nhìn (view): Hướng nhìn chỉ ra những khía cạnh khác nhau của hệ thống cần phải được mô hình hóa. Một hướng nhìn không phải là một bản vẽ, mà là một sự trừu tượng hóa bao gồm một loạt các biểu đồ khác nhau. Chỉ qua việc định nghĩa của một loạt các hướng nhìn khác nhau, mỗi hướng nhìn chỉ ra một khía cạnh riêng biệt của hệ thống, người ta mới có thể tạo dựng nên một bức tranh hoàn thiện về hệ thống. Cũng chính các hướng nhìn này nối kết ngôn ngữ mô hình hóa với quy trình được chọn cho giai đoạn phát triển.

Biểu đồ (diagram): Biểu đồ là các hình vẽ miêu tả nội dung trong một hướng nhìn. UML có tất cả 9 loại biểu đồ khác nhau được sử dụng trong những sự kết hợp khác nhau để cung cấp tất cả các hướng nhìn của một hệ thống.

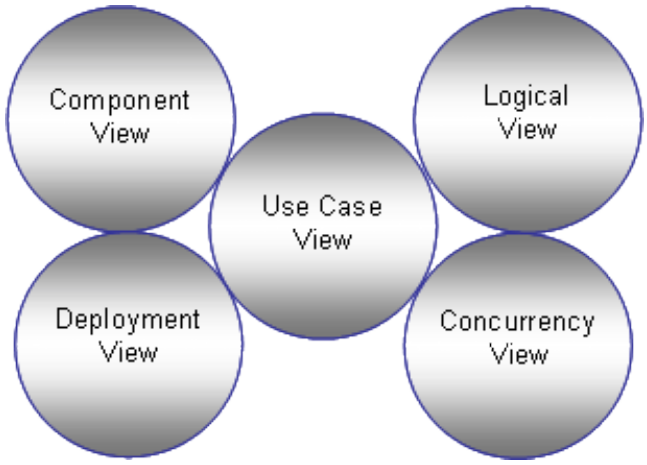
Phần tử mô hình hóa (model element): Các khái niệm được sử dụng trong các biểu đồ được gọi là các phần tử mô hình, thể hiện các khái niệm hướng đối tượng quen thuộc. Ví dụ như lớp, đối tượng, thông điệp cũng như các quan hệ giữa các khái niệm này, bao gồm cả liên kết, phụ thuộc, khái quát hóa. Một phần tử mô hình thường được sử dụng trong nhiều biểu đồ khác nhau, nhưng nó luôn luôn có chỉ một ý nghĩa và một kí hiệu.

Cơ chế chung: Cơ chế chung cung cấp thêm những lời nhận xét bổ sung, các thông tin cũng như các quy tắc ngữ pháp chung về một phần tử mô hình; chúng còn cung cấp thêm các cơ chế để có thể mở rộng ngôn ngữ UML cho phù

hợp với một phương pháp xác định (một quy trình, một tổ chức hoặc một người dùng).

Hướng nhìn (View)

Mô hình hóa một hệ thống phức tạp là một việc làm khó khăn. Lý tưởng nhất là toàn bộ hệ thống được miêu tả chỉ trong một bản vẽ, một bản vẽ định nghĩa một cách rõ ràng và mạch lạc toàn bộ hệ thống, một bản vẽ ngoài ra lại còn dễ giao tiếp và dễ hiểu. Mặc dù vậy, thường thì đây là chuyện bất khả thi. Một bản vẽ không thể nắm bắt tất cả các thông tin cần thiết để miêu tả một hệ thống. Một hệ thống cần phải được miêu tả với một loạt các khía cạnh khác nhau: Về mặt chức năng (cấu trúc tĩnh của nó cũng như các tương tác động), về mặt phi chức năng (yêu cầu về thời gian, về độ đáng tin cậy, về quá trình thực thi, v.v. và v.v.) cũng như về khía cạnh tổ chức (tổ chức làm việc, ánh xạ nó vào các code module, ...). Vì vậy một hệ thống thường được miêu tả trong một loạt các hướng nhìn khác nhau, mỗi hướng nhìn sẽ thể hiện một bức ảnh ánh xạ của toàn bộ hệ thống và chỉ ra một khía cạnh riêng của hệ thống.



Hình - Các View trong UML

Mỗi một hướng nhìn được miêu tả trong một loạt các biểu đồ, chứa đựng các thông tin nêu bật khía cạnh đặc biệt đó của hệ thống. Trong thực tế khi phân tích và thiết kế rất dễ xảy ra sự trùng lặp thông tin, cho nên một biểu đồ trên thật tế có thể là thành phần của nhiều hướng nhìn khác nhau. Khi nhìn hệ thống từ nhiều hướng nhìn khác nhau, tại một thời điểm có thể người ta chỉ tập trung vào

một khía cạnh của hệ thống. Một biểu đồ trong một hướng nhìn cụ thể nào đó cần phải đủ độ đơn giản để tạo điều kiện giao tiếp dễ dàng, để dính liền với các biểu đồ khác cũng như các hướng nhìn khác, làm sao cho bức tranh toàn cảnh của hệ thống được miêu tả bằng sự kết hợp tất cả các thông tin từ tất cả các hướng nhìn. Một biểu đồ chứa các kí hiệu hình học mô tả các phần tử mô hình của hệ thống. UML có tất cả các hướng nhìn sau:

- Hướng nhìn Use case (use case view) : đây là hướng nhìn chỉ ra khía cạnh chức năng của một hệ thống, nhìn từ hướng tác nhân bên ngoài.

- Hướng nhìn logic (logical view): chỉ ra chức năng sẽ được thiết kế bên trong hệ thống như thế nào, qua các khái niệm về cấu trúc tĩnh cũng như ứng xử động của hệ thống.

- Hướng nhìn thành phần (component view): chỉ ra khía cạnh tổ chức của các thành phần code.

- Hướng nhìn song song (concurrency view): chỉ ra sự tồn tại song song/ trùng hợp trong hệ thống, hướng đến vấn đề giao tiếp và đồng bộ hóa trong hệ thống.

- Hướng nhìn triển khai (deployment view): chỉ ra khía cạnh triển khai hệ thống vào các kiến trúc vật lý (các máy tính hay trang thiết bị được coi là trạm công tác).

Khi chọn công cụ để vẽ biểu đồ, hãy chọn công cụ nào tạo điều kiện dễ dàng chuyển từ hướng nhìn này sang hướng nhìn khác. Ngoài ra, cho mục đích quan sát một chức năng sẽ được thiết kế như thế nào, công cụ này cũng phải tạo điều kiện dễ dàng cho bạn chuyển sang hướng nhìn Use case (để xem chức năng này được miêu tả như thế nào từ phía tác nhân), hoặc chuyển sang hướng nhìn triển khai (để xem chức năng này sẽ được phân bố ra sao trong cấu trúc vật lý - Nói một cách khác là nó có thể nằm trong máy tính nào).

Ngoài các hướng nhìn kể trên, ngành công nghiệp phần mềm còn sử dụng cả các hướng nhìn khác, ví dụ hướng nhìn tĩnh-động, hướng nhìn logic-vật lý, quy trình nghiệp vụ (workflow) và các hướng nhìn khác. UML không yêu cầu chúng ta phải sử dụng các hướng nhìn này, nhưng đây cũng chính là những hướng nhìn mà

các nhà thiết kế của UML đã nghĩ tới, nên có khả năng nhiều công cụ sẽ dựa trên các hướng nhìn đó.

Hướng nhìn Use case (Use case View):

Hướng nhìn Use case miêu tả chức năng của hệ thống sẽ phải cung cấp do được tác nhân từ bên ngoài mong đợi. Tác nhân là thực thể tương tác với hệ thống; đó có thể là một người sử dụng hoặc là một hệ thống khác. Hướng nhìn Use case là hướng nhìn dành cho khách hàng, nhà thiết kế, nhà phát triển và người thử nghiệm; nó được miêu tả qua các biểu đồ Use case (use case diagram) và thỉnh thoảng cũng bao gồm cả các biểu đồ hoạt động (activity diagram). Cách sử dụng hệ thống nhìn chung sẽ được miêu tả qua một loạt các Use case trong hướng nhìn Use case, nơi mỗi một Use case là một lời miêu tả mang tính đặc thù cho một tính năng của hệ thống (có nghĩa là một chức năng được mong đợi).

Hướng nhìn Use case mang tính trung tâm, bởi nó đặt ra nội dung thúc đẩy sự phát triển các hướng nhìn khác. Mục tiêu chung của hệ thống là cung cấp các chức năng miêu tả trong hướng nhìn này – cùng với một vài các thuộc tính mang tính phi chức năng khác – vì thế hướng nhìn này có ảnh hưởng đến tất cả các hướng nhìn khác. Hướng nhìn này cũng được sử dụng để thẩm tra (verify) hệ thống qua việc thử nghiệm xem hướng nhìn Use case có đúng với mong đợi của khách hàng (Hỏi: "Đây có phải là thứ bạn muốn") cũng như có đúng với hệ thống vừa được hoàn thành (Hỏi: "Hệ thống có hoạt động như đã đặc tả?").

Hướng nhìn logic (Logical View):

Hướng nhìn logic miêu tả phương thức mà các chức năng của hệ thống sẽ được cung cấp. Chủ yếu nó được sử dụng cho các nhà thiết kế và nhà phát triển. Ngược lại với hướng nhìn Use case, hướng nhìn logic nhìn vào phía bên trong của hệ thống. Nó miêu tả kể cả cấu trúc tĩnh (lớp, đối tượng, và quan hệ) cũng như sự tương tác động sẽ xảy ra khi các đối tượng gửi thông điệp cho nhau để cung cấp chức năng đã định sẵn. Hướng nhìn logic định nghĩa các thuộc tính như trường tồn (persistency) hoặc song song (concurrency), cũng như các giao diện cũng như cấu trúc nội tại của các lớp.

Cấu trúc tĩnh được miêu tả bằng các biểu đồ lớp (class diagram) và biểu đồ đối tượng (object diagram). Quá trình mô hình hóa động được miêu tả trong các biểu đồ trạng thái (state diagram), biểu đồ trình tự (sequence diagram), biểu đồ tương tác (collaboration diagram) và biểu đồ hoạt động (activity diagram).

Hướng nhìn thành phần (Component View):

Là một lời miêu tả của việc thực thi các modul cũng như sự phụ thuộc giữa chúng với nhau. Nó thường được sử dụng cho nhà phát triển và thường bao gồm nhiều biểu đồ thành phần. Thành phần ở đây là các modul lệnh thuộc nhiều loại khác nhau, sẽ được chỉ ra trong biểu đồ cùng với cấu trúc cũng như sự phụ thuộc của chúng. Các thông tin bổ sung về các thành phần, ví dụ như vị trí của tài nguyên (trách nhiệm đối với một thành phần), hoặc các thông tin quản trị khác, ví dụ như một bản báo cáo về tiến trình của công việc cũng có thể được bổ sung vào đây.

Hướng nhìn song song (Concurrency View):

Hướng nhìn song song nhắm tới sự chia hệ thống thành các qui trình (process) và các bộ xử lý (processor). Khía cạnh này, vốn là một thuộc tính phi chức năng của hệ thống, cho phép chúng ta sử dụng một cách hữu hiệu các nguồn tài nguyên, thực thi song song, cũng như xử lý các sự kiện không đồng bộ từ môi trường. Bên cạnh việc chia hệ thống thành các tiểu trình có thể được thực thi song song, hướng nhìn này cũng phải quan tâm đến vấn đề giao tiếp và đồng bộ hóa các tiểu trình đó.

Hướng nhìn song song giành cho nhà phát triển và người tích hợp hệ thống, nó bao gồm các biểu đồ động (trạng thái, trình tự, tương tác và hoạt động) cùng các biểu đồ thực thi (biểu đồ thành phần và biểu đồ triển khai).

Hướng nhìn triển khai (Deployment View):

Cuối cùng, hướng nhìn triển khai chỉ cho chúng ta sơ đồ triển khai về mặt vật lý của hệ thống, ví dụ như các máy tính cũng như các máy móc và sự liên kết giữa chúng với nhau. Hướng nhìn triển khai giành cho các nhà phát triển, người tích hợp cũng như người thử nghiệm hệ thống và được thể hiện bằng các biểu đồ triển khai. Hướng nhìn này cũng bao gồm sự ánh xạ các thành phần của hệ thống

vào cấu trúc vật lý; ví dụ như chương trình nào hay đối tượng nào sẽ được thực thi trên máy tính nào.

Biểu đồ (diagram)

Biểu đồ là các hình vẽ bao gồm các ký hiệu phần tử mô hình hóa được sắp xếp để minh họa một thành phần cụ thể hay một khía cạnh cụ thể của hệ thống. Một mô hình hệ thống thường có nhiều loại biểu đồ, mỗi loại có nhiều biểu đồ khác nhau. Một biểu đồ là một thành phần của một hướng nhìn cụ thể; và khi được vẽ ra, nó thường thường cũng được xếp vào một hướng nhìn. Mặt khác, một số loại biểu đồ có thể là thành phần của nhiều hướng nhìn khác nhau, tùy thuộc vào nội dung của biểu đồ.

Phần sau miêu tả các khái niệm căn bản nằm đằng sau mỗi loại biểu đồ. Tất cả các chi tiết về biểu đồ, ngữ cảnh của chúng, ý nghĩa chính xác của chúng và sự tương tác giữa chúng với nhau được miêu tả chi tiết trong các chương sau (mô hình đối tượng – mô hình động). Các biểu đồ lấy làm ví dụ ở đây được lấy ra từ nhiều loại hệ thống khác nhau để chỉ ra nét phong phú và khả năng áp dụng rộng khắp của ULM.

Chương 2: Rational Rose – Công cụ hỗ trợ ngôn ngữ UML

1. Giao diện đồ họa

Rose hỗ trợ để xây dựng các biểu đồ UML mô hình hoá các lớp, các thành phần và mối quan hệ của chúng trong hệ thống một cách trực quan và thống nhất.

Nó cho phép mô tả chi tiết hệ thống bao gồm những cái gì, trao đổi tương tác với nhau và hoạt động như thế nào để người phát triển hệ thống có thể sử dụng mô hình như kế hoạch chi tiết cho việc xây dựng hệ thống.

Rose còn hỗ trợ rất tốt trong giao tiếp với khách hàng và làm hồ sơ, tài liệu cho từng phần tử trong mô hình.

Rose hỗ trợ cho việc kiểm tra tính đúng đắn của mô hình, thực hiện việc chuyển bản thiết kế chi tiết sang mã chương trình trong một ngôn ngữ lập trình lựa chọn và ngược lại, mã chương trình có thể chuyển trở lại yêu cầu hệ thống. Rose hỗ trợ phát sinh mã khung chương trình trong nhiều ngôn ngữ lập trình khác nhau như: C++, Java, Visual Basic, Oracle 8, v.v.

Ngoài ra Rose hỗ trợ cho các nhà phân tích, thiết kế và phát triển phần mềm:

Tổ chức mô hình hệ thống thành một hay nhiều tệp, được gọi là đơn vị điều khiển được. Cho phép phát triển song song các đơn thể điều khiển được của mô hình,

Hỗ trợ mô hình dịch vụ nhiều tầng (ba tầng) và mô hình phân tán, cơ chế khách/chủ (Client/Server).

Cho phép sao chép hay chuyển dịch các tệp mô hình, các đơn vị điều khiển được giữa các không gian làm việc khác nhau theo cơ chế "ánh xạ đường dẫn ảo" (Virtual Path Map),

Cho phép quản lý mô hình và tích hợp với những hệ thống điều khiển chuẩn, Rose cung cấp khả năng tích hợp với ClearCase và Microsoft Visual SourceSafe, v.v.

Sử dụng các bộ tích hợp mô hình (Model Integator) để so sánh và kết hợp các mô hình, các đơn vị điều khiển được với nhau.

Bản thân UML không định nghĩa quá trình phát triển phần mềm, nhưng UML và Rose hỗ trợ rất hiệu quả trong cả quá trình xây dựng phần mềm.

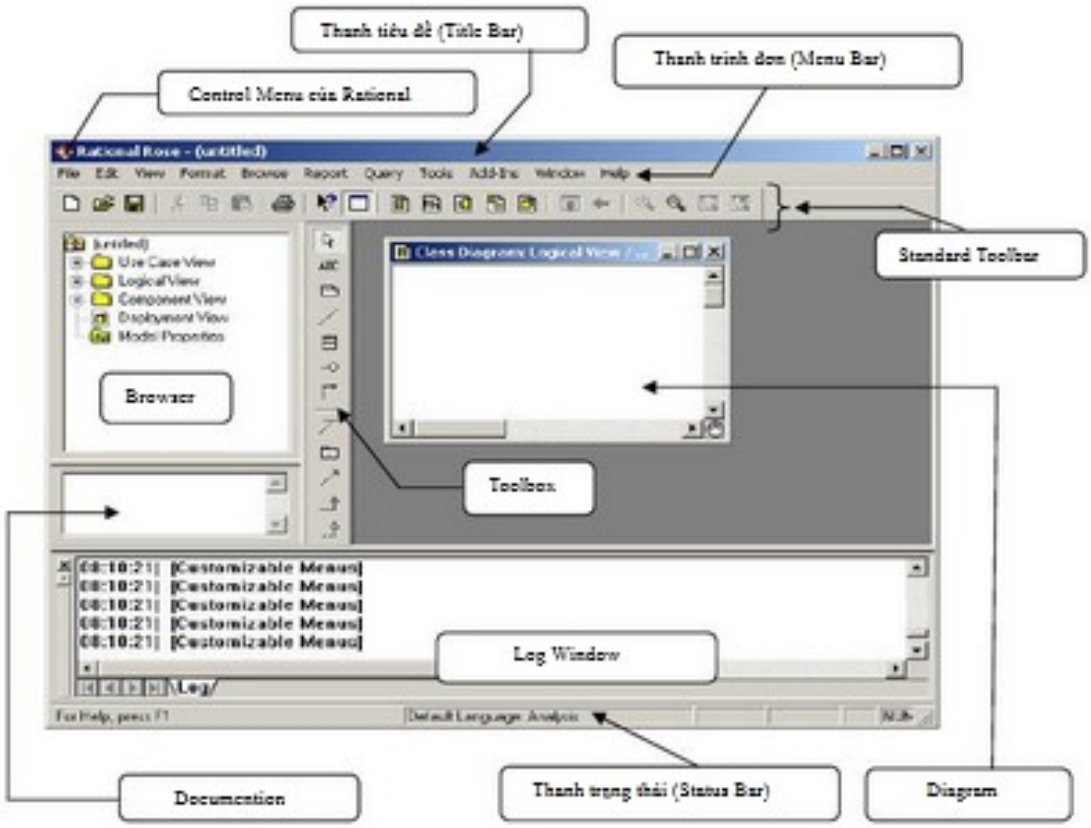
Có ba phiên bản khác nhau của Rose :

+ Rose Modeler: cho phép bạn tạo mô hình cho hệ thống, nhưng không hỗ trợ tiến trình phát sinh mã hoặc thiết kế kỹ thuật đảo ngược.

+ Rose Professional: cho phép phát sinh mã trong một ngôn ngữ

+ Rose Enterprise: cho phép phát sinh mã cho C++, Java, Ada, Corba, Visual Basic, Oracle … Một mô hình có thể có các thành phần được phát sinh bằng các ngôn ngữ khác nhau.

Giao diện chính của chương trình:



Dòng trên cùng gọi là Thanh tiêu đề (Title Bar) ở đó có tên của ứng dụng là Rational Rose. Bên trái có biểu tượng, là hộp Application Control Box của Rose, khi nhắp chuột vào đó sẽ xuất hiện.

Control Menu (hình bên cạnh), nếu Double Click vào biểu tượng này hoặc Close hoặc Alt+F4, sẽ kết thúc Rose.

Dòng thứ hai gọi là Menu Bar (Thanh trình đơn) gồm các mục từ File đến Help và sẽ được kích hoạt bằng phím Alt nếu dùng bàn phím.

Dòng thứ ba là Standard Toolbar (Thanh công cụ chuẩn) chứa biểu tượng của các lệnh thường dùng.

1.1. Trình duyệt

- Dùng để nhanh chóng điều hướng qua mô hình.

- Là một cấu trúc phân cấp mà bạn có thể dùng để điều hướng qua mô hình Rose. Mọi thứ bổ sung vào mô hình : các lớp, các thành phần đều hiển thị trong trình duyệt.

- Dùng trình duyệt, có thể :

+ Bổ sung các phần tử mô hình (các lớp, các thành phần, các sơ đồ…)
+ Xem các phần tử mô hình hiện có.
+ Xem các mối liên hệ hiện có giữa các phần tử mô hình.
+ Dời các phần tử mô hình.
+ Đặt tên lại các phần tử mô hình.
+ Bổ sung một phần tử mô hình vào một sơ đồ.
+ Mở một sơ đồ.
+ …

- Có bốn kiểu xem trong trình duyệt : Use Case View, Logical View, Component View và Deployment View.

-Trình duyệt được tổ chức theo kiểu xem hình cây. Mỗi phần tử mô hình có thể chứa các phần tử khác bên dưới nó trong hệ phân cấp.

- Mặc định, trình duyệt sẽ xuất hiện trong vùng bên trái của màn hình, có thể di chuyển đến một nơi khác hoặc ẩn trình duyệt.

- Để ẩn hoặc hiện trình duyệt cần:

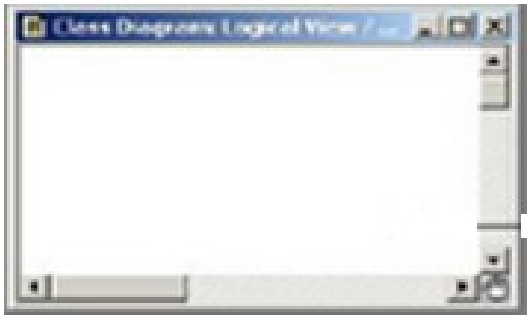
+ Nhắp phải chuột trong cửa sổ trình duyệt.

+ Chọn Hide -> trình duyệt sẽ ẩn đi.

+ Muốn hiện lại, chọn View từ thanh trình đơn, chọn Browser.

1.2. Cửa sổ soạn thảo

Cửa sổ soạn thảo có dạng như hình vẽ:



Cửa sổ này là nơi tạo lập, sửa đổi văn bản để gắn vào phần tử mô hình (tác nhân, UC, quan hệ, thuộc tính, thao tác, thành phần, nút).

Để tạo tài liệu cho mô hình ta làm như sau: chọn phần tử (click chuột trên phần tử), nhập tài liệu vào cửa sổ tài liệu. Cửa sổ tài liệu cũng tắt/mở, trôi nổi hay bám dính như cửa sổ Browser.

1.3. Các công cụ

Biểu tượng	Tên gọi	Chức năng

	Create New Model	Tạo một tập tin mô hình mới
	Open Existing Model	Mở một tập tin mô hình hiện có
	Save Model Or Log	Lưu tập tin mô hình
	Cut	Cắt rời văn bản ra Clipboard
	Copy	Chép văn bản ra Clipboard
	Paste	Dán văn bản từ Clipboard
	Print Diagrams	In một hay nhiều sơ đồ từ mô hình hiện hành

	Context Sensitive Help	Truy cập tập tin trợ giúp
	View Documentation	Xem cửa sổ Documentation
	Browse Class Diagram	Định vị và mở sơ đồ Class
	Browse Interaction Diagram	Định vị và mở sơ đồ Sequence hoặc Collaboration
	Browse Componet Diagram	Định vị và mở sơ đồ Component
	Browse state Machine Diagram	Định vị và mở sơ đồ Statechart hoặc Activity
	Browse Deployment Diagram	Mở sơ đồ Deployment của mô hình

	Browse Parent	Mở sơ đồ cha của một sơ đồ.
	Browse Previous	Mở sơ đồ vừa xem
	Zoom In	Tăng độ Zoom
	Zoom out	Giảm độ Zoom
	Fit In Windows	Ấn định độ Zoom để nguyên cả sơ đồ vừa lọt trong cửa sổ
	Undo Fit In Window	Thôi lệnh Fit In Windows

1.4. Cửa sổ biểu đồ (Diagram Window)

Cửa sổ biểu đồ là nơi cho phép ta tạo lập và sửa đổi khung nhìn đồ họa mô hình hiện hành.

Mỗi biểu tượng trong biểu đồ biểu diễn một thành phần mô hình hóa khác nhau. Cửa sổ biểu đồ xuất hiện khi nhấp đúp chuột trên cửa sổ biểu đồ trong cửa sổ Browser.

1.5. Log

Khi làm việc trên mô hình Rose, một số thông tin sẽ được đăng vào cửa sổ theo dõi. Ví dụ khi phát sinh mã, các lỗi phát sinh sẽ được đăng trong cửa sổ theo dõi.

2. Các hướng nhìn (View) trong mô hình Rose

2. 1. Hướng nhìn trường hợp sử dụng (User Cases)

Use Case View : bao gồm tất cả các sơ đồ Use Case trong hệ thống. Nó cũng có thể gộp vài sơ đồ Sequence và Collaboration. Use Case View là một cách nhìn hệ thống độc lập với thực thi. Nó tập trung vào bức tranh tổng thể về nội dung thực hiện của hệ thống, không quan tâm đến chi tiết về cách thực hiện nó.

Các thành phần của Use Case View :

Business Actors

Business Workers

Business Use Cases

Business Use Cases Diagrams

Actors

Use Cases

Use Case Diagrams

Activity Diagrams

Sequence Diagrams

Collaboration Diagrams

Packages

…

2.2. Hướng nhìn logic (Logicals)

Logical View : tập trung vào cách hệ thống thực thi cách ứng xử trong các tác vụ. Nó cung cấp bức tranh chi tiết về các mẫu hệ thống, mô tả tính tương quan giữa các mẫu với nhau. Logical View bao gồm các lớp cụ thể cần thiết, các sơ đồ Class …

Các thành phần của Logical View:

Classes

Class Diagrams

Sequence Diagrams

Collaboration Diagrams

Statechart Diagrams

Packages

…

2.3. Hướng nhìn thành phần (Components)

Component View : chứa đựng thông tin về các tập tin thi hành, các thư viện thời gian chạy, các thành phần khác trong mô hình …

Các thành phần của Component View:

Components

Component Diagrams

Packages

2.4. Hướng nhìn triển khai (Deployments)

Deployment View: liên quan đến tiến trình triển khai vật lý của hệ thống

Các thành phần của Deployment View :

Processes

Processors

Connectors

Devices

Deployment Diagrams

3. Làm việc với Rose

3.1. Tạo mô hình

Bước đầu tiên khi làm việc với Rose đó là tạo một mô hình. Các mô hình có thể được tạo từ đầu hoặc sử dụng một framework model hiện có (là các mô hình cài đặt sẵn trong máy cho một số ngôn ngữ như Visual Basic, Java, C++,

…). Mô hình Rose và tất cả các sơ đồ, các đối tượng, các phần tử mô hình khác được lưu trong một tập tin đơn lẻ có đuôi .mdl

– Để tạo một mô hình :

Chọn File -> New từ trình đơn, hoặc nhấn nút trên thanh công cụ chuẩn.

Nếu đã cài đặt Framework Wizard, danh sách các cơ cấu sẵn có sẽ xuất hiện (như hình dưới đây). Lựa chọn cơ cấu rồi nhấp nút OK, hoặc Cancel nếu không dùng.

3.2. Lưu mô hình

Giống như các ứng dụng khác, bạn nên tạo thói quen lưu tập tin định kỳ, Rose cũng vậy. Như đã nêu trên, nguyên cả mô hình được lưu trong một tập tin. Ngoài ra, có thể lưu sổ theo dõi (Log Window) ra một tập tin.

– Để lưu một mô hình :

+ Chọn File -> Save từ thanh trình đơn

+ Hoặc nhấn nút trên thanh công cụ chuẩn.

Giống như các ứng dụng khác, nên tạo thói quen lưu tập tin định kỳ, Rose cũng vậy. Như đã nêu trên, nguyên cả mô hình được lưu trong một tập tin. Ngoài ra, bạn có thể lưu sổ theo dõi (Log Window) ra một tập tin.

– Để lưu một mô hình :

+ Chọn File -> Save từ thanh trình đơn

+ Hoặc nhấn nút  trên thanh công cụ chuẩn.

Để lưu sổ theo dõi :

+ Lựa chọn cửa sổ theo dõi.

+ Chọn File -> Save Log As từ thanh trình đơn.

+ Nhập tên tập tin cửa sổ theo dõi.

Hoặc :

+ Lựa chọn cửa sổ theo dõi.

+ Nhấn nút  trên thanh công cụ chuẩn.

+ Nhập tên tập tin cửa sổ theo dõi.

3.3. Exporting và Importing mô hình

– Một trong những lợi ích chính của hướng đối tượng là khả năng dùng lại. Khả năng dùng lại có thể áp dụng không những cho mã mà còn cho các mô hình. Để vận dụng đầy đủ khả năng dùng lại, Rose hỗ trợ phương pháp xuất (nhập) các mô hình và các phần tử của mô hình. Bạn có thể xuất một mô hình hoặc một phần của mô hình và nhập nó vào các mô hình khác.

– Để nhập một mô hình, gói hoặc lớp :

+ Chọn File -> Import Model từ thanh trình đơn.

+ Chọn tập tin để nhập. Các kiểu tập tin được phép nhập bao gồm : model (.mdl), petal (.prl), category (.cat), subsystem (.sub).

– Để xuất một mô hình :

+ Chọn File -> Export Model từ thanh trình đơn.

+ Nhập tên của tập tin để xuất.

– Để xuất một gói các lớp :

+ Chọn gói để xuất từ một sơ đồ Class.
+ Chọn File -> Export <Packages> từ thanh trình đơn.
+ Nhập tên của tập tin để xuất.

– Để xuất một lớp :

+ Chọn lớp để xuất từ một sơ đồ Class.
+ Chọn File -> Export <Class> từ thanh trình đơn.
+ Nhập tên của tập tin để xuất.

3.4. Xuất bản mô hình lên Web

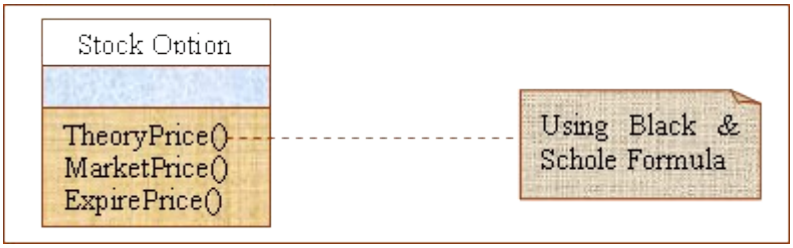
3.5. Làm việc với các đơn vị điều khiển

3.6. Sử dụng hợp nhất mô hình (Model integrator)

3.7. Làm việc với hàng chú thích

Cho dù một ngôn ngữ mô hình hóa có được mở rộng đến bao nhiêu chăng nữa, nó cũng không thể định nghĩa tất cả mọi việc. Nhằm tạo điều kiện bổ sung thêm cho một mô hình những thông tin không thể được thể hiện bằng phần tử mô hình, UML cung cấp khả năng kèm theo lời ghi chú. Một lời ghi chú có thể được để bất kỳ nơi nào trong bất kỳ biểu đồ nào, và nó có thể chứa bất kỳ loại thông tin nào. Dạng thông tin của bản thân nó là chuỗi ký tự (string), không được UML diễn giải. Lời ghi chú thường đi kèm theo một số các phần tử mô hình trong biểu đồ, được nối bằng một đường chấm chấm, chỉ ra phần tử mô hình nào được chi tiết hóa hoặc được giải thích.

Một lời ghi chú thường chứa lời nhận xét hoặc các câu hỏi của nhà tạo mô hình, ví dụ lời nhắc nhở cần phải xử lý vấn đề nào đó trong thời gian sau này. Lời ghi chú cũng có thể chứa các thông tin dạng khuôn mẫu (stereotype).



Hình - Một ví dụ về ghi chú

3.8. Làm việc với gói

Nhóm hay còn gọi là gói (backage), nó dùng để tổ chức các lớp có chức năng chung lại với nhau.

3.9. Thêm file và URLs đến Rose

3.10. Thêm và xóa biểu đồ

Để tạo biểu đồ làm theo các bước sau:

Click chuột phải lên biểu đồ cần tạo chọn New chọn Diagram.

Đặt tên cho biểu đồ mới được tạo.

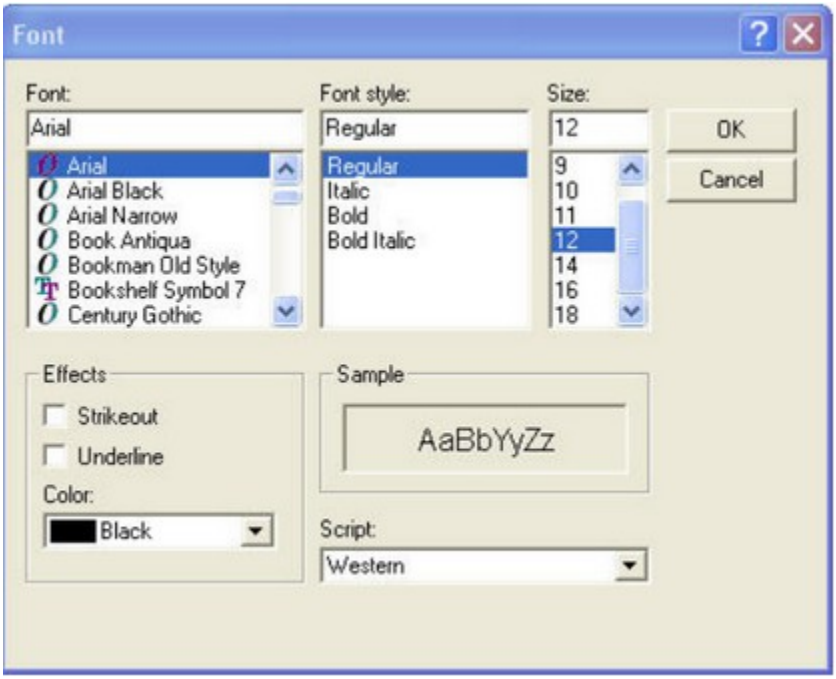
Để xóa biểu đồ, click chuột phải lên biểu đồ cần xóa chọn Delete.

3.11. Thiết lập các tùy chọn chung

Xác lập các tuỳ chọn như font chữ, màu sắc :

– Đối với font chữ :

+ Trong Rose, bạn có thể thay đổi font chữ của các đối tượng riêng lẻ trên một sơ đồ, nhờ đó cải thiện khả năng dễ đọc của mô hình. Các font chữ và cỡ chữ cũng như các thành phần liên quan nằm trong cửa sổ Font như hình dưới đây.



Để chọn một font chữ nào đó hay cỡ chữ của một đối tượng trên một sơ đồ :

Lựa chọn các đối tượng muốn dùng

Chọn Format -> Font từ thanh trình đơn. Sau đó chọn font chữ, kích cỡ … muốn dùng.

Đối với màu sắc :

+ Ngoài việc thay đổi các font chữ, cũng có thể thay đổi màu sắc riêng lẻ cho các đối tượng. Để thay đổi màu sắc đường kẻ và màu tô của một đối tượng dùng cửa sổ Color.

+ Để thay màu đường kẻ của đối tượng :

Lựa chọn đối tượng muốn thay đổi

Chọn Format -> Line Color từ thanh trình đơn

Chọn màu đường kẻ muốn dùng

+ Để thay đổi màu tô của đối tượng :

Lựa chọn đối tượng muốn thay đổi

Chọn Format -> Fill Color từ thanh trình đơn

Chọn màu tô muốn dùng.


Color
Basic colors:
Custom colors:
Define Custom Colors >>
OK
Cancel

Chương 3. Use Cases và Actors (Trường hợp sử dụng và các tác nhân)

1. Mô hình hóa các trường hợp sử dụng

1.1. Các khái niệm (Actors; User Cases; Luồng sự kiện; Quan hệ;…)

Actor (tác nhân)

- Là người dùng của hệ thống, một tác nhân có thể là một người dùng thực hoặc các hệ thống máy tính khác có vai trò nào đó trong hoạt động của hệ thống.

- Ký hiệu :

Use Case :

- Use case dùng để mô tả yêu cầu của hệ thống mới về mặt chức năng, mỗi chức năng sẽ được biểu diễn như một hoặc nhiều use case . Đây là thành phần cơ bản của biểu đồ use case. Các use case được biểu diễn bởi các hình elip. Tên các use case thể hiện một chức năng xác định của hệ thống.

- Ký hiệu :

Relationships (Mối quan hệ):

- Là các mỗi quan hệ giữa tác nhân với tác nhân, tác nhân với Use Case và Use Case với Use Case:

+ include: Use Case này sử dụng lại chức năng của Use Case kia.

+ extend: Use Case này mở rộng từ use case kia bằng cách thêm vào một chức năng cụ thể.

+ Generalization: Use Case này được kế thừa các chức năng từ Use Case kia.

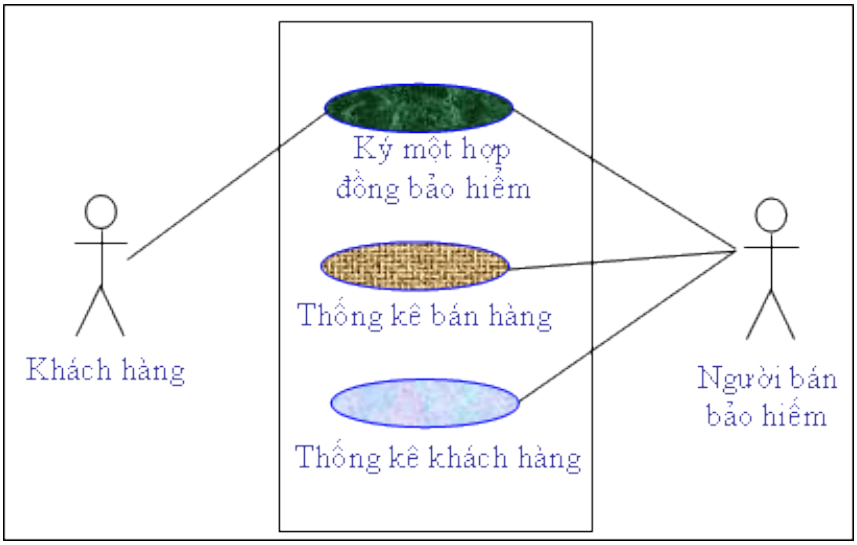
- Ký hiệu :

1.2. Phân tích các trường hợp sử dụng

1.3. Biểu đồ User Case

Biểu đồ Use case (Use Case Diagram):

Một biểu đồ Use case chỉ ra một số lượng các tác nhân ngoại cảnh và mối liên kết của chúng đối với Use case mà hệ thống cung cấp. Một Use case là một lời miêu tả của một chức năng mà hệ thống cung cấp. Lời miêu tả Use case thường là một văn bản tài liệu, nhưng kèm theo đó cũng có thể là một biểu đồ hoạt động. Các Use case được miêu tả duy nhất theo hướng nhìn từ ngoài vào của các tác nhân (hành vi của hệ thống theo như sự mong đợi của người sử dụng), không miêu tả chức năng được cung cấp sẽ hoạt động nội bộ bên trong hệ thống ra sao. Các Use case định nghĩa các yêu cầu về mặt chức năng đối với hệ thống..



Biểu đồ use case của một công ty bảo hiểm

1.4. Các ví dụ và bài tập

Nhà băng ABC đưa ra các yêu cầu sau:

Một khách hàng có thể muốn gửi tiền vào, rút tiền ra hoặc đơn giản kiểm tra lại số tiền trong tài khoản của anh ta qua máy tự động rút tiền (ATM). Khi đưa tiền vào hoặc rút tiền ra, cần phải ghi ra giấy kết quả những chuyển dịch đã thực hiện và trao tờ giấy này cho khách hàng.

Quan sát các chức năng căn bản và các thành phần tham gia, ta thấy có hai tác nhân dễ nhận ra nhất là khách hàng và nhân viên thu ngân.

Qua đó, có thể nhân dạng các Use Case sau:

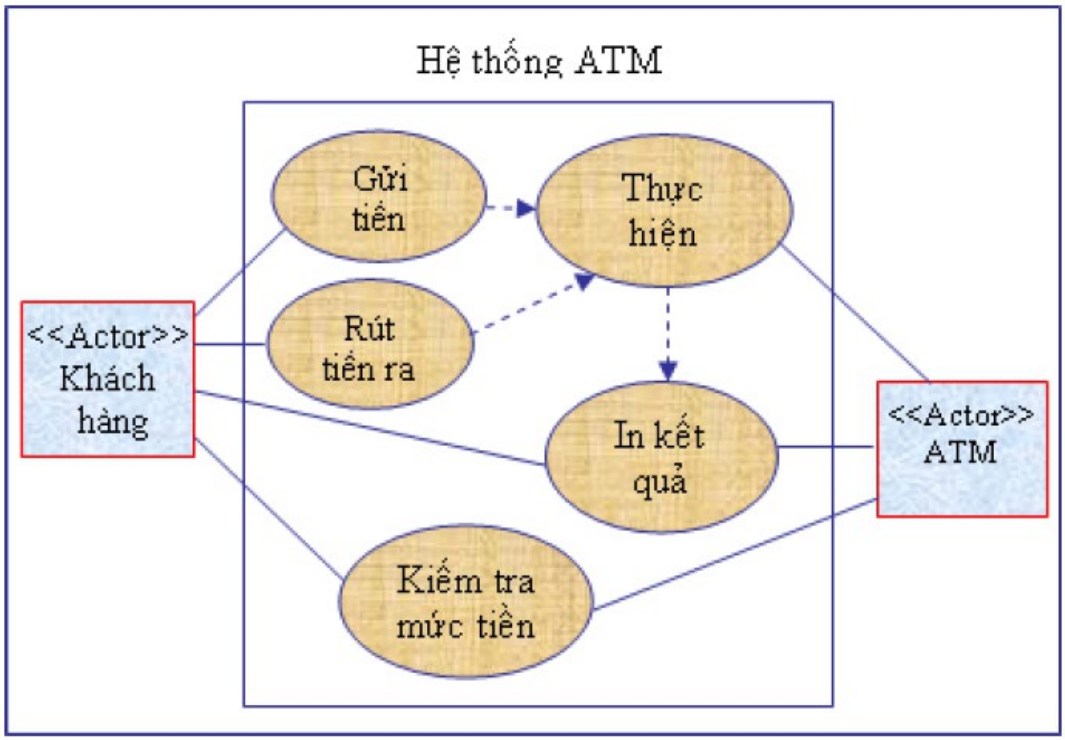
Gửi tiền vào.

Rút tiền ra.

Kiểm tra mức tiền trong tài khoản

Thực hiện các chuyển dịch nội bộ hệ thống

In kết quả các chuyển dịch đã thực hiện.



Hình– Các Use case trong hệ thống ATM

Use Case gửi tiền vào và rút tiền ra phụ thuộc vào Use Case thực hiện các chuyển dịch trong nội bộ hệ thống, việc thực hiện này về phần nó lại phụ thuộc

vào chức năng in ra các công việc đã được thực hiện. Kiểm tra mức tiền trong tài khoản là một Use Case độc lập, không phụ thuộc vào các Use Case khác.

Bài tập:

Bài 1. Một tác nhân (Actor) trong một Use Case luôn là một con người

Bài 2. Hệ thống khác cũng có thể đóng vai trò tác nhân trong một Use Case?

Bài 3. Mỗi hệ thống chỉ có một Use Case?

Bài 4. Biểu đồ Use case mô tả chức năng hệ thống?

2. Mô hình tương tác

2.1. Biểu đồ tương tác

2.2. Biểu đồ tuần tự

Biểu đồ tuần tự thể hiện một chuỗi các sự kiện, hành vi của đối tượng theo một trình tự thời gian. Nó được sử dụng để mô tả dòng thông điệp được gửi đi và và các đối tượng phối hợp nhận và xử lý để trả về kết quả mà theo yêu cầu mà thông điệp gửi đến.

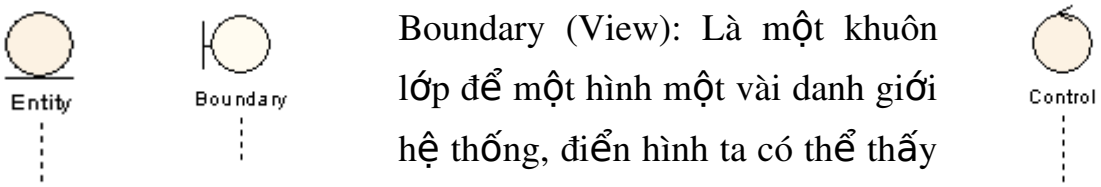
Biểu đồ tuần tự thông thường được sử dụng như một mô hình giải thích cho kịch bản usecase.

Biểu đồ tuần tự thể hiện rất rõ đối tượng nào tương tác với đối tượng nào và thông điệp là gì.

Khi đọc một biểu đồ tuần tự ta đọc từ trái qua phải và từ trên xuống dưới.

Các ký hiệu và quy tắc sử dụng:

Thành phần tham gia vào biểu đồ tuần tự có thể là Actor, các Object và một số thành phần sau:



Boundary (View): Là một khuôn lớp để một hình một vài danh giới hệ thống, điển hình ta có thể thấy Boundary là giao diện chương trình. Nó thể hiện các tương tác giữa người dùng với hệ thống ở múc độ giao diện màn hình. Nó thường được sử dụng trong biểu đồ tuần tự và nhìn nhận dưới mô hinh MVC.

Entity (Model): Là một kho (Store) để thu thập thông tin và knowledge trong hệ thống và được áp dụng trong MVC.

Controller: Là một khuôn lớp để thể hiện các control, quản lý các thực thể. Control được tổ chức và schedule cho các tương tác khác nhau với các thành phần khác nhau.

Lifelines: Thể hiện sự tồn tại của đối tượng theo thời gian (thời gian sống của đối tượng). Trong UML nó được biểu diễn bởi được đường nét rời đứng.

Activations: Thể hiện thời gian, chu trình sống của đối tượng khi thực hiện một service nào đó (thời gian mà service đó còn tồn tại). Trong UML nó được biểu diễn bằng một hình chữ nhật hẹp, đứng.

Message: Một object gửi một thông điệp đến một object khác nhờ thực hiện một service nào đó mà nó không có. Khi gửi một message có thể có tham số kèm theo và đó là knowledge (data) của object gửi.

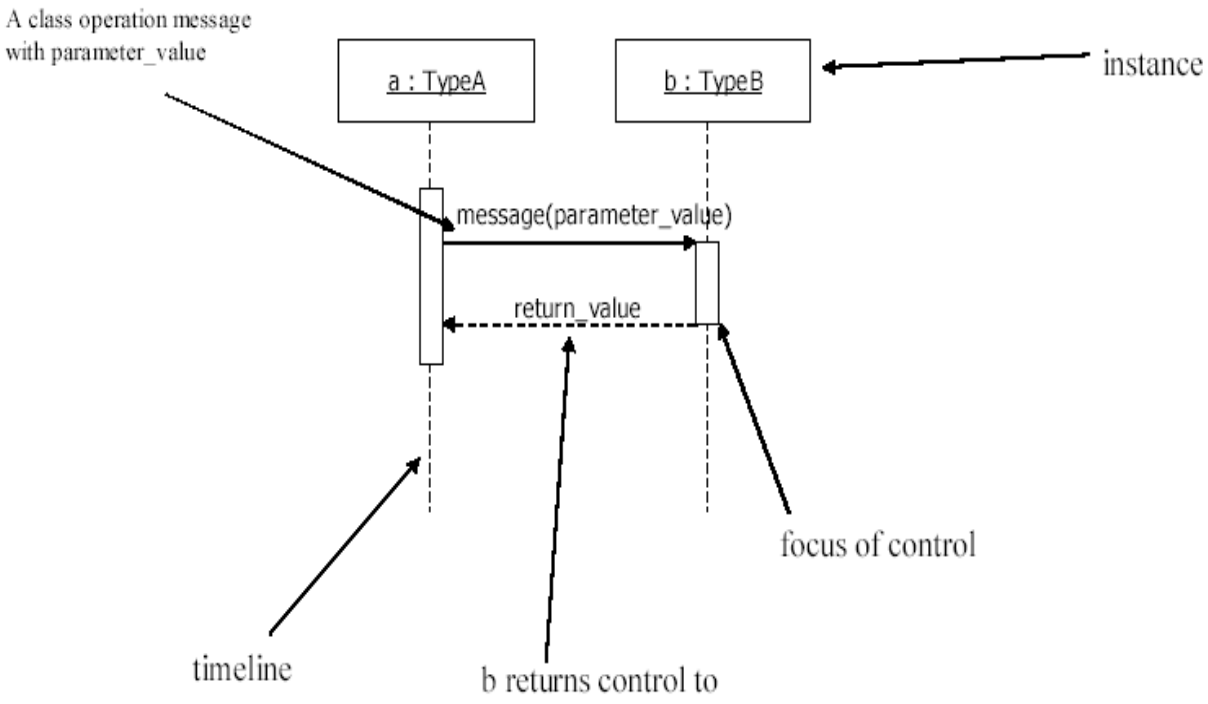
Message (Call, Procedure): gọi một phương thức cụ thể của object đích.

Self message: Self message là tự gọi một phương thức ngay tại trong lớp đó.

Return message: Là message kết quả trả về sau khi đã thực hiện một message gửi

Destroy: Kết thúc chu trình sống của đối tượng thì ta phải huỷ đối tượng.

Biểu đồ sequence(diagrams):



2.3. Biểu đồ cộng tác

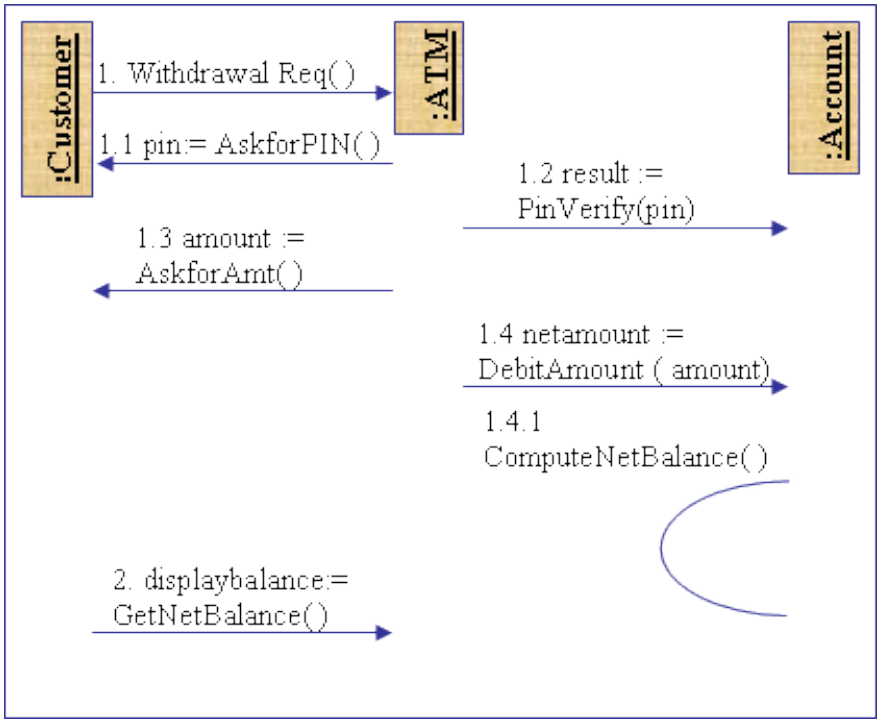
Một biểu đồ cộng tác miêu tả tương tác giữa các đối tượng cũng giống như biểu đồ tuần tự, nhưng nó tập trung trước hết vào các sự kiện, tức là tập trung chủ yếu vào sự tương tác giữa các đối tượng.

Trong một biểu đồ cộng tác, các đối tượng được biểu diễn bằng kí hiệu lớp. Thứ tự trong biểu đồ cộng tác được thể hiện bằng cách đánh số các thông điệp. Kỹ thuật đánh số được coi là hơi có phần khó hiểu hơn so với kỹ thuật mũi tên sử dụng trong biểu đồ tuần tự. Nhưng ưu điểm của biểu đồ cộng tác là nó có thể chỉ ra các chi tiết về các lệnh gọi hàm (thủ tục), yếu tố được né tránh trong biểu đồ tuần tự.

Biểu đồ sau đây là một ví dụ cho một biểu đồ cộng tác, được chuẩn bị cũng cho một cảnh kịch rút tiền mặt như trong biểu đồ tuần tự của phần trước. Hãy quan sát các thứ tự số trong biểu đồ. Đầu tiên thủ tục WithdrawalReq() được gọi từ lớp khách hàng. Đó là lệnh gọi số 1. Bước tiếp theo trong tuần tự là hàm AskForPin(), số 1.1, được gọi từ lớp ATM. Thông điệp trong biểu đồ được viết

dưới dạng pin:= AskForPin(), thể hiện rằng "giá trị trả về" của hàm này chính là mã số mà lớp khách hàng sẽ cung cấp.

Hình cung bên lớp tài khoản biểu thị rằng hàm ComputeNetBalance() được gọi trong nội bộ lớp tài khoản và nó xử lý cục bộ. Thường thì nó sẽ là một thủ tục riêng (private) của lớp.



Hình- Một biểu đồ cộng tác của kích cảnh rút tiền ở máy ATM

2.4. Làm việc với các đối tượng

2.5. Làm việc với các thông điệp

2.6. Các ví dụ và bài tập

3. Hoạt động của đối tượng

3.1. Các khái niệm

3.2. Biểu đồ hoạt động

Biểu đồ hoạt động nắm bắt hành động và các kết quả của chúng. Biểu đồ hoạt động tập trung vào công việc được thực hiện trong khi thực thi một thủ tục (hàm), các hoạt động trong một lần thực thi một trường hợp sử dụng hoặc trong một đối tượng. Biểu đồ hoạt động là một biến thể của biểu đồ trạng thái và có

một mục tiêu tương đối khác, đó là nắm bắt hành động (công việc và những hoạt động phải được thực hiện) cũng như kết quả của chúng theo sự biến đổi trạng thái. Các trạng thái trong biểu đồ hoạt động (được gọi là các trạng thái hành động) sẽ chuyển sang giai đoạn kế tiếp khi hành động trong trạng thái này đã được thực hiện xong (mà không xác định bất kỳ một sự kiện nào theo như nội dung của biểu đồ trạng thái). Một sự điểm phân biệt khác giữa biểu đồ hoạt động và biểu đồ trạng thái là các hành động của nó được định vị trong các luồng (swimlane). Một luồng sẽ gom nhóm các hoạt động, chú ý tới khái niệm người chịu trách nhiệm cho chúng hoặc chúng nằm ở đâu trong một tổ chức. Một biểu đồ hoạt động là một phương pháp bổ sung cho việc miêu tả tương tác, đi kèm với trách nhiệm thể hiện rõ các hành động xảy ra như thế nào, chúng làm gì (thay đổi trạng thái đối tượng), chúng xảy ra khi nào (chuỗi hành động), và chúng xảy ra ở đâu (luồng hành động).

Biểu đồ hoạt động có thể được sử dụng cho nhiều mục đích khác nhau, ví dụ như:

- Để nắm bắt công việc (hành động) sẽ phải được thực thi khi một thủ tục được thực hiện. Đây là tác dụng thường gặp nhất và quan trọng nhất của biểu đồ hoạt động.

- Để nắm bắt công việc nội bộ trong một đối tượng.

- Để chỉ ra một nhóm hành động liên quan có thể được thực thi ra sao, và chúng sẽ ảnh hưởng đến những đối tượng nằm xung quanh chúng như thế nào.

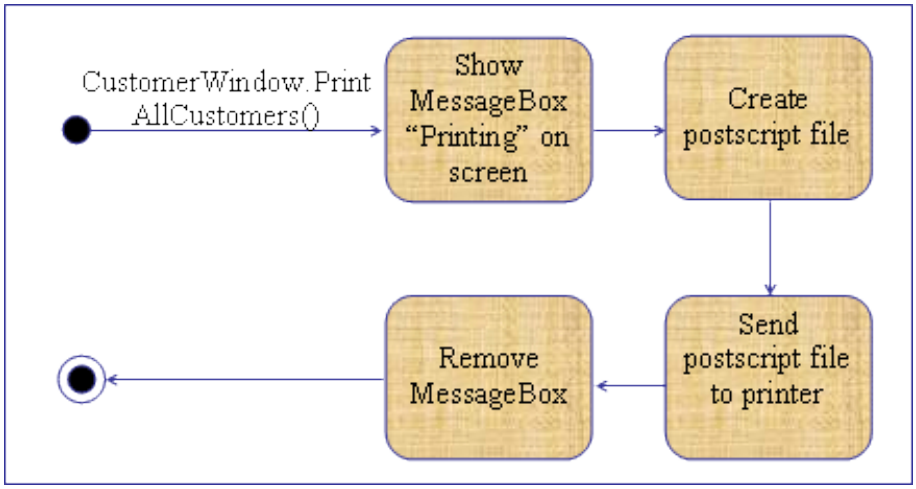
- Để chỉ ra một trường hợp sử dụng có thể được thực thể hóa như thế nào, theo khái niệm hành động và các sự biến đổi trạng thái của đối tượng.

- Để chỉ ra một doanh nghiệp hoạt động như thế nào theo các khái niệm công nhân (tác nhân), qui trình nghiệp vụ (workflow), hoặc tổ chức và đối tượng (các khía cạnh vật lý cũng như tri thức được sử dụng trong doanh nghiệp).

Biểu đồ hoạt động có thể được coi là một loại Flow chart. Điểm khác biệt là Flow Chart bình thường ra chỉ được áp dụng đối với các qui trình tuần tự, biểu đồ hoạt động có thể xử lý cả các các qui trình song song.

Hành động và sự thay đổi trạng thái

Một hành động được thực hiện để sản sinh ra một kết quả. Việc thực thi của thủ tục có thể được miêu tả dưới dạng một tập hợp của các hành động liên quan, sau này chúng sẽ được chuyển thành các dòng code thật sự. Theo như định nghĩa ở phần trước, một biểu đồ hoạt động chỉ ra các hành động cùng mối quan hệ giữa chúng và có thể có một điểm bắt đầu và một điểm kết thúc. Biểu đồ hoạt động sử dụng cũng cùng những ký hiệu như trong biểu đồ trạng thái bình thường.



Hình - Khi một người gọi tác vụ PrintAllCustomer (trong lớp CustomerWindow), các hành động khởi động. hành động đầu tiên là hiện một hộp thông báo lên màn hình; hành động thứ hai là tạo một tập tin postscript; hành động thứ ba là gởi file postscript đến máy in; và hành động thứ tư là xóa hộp thông báo trên màn hình. Các hành động được chuyển tiếp tự động; chúng xảy ra ngay khi hành động trong giai đoạn nguồn được thực hiện.

Các sự thay đổi có thể được bảo vệ bởi các điều kiện canh giữ (Guard-condition), các điều kiện này phải được thỏa mãn thì sự thay đổi mới nổ ra. Một ký hiệu hình thoi được sử dụng để thể hiện một quyết định. Ký hiệu quyết định có thể có một hoặc nhiều sự thay đổi đi vào và một hoặc nhiều sự thay đổi đi ra được dán nhãn đi kèm các điều kiện bảo vệ. Bình thường ra, một trong số các sự thay đổi đi ra bao giờ cũng được thỏa mãn (là đúng).

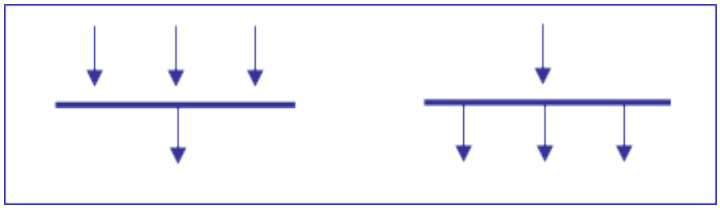
Một sự thay đổi được chia thành hai hay nhiều sự thay đổi khác sẽ dẫn đến các hành động xảy ra song song. Các hành động được thực hiện đồng thời, mặc

dù chúng cũng có thể được thực hiện lần lượt từng cái một. Yếu tố quan trọng ở đây là tất cả các thay đổi đồng thời phải được thực hiện trước khi chúng được thống nhất lại với nhau (nếu có). Một đường thẳng nằm ngang kẻ đậm (còn được gọi là thanh đồng hộ hóa – Synchronisation Bar) chỉ rằng một sự thay đổi được chia thành nhiều nhánh khác nhau và chỉ ra một sự chia sẻ thành các hành động song song. Cũng đường thẳng đó được sử dụng để chỉ ra sự thống nhất các nhánh.

Kí hiệu UML cho các thành phần căn bản của biểu đồ hoạt động:

- Hoạt động (Activity): là một qui trình được định nghĩa rõ ràng, có thể được thực thi qua một hàm hoặc một nhóm đối tượng. Hoạt động được thể hiện bằng hình chữ nhật bo tròn cạnh.

- Thanh đồng bộ hóa (Synchronisation bar): chúng cho phép ta mở ra hoặc là đóng lại các nhánh chạy song song nội bộ trong tiến trình.



Hình - Thanh đồng bộ hóa

- Điều kiện canh giữ (Guard Condition): các biểu thức logic có giá trị hoặc đúng hoặc sai. Điều kiện canh giữ được thể hiện trong ngoặc vuông, ví dụ:

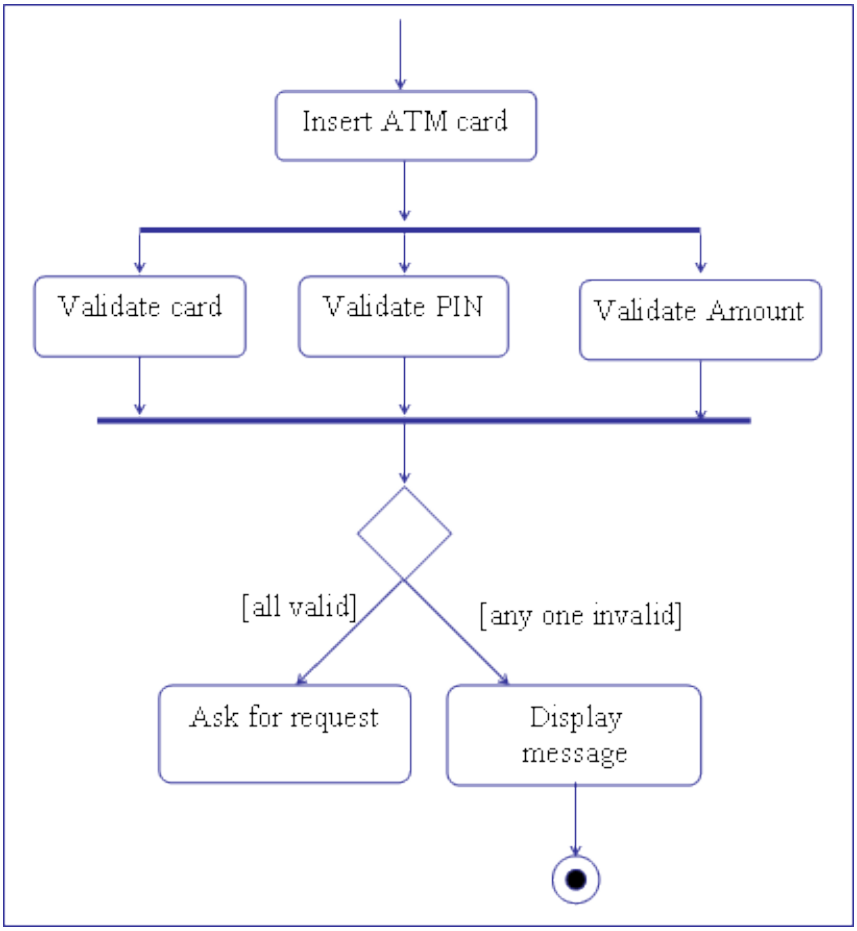
[Customer existing].

- Điểm quyết định (Decision Point): được sử dụng để chỉ ra các sự thay đổi khả thi. Kí hiệu là hình thoi.

Hình sau đây miêu tả một đoạn biểu đồ hoạt động của máy ATM. Sau khi thẻ được đưa vào máy, ta thấy có ba hoạt động song song:

- Xác nhận thẻ
- Xác nhận mã số PIN
- Xác nhận số tiền yêu cầu được rút

Chỉ khi sử dụng biểu đồ hoạt động, các hoạt động song song như vậy mới có thể được miêu tả. Mỗi một hoạt động xác nhận bản thân nó cũng đã có thể là một quá trình riêng biệt.



Hình - Biểu đồ hoạt động của máy ATM

3.3. Biểu đồ chuyển trạng thái

Biểu đồ trạng thái được sử dụng để biểu diễn các trạng thái và sự chuyển tiếp giữa các trạng thái của các đối tượng trong một lớp xác định. Thông thường, mỗi lớp sẽ có một biểu đồ trạng thái (trừ lớp trừu tượng là lớp không có đối tượng).

Biểu đồ trạng thái được biểu diễn dưới dạng máy trạng thái hữu hạn với các trạng thái và sự chuyển tiếp giữa các trạng thái đó. Có 2 dạng biểu đồ trạng thái:

Biểu đồ trạng thái cho một use case: mô tả các trạng thái và chuyển tiếp trạng thái của một đối tượng thuộc một lớp nào đó trong hoạt động của một use case cụ thể.

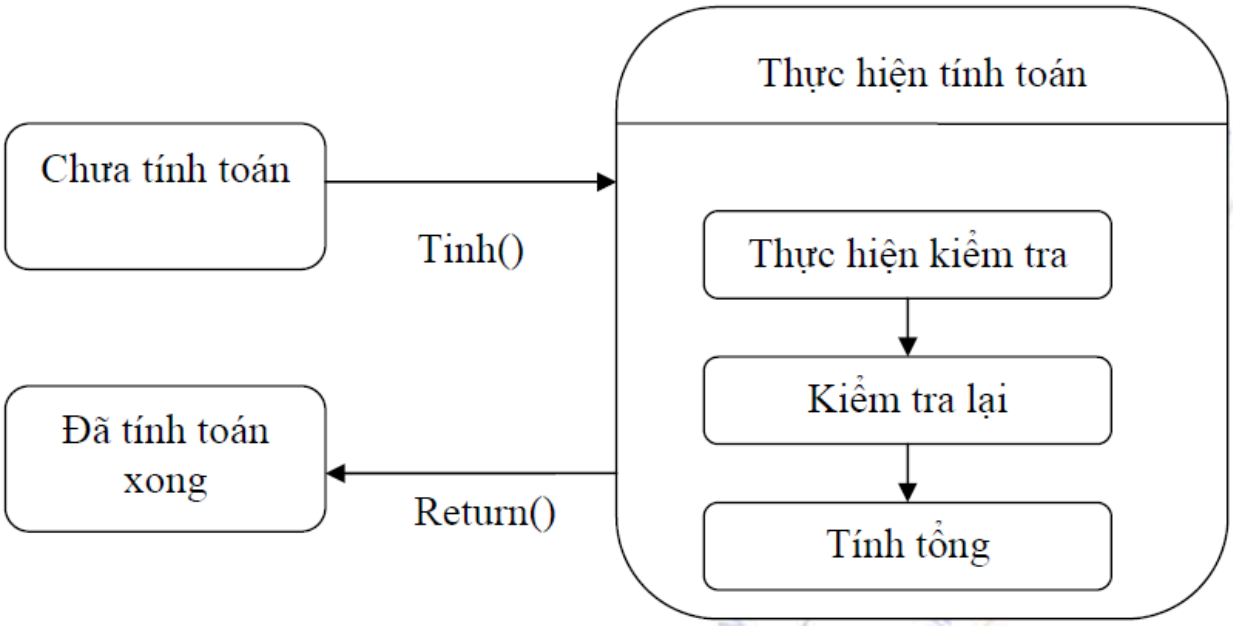
Biểu đồ trạng thái hệ thống mô tả tất cả các trạng thái của một đối tượng trong toàn bộ hoạt động của cả hệ thống.

Tập ký hiệu UML cho biểu đồ trạng thái:

- Các thành phần trong một biểu đồ trạng thái bao gồm:

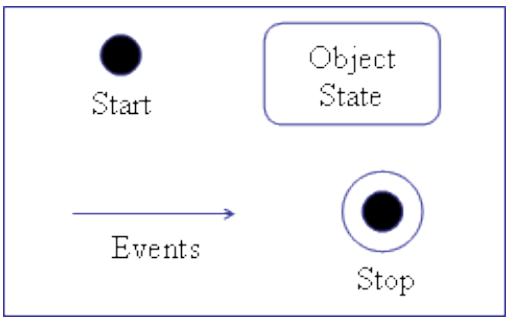
+ Trạng thái (state): Bên trong các trạng thái có thể miêu tả các biến trạng thái hoặc các hành động (action) tương ứng với trạng thái đó.

+ Trạng thái con (substate): là một trạng thái chứa bên trong một trạng thái khác. Trạng thái có nhiều trạng thái con gọi là trạng thái tổ hợp. Ví dụ trạng thái con như hình sau:

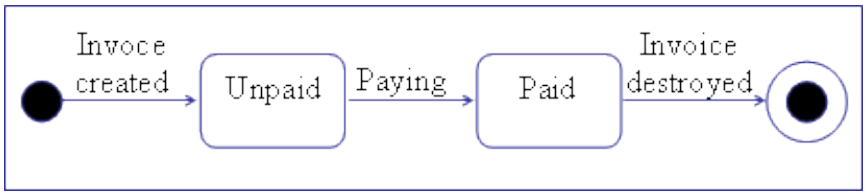


Hình: Biểu đồ trạng thái có trạng thái con

Hình sau sẽ chỉ ra các kí hiệu UML thể hiện trạng thái bắt đầu và trạng thái kết thúc, sự kiện cũng như các trạng thái của một đối tượng.

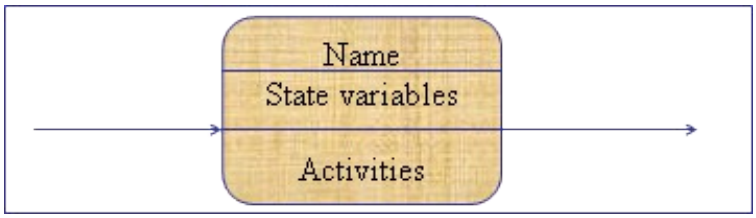


Các ký hiệu UML thể hiện bắt đầu, kết thúc, sự kiện và trạng thái của một đối tượng.



Hình: Biểu đồ trạng thái thực hiện hoá đơn.

Một trạng thái có thể có ba thành phần, như được chỉ trong hình sau :



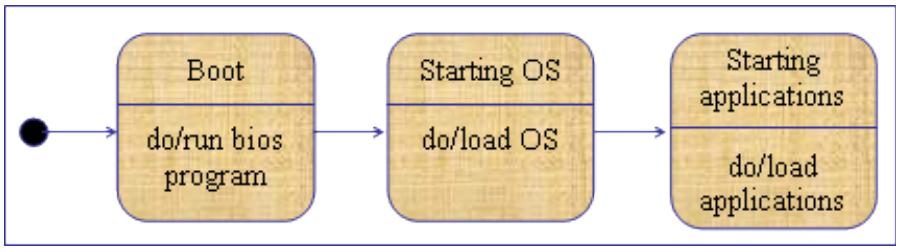
Hình: Các ngăn Tên, Biến trạng thái và hành động

Phần thứ nhất chỉ ra tên của trạng thái, ví dụ như chờ, đã được trả tiền hay đang chuyển động. Phần thứ hai (không bắt buộc) dành cho các biến trạng thái. Đây là những thuộc tính của lớp được thể hiện qua biểu đồ trạng thái; nhiều khi các biến tạm thời cũng tỏ ra rất hữu dụng trong trạng thái, ví dụ như các loại biến đếm (counter). Phần thứ ba (không bắt buộc) là phần dành cho hoạt động, nơi các sự kiện và các hành động có thể được liệt kê. Có ba loại sự kiện chuẩn hóa có thể được sử dụng cho phần hành động: entry (đi vào), exit (đi ra), và do (thực hiện). Loại sự kiệnđi vào được sử dụng để xác định các hành động khởi nhập trạng thái, ví dụ gán giá trị cho một thuộc tính hoặc gửi đi một thông điệp. Sự kiện đi ra có thể được sử dụng để xác định hành động khi rời bỏ trạng thái. Sự kiện thực hiện được sử dụng để xác định hành động cần phải được thực hiện

trong trạng thái, ví dụ như gửi một thông điệp, chờ, hay tính toán. Ba loại sự kiện chuẩn này không thể được sử dụng cho các mục đích khác.

Một sự biến đổi trạng thái thường có một sự kiện đi kèm với nó, nhưng không bắt buộc. Nếu có một sự kiện đi kèm, sự thay đổi trạng thái sẽ được thực hiện khi sự kiện kia xảy ra. Một hành động loại thực hiện trong trạng thái có thể là một quá trình đang tiếp diễn (ví dụ chờ, điều khiển các thủ tục,...) phải được thực hiện trong khi đối tượng vẫn ở nguyên trong trạng thái này. Một hành động thực hiện có thể bị ngắt bởi các sự kiện từ ngoài, có nghĩa là một sự kiện kiện gây nên một sự biến đổi trạng thái có thể ngưng ngắt một hành động thực hiện mang tính nội bộ đang tiếp diễn.

Trong trường hợp một sự biến đổi trạng thái không có sự kiện đi kèm thì trạng thái sẽ thay đổi khi hành động nội bộ trong trạng thái đã được thực hiện xong (hành động nội bộ kiểu đi vào, đi ra, thực hiện hay các hành động do người sử dụng định nghĩa). Theo đó, khi tất cả các hành động thuộc trạng thái đã được thực hiện xong, một sự thay đổi trạng thái sẽ tự động xảy ra mà không cần sự kiện từ ngoài.



Biến đổi trạng thái không có sự kiện từ ngoài. Sự thay đổi trạng thái xảy ra khi các hoạt động trong mỗi trạng thái được thực hiện xong.

Nhận biết trạng thái và sự kiện:

Quá trình phát hiện sự kiện và trạng thái về mặt bản chất bao gồm việc hỏi một số các câu hỏi thích hợp:

- Một đối tượng có thể có những trạng thái nào?: Hãy liệt kê ra tất cả những trạng thái mà một đối tượng có thể có trong vòng đời của nó.

- Những sự kiện nào có thể xảy ra?: Bởi sự kiện gây ra việc thay đổi trạng thái nên nhận ra các sự kiện là một bước quan trọng để nhận diện trạng thái.

- Trạng thái mới sẽ là gì?: Sau khi nhận diện sự kiện, hãy xác định trạng thái khi sự kiện này xảy ra và trạng thái sau khi sự kiện này xảy ra.

- Có những thủ tục nào sẽ được thực thi?: Hãy để ý đến các thủ tục ảnh hưởng đến trạng thái của một đối tượng.

- Chuỗi tương tác giữa các đối tượng là gì?: Tương tác giữa các đối tượng cũng có thể ảnh hưởng đến trạng thái của đối tượng.

- Qui định nào sẽ được áp dụng cho các phản ứng của các đối tượng với nhau?: Các qui định kiềm tỏa phản ứng đối với một sự kiện sẽ xác định rõ hơn các trạng thái.

- Những sự kiện và sự chuyển tải nào là không thể xảy ra?: Nhiều khi có một số sự kiện hoặc sự thay đổi trạng thái không thể xảy ra. Ví dụ như bán một chiếc ô tô đã được bán rồi.

- Cái gì khiến cho một đối tượng được tạo ra?: Đối tượng được tạo ra để trả lời cho một sự kiện. Ví dụ như một sinh viên ghi danh cho một khóa học.

- Cái gì khiến cho một đối tượng bị hủy?: Đối tượng sẽ bị hủy đi khi chúng không được cần tới nữa. Ví dụ khi một sinh viên kết thúc một khóa học.

- Cái gì khiến cho đối tượng cần phải được tái phân loại (reclassfied)?: Những loại sự kiện như một nhân viên được tăng chức thành nhà quản trị sẽ khiến cho động tác tái phân loại của nhân viên đó được thực hiện.

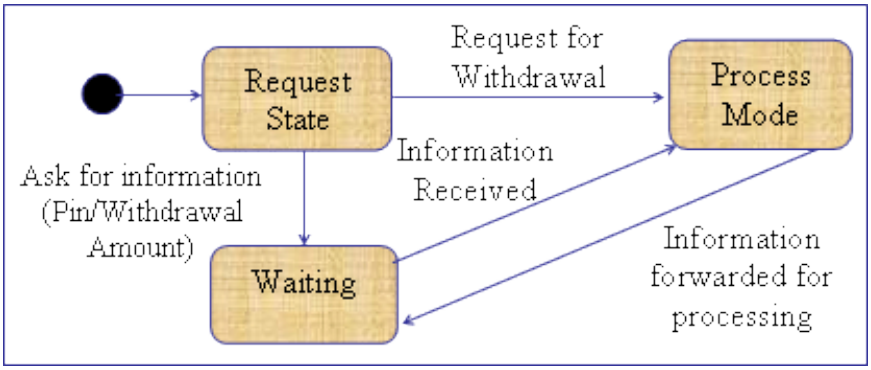
Chuyển biểu đồ tuần tự thành biểu đồ trạng thái:

Hãy dõi theo một chuỗi các sự kiện được miêu tả trong biểu đồ, chuỗi này phải mang tính tiêu biểu cho các tương tác trong hệ thống. Hãy quan sát các sự kiện ảnh hưởng đến đối tượng mà ta đang nghiên cứu.

Hãy sắp xếp các sự kiện thành một đường dẫn, dán nhãn input (hoặc entry) và output (exit) cho các sự kiện. Khoảng cách giữa hai sự kiện này sẽ là một trạng thái.

Nếu cảnh kịch có thể được nhắc đi nhắc lại rất nhiều lần (vô giới hạn), hãy nối đường dẫn từ trạng thái cuối cùng đến trạng thái đầu tiên.

Biểu đồ sau đây chỉ ra biểu đồ trạng thái của một lớp máy ATM, được chiết suất từ biểu đồ tuần tự hoặc biểu đồ cộng tác đã được trình bày trong các phần trước.



Hình: Chuyển một biểu đồ tuần tự sang biểu đồ trạng thái

Trộn lẫn các cảnh kịch khác vào trong biểu đồ trạng thái:

Một khi biểu đồ trạng thái cho một đối tượng đã sẵn sàng, chúng ta cần phải trộn những chuỗi sự kiện có ảnh hưởng đến đối tượng này vào trong biểu đồ trạng thái. Điều này có nghĩa là chúng ta cần phải quan sát các hiệu ứng gián tiếp của các sự kiện khác đối với đối tượng đang là chủ đề chính của biểu đồ trạng thái. Đây là việc quan trọng, bởi các đối tượng trong một hệ thống tương tác với nhau và vì các đối tượng khác cũng có khả năng gây nên sự kiện cho một đối tượng định trước, nên lối ứng xử này cũng cần phải được thể hiện trong biểu đồ trạng thái.

Điểm bắt đầu cho công việc này là:

- Ấn định một điểm bắt đầu chung cho tất cả các chuỗi sự kiện bổ sung.
- Xác định điểm nơi các ứng xử bắt đầu khác biệt với những ứng xử đã được mô hình hóa trong biểu đồ trạng thái.

Bổ sung thêm sự các biến đổi mới từ trạng thái này, trong tư cách một đường dẫn thay thế. Cần để ý đến những đường dẫn có vẻ ngoài đồng nhất nhưng thật ra có khác biệt trong một tình huống nhất định nào đó.

Hãy chú ý đến các sự kiện xảy ra trong những tình huống bất tiện. Mỗi sự kiện do khách hàng hay người sử dụng gây nên đều có thể sa vào trạng thái của các sự kiện bất tiện. Hệ thống không nắm quyền điều khiển đối với người sử

dụng và người sử dụng có thể quyết định để làm nảy ra một sự kiện tại một thời điểm tiện lợi đối với anh ta. Ví dụ như khách hàng có thể quyết định kết thúc trước kỳ hạn một tài khoản đầu tư.

Một trường hợp khác cũng cần phải được xử lý là sự kiện do người sử dụng gây nên không thể xảy ra. Có một loạt các lý do (người sử dụng thiếu tập trung, buồn nản, lơ đãng...) khiến cho sự kiện loại này không xảy ra. Cả trường hợp này cũng phải được xử lý thấu đáo. Ví dụ một khách hàng thất bại trong việc báo cho nhà băng biết những mệnh lệnh của anh ta về kỳ hạn của tài khoản, một tấm séc được viết ra nhưng lại không có khả năng giải tỏa mức tiền cần thiết.

Nhìn theo phương diện các biểu đồ trạng thái như là một thành phần của mô hình động, cần chú ý những điểm sau:

- Biểu đồ trạng thái chỉ cần được tạo dựng nên cho các lớp đối tượng có ứng xử động quan trọng.

- Hãy thẩm tra biểu đồ trạng thái theo khía cạnh tính nhất quán đối với những sự kiện dùng chung để cho toàn bộ mô hình động được đúng đắn.

- Dùng các trường hợp sử dụng để hỗ trợ cho quá trình tạo dựng biểu đồ trạng thái.

- Khi định nghĩa một trạng thái, hãy chỉ để ý đến những thuộc tính liên quan.

3.4. Các ví dụ và bài tập

Chương 4. Lớp, gói và quan hệ

1. Lớp và gói

1.1. Lớp và tìm kiếm lớp

Lớp:

Một lớp là một lời miêu tả của một nhóm các đối tượng có chung thuộc tính, chung phương thức (ứng xử), chung các mối quan hệ với các đối tượng khác và chung ngữ nghĩa (semantic). Nói như thế có nghĩa lớp là một khuôn mẫu để tạo ra đối tượng. Mỗi đối tượng là một thực thể của một lớp nào đó và một đối tượng không thể là kết quả thực thể hóa của nhiều hơn một lớp. Chúng ta sử dụng khái niệm lớp để bàn luận về các hệ thống và để phân loại các đối tượng mà chúng ta đã nhận dạng ra trong thế giới thực.

Một lớp tốt sẽ nắm bắt một và chỉ một sự trừu tượng hóa - nó phải có một chủ đề chính. Ví dụ, một lớp vừa có khả năng giữ tất cả các thông tin về một sinh viên và thông tin về tất cả những lớp học mà người sinh viên đó đã trải qua trong nhiều năm trước không phải là một lớp tốt, bởi nó không có chủ đề chính. Lớp này cần phải được chia ra làm hai lớp liên quan đến nhau: lớp sinh viên và lớp lịch sử của sinh viên.

Tìm lớp:

Hầu như không có một công thức chung cho việc phát hiện ra các lớp. Đi tìm các lớp là một công việc đòi hỏi trí sáng tạo và cần phải được thực thi với sự trợ giúp của chuyên gia ứng dụng. Vì qui trình phân tích và thiết kế mang tính vòng lặp, nên danh sách các lớp sẽ thay đổi theo thời gian. Tập hợp ban đầu của các lớp tìm ra chưa chắc đã là tập hợp cuối cùng của các lớp sau này sẽ được thực thi và biến đổi thành code. Vì thế, thường người ta hay sử dụng đến khái niệm các lớp ứng cử viên (Candidate Class) để miêu tả tập hợp những lớp đầu tiên được tìm ra cho hệ thống.

Như đã biết trường hợp sử dụng là những lời miêu tả chức năng của hệ thống, còn trách nhiệm thực thi thuộc về các đối tượng cộng tác thực thi chức

năng đó. Nói một cách khác, chúng ta đi tìm các lớp là để tiến tới tìm giải pháp cung cấp những ứng xử hướng ngoại đã được xác định trong các trường hợp sử dụng.

Có nhiều phương pháp khác nhau để thực hiện công việc đó. Có phương pháp đề nghị tiến hành phân tích phạm vi bài toán, chỉ ra tất cả các lớp thực thể (thuộc phạm vi bài toán) với mối quan hệ của chúng với nhau. Sau đó nhà phát triển sẽ phân tích từng trường hợp sử dụng và phân bổ trách nhiệm cho các lớp trong mô hình phân tích (analysis model), nhiều khi sẽ thay đổi chúng hoặc bổ sung thêm các lớp mới. Có phương pháp đề nghị nên lấy các trường hợp sử dụng làm nền tảng để tìm các lớp, làm sao trong quá trình phân bổ trách nhiệm thì mô hình phân tích của phạm vi bài toán sẽ từng bước từng bước được thiết lập.

Phân tích phạm vi bài toán để tìm lớp:

Quá trình phân tích phạm vi bài toán thường được bắt đầu với các khái niệm then chốt (Key Abstraction), một công cụ thường được sử dụng để nhận diện và lọc ra các lớp ứng cử viên (Candidate class).

Khái niệm then chốt:

Hãy lấy ví dụ một nhà băng ABC, điều đầu tiên ta nghĩ tới là gì? Tiền! Bên cạnh đó, ABC còn phải có những thực thể liên quan tới tiền như sau:

Khách hàng

Sản phẩm (các tài khoản được coi là các sản phẩm của một nhà băng)

Lực lượng nhân viên

Ban quản trị nhà băng

Phòng máy tính trong nhà băng

Những thực thể này được gọi là các khái niệm then chốt cho những gì mà nhà băng có thể có. Khái niệm then chốt hoặc mang tính cấu trúc (structural) hoặc mang tính chức năng (functional). Thực thể mang tính cấu trúc là những thực thể vật lý tương tác với nhà băng, ví dụ khách hàng. Thực thể mang tính chức năng là những chức năng mà nhà băng phải thực hiện, ví dụ duy trì một tài khoản hoặc

chuyển tiền từ tài khoản này sang tài khoản khác. Khái niệm then chốt là các thực thể ta để ý đến đầu tiên. Chúng rất quan trọng vì giúp ta:

Định nghĩa ranh giới của vấn đề

Nhấn mạnh đến các thực thể có liên quan đến thiết kế của hệ thống

Loại bỏ thực thể nằm ngoài phạm vi hệ thống

Các khái niệm then chốt thường sẽ trở thành các lớp trong mô hình phân tích

Một khái niệm then chốt tóm lại là một lớp hay đối tượng thuộc chuyên ngành của phạm vi bài toán. Khi trình bày với người sử dụng, chúng có một ánh xạ 1-1 giữa với những thực thể liên quan tới người sử dụng như hóa đơn, sec, giấy đề nghị rút tiền, sổ tiết kiệm, thẻ rút tiền tự động, nhân viên thu ngân, nhân viên nhà băng, các phòng ban,….

Mức độ trừu tượng:

Khi phân tích phạm vi bài toán, cần chú ý rằng mức độ trừu tượng của các khái niệm then chốt là rất quan trọng, bởi mức độ trừu tượng quá cao hay quá thấp đều rất dễ gây nhầm lẫn.

Mức trừu tượng quá cao dẫn tới những định nghĩa quá khái quát về một thực thể, tạo nên một cái nhìn vĩ mô và thường không nhắm vào một mục tiêu cụ thể. Ví dụ trong một nhà băng, ta không thể chọn khái niệm then chốt là "người", bởi nó sẽ dẫn đến lời miêu tả: "Một người đến nhà băng để gửi tiền vào, và số tiền đó được một người khác tiếp nhận." – trong khi một yêu cầu quan trọng ở đây là phải phân biệt giữa nhân viên với khách hàng vì chức năng của họ là khác hẳn nhau.

Tương tự như vậy, mức trừu tượng quá thấp cũng dễ gây hiểu lầm, bởi những thông tin quá vụn vặt chưa thích hợp với thời điểm này. Ví dụ những quyết định dạng:

Form mở tài khoản đòi hỏi tất cả 15 Entry.

Những dữ liệu trên Form này đều phải được căn phải.

Không có nhiều chỗ để ghi địa chỉ của khách hàng trên Form.

nên được để dành cho các giai đoạn sau.

Vài điểm cần chú ý về khái niệm then chốt:

Những thực thể xuất hiện đầu tiên trong óc não chúng ta là những thực thể dễ có khả năng trở thành khái niệm then chốt cho một vấn đề định trước.

Mỗi lần tìm thấy một khái niệm then chốt mới, cần xem xét nó theo cách nhìn của vấn đề, có thể hỏi các câu hỏi sau:

Những chức năng nào có thể được thực hiện đối với thực thể này?

Điều gì khiến những thực thể loại này được tạo ra?

Nếu không có câu trả lời thích hợp, cần phải suy nghĩ lại về thực thể đó.

Mỗi khái niệm then chốt mới cần phải được đặt tên cho thích hợp, miêu tả đúng chức năng của khái niệm.

Nhận dạng lớp và đối tượng:

Nắm vững khái niệm lớp, chúng ta có thể tương đối dễ dàng tìm thấy các lớp và đối tượng trong phạm vi vấn đề. Một nguyên tắc thô sơ thường được áp dụng là danh từ trong các lời phát biểu bài toán thường là các ứng cử viên để chuyển thành lớp và đối tượng.

Một số gợi ý thực tế cho việc tìm lớp trong phạm vi vấn đề:

Bước đầu tiên là cần phải tập trung nghiên cứu kỹ:

Các danh từ trong những lời phát biểu bài toán

Kiến thức chuyên ngành thuộc phạm vi bài toán

Các Trường hợp sử dụng

Ví dụ trong lời phát biểu "Có một số tài khoản mang lại tiền lãi", ta thấy có hai danh từ là tài khoản và tiền lãi. Chúng có thể là các lớp tiềm năng cho mô hình nhà băng lẻ.

Thứ hai, chúng ta cần chú ý đến các nhóm vật thể trong hệ thống hiện thời như:

Các thực thể vật lý của hệ thống: những vật thể tương tác với hệ thống, ví dụ khách hàng.

Các vật thể hữu hình: các vật thể vật lý mà ta có thể nhìn và sờ thấy. Ví dụ như công cụ giao thông, sách vở, một con người, một ngôi nhà,…. Trong một nhà băng ABC, đó có thể là tập sec, phiếu đề nghị rút tiền, sổ tiết kiệm, các loại Form cần thiết.

Các sự kiện (Events): Một chiếc xe bị hỏng, một cái cửa được mở ra. Trong một nhà băng là sự đáo hạn một tài khoản đầu tư, hiện tượng rút quá nhiều tiền mặt trong một tài khoản bình thường.

Các vai trò (Role): Ví dụ như mẹ, khách hàng, người bán hàng, …. Trong một nhà băng, vai trò có thể là nhân viên, nhà quản trị, khách hàng,...

Các sự tương tác (Interactions): Ví dụ việc bán hàng là một chuỗi tương tác bao gồm khách hàng, người bán hàng và sản phẩm. Trong một nhà băng, việc mở một tài khoản mới sẽ yêu cầu một chuỗi tương tác giữa nhân viên và khách hàng.

Vị trí (Location): Một đồ vật nào đó hoặc một người nào đó được gán cho một vị trí nào đó. Ví dụ: Ôtô đối với nhà để xe. Trong một nhà băng ta có thể thấy nhân viên thu ngân luôn đứng ở cửa sổ của mình.

Đơn vị tổ chức (Organisation Unit): Ví dụ các phòng ban, phòng trưng bày sản phẩm, các bộ phận. Trong một nhà băng có thể có bộ phận tài khoản bình thường, bộ phận tài khoản tiết kiệm, bộ phận tài khoản đầu tư.

Bên cạnh đó, còn nhiều câu hỏi khác giúp ta nhận dạng lớp. Ví dụ như:

Ta có thông tin cần được lưu trữ hoặc cần được phân tích không? Nếu có thông tin cần phải được lưu trữ, biến đổi, phân tích hoặc xử lý trong một phương thức nào đó thì chắc chắn đó sẽ là ứng cử viên cho lớp. Những thông tin này có thể là một khái niệm luôn cần phải được ghi trong hệ thống hoặc là sự kiện, giao dịch xảy ra tại một thời điểm cụ thể nào đó.

Ta có các hệ thống ngoại vi không? Nếu có, thường chúng cũng đáng được quan tâm tới khi tạo dựng mô hình. Các hệ thống bên ngoài có thể được coi là các lớp chứa hệ thống của chúng ta hoặc tương tác với hệ thống của chúng ta.

Chúng ta có các mẫu, thư viện lớp, thành phần và những thứ khác không? Nếu chúng ta có mẫu, thư viện, thành phần từ các dự án trước (xin được của các bạn đồng nghiệp, mua được từ các nhà cung cấp) thì chúng thường cũng sẽ chứa các ứng cử viên lớp.

Có thiết bị ngoại vi mà hệ thống của chúng ta cần xử lý không? Mỗi thiết bị kỹ thuật được nối với hệ thống của chúng ta thường sẽ trở thành ứng cử viên cho lớp xử lý loại thiết bị ngoại vi này.

Chúng ta có phần công việc tổ chức không? Miêu tả một đơn vị tổ chức là công việc được thực hiện với các lớp, đặc biệt là trong các mô hình doanh nghiệp.

Tổng kết về các nguồn thông tin cho việc tìm lớp:

Nhìn chung, các nguồn thông tin chính cần đặc biệt chú ý khi tìm lớp là:

Các lời phát biểu yêu cầu

Các Trường hợp sử dụng

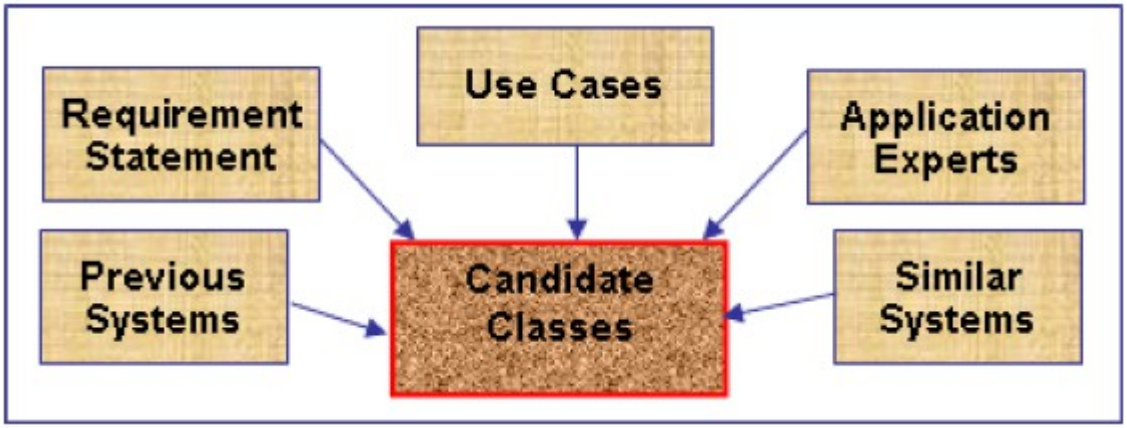
Sự trợ giúp của các chuyên gia ứng dụng

Nghiên cứu hệ thống hiện thời

Loạt các lớp đầu tiên được nhận dạng qua đây thường được gọi là các lớp ứng cử viên (Candidate Class). Ngoài ra, nghiên cứu những hệ thống tương tự cũng có thể sẽ mang lại cho ta các lớp ứng cử viên khác:

Khi nghiên cứu hệ thống hiện thời, hãy để ý đến các danh từ và các khái niệm then chốt để nhận ra lớp ứng cử viên. Không nên đưa các lớp đã được nhận diện một lần nữa vào mô hình chỉ bởi vì chúng được nhắc lại ở đâu đó theo một tên gọi khác. Ví dụ, một hệ thống nhà băng có thể coi cùng một khách hàng với nhiều vị trí khác nhau là nhiều khách hàng khác nhau. Cần chú ý khi phân tích những lời miêu tả như thế để tránh dẫn đến sự trùng lặp trong quá trình nhận diện lớp.

Có nhiều nguồn thông tin mà nhà thiết kế cần phải chú ý tới khi thiết kế lớp và chỉ khi làm như vậy, ta mới có thể tin chắc về khả năng tạo dựng một mô hình tốt. Hình sau tổng kết các nguồn thông tin kể trên.



Hình 3.4 - Nguồn thông tin hỗ trợ tìm lớp

Các trường hợp sử dụng là nguồn tốt nhất cho việc nhận diện lớp và đối tượng. Cần nghiên cứu kỹ các Trường hợp sử dụng để tìm các thuộc tính (attribute) báo trước sự tồn tại của đối tượng hoặc lớp tiềm năng. Ví dụ nếu Trường hợp sử dụng yêu cầu phải đưa vào một số tài khoản (account-number) thì điều này trỏ tới sự tồn tại của một đối tượng tài khoản.

Một nguồn khác để nhận ra lớp/đối tượng là các Input và Output của hệ thống. Nếu Input bao gồm tên khách hàng thì đây là tín hiệu cho biết sự tồn tại của một đối tượng khách hàng, bởi nó là một attribute của khách hàng.

Nói chuyện với người sử dụng cũng gợi mở đến các khái niệm then chốt. Thường thì người sử dụng miêu tả hệ thống theo lối cần phải đưa vào những gì và mong chờ kết quả gì. Thông tin đưa vào và kết quả theo lối miêu tả của người sử dụng cần phải được tập hợp lại với nhau để nhận dạng khái niệm then chốt.

1.2. Biểu đồ lớp

Một biểu đồ lớp là một dạng mô hình tĩnh. Một biểu đồ lớp miêu tả hướng nhìn tĩnh của một hệ thống bằng các khái niệm lớp và mối quan hệ giữa chúng với nhau. Mặc dù nó cũng có những nét tương tự với một mô hình dữ liệu, nhưng nên nhớ rằng các lớp không phải chỉ thể hiện cấu trúc thông tin mà còn miêu tả cả hành vi. Một trong các mục đích của biểu đồ lớp là tạo nền tảng cho các biểu đồ khác, thể hiện các khía cạnh khác của hệ thống (ví dụ như trạng thái của đối tượng hạy công tác động giữa các đối tượng, được chỉ ra trong các biểu đồ động). Một lớp trong biểu đồ lớp có thể được thực thi trực tiếp trong một ngôn ngữ hướng đối tượng có hỗ trợ trực tiếp khái niệm lớp. Một biểu đồ lớp chỉ ra các lớp, nhưng bên cạnh đó còn có một biến tấu hơi khác một chút chỉ ra các đối tượng thật sự là các thực thể của các lớp này (biểu đồ đối tượng).

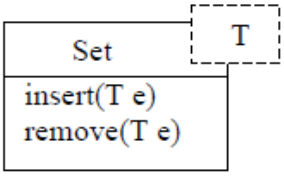
Để tạo một biểu đồ lớp, đầu tiên ta phải nhận diện và miêu tả các lớp. Một khi đã có một số lượng các lớp, ta sẽ xét đến quan hệ giữa các lớp đó với nhau.

Các loại lớp trong biểu đồ:

Biểu đồ lớp có thể chứa nhiều loại lớp khác nhau, chúng có thể là *những lớp thông thường, lớp tham số hoá, lớp hiện thực, lớp tiện ích, và lớp metaclass (siêu lớp).*

Lớp tham số hoá*(ParameterizedClass)*

Lớp tham số hoá là lớp được sử dụng để tạo ra một họ các lớp khác. Trong những ngôn ngữ lập trình có kiểu mạnh như C++, lớp tham số hoá chính là lớp mẫu (template). Trong UML, có thể khai báo lớp tham số hoá (lớp mẫu) Set cho họ các lớp có các phần tử là kiểu T bất kỳ, được xem như là tham số như sau:



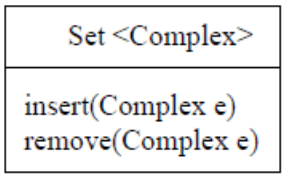
Hình: Lớp được tham số hoá

Lớp tham số hoá có thể sử dụng để thể hiện quyết định thiết kế về các giao thức trao đổi giữa các lớp. Lớp tham số hoá ít được sử dụng trong mô hình khái niệm mà chủ yếu được sử dụng trong các mô hình cài đặt, nhưng cũng chỉ khi ngôn ngữ lập trình được chọn để lập trình có hỗ trợ cơ chế lớp mẫu (*template class*) như C++ chẳng hạn. Cũng cần lưu ý là không phải tất cả các ngôn ngữ lập trình hướng đối tượng đều hỗ trợ kiểu lớp mẫu, ví dụ Java không hỗ trợ, nhưng tất cả các lớp trong Java lại tạo ra cấu trúc cây phân cấp có gốc là lớp Object. Do tính chất phân cấp của cây và nguyên lý chung bảo toàn mối quan hệ giữa các lớp con với lớp cha (nguyên lý thành viên và nguyên lý 100%) mà ta vẫn có thể tạo ra được những cấu trúc tổng quát, không thuần nhất tương tự như lớp mẫu [15].

Lớp hiện thực(*InstantiatedClass*)

Lớp hiện thực là loại lớp tham số hoá mà đối số của nó là một kiểu trị cụ thể. Như vậy, lớp tham số hoá là khuôn để tạo ra các lớp hiện thực.

Lớp Set<Complex> tập các số phức (Complex) là lớp hiện thực được biểu diễn trong UML như hình 2.



Hình: Lớp hiện thực hoá

Lớp tiện ích (*Class Utility)*

Lớp tiện ích là tập hợp các thao tác được sử dụng nhiều nơi trong hệ thống, chúng được tổ chức thành lớp tiện ích để các lớp khác có thể cùng sử dụng. Trong biểu đồ, lớp tiện ích được thể hiện bằng lớp có đường viền bóng như hình 3 (a).

(a) Lớp tiện ích (b) Giao diện

Giao diện*(Interface)*

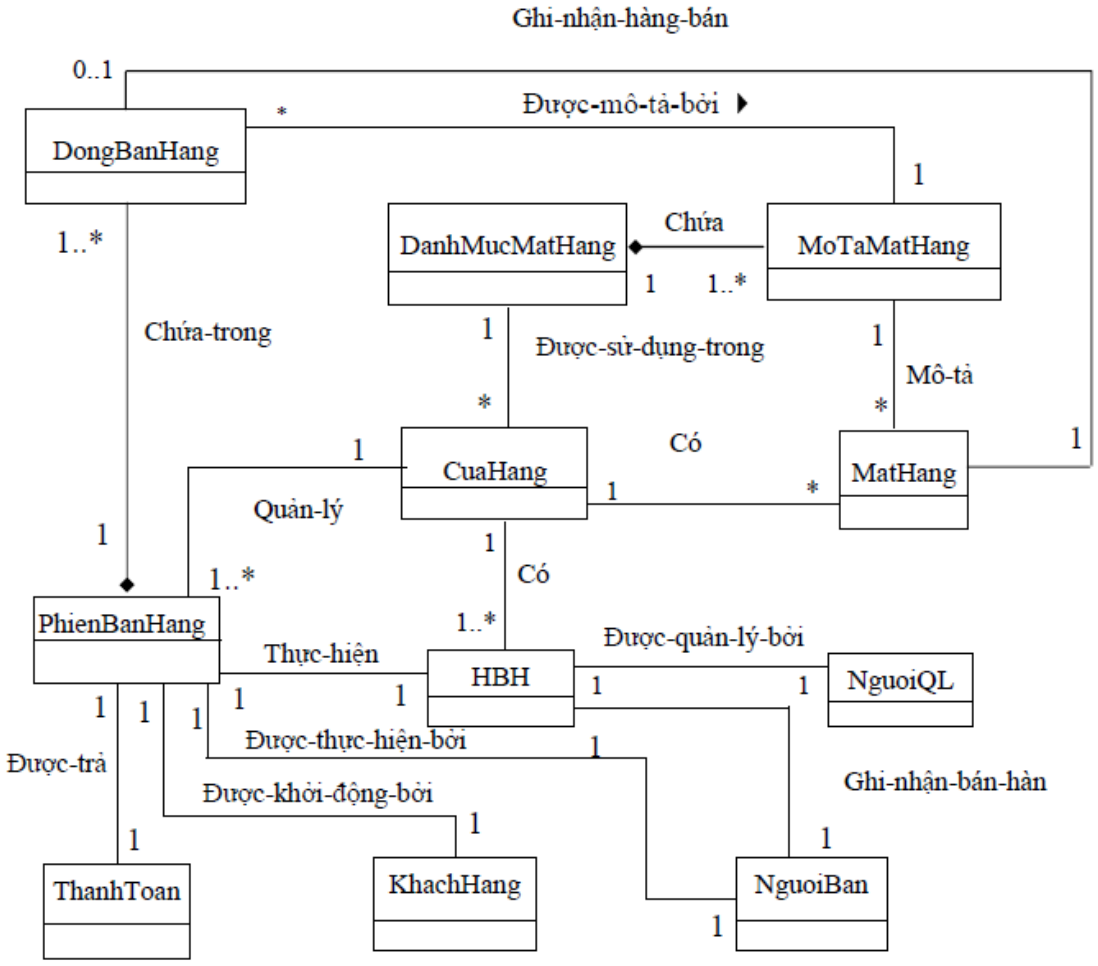
Giao diện là tập những thao tác quan sát được từ bên ngoài của một lớp và/hoặc một thành phần, và không có nội dung cài đặt của riêng lớp đó. Giao diện thuộc quan sát logic và có thể xuất hiện trong cả biểu đồ lớp và biểu đồ thành phần với ký hiệu đồ hoạ như hình 3 (b).

MetaClass*(siêu lớp)*

MetaClass là lớp để tạo ra các lớp khác, nghĩa là thể hiện của nó là lớp chứ không phải là đối tượng. Lớp tham số hoá chính là một loại siêu lớp.

Biểu đồ lớp trong Hệ HBH

Trong số những quan hệ kết hợp đã phát hiện ở mục trên, dễ nhận thấy lớp **DanhMucMatHang** chứa các **MoTaMatHang**, do vậy giữa hai lớp này có quan hệ kết nhập. Tương tự, trong mỗi **PhienBanHang** có một số **DongBanHang**, nghĩa là giữa hai lớp này cũng sẽ có quan hệ kết nhập. Kết hợp tất cả các mối quan hệ giữa các lớp như đã phân tích ở trên, chúng ta xây dựng được biểu đồ lớp như sau:



Hình: Biểu đồ lớp của HBH

1.3. Stereotype của lớp

Mẫu rập khuôn (stereotype) của các lớp:

Mẫu rập khuôn (Stereotype) là cơ chế mở rộng các phần tử của mô hình để tạo ra những phần tử mới. Nó cho phép dễ dàng bổ sung thêm các thông tin cho các phần tử của mô hình và những phần tử này được đặc tả trong các dự án hay trong quá trình phát triển phần mềm. Nó được sử dụng để phân loại các lớp đối tượng.

Rational Rose đã xây dựng một số *stereotype* như <<boundary>>, <<entity>>, <<control>>, <<interface>>, v.v., ngoài ra chúng ta có thể định nghĩa những loại kiểu mới cho mô hình hệ thống.

Lớp biên(*EntityClass*)

Lớp biên là lớp nằm trên đường biên của hệ thống với phần thế giới bên ngoài. Nó có thể là biểu mẫu (*form*), báo cáo (*report*), giao diện với các thiết bị phần cứng như máy in, máy đọc ảnh (*Scanner*), v.v. hoặc là giao diện với các hệ thống khác. Trong UML, lớp biên được ký hiệu .

Để tìm lớp biên hãy khảo sát biểu đồ ca sử dụng, một tác nhân có thể xác định tương ứng một lớp biên. Nếu có hai tác nhân cùng kích hoạt một ca sử dụng thì chỉ cần tạo ra một lớp biên cho cả hai.

Lớp thực thể(*ControlClass*)

Lớp thực thể là lớp lưu giữ các thông tin mà nó được ghi vào bộ nhớ ngoài.

Lớp **SinhVien** là lớp thực thể. Trong UML, lớp thực thể được ký hiệu như trong hình 4 (b).

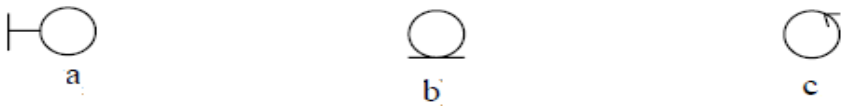
Lớp thực thể có thể tìm thấy trong các luồng sự kiện và biểu đồ tương tác. Thông thường phải tạo ra các bảng dữ liệu trong CSDL cho mỗi lớp thực thể. Mỗi thuộc tính của lớp thực thể trở thành trường dữ liệu trong bảng dữ liệu.

Lớp điều khiển

Lớp điều khiển là lớp làm nhiệm vụ điều phối hoạt động của các lớp khác. Thông thường mỗi ca sử dụng có một lớp điều khiển để điều khiển trình tự các sự kiện xảy ra trong nó.

Lớp điều khiển không tự thực hiện các chức năng nghiệp vụ của hệ thống, nhưng chúng lại gửi nhiều thông điệp cho những lớp có liên quan, do vậy còn được gọi là lớp quản lý.

Trong UML, lớp điều khiển được ký hiệu như trong hình (c).

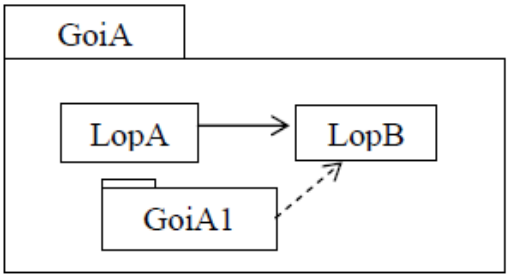


(a) Lớp biên (b) Lớp thực thể (c) Lớp điều khiển

1.4. Gói

Để hiểu được những hệ thống lớn, phức tạp có nhiều lớp đối tượng, thường chúng ta phải có cách chia các lớp đó thành một số nhóm được gọi là gói.

Gói là một nhóm các phần tử của mô hình gồm các lớp, các mối quan hệ và các gói nhỏ hơn. Cách tổ chức hệ thống thành các gói (hệ thống con) chính là cách phân hoạch mô hình thành các đơn vị nhỏ hơn để trị dễ hiểu và dễ quản lý hơn. Gói được mô tả trong UML gồm tên của gói, có thể có các lớp, gói nhỏ khác và được ký hiệu như hình 1.



Gói các lớp trong UML

Khi phân chia các lớp thành các gói, chúng ta có thể dựa vào: các lớp chính (lớp thống trị), các mối quan hệ chính, các chức năng chính. Theo cách phân chia đó chúng ta có thể chia hệ thống thành các phân hệ phù hợp với cách phân chia trong hệ thống thực.

Hệ thống quản lý thư viện có thể tổ chức thành bốn gói: gói giao diện, gói nghiệp vụ, gói CSDL và gói tiện ích như hình 2. Trong đó,

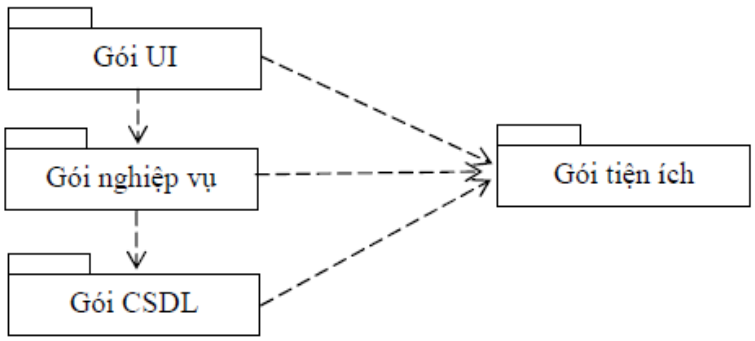
Gói giao diện (UI – User Interface): bao gồm các lớp giao diện với người dùng, cho các khả năng quan sát và truy nhập vào dữ liệu. Các đối tượng trong gói này có thể thực hiện các thao tác trên các đối tượng tác nghiệp để truy vấn hay nhập dữ liệu.

Gói nghiệp vụ (Business Package): chứa các lớp thực thể thuộc lĩnh vực bài toán ứng dụng.

Gói CSDL: chứa các lớp dịch vị cho các lớp ở gói tác nghiệp trong việc tổ chức, quản lý và lưu trữ dữ liệu.

Gói tiện ích: chứa các lớp dịch vụ cho các gói khác của hệ thống.

Các gói của một hệ thống thường có mối quan hệ với nhau, như quan hệ phụ thuộc.



Hình: Tổ chức các gói của hệ thống thư viện

1.5. Thuộc tính và thao tác

Lớp có thuộc tính miêu tả những đặc điểm của đối tượng. Giá trị của thuộc tính thường là những dạng dữ liệu đơn giản được đa phần các ngôn ngữ lập trình hỗ trợ như Integer, Boolean, Floats, Char, …

Thuộc tính có thể có nhiều mức độ trông thấy được (visibility) khác nhau, miêu tả liệu thuộc tính đó có thể được truy xuất từ các lớp khác, khác với lớp định nghĩa ra nó. Nếu thuộc tính có tính trông thấy là công cộng (public), thì nó có thể được nhìn thấy và sử dụng ngoài lớp đó. Nếu thuộc tính có tính trông thấy là riêng (private), bạn sẽ không thể truy cập nó từ bên ngoài lớp đó. Một tính trông thấy khác là bảo vệ (protected), được sử dụng chung với công cụ khái quát hóa và chuyên biệt hóa. Nó cũng giống như các thuộc tính riêng nhưng được thừz kế bởi các lớp dẫn xuất.

Trong UML, thuộc tính công cộng mang kí hiệu "+" và thuộc tính riêng mang dấu "-".

Giá trị được gán cho thuộc tính có thể là một cách để miêu tả trạng thái của đối tượng. Mỗi lần các giá trị này thay đổi là biểu hiện cho thấy có thể đã xảy ra một sự thay đổi trong trạng thái của đối tượng.

Lưu ý: Mọi đặc điểm của một thực thể là những thông tin cần lưu trữ đều có thể chuyển thành thuộc tính của lớp miêu tả loại thực thể đó.

1.6. Các bài tập và ví dụ

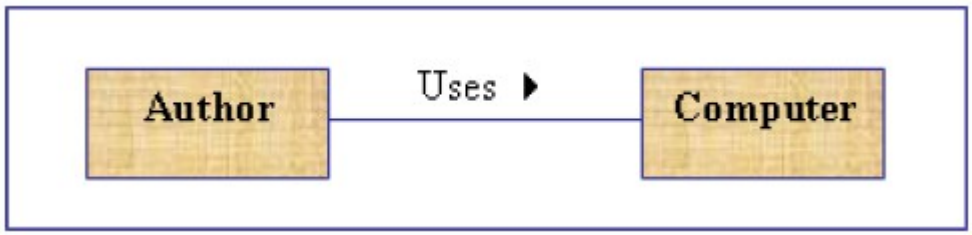
2. Quan hệ

2.1. Quan hệ giữa các lớp

2.2. Liên kết (association)

Một liên hệ là một sự nối kết giữa các lớp, một liên quan về ngữ nghĩa giữa các đối tượng của các lớp tham gia. Liên hệ thường thường mang tính hai chiều, có nghĩa khi một đối tượng này có liên hệ với một đối tượng khác thì cả hai đối tượng này nhận thấy nhau. Một mối liên hệ biểu thị bằng các đối tượng của hai lớp có nối kết với nhau, ví dụ rằng "chúng biết về nhau", "được nối với nhau", "cứ mỗi X lại có một Y",.... Lớp và liên hệ giữa các lớp là những công cụ rất mạnh mẽ cho việc mô hình hóa các hệ thống phức tạp, ví dụ như cấu trúc sản phẩm, cấu trúc văn bản và tất cả các cấu trúc thông tin khác.

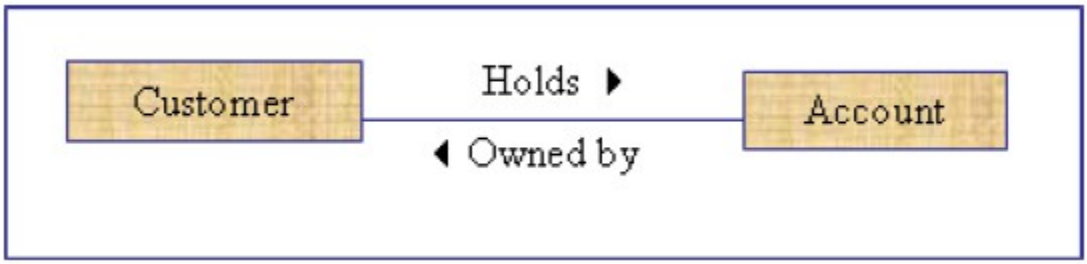
Mối liên kết được thể hiện trong biểu đồ UML bằng một đường thẳng nối hai lớp



Hình -Một lớp Author kết hợp với lớp Computer

1- Vai trò trong liên hệ

Một liên hệ có thể có các vai trò (Roles). Các vai trò được nối với mỗi lớp bao chứa trong quan hệ. Vai trò của một lớp là chức năng mà nó đảm nhận nhìn từ góc nhìn của lớp kia. Tên vai trò được viết kèm với một mũi tên chỉ từ hướng lớp chủ nhân ra, thể hiện lớp này đóng vai trò như thế nào đối với lớp mà mũi tên chỉ đến.



Hình 4.10- Vai trò trong liên hệ giữa Customer và Account

Trong ví dụ trên: một khách hàng có thể là chủ nhân của một tài khoản và tài khoản được chiếm giữ bởi khách hàng. Đường thẳng thể hiện liên hệ giữa hai lớp.

Một số điểm cần chú ý khi đặt tên vai trò:

Tên vai trò có thể bỏ đi nếu trùng với tên lớp

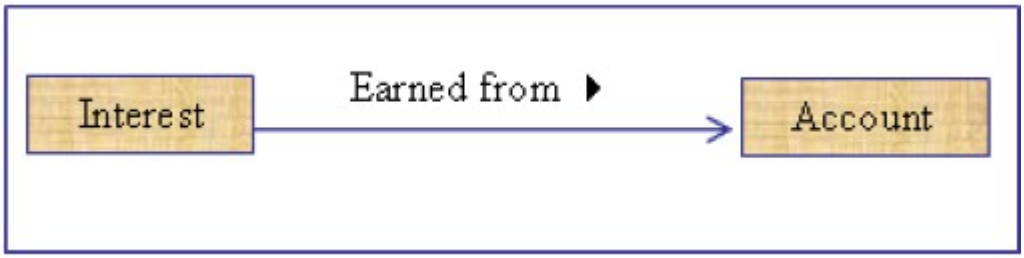
Tên vai trò phải là duy nhất.

Tên vai trò phải khác với các thuộc tính của lớp.

Tên vai trò phải miêu tả được chức năng mà lớp này đảm nhận trong quan hệ, tức cần phải là các khái niệm lấy ra từ phạm vi vấn đề, giống như tên các lớp.

2- Liên hệ một chiều (Uni-Directional Association)

Ta cũng có thể sử dụng mối liên hệ một chiều bằng cách thêm một mũi tên và một đầu của đường thẳng nối kết. Mũi tên chỉ ra rằng sự nối kết chỉ có thể được sử dụng duy nhất theo chiều của mũi tên.

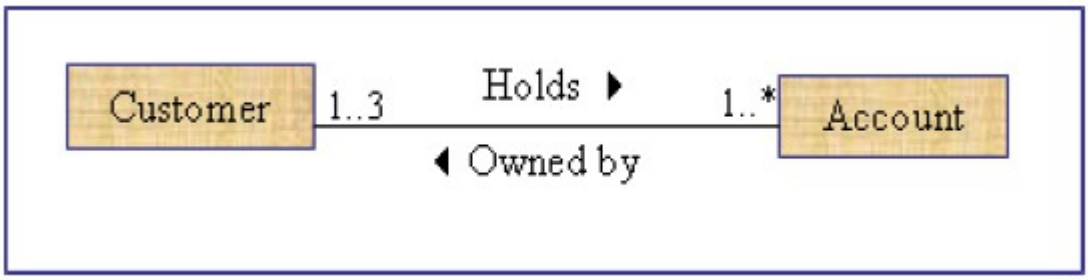


Hình - Liên hệ một chiều giữa Interest và Account

Biểu đồ phần 4.11 thể hiện rằng giữa hai lớp có liên hệ, nhưng không hề có thông tin về số lượng các đối tượng trong quan hệ. Ta không thể biết một khách hàng có thể có bao nhiêu tài khoản và một tài khoản có thể là của chung cho bao

nhiêu khách hàng. Trong UML, loại thông tin như thế được gọi là số lượng phần tử (Cardinality) trong quan hệ.

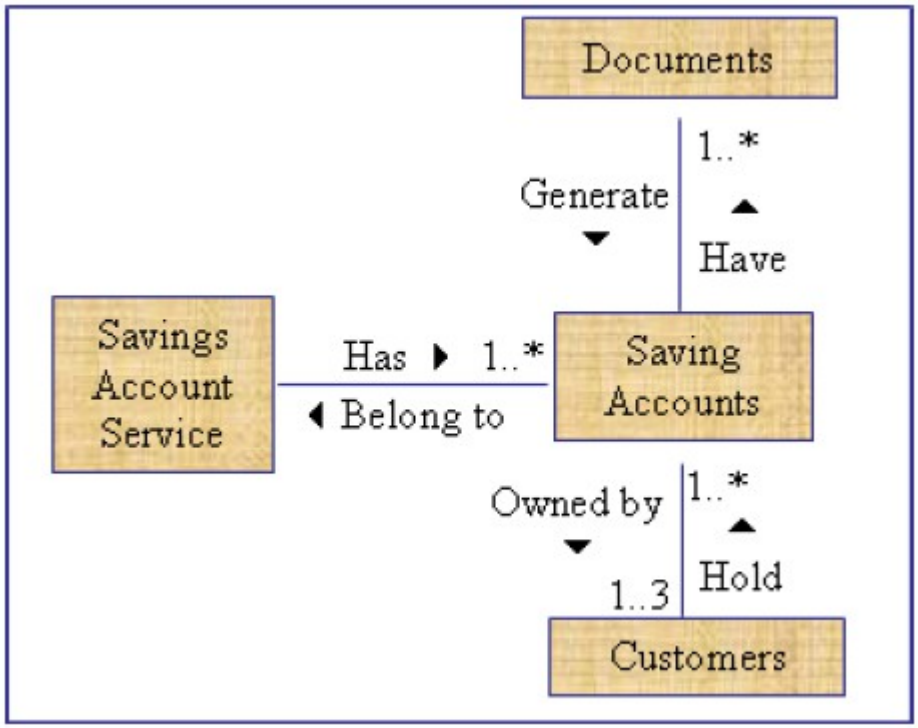
3- Số lượng (Cardinality) trong liên hệ:



Hình - Số lượng trong liên hệ giữa Customer và Account

Biểu đồ trên nói rõ một khách hàng có thể mở một hoặc nhiều tài khoản và một tài khoản có thể thuộc về một cho tới ba khách hàng.

Số lượng được ghi ở phía đầu đường thẳng thể hiện liên hệ, sát vào lớp là miền áp dụng của nó. Phạm vi của số lượng phần tử trong liên hệ có thể từ 0-tới-1 (0..1), 0-tới-nhiều (0..* hay), một-tới-nhiều (1..), hai (2), năm-tới-mười một (5..11). Cũng có thể miêu tả một dãy số ví dụ (1,4,6, 8..12). Giá trị mặc định là 1.



Hình - Một sơ đồ lớp tiêu biểu

Hình trên là ví dụ cho một biểu đồ lớp tiêu biểu. Biểu đồ giải thích rằng bộ phận dịch vụ tài khoản tiết kiệm của một nhà băng có thể có nhiều tài khoản tiết kiệm nhưng tất cả những tài khoản này đều thuộc về bộ phận đó. Một tài khoản tiết kiệm về phần nó lại có thể có nhiều tài liệu, nhưng những tài liệu này chỉ thuộc về một tài khoản tiết kiệm mà thôi. Một tài khoản tiết kiệm có thể thuộc về từ 1 cho tới nhiều nhất là 3 khách hàng. Mỗi khách hàng có thể có nhiều hơn một tài khoản.

4- Phát hiện liên hệ:

Thường sẽ có nhiều mối liên hệ giữa các đối tượng trong một hệ thống. Quyết định liên hệ nào cần phải được thực thi là công việc thụôc giai đoạn thiết kế. Có thể tìm các mối liên hệ qua việc nghiên cứu các lời phát biểu vấn đề, các yêu cầu. Giống như danh từ đã giúp chúng ta tìm lớp, các động từ ở đây sẽ giúp ta tìm ra các mối quan hệ.

Một vài gợi ý khi tìm liên hệ:

Vị trí về mặt vật lý hoặc sự thay thế, đại diện: Mỗi cụm động từ xác định hay biểu lộ một vị trí đều là một biểu hiện chắc chắn cho liên hệ. Ví dụ: tại địa điểm, ngồi trong, …

Sự bao chứa: Cụm động từ biểu lộ sự bao chứa, ví dụ như: là thành phần của....

Giao tiếp: Có nhiều cụm động từ biểu lộ sự giao tiếp, ví dụ truyền thông điệp, nói chuyện với, …

Quyền sở hữu: Ví dụ: thuộc về, của, …

Thoả mãn một điều kiện: Những cụm từ như: làm việc cho, là chồng/vợ của, quản trị, ….

5- Xử lý các liên hệ không cần thiết:

Sau khi tìm các mối liên hệ, bước tiếp theo đó là phân biệc các liên hệ cần thiết ra khỏi các liên hệ không cần thiết. Liên hệ không cần thiết có thể bao gồm những liên hệ bao chứa các lớp ứng cử viên đã bị loại trừ hoặc các liên hệ không liên quan đến hệ thống. Có những liên hệ được tạo ra nhằm mục đích tăng hiệu

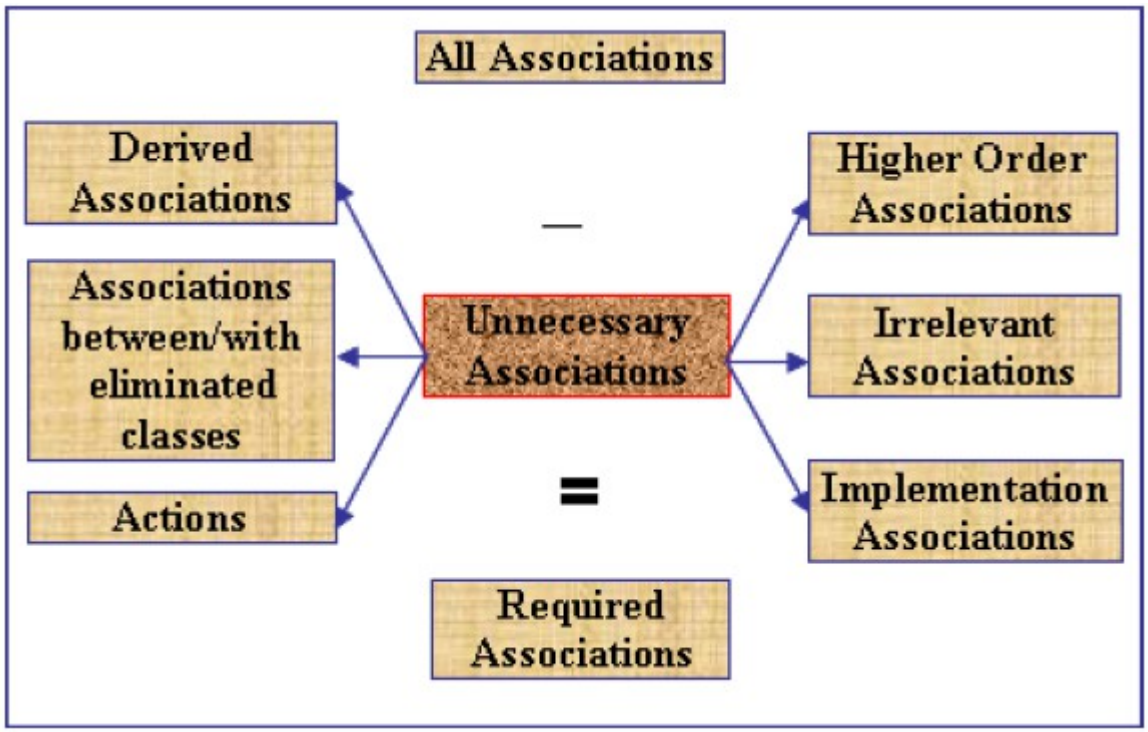
quả. Những liên hệ như thế là ví dụ tiêu tiểu của các chi tiết thực thi và không liên quan tới giai đoạn này.

Cần chú ý phân biệt giữa hành động và mối liên hệ. Người ta thường có xu hướng miêu tả hành động như là liên hệ, bởi cả liên hệ lẫn hành động đều được dẫn xuất từ những cụm từ mang tính động từ trong bản miêu tả yêu cầu. Các hành động đã được thể hiện sai thành liên hệ cũng cần phải được loại bỏ. Khi làm việc này, có thể áp dụng một nguyên tắc: liên hệ là nối kết mang tính tĩnh giữa các đối tượng, trong khi hành động chỉ là thao tác xảy ra một lần.

Hành động vì vậy nên được coi là Phương thức đối với một đối tượng chứ không phải quan hệ giữa các lớp.

Ví dụ với "Ban quản trị nhà băng đuổi việc một nhân viên", động từ “đuổi việc” thể hiện hành động. Trong khi đó với “Một nhân viên làm việc cho hãng" thì động từ “làm việc" miêu tả liên hệ giữa hai lớp nhân viên và hãng.

Trong khi cố gắng loại bỏ các liên hệ dư thừa, bạn sẽ thấy có một số liên hệ dư thừa đã "lẻn vào" mô hình của chúng ta trong giai đoạn thiết kế. Hình sau chỉ ra một số loại liên hệ dư thừa cần đặc biệt chú trọng.



Hình - Loại bỏ các liên hệ không cần thiết

2.3. Phụ thuộc và gói phụ thuộc

2.4. Quan hệ kết tập (Aggregations)

Kết tập là một trường hợp đặc biệt của liên hệ. Kết tập biểu thị rằng quan hệ giữa các lớp dựa trên nền tảng của nguyên tắc "một tổng thể được tạo thành bởi các bộ phận". Nó được sử dụng khi chúng ta muốn tạo nên một thực thể mới bằng cách tập hợp các thực thể tồn tại với nhau. Một ví dụ tiêu biểu của kết tập là chiếc xe ô tô gồm có bốn bánh xe, một động cơ, một khung gầm, một hộp số, v.v....

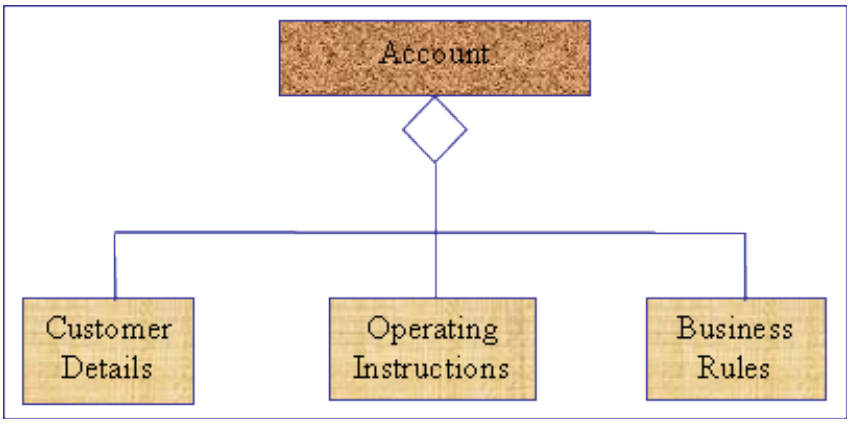
Quá trình ghép các bộ phận lại với nhau để tạo nên thực thể cần thiết được gọi là sự kết tập. Trong quá trình tìm lớp, kết tập sẽ được chú ý tới khi gặp các loại động từ “được tạo bởi", "gồm có", …. Quan hệ kết tập không có tên riêng. Tên ngầm chứa trong nó là "bao gồm các thành phần".

Kí hiệu kết tập:

Kí hiệu UML cho kết tập là đường thẳng với hình thoi (diamond) đặt sát lớp biểu thị sự kết tập (tổng thể).

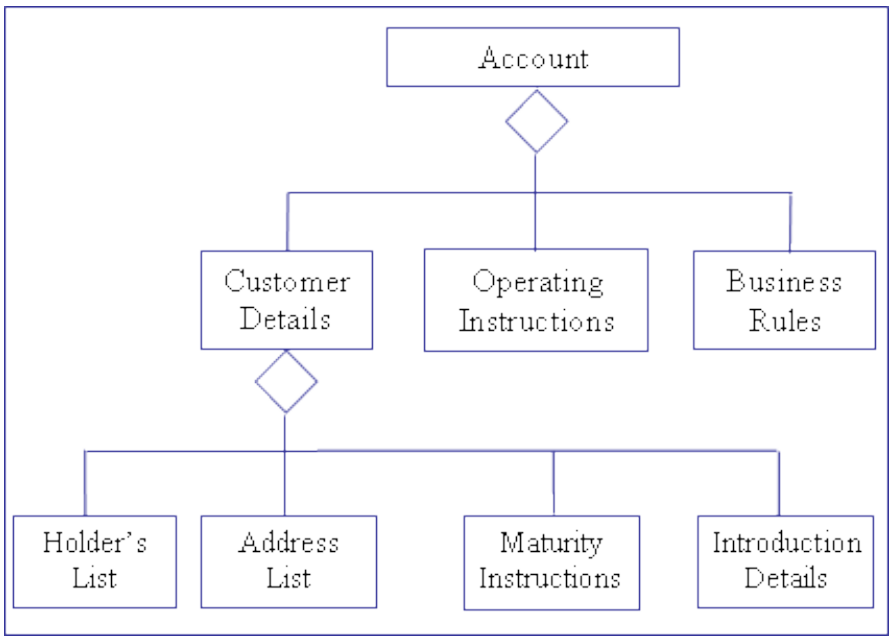
Một lớp tài khoản được tạo bởi các lớp chi tiết về khách hàng, các lệnh giao dịch đối với tài khoản cũng như các quy định của nhà băng.

Quan hệ trên có thể được trình bày như sau:



Quan hệ kết tập (1)

Mỗi thành phần tạo nên kết tập (tổng thể) được gọi là một bộ phận (aggregates). Mỗi bộ phận về phần nó lại có thể được tạo bởi các bộ phận khác.



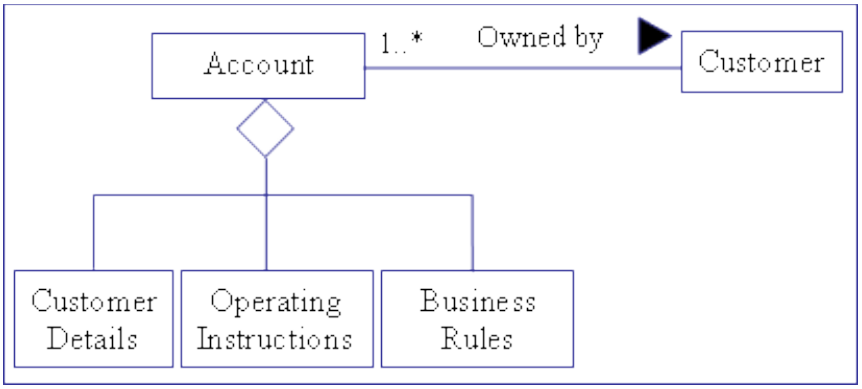
Quan hệ kết tập (2)

Trong trường hợp tài khoản kể trên, một trong các bộ phận của nó là các chi tiết về khách hàng. Các chi tiết về khách hàng lại bao gồm danh sách chủ tài khoản, danh sách địa chỉ, các quy định về kỳ hạn cũng như các chi tiết khác khi mở tài khoản.

Kết tập và liên hệ:

Khái niệm kết tập nảy sinh trong tình huống một thực thể bao gồm nhiều thành phần khác nhau. Liên hệ giữa các lớp mặt khác là mối quan hệ giữa các thực thể.

Quan sát hình sau:



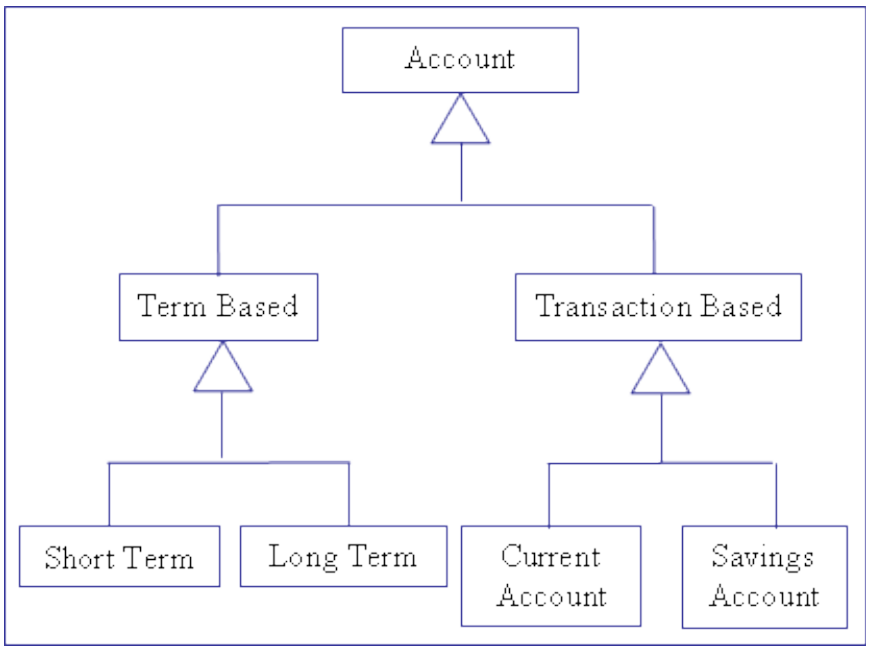
Kết tập và liên hệ

Một tài khoản được tạo bởi các chi tiết về khách hàng, các lệnh giao dịch đối với tài khoản cũng như các quy định của nhà băng. Khách hàng không phải là là bộ phận của tài khoản, nhưng có quan hệ với tài khoản.

Nhìn chung, nếu các lớp được nối kết với nhau một cách chặt chẽ qua quan hệ "toàn thể – bộ phận" thì người ta có thể coi quan hệ là kết tập. Không có lời hướng dẫn chắc chắn và rõ ràng cho việc bao giờ nên dùng kết tập và bao giờ nên dùng liên hệ. Một lối tiệm cận nhất quán đi kèm với những kiến thức sâu sắc về phạm vi vấn đề sẽ giúp nhà phân tích chọn giải pháp đúng đắn.

Khái quát hóa và chuyên biệt hóa (Generalization & Specialization)

Hãy quan sát cấu trúc lớp trong biểu đồ sau:



Chuyên biệt hoá (Specialization)

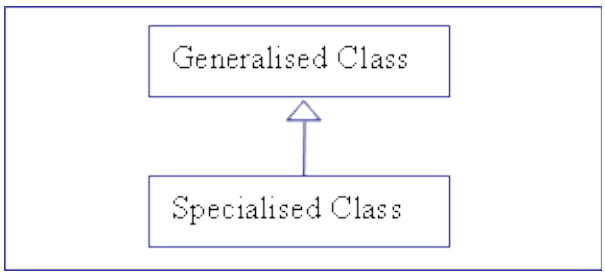
Trong hình trên, tài khoản là khái niệm chung của các loại tài khoản khác nhau và chứa những đặc tả cần thiết cho tất cả các loại tài khoản. Ví dụ như nó có thể chứa số tài khoản và tên chủ tài khoản. Ta có thể có hai loại tài khoản đặc biệt suy ra từ dạng tài khoản chung này, một loại mang tính kỳ hạn và một loại mang tính giao dịch. Yếu tố chia cách hai lớp này với nhau là các quy định chuyên ngành hay đúng hơn là phương thức hoạt động của hai loại tài khoản.

Tương tự như vậy, tài khoản đầu tư trung hạn và dài hạn lại là những khái niệm chuyên biệt của khái niệm tài khoản có kỳ hạn. Mặt khác, tài khoản bình thường và tài khoản tiết kiệm là những trường hợp đặc biệt của loại tài khoản giao dịch.

Loại cấu trúc lớp như thế được gọi là một cấu trúc hình cây hoặc cấu trúc phân cấp. Khi chúng ta dịch chuyển từ điểm xuất phát của cây xuống dưới, chúng ta sẽ gặp các khái niệm càng ngày càng được chuyên biệt hóa nhiều hơn. Theo con đường đi từ tài khoản đến tài khoản tiết kiệm, ta sẽ phải đi qua lớp tài khoản giao dịch. Lớp này tiếp tục phân loại các lớp chuyên biệt hóa cao hơn, tùy thuộc vào chức năng của chúng.

Kí hiệu khái quát hóa và chuyên biệt hóa

Trong biểu đồ trên, các lớp trong một cấu trúc cây được nối với nhau bằng một mũi tên rỗng , chỉ từ lớp chuyên biệt hơn tới lớp khái quát hơn.



Khái quát hóa

Quá trình bắt đầu với một lớp khái quát để sản xuất ra các lớp mang tính chuyên biệt cao hơn được gọi là quá trình **chuyên biệt hoá(Specialization)**

Chuyên biệt hóa: là quá trình tinh chế một lớp thành những lớp chuyên biệt hơn. Chuyên biệt hóa bổ sung thêm chi tiết và đặc tả cho lớp kết quả. Lớp mang tính khái quát được gọi là **lớp cha (superclass)**, kết quả chuyên biệt hóa là việc tạo ra các **lớp con (Subclass).**

Mặt khác, nếu chúng ta đi dọc cấu trúc cây từ dưới lên, ta sẽ gặp các lớp ngày càng mang tính khái quát cao hơn - Ví dụ từ lớp tài khoản tiết kiệm lên tới lớp tài khoản. Con đường bắt đầu từ một lớp chuyên biệt và khiến nó ngày càng mang tính khái quát cao hơn được gọi là quá trình **khái quát hóa (Generalization)**.

Lớp chuyên biệt ở đây được gọi là lớp con, trong ví dụ trên là tài khoản tiết kiệm, trong khi lớp khái quát kết quả được gọi là lớp cha.

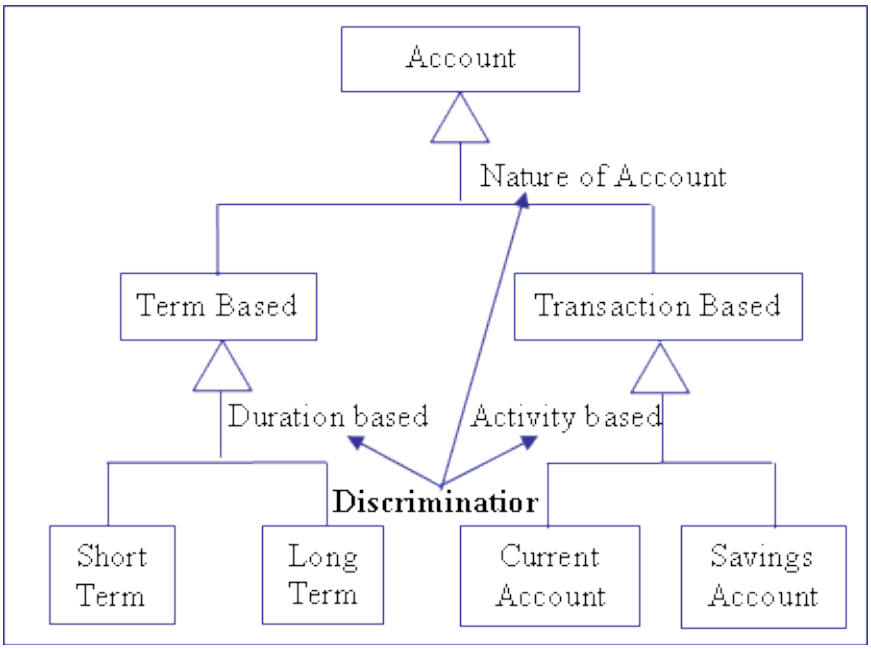
Chuyên biệt hóa và khái quát hóa là hai con đường khác nhau để xem xét cùng một mối quan hệ.

Một lớp là lớp con của một lớp này có thể đóng vài trò là một lớp cha của lớp khác.

Yếu tố phân biệt (Discriminatior)

Để tạo một cấu trúc phân cấp, cần phải có một số thuộc tính làm nền tảng cho quá trình chuyên biệt hóa. Thuộc tính đó được gọi là **yếu tố phân biệt** (Discriminator).

Với mỗi giá trị có thể gán cho yếu tố phân biệt trong lớp cha, ta sẽ có một lớp con tương ứng.



Yếu tố phân biệt (Discriminatior)

Trong hình trên, yếu tố phân biệt trong lớp tài khoản là "loại tài khoản". Chúng ta giả thiết rằng chỉ có hai loại tài khoản, một mang tính kỳ hạn và một mang tính giao dịch. Theo đó, ta phải tạo ra hai lớp con, một cho các tài khoản mang tính kỳ hạn và một cho các tài khoản mang tính giao dịch.

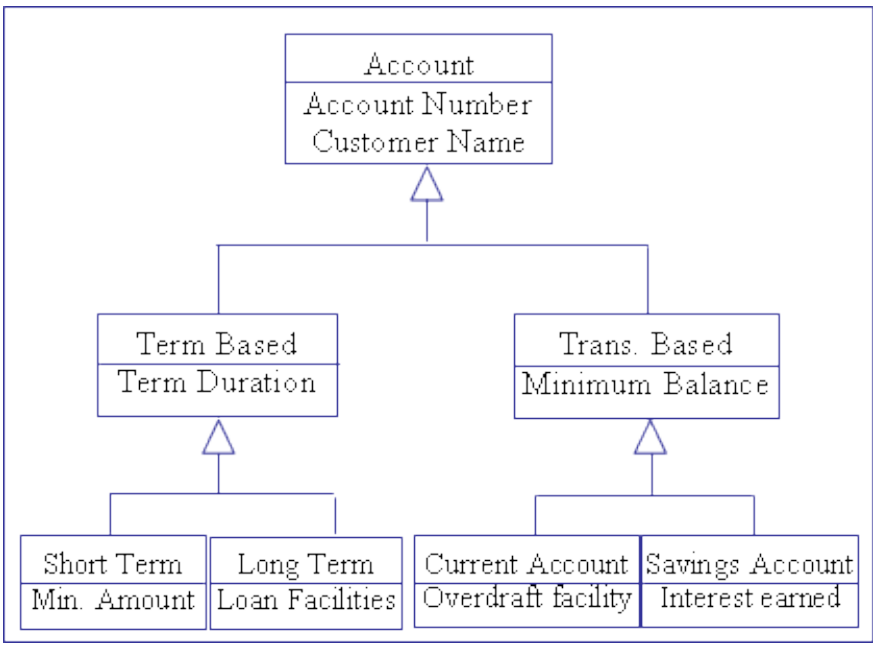
Trong mô hình đối tượng, không nhất thiết phải nêu bật yếu tố phân biệt. Yếu tố phân biệt luôn có mặt trong một cấu trúc phân cấp lớp cha/ con, dù có được nhấn mạnh trong mô hình đối tượng hay không. Mặc dầu vậy, để đảm bảo cho một mô hình được định nghĩa rõ ràng, trình bày yếu tố phân biệt vẫn luôn là công việc nên thực hiện.

Lớp trừu tượng

Quan sát cấu trúc trong hình trên, ta thấy lớp tài khoản sẽ không bao giờ được thực thể hóa, có nghĩa là hệ thống sẽ không bao giờ tạo ra các đối tượng thuộc lớp này. Nguyên nhân là vì lớp tài khoản mang tính khái quát cao đến mức độ việc khởi tạo lớp này sẽ không có một ý nghĩa nào đáng kể. Lớp tài khoản mặc dù vậy vẫn đóng một vai trò quan trọng trong việc khái quát hóa các thuộc tính sẽ được cần đến trong các lớp dẫn xuất từ nó. Những loại lớp như thế được dùng để cung cấp một cây cấu trúc lớp và không có sự tồn tại đầy đủ ý nghĩa trong một mô hình thật sự ngoài đời, chúng được gọi là **lớp trừu trượng (abstract class).**

Tạo lớp trừu tượng

Các lớp trừu trượng là kết quả của quá trình khái quát hóa. Hãy quan sát ví dụ cấu trúc lớp sau đây. Lớp tài khoản đứng đầu cây cấu trúc và được gọi là lớp căn bản. Lớp căn bản của một cây cấu trúc chứa những thuộc tính đã được khái quát hóa và có thể được áp dụng cho mọi lớp dẫn xuất từ nó. Trong quá trình khái quát hóa, các thuộc tính được dùng chung trong các lớp chuyên biệt được đưa lên lớp cha. Lớp cha về cuối được tạo bởi các thuộc tính chung của tất cả các lớp dẫn xuất từ nó. Những lớp cha dạng như vậy trong rất nhiều trường hợp sẽ mang tính khái quát tuyệt đối và sẽ không theo đuổi mục đích khởi tạo, chúng có lối ứng xử giống như một thùng chứa (container) cho tất cả các thuộc tính chung của các lớp dẫn xuất. Những lớp như thế trong trường hợp chung thường là kết quả ánh xạ của những danh từ trừu tượng, là hệ quả của phương pháp sử dụng các danh từ để nhận diện lớp .



Tạo lớp trừu tượng

Biểu đồ trên cho ta một ví dụ về khái quát hóa và các thuộc tính chung, nó chỉ ra nhiều lớp chuyên biệt. Chú ý rằng cứ theo mỗi mức chuyên biệt hóa lại có thêm các thuộc tính được bổ sung thêm cho các lớp, khiến chúng mang tính chuyên biệt cao hơn so với các lớp cha ở mức trừu tượng bên trên. Ví dụ lớp tài khoản có thuộc tính là số tài khoản và tên khách hàng. Đây là những thuộc tính hết sức chung chung. Tất cả các lớp dẫn xuất từ nó, dù là trực tiếp hay gián tiếp (ở các mức độ trừu tượng thấp hơn nữa), đều có quyền sử dụng các thuộc tính đó của lớp tài khoản. Các lớp tài khoản có kỳ hạn và tài khoản giao dịch là hai lớp chuyên biệt dẫn xuất từ lớp tài khoản. Chúng có những thuộc tính chuyên biệt riêng của chúng - ví dụ mức thời gian (duration) đối với lớp tài khoản có kỳ hạn và mức tiền tối thiểu đối với lớp tài khoản giao dịch – bên cạnh hai thuộc tính số tài khoản và tên khách hàng mà chúng thừa kế từ lớp tài khoản. Cũng tương tự như thế với tài khoản đầu tư ngắn hạn và tài khoản đầu tư trung hạn là các loại lớp thuộc tài khoản có kỳ hạn, tài khoản tiết kiệm và tài khoản bình thường là các loại lớp thuộc lớp tài khoản giao dịch.

Lớp cụ thể (concrete class)

Lớp cụ thể là những lớp có thể thực thể hóa. Như đã nói từ trước, các lớp cụ thể khi thực thể hóa được gọi là các đối tượng. Trong ví dụ trên, các lớp tài khoản đầu tư ngắn hạn và tài khoản đầu tư dài hạn có thể được thực thể hóa thành đối tượng. Tương tự đối với tài khoản tiết kiệm và tài khoản bình thường.

Tổng kết về phát triển cây cấu trúc

Cơ chế dùng chung thuộc tính và thủ tục sử dụng nguyên tắc khái quát hóa được gọi là **tính thừa kế (inheritance)**. Sử dụng tính thừa kế để tinh chế (refine) các lớp sẽ dẫn tới việc phát triển một cây cấu trúc. Nên phát hiện những ứng xử (behaviour) chung trong một loạt lớp rồi thể hiện nó thành một lớp cha. Sự khác biệt trong ứng xử của cùng một lớp sẽ dẫn tới việc tạo ra các lớp con.

Khi phát triển cây cấu trúc, hãy quan sát ứng xử của các lớp. Trong trường hợp có một liên hệ tồn tại từ một lớp cụ thể đến tất cả các lớp con của một lớp cha, nên dịch chuyển liên hệ này lên lớp cha.

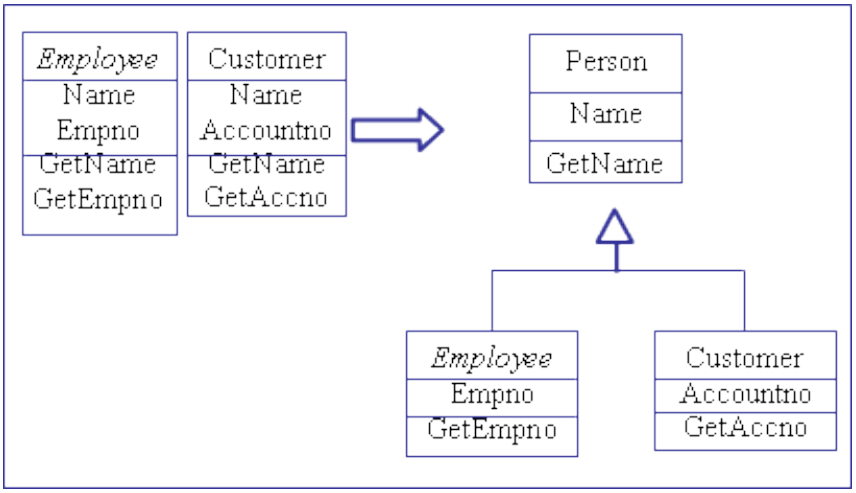
Nếu tồn tại một liên hệ giữa một lớp nào đó và một lớp cha, hãy chuyên biệt hóa và nâng cao cấu trúc để xác định xem liệu liên hệ này có được áp dụng cho tất cả các lớp con của lớp cha nọ hay không. Nếu có thì gán nó vào lớp cha, nếu không thì dịch xuống cho những lớp con phù hợp.

Trong khi tiến hành khái quát hóa, trọng tâm công việc là xác định các ứng xử chung trong một nhóm nhiều lớp chuyên biệt bậc trung. Khi đã xây dựng được một thủ tục hoặc một thuộc tính chung, nên kiểm tra lại xem chúng có thật sự là yếu tố chung của tất cả các lớp chuyên biệt trong phạm vi này. Khái quát hóa được áp dụng chỉ khi chúng ta có một tập hợp các lớp định nghĩa một loại đối tượng riêng biệt và có một số lượng lớn các ứng xử chung. Trọng tâm ở đây là tạo nên lớp cha chứa các ứng xử chung đó.

Khi chuyên biệt hóa, ta đi tìm các sự khác biệt trong ứng xử để tạo các lớp con thích ứng. Có nghĩa là ta xem xét một lớp tồn tại, kiểm tra xem có phải tất cả các ứng xử của nó đều có khả năng áp dụng cho mọi đối tượng. Nếu không, ta lọc ra ứng xử không phải lúc nào cũng cần thiết và chia trường hợp nó ra thành các lớp con. Trọng tâm của chuyên biệt hóa là tạo các lớp con.

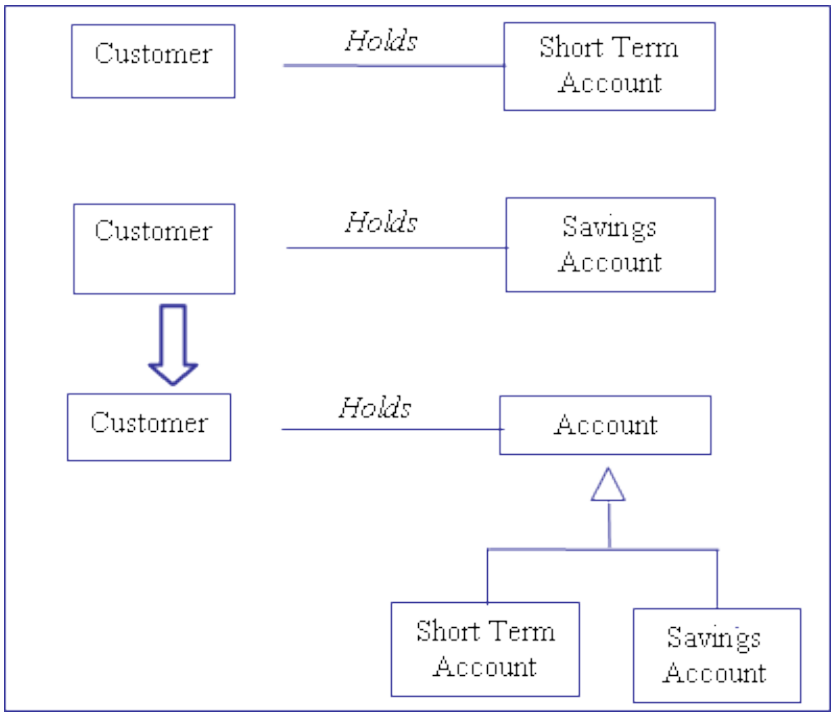
Với cơ chế thừa kế, một lớp con sẽ kế thừa mọi thuộc tính à thủ tục của tất cả các lớp cha của nó.

Hình sau làm rõ việc tạo cấu trúc lớp sử dụng tính khái quát.



Phát triển hệ thống lớp (1)

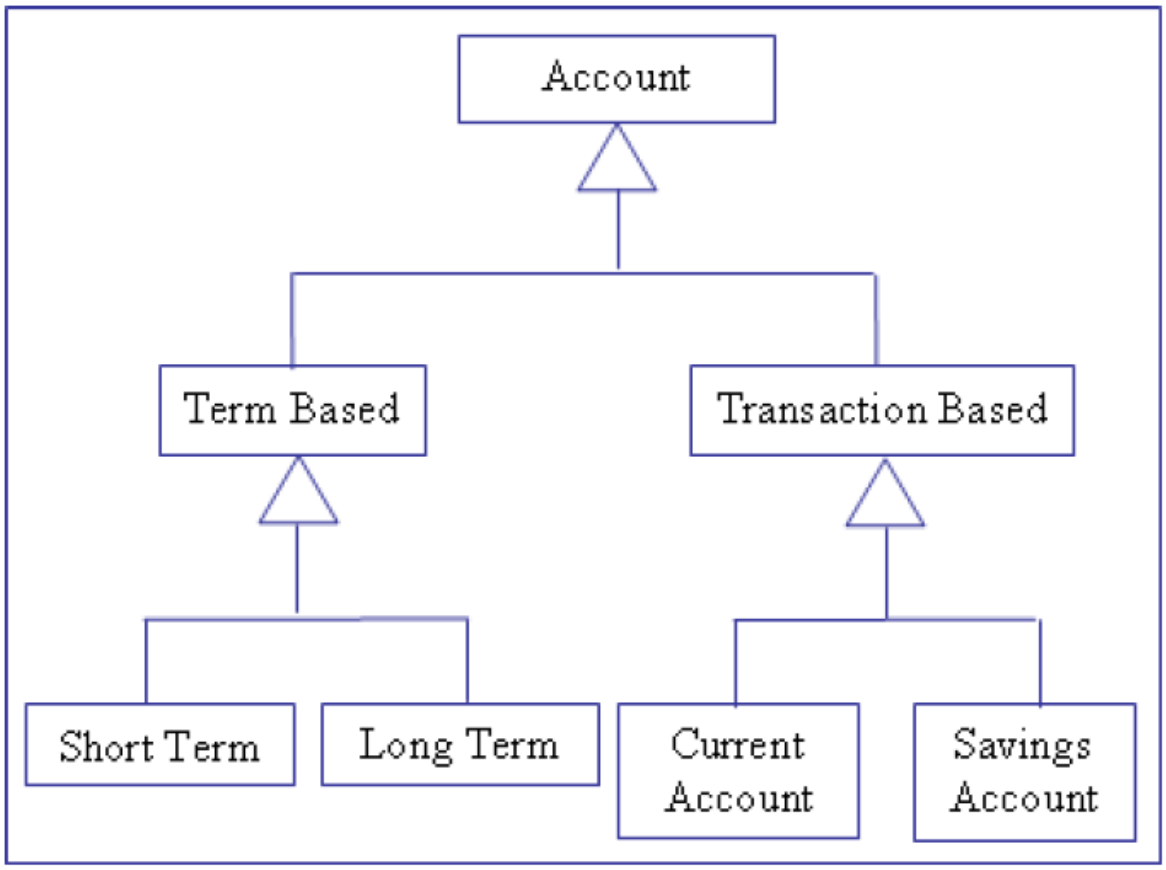
Thường xảy ra trường hợp tất cả các lớp con cùng tham gia vào một liên hệ hoặc kết tập. Trong trường hợp này nên tạo lớp cha định nghĩa liên hệ /kết tập đó. Hình sau giải thích thêm điểm này:



Phát triển hệ thống lớp (2)

2.5. Khái quát hóa – chuyên biệt hóa

Hãy quan sát cấu trúc lớp trong biểu đồ sau:



Hình - Chuyên biệt hoá (Specialization)

Trong hình trên, tài khoản là khái niệm chung của các loại tài khoản khác nhau và chứa những đặc tả cần thiết cho tất cả các loại tài khoản. Ví dụ như nó có thể chứa số tài khoản và tên chủ tài khoản. Ta có thể có hai loại tài khoản đặc biệt suy ra từ dạng tài khoản chung này, một loại mang tính kỳ hạn và một loại mang tính giao dịch. Yếu tố chia cách hai lớp này với nhau là các quy định chuyên ngành hay đúng hơn là phương thức hoạt động của hai loại tài khoản.

Tương tự như vậy, tài khoản đầu tư trung hạn và dài hạn lại là những khái niệm chuyên biệt của khái niệm tài khoản có kỳ hạn. Mặt khác, tài khoản bình thường và tài khoản tiết kiệm là những trường hợp đặc biệt của loại tài khoản giao dịch.

Loại cấu trúc lớp như thế được gọi là một cấu trúc hình cây hoặc cấu trúc phân cấp. Khi chúng ta dịch chuyển từ điểm xuất phát của cây xuống dưới, chúng

ta sẽ gặp các khái niệm càng ngày càng được chuyên biệt hóa nhiều hơn. Theo con đường đi từ tài khoản đến tài khoản tiết kiệm, ta sẽ phải đi qua lớp tài khoản giao dịch. Lớp này tiếp tục phân loại các lớp chuyên biệt hóa cao hơn, tùy thuộc vào chức năng của chúng.

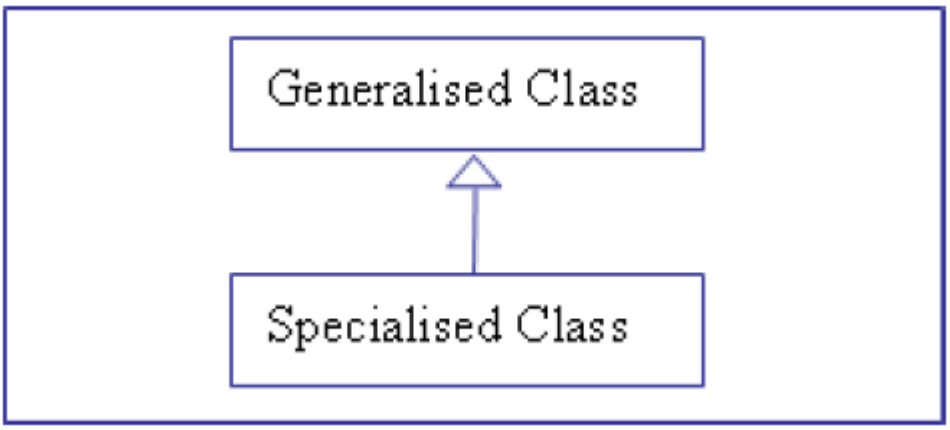
Trong hình trên, tài khoản là khái niệm chung của các loại tài khoản khác nhau và chứa những đặc tả cần thiết cho tất cả các loại tài khoản. Ví dụ như nó có thể chứa số tài khoản và tên chủ tài khoản. Ta có thể có hai loại tài khoản đặc biệt suy ra từ dạng tài khoản chung này, một loại mang tính kỳ hạn và một loại mang tính giao dịch. Yếu tố chia cách hai lớp này với nhau là các quy định chuyên ngành hay đúng hơn là phương thức hoạt động của hai loại tài khoản.

Tương tự như vậy, tài khoản đầu tư trung hạn và dài hạn lại là những khái niệm chuyên biệt của khái niệm tài khoản có kỳ hạn. Mặt khác, tài khoản bình thường và tài khoản tiết kiệm là những trường hợp đặc biệt của loại tài khoản giao dịch.

Loại cấu trúc lớp như thế được gọi là một cấu trúc hình cây hoặc cấu trúc phân cấp. Khi chúng ta dịch chuyển từ điểm xuất phát của cây xuống dưới, chúng ta sẽ gặp các khái niệm càng ngày càng được chuyên biệt hóa nhiều hơn. Theo con đường đi từ tài khoản đến tài khoản tiết kiệm, ta sẽ phải đi qua lớp tài khoản giao dịch. Lớp này tiếp tục phân loại các lớp chuyên biệt hóa cao hơn, tùy thuộc vào chức năng của chúng.

1- Kí hiệu khái quát hóa và chuyên biệt hóa

Trong biểu đồ trên, các lớp trong một cấu trúc cây được nối với nhau bằng một mũi tên rỗng, chỉ từ lớp chuyên biệt hơn tới lớp khái quát hơn.



Hình - Khái quát hóa

Quá trình bắt đầu với một lớp khái quát để sản xuất ra các lớp mang tính chuyên biệt cao hơn được gọi là quá trình chuyên biệt hoá (Specialization)

Chuyên biệt hóa: là quá trình tinh chế một lớp thành những lớp chuyên biệt hơn. Chuyên biệt hóa bổ sung thêm chi tiết và đặc tả cho lớp kết quả. Lớp mang tính khái quát được gọi là lớp cha (superclass), kết quả chuyên biệt hóa là việc tạo ra các lớp con (Subclass).

Mặt khác, nếu chúng ta đi dọc cấu trúc cây từ dưới lên, ta sẽ gặp các lớp ngày càng mang tính khái quát cao hơn - Ví dụ từ lớp tài khoản tiết kiệm lên tới lớp tài khoản. Con đường bắt đầu từ một lớp chuyên biệt và khiến nó ngày càng mang tính khái quát cao hơn được gọi là quá trình khái quát hóa (Generalization). Lớp chuyên biệt ở đây được gọi là lớp con, trong ví dụ trên là tài khoản tiết kiệm, trong khi lớp khái quát kết quả được gọi là lớp cha.

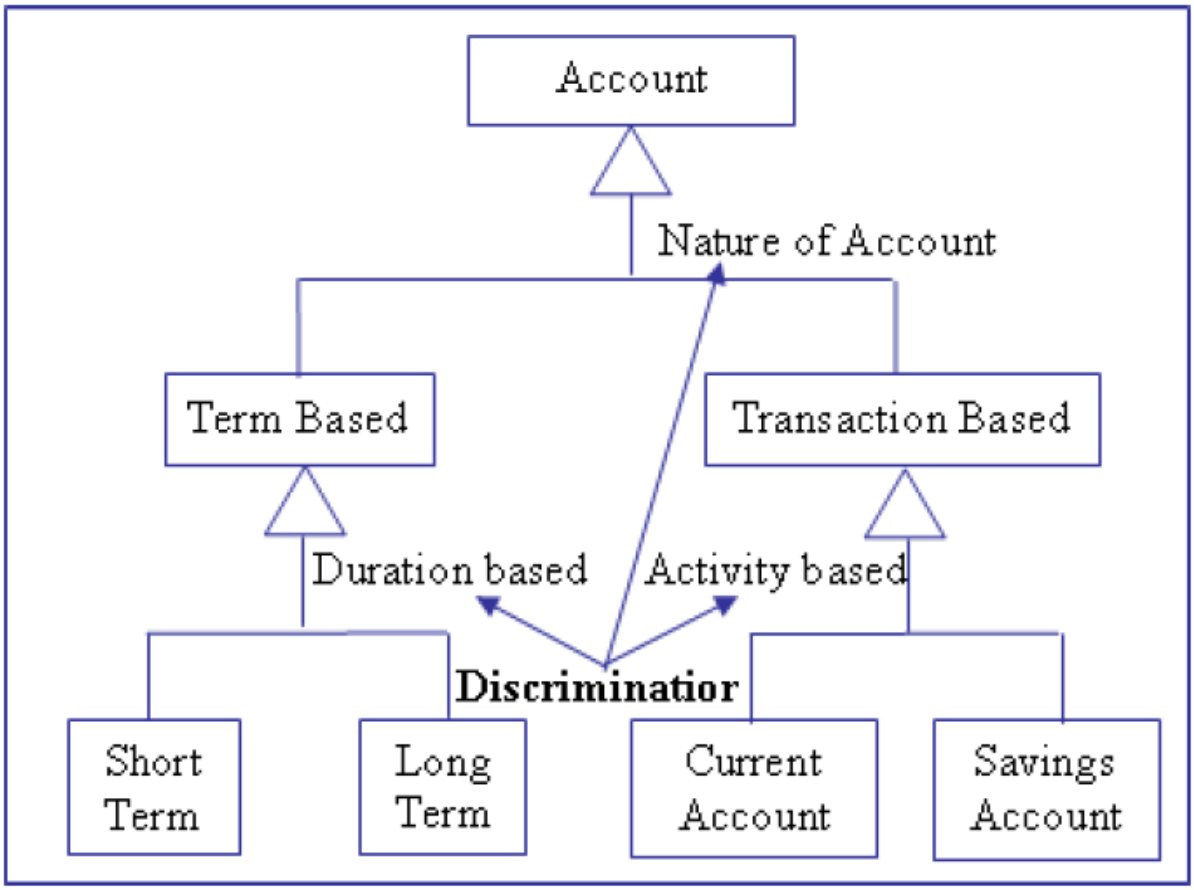
Chuyên biệt hóa và khái quát hóa là hai con đường khác nhau để xem xét cùng một mối quan hệ.

Một lớp là lớp con của một lớp này có thể đóng vài trò là một lớp cha của lớp khác.

2- Yếu tố phân biệt (Discriminatior)

Để tạo một cấu trúc phân cấp, cần phải có một số thuộc tính làm nền tảng cho quá trình chuyên biệt hóa. Thuộc tính đó được gọi là yếu tố phân biệt (Discriminator).

Với mỗi giá trị có thể gán cho yếu tố phân biệt trong lớp cha, ta sẽ có một lớp con tương ứng.



Hình - Yếu tố phân biệt (Discriminatior)

Trong hình trên, yếu tố phân biệt trong lớp tài khoản là "loại tài khoản". Chúng ta giả thiết rằng chỉ có hai loại tài khoản, một mang tính kỳ hạn và một mang tính giao dịch. Theo đó, ta phải tạo ra hai lớp con, một cho các tài khoản mang tính kỳ hạn và một cho các tài khoản mang tính giao dịch.

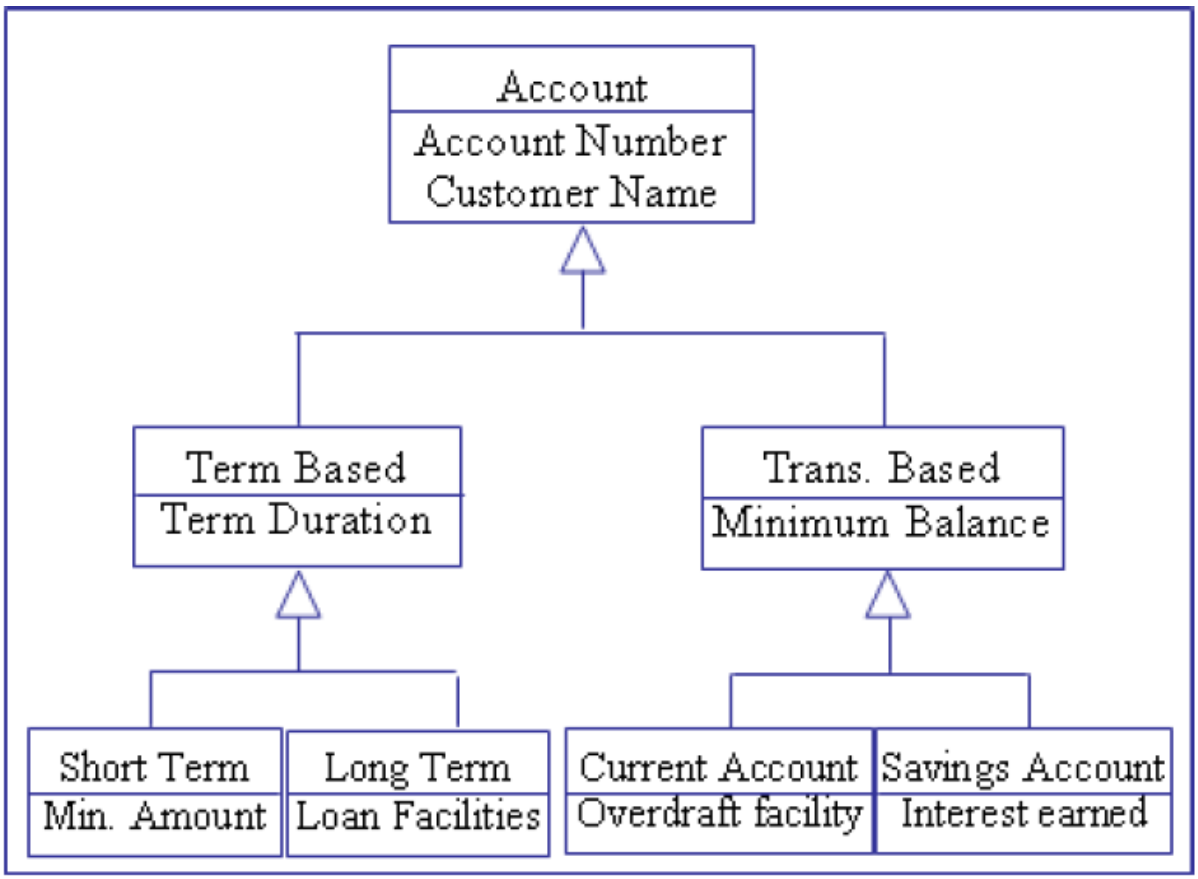
Trong mô hình đối tượng, không nhất thiết phải nêu bật yếu tố phân biệt. Yếu tố phân biệt luôn có mặt trong một cấu trúc phân cấp lớp cha/ con, dù có được nhấn mạnh trong mô hình đối tượng hay không. Mặc dầu vậy, để đảm bảo cho một mô hình được định nghĩa rõ ràng, trình bày yếu tố phân biệt vẫn luôn là công việc nên thực hiện.

2.1- Lớp trừu tượng

Quan sát cấu trúc trong hình trên, ta thấy lớp tài khoản sẽ không bao giờ được thực thể hóa, có nghĩa là hệ thống sẽ không bao giờ tạo ra các đối tượng thuộc lớp này. Nguyên nhân là vì lớp tài khoản mang tính khái quát cao đến mức độ việc khởi tạo lớp này sẽ không có một ý nghĩa nào đáng kể. Lớp tài khoản mặc dù vậy vẫn đóng một vai trò quan trọng trong việc khái quát hóa các thuộc tính sẽ được cần đến trong các lớp dẫn xuất từ nó. Những loại lớp như thế được dùng để cung cấp một cây cấu trúc lớp và không có sự tồn tại đầy đủ ý nghĩa trong một mô hình thật sự ngoài đời, chúng được gọi là lớp trừu trượng (abstract class).

2.2- Tạo lớp trừu tượng

Các lớp trừu trượng là kết quả của quá trình khái quát hóa. Hãy quan sát ví dụ cấu trúc lớp sau đây. Lớp tài khoản đứng đầu cây cấu trúc và được gọi là lớp căn bản. Lớp căn bản của một cây cấu trúc chứa những thuộc tính đã được khái quát hóa và có thể được áp dụng cho mọi lớp dẫn xuất từ nó. Trong quá trình khái quát hóa, các thuộc tính được dùng chung trong các lớp chuyên biệt được đưa lên lớp cha. Lớp cha về cuối được tạo bởi các thuộc tính chung của tất cả các lớp dẫn xuất từ nó. Những lớp cha dạng như vậy trong rất nhiều trường hợp sẽ mang tính khái quát tuyệt đối và sẽ không theo đuổi mục đích khởi tạo, chúng có lối ứng xử giống như một thùng chứa (container) cho tất cả các thuộc tính chung của các lớp dẫn xuất. Những lớp như thế trong trường hợp chung thường là kết quả ánh xạ của những danh từ trừu tượng, là hệ quả của phương pháp sử dụng các danh từ để nhận diện lớp.



Hình - Tạo lớp trừu tượng

Biểu đồ trên cho ta một ví dụ về khái quát hóa và các thuộc tính chung, nó chỉ ra nhiều lớp chuyên biệt. Chú ý rằng cứ theo mỗi mức chuyên biệt hóa lại có thêm các thuộc tính được bổ sung thêm cho các lớp, khiến chúng mang tính chuyên biệt cao hơn so với các lớp cha ở mức trừu tượng bên trên. Ví dụ lớp tài khoản có thuộc tính là số tài khoản và tên khách hàng. Đây là những thuộc tính hết sức chung chung. Tất cả các lớp dẫn xuất từ nó, dù là trực tiếp hay gián tiếp (ở các mức độ trừu tượng thấp hơn nữa), đều có quyền sử dụng các thuộc tính đó của lớp tài khoản. Các lớp tài khoản có kỳ hạn và tài khoản giao dịch là hai lớp chuyên biệt dẫn xuất từ lớp tài khoản. Chúng có những thuộc tính chuyên biệt riêng của chúng - ví dụ mức thời gian (duration) đối với lớp tài khoản có kỳ hạn và mức tiền tối thiểu đối với lớp tài khoản giao dịch – bên cạnh hai thuộc tính số tài khoản và tên khách hàng mà chúng thừa kế từ lớp tài khoản. Cũng tương tự như thế với tài khoản đầu tư ngắn hạn và tài khoản đầu tư trung hạn là các loại lớp

thuộc tài khoản có kỳ hạn, tài khoản tiết kiệm và tài khoản bình thường là các loại lớp thuộc lớp tài khoản giao dịch.

2.6. Làm việc với quan hệ

Chương 5. Biểu đồ kiến trúc vật lý và phát sinh mã trình

1. Biểu đồ kiến trúc vật lý (biểu đồ thành phần và triển khai)

1.1. Biểu đồ thành phần

Biểu đồ này cho ta cái nhìn vật lý của mô hình, đồng thời cho thấy các thành phần phần mềm trong hệ thống và quan hệ giữa chúng.

Một sơ đồ thành phần thì được chứa đựng hay tại mức đỉnh của mô hình hay bởi một gói. Cái này có nghĩa là sơ đồ sẽ miêu tả những thành phần và những gói mà cái sơ đồ đó chứa đựng.

Thành phần mã nguồn: thành phần mã nguồn có ý nghĩa vào thời điểm dịch chương trình

Thành phần nhị phân: thường là mã trình có được sau khi dịch thành phần mã nguồn.

Thành phần khả thi: thành phần thực hiện được là tệp chương trình thực hiện được (các tệp .EXE), là kết quả của liên kết các thành phần nhị phân thành phần thực hiện được biểu diễn đơn vị thực hiện được chạy trên bộ xử lý máy tính.

Sử dụng Rational để thiết kế:

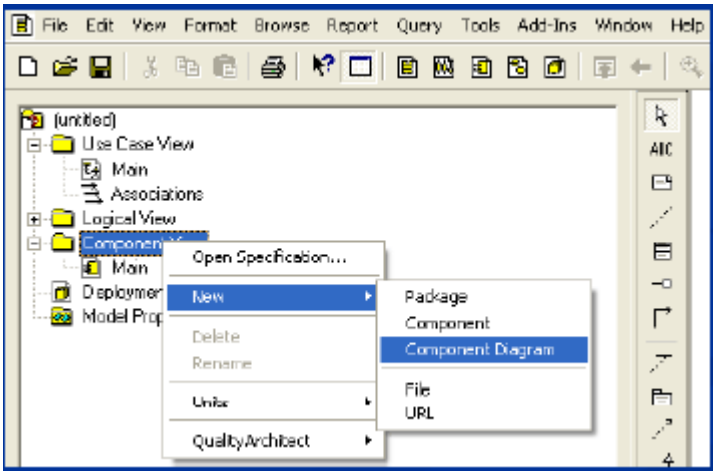
Tạo và hủy biểu đồ thành phần:

Tạo biểu đồ thành phần trong khung nhìn thành phần theo các bước như sau:

Trong Browser, nhấn chuột phải trên gói chứa component diagram.

Chọn thực đơn New Component Diagram.

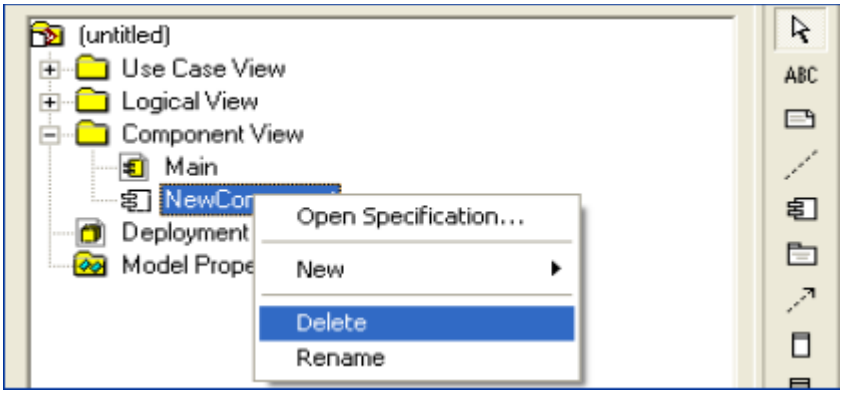
Nhập tên cho biểu đồ thành phần mới.



Hủy bỏ biểu đồ thành phần theo các bước sau:

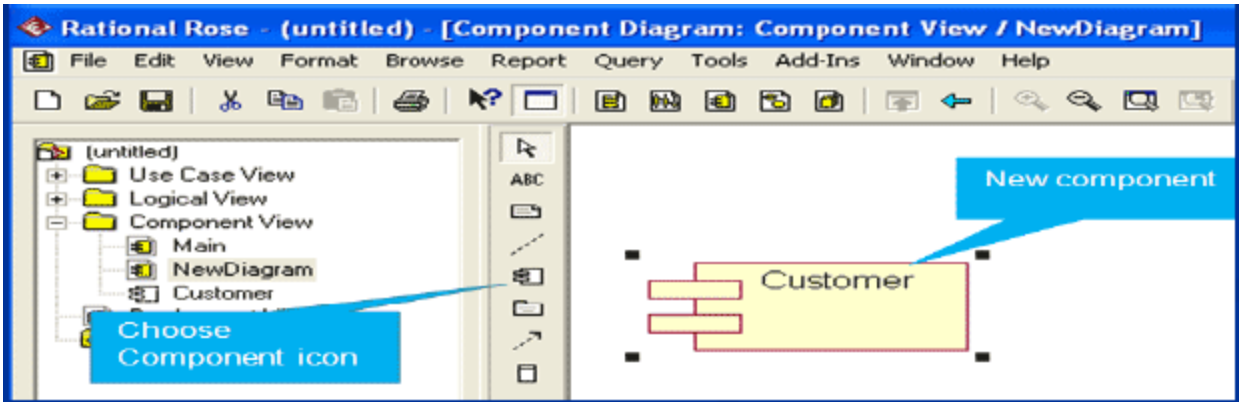
Trong Browser, nhấn chuột phải trên Component diagram.

Chọn thực đơn Delete.



Bổ sung và hủy bỏ thành phần:

Bổ sung thành phần vào biểu đồ theo các bước sau:



Hủy bỏ thành phần khỏi biểu đồ như sau:

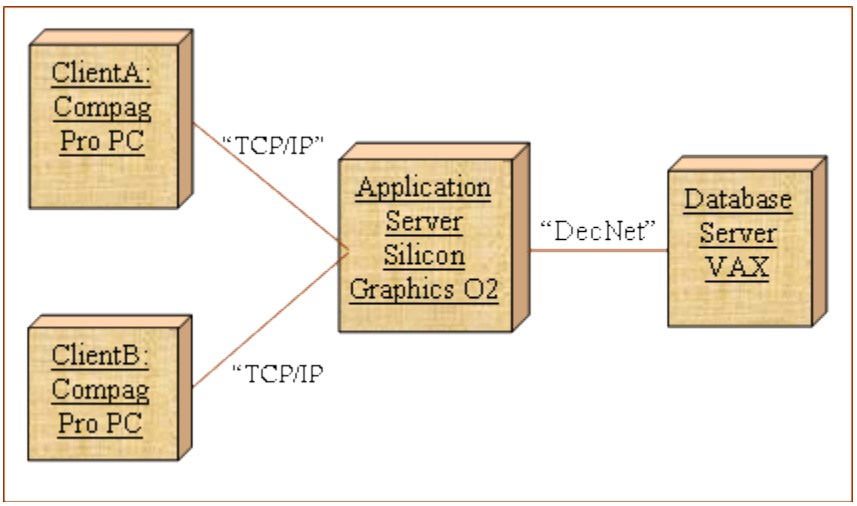
Nhấn phím phải trên thành phần trong browser.

Nhấn các phím Ctrl+D.

1.2. Biểu đồ triển khai

Biểu đồ triển khai chỉ ra kiến trúc vật lý của phần cứng cũng như phần mềm trong hệ thống. Bạn có thể chỉ ra từng máy tính cụ thể và từng trang thiết bị cụ thể (node) đi kèm sự nối kết giữa chúng với nhau, bạn cũng có thể chỉ ra loại của các mối nối kết đó. Bên trong các nút mạng (node), các thành phần thực thi được cũng như các đối tượng sẽ được xác định vị trí để chỉ ra những phần mềm nào sẽ được thực thi tại những nút mạng nào. Bạn cũng có thể chỉ ra sự phụ thuộc giữa các thành phần.

Biểu đồ triển khai chỉ ra hướng nhìn triển khai, miêu tả kiến trúc vật lý thật sự của hệ thống. Đây là một hướng nhìn rất xa lối miêu tả duy chức năng của hướng nhìn Use case. Mặc dù vậy, trong một mô hình tốt, người ta có thể chỉ tất cả những con đường dẫn từ một nút mạng trong một kiến trúc vật lý cho tới những thành phần của nó, cho tới lớp mà nó thực thi, cho tới những tương tác mà các đối tượng của lớp này tham gia để rồi cuối cùng, tiến tới một Use case. Rất nhiều hướng nhìn khác nhau của hệ thống được sử dụng đồng thời để tạo ra một lời miêu tả thấu đáo đối với hệ thống trong sự tổng thể của nó.



Hình - Một biểu đồ triển khai chỉ ra kiến trúc vật lý của hệ thống

1.3. Làm việc với hướng nhìn Components và Deployments

2. Phát sinh mã trình trong Rose

2.1. Sáu bước cơ bản để phát sinh mã trình bằng Rose

Có sáu bước cơ bản thực hiện để phát sinh mã chương trình:

1. Thiết lập các thuộc tính của mô hình
2. Kiểm tra mô hình
3. Tạo lập các thành phần
4. Gán các lớp vào thành phần
5. Chọn lớp, thành phần hay gói để phát sinh mã.
6. Phát sinh mã chương trình.

Không phải ngôn ngữ nào cũng cần đầy đủ các bước nêu trên. Ví dụ, khi phát sinh mã chương trình bằng C++ (hay Java) cho các lớp đã được thiết kế chi tiết thì chỉ cần thực hiện bước 5 và 6, hay bước một cũng không bắt buộc phải thực hiện. Tuy nhiên, về mặt qui trình công nghiệp nên thực hiện cả sáu bước trên.

Bước 1: Thiết lập các đặc tính của mô hình

Có nhiều đặc tính được sử dụng để phát sinh mã nguồn có thể gán cho lớp, vai trò(Role), thuộc tính, hàm và các thành phần khác của lớp.

+ Các đặc tính của lớp bao gồm: các toán tử tạo lập, huỷ tử, toán tử tạo lập nhân bản, phép đối sánh, các phương thức truy cập dữ liệu (get/set methods)

+ Các đặc tính của vai trò bao gồm: thiết lập các phương thức truy cập, lớp chứa.

+ Các đặc tính của phương thức bao gồm: những phép toán chung như phương thức abstract, virtual, static, friend, v.v.

Các đặc tính này điều khiển việc phát sinh mã chương trình tự động.

Đặc tính GenrateGetOperation trong C++ sẽ điều khiển để sinh mã các hàm có tiếp đầu ngữ là getX để đọc dữ liệu bị che giấu (khai báo private) trong lớp nếu nó được chọn. Trước khi phát sinh mã chương trình nên xem xét các đặc tính của mô hình và có thể bổ sung, hay thay đổi chúng nếu cần.

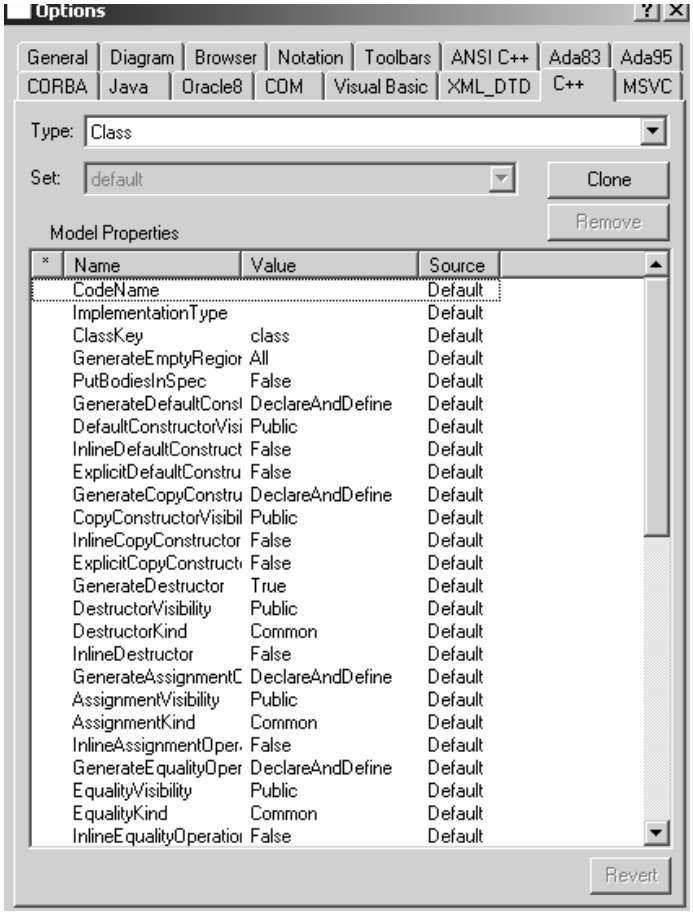
Để quan sát đặc tính của mô hình có thể chọn *Tools > Options (hoặc nhấn đúp vào Model Properties từ Browser)*, sau đó chọn ngôn ngữ lập trình, ví dụ chọn C++ như hình 1.

Có thể thay đổi Value và Source của các đặc tính bằng nhấn chuột vào những đặc tính cần thay đổi và lựa chọn các đại lượng tương ứng trong thực đơn đẩy xuống.

Tập đặc tính tạm thời: thay vì những đặc tính mặc định, ta có thể tạo lập đặc tính tạm thời để sử dụng. Để tạo lập tập đặc tính tạm thời cho C++, có thể chọn

Tools > Model Properties > Edit > C++ tab > Edit Set

sau đó nhấn nút Close trong cửa sổ Clone the Property Set và nhập tên mới cho tập đặc tính đó. Khi không còn cần đến tập đặc tính tạm thời thì có thể chọn Remove để loại bỏ nó.



Hình: Các đặc tính của mô hình để sinh mã cho lớp trong C++

Bước 2: Kiểm tra mô hình

Chức năng *Check Model* ở *Tool*s được thiết kế để kiểm tra sự nhất quán giữa các đơn thể khi mô hình của bạn được lưu trành nhiều đơn vị điều khiển. Kiểm tra mô hình để phát hiện những sai phạm, những điểm không thống nhất và các lỗi trong mô hình. Sau khi chọn *Tools* > *Check Model*, lỗi của mô hình sẽ được hiển thị ở cửa sổ Log. Các lỗi hay xảy ra là các thông điệp trong biểu đồ tương tác không được ánh xạ thành các phương thức của lớp tương ứng. Mục Access Violation sẽ tìm ra những vi phạm khi có những quan hệ giữa hai lớp ở hai gói khác nhau, nhưng hai gói đó lại không có quan hệ với nhau. Chọn *Report* > *Show Access Violation* để biết được những vi phạm đó.

Bước 3: Tạo lập thành phần hay mở biểu đồ thành phần

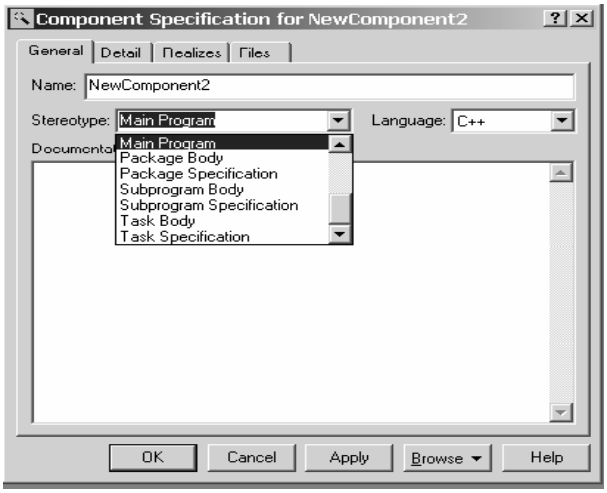
Ta có thể sinh mã trình cho từng lớp hoặc cho từng thành phần chứa một số lớp nhất định và các mẫu rập *khuôn (stereotype). Các thành phần cần thiết để ánh xạ các lớp trong mô hình sang ngôn ngữ lập trình đã xác định và các đơn thể phần mềm. Có nhiều loại thành phần như thành phần chứa mã nguồn, tệp thực thi (.exe), tệp thư viện*, v.v. Các lớp và các giao diện phải được gán vào một thành phần của một ngữ cài đặthoặc được gán vào một số thành phần của cùng một ngôn ngữ.

3.1 Tạo lập thành phần mới

1. Chọn Component view ở vùng Browser (thường hiển thị ở bên trái nhất).

2. Nhấn chuột phải để hiển thị thực đơn tắt (shortcut). Chọn *New* > *Package* hoặc New > Component rồi đặt tên cho chúng, nếu không chúng sẽ được đặt tên mặc định. Bạn chọn package nếu muốn tạo lập một gói mới chứa một số lớp để phát sinh mã chương trình.

3. Hoặc nhấn chuột phải để mở thực đơn tắt và nhấn đúp chuột để mở Open Specification. Trong đó bạn có thể chọn mẫu rập khuôn cho thành phần trong Stereotype và gán ngôn ngữ cho thành phần ở hộp Language (xem hình 2).



Hình: Tạo lập thành phần mới Để tạo ra header file thì chọn Package Specification, còn nếu muốn tạo ra tệp nội dung thì chọn Package Body ở Stereotype.

3.2 Mở biểu đồ và tạo lập các thành phần

Để tạo lập các thành phần trong biểu đồ thành phần chúng ta làm như sau:

1. Mở *Component Diagram*

2. Sử dụng biểu tượng *Component* trong thanh công cụ để bổ sung những loại thành phần mới vào biểu đồ.

Bước 4: Gán lớp, giao diện vào cho thành phần

Mỗi thành phần mã nguồn biểu diễn tệp mã nguồn của một hay một vài lớp. Trong C++, mỗi lớp được gán vào hai thành phần mã nguồn: tệp header *.h* và tệp chứa thân của lớp *.cpp*. Với C++, Rose có thể không qua bước này, khi phát sinh mã chương trình cho một lớp nó có thể yêu cầu gán trực tiếp vào thành phần lựa chọn.

Một lớp hoặc giao diện có thể được gán vào thành phần của ngôn ngữ lập trình hoặc gán vào một số thành phần của cùng một ngôn ngữ như sau:

1. Chọn lớp ở Browser, hoặc ở biểu đồ, và mở Open Specification của lớp đó.

2. Ở mục Components tab, chọn Show All Components.

3. Nhấn chuột phải ở thành phần tương ứng và kích vào Assign để gán lớp đã chọn vào thành phần đó.

Cũng có thể chọn thành phần ở Browser di chuột để kéo nó đến lớp tương ứng ở một biểu đồ ở Browser, hoặc mở Components tab ở phần đặc tả lớp (Class Specification). Sau khi lớp được gán vào cho thành phần thì ở Browser, tên của lớp đã được đính với tên của thành phần ở trong ngoặc đơn.

Để gán nhiều lớp (giao diện) vào một thành phần ta làm như sau:

1. Chọn thành phần ở Browser, hoặc ở biểu đồ thành phần, và mở Open Specification của thành phần đó.

2. Về Realizes tab, kích vào mục Show All Classes.

3. Đối với những lớp cần gán vào thành phần này thì nhấn chuột phải vào lớp đó và kich vào Assign để gán nó vào thành phần đã chọn.

4. Về mục General tab, có thể đặt tên mới cho thành phần ở hộp Name, chọn kiểu cho thành phần đó ở hộp Stereotype, và gán ngôn ngữ cài đặt cho thành phần này ở hộp Language.

Tương tự như trên, cũng có thể chọn từng lớp (giao diện) và di chuột để gán những lớp đã chọn tới thành phần tương ứng trong biểu đồ hoặc ở Browser.

Để gán một ngôn ngữ cho một lớp (tương tự đối với thành phần):

1. Mở phần đặc tả Open Specification của lớp (nhấn chuột phải ở tên lớp).

2. Về mục Components tab, kích vào mục Show All Components. Ở trường Language hiển thị ngôn ngữ đã được gán cho thành phần.

3. Nhấn chuột phải ở thành phần mà bạn định gán lớp vào và nhấn Assign.

Để tìm xem ngôn ngữ nào được gán cho một lớp ta thực hiện như sau:

1. Mở phần đặc tả Open Specification của lớp (nhấn chuột phải ở tên lớp).

2. Sẽ nhìn thấy ngôn ngữ lập trình đã được gán cho lớp đó ở trường Language khi chọn mục Component.

Bước 5: Chọn lớp, thành phần hay gói

Bạn có thể chọn lớp, thành phần hay gói ở Browser để phát sinh mã chương trình đồng thời. Nếu phát sinh mã chương trình từ gói ở Logical View ở biểu đồ lớp hay biểu đồ thành phần ở *Component View.*

Bước 6: Phát sinh mã chương trình

Nếu cài đặt Rose Enterprise, bạn có thể lựa chọn khá nhiều ngôn ngữ để cài đặt chương trình. Để hiển thị hay ẩn các lựa chọn ngôn ngữ, hãy chọn *Add-Ins* > *Add-In Manager*

Khi chọn lớp hay thành phần để phát sinh mã, hãy chọn ngôn ngữ tương ứng trong thực đơn trên.

Để sinh mã chương trình cho một lớp hay thành phần có thể thực hiện theo hai cách.

Cách thứ nhất thực hiện như sau:

1. Chọn lớp, thành phần, hay gói ở biểu đồ hoặc ở *Browser*.

2. Từ Tools chọn ngôn ngữ lập trình, ví dụ C++ rồi nhấn Code Generation. Mỗi lớp sẽ tạo ra hai tệp tương ứng như đã nói ở trên. Khi bạn muốn xem mã chương trình thì chọn lớp tương ứng, nhấn chuột phải, về C++ rồi nhấn Browse Header hoặc Browse Body tuỳ bạn muốn hiển thị header file hay thân của lớp.

Cách thứ hai thực hiện như sau:

1. Chọn lớp, hay thành phần ở biểu đồ hoặc ở Browser.

2. Nhấn chuột phải ở lớp, thành phần đã chọn, và chọn ngôn ngữ (C++) rồi nhấn *Code Generation.*

Nếu có lỗi (Errors) hay cảnh báo (Warning) trong quá trình sinh mã thì những lỗi, hay cảnh báo đó sẽ hiển thị ở cửa sổ Log.

Bạn cũng có thể sinh mã chương trình cho lớp hoặc thành phần mà không cần phải thực hiện những bước nêu trên.

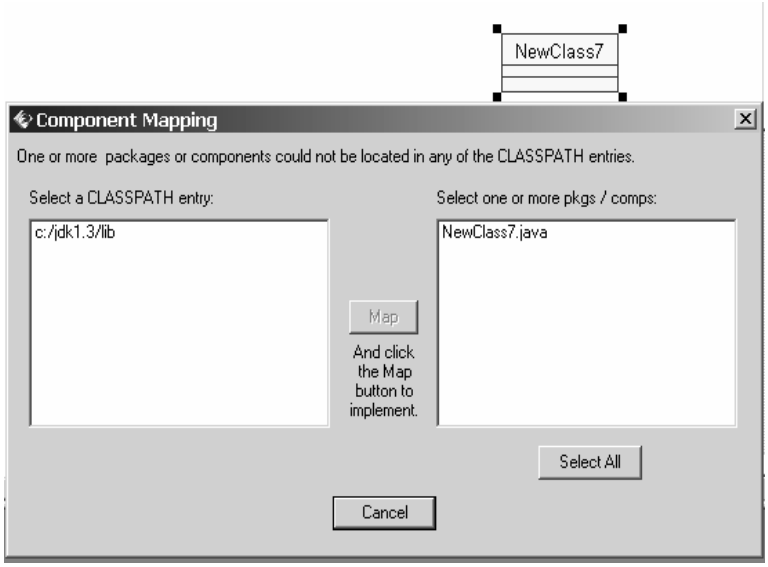
Để sinh mã chương trình trực tiếp ta thực hiện như sau:

1. Trước tiên chọn lớp (thành phần) ở biểu đồ.

2. Từ Tools, chọn ngôn ngữ, ví dụ C++.

3. Chọn Code Generation để phát sinh mã chương trình bằng C++ cho lớp đã chọn.

Nếu chọn Java thì trước khi phát sinh mã chương trình, bạn phải chọn thư mục và các lớp tương ứng rồi nhấn Map để ánh xạ những chương trình được phát sinh vào thư mục đã chọn như hình dưới.



Hình: Phát sinh mã bằng Java cho lớp NewClass7

Cái gì được phát sinh?

Khi phát sinh mã chương trình, Rose tập hợp các thông tin từ Logical View và

Component View. Rose chỉ phát sinh khung chương trình bao gồm.

Class: mọi lớp trong mô hình đều có thể được phát sinh mã chương trình,

Attribute: thuộc tính được xác định miền tác động, kiểu dữ liệu và giá trị mặc định,

Operation: các thao tác được khai báo kèm theo danh sách các tham số, kiểu của tham số, kiểu dữ liệu trả lại,

*Relationship*s: một số mối quan hệ sẽ được phát sinh mã tương ứng,

Component: mỗi thành phần sẽ được cài đặt trong một tệp mã nguồn tương ứng.

Rose không hướng vào thiết kế giao diện đồ hoạ. Do vậy, sau khi phát sinh khung chương trình như trên, nhiệm vụ tiếp theo của người phát triển là tập hợp những tệp nguồn đã được tạo lập, lập trình chi tiết cho những thao tác của lớp chưa được sinh mã và thiết kế giao diện đồ hoạ cho hệ thống ứng dụng.

2.2. Phát sinh mã trình C++

2.3. Ví dụ

Hãy cho biết những mệnh đề sau đúng hay sai (true / false), giải thích tại sao?

+ Kiến trúc vật lý thể hiện các lớp đối tượng, các quan hệ và sự cộng tác để hình thành chức năng của hệ thống.

+ Kiến trúc phổ biến chung hiện nay cho các hệ thống phần mềm là kiến trúc ba tầng: tầng giao diện, tầng tác nghiệp và tầng lưu trữ.

+ Biểu đồ thành phần mô tả các thành phần và sự phụ thuộc của chúng trong hệ thống.

+ Có thể chọn mô hình dữ liệu quan hệ để lưu trữ dữ liệu cho hệ thống được phân tích, thiết kế hướng đối tượng.

+ Tất cả các tên gọi, biến và kiểu dữ liệu của các lớp trong biểu đồ lớp được thiết kế phải được chuyển tương ứng sang mã chương trình trong các định nghĩa lớp ở ngôn ngữ lập trình đã được lựa chọn.

Xây dựng biểu đồ thành phần cho hệ thống "Đăng ký môn học".

Xây dựng biểu đồ thành phần cho hệ thống "Quản lý thư viện"

Thực hiện sinh mã tự động trong Rose cho các lớp ở hình 7.8.

Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về kiến trúc của hệ thống phần mềm.

Kiến trúc phần mềm là [(1)] về các hệ thống con, [(2)] và [(3)] giữa chúng. Các hệ thống con và [(2)] được xác định theo nhiều góc nhìn khác nhau để chỉ ra các

[(4)] và của hệ thống phần mềm. Kiến trúc hệ thống được chia thành hai loại:logic và vật lý.

Chọn câu trả lời:

a. thuộc tính chức năng .

b. mối quan hệ.

c. các thành phần.

d. một mô tả.

e. phi chức năng

Chọn từ danh sách dưới đây những thuật ngữ thích hợp để điền vào các chỗ [(…)] trong đoạn văn mô tả về biểu đồ thành phần.

Biểu đồ thành phần được xem như là tập các biểu tượng thành phần biểu diễn cho các [(1)] vật lý trong một [(2)]. Ý tưởng cơ bản của biểu đồ [(1)] là tạo ra cho những [(3)] và phát triển một bức tranh chung về các thành phần của [(2)].

Chọn câu trả lời:

a. hệ thống.

b. người thiết kế.

c. thành phần vật lý.

d. một mô tả.

3. Tham khảo: tham khảo ví dụ áp dụng