VIỆN KHOA HỌC VÀ CÔNG NGHỆ VIỆT NAM
VIỆN CÔNG NGHỆ THÔNG TIN

PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG

UNIFIED MODELING LANGUAGE

PGS.TS. Đặng Văn Đức
Email: dvduc@ioit.ac.vn

Xuất bản năm 2002

Chuyển sang ebook bởi

Sinh viên lớp DHTH4LT – Trường ĐH Công Nghiệp

10/2009

Mục lục

LỜI NÓI ĐẦU

Hệ thống tin học ngày càng phức tạp. Xu thế áp dụng phương pháp hướng đối tượng (phương pháp mới) thay cho phương pháp cấu trúc (phương pháp truyền thống) ngày càng phổ biến khi xây dựng các hệ thống phần mềm lớn và phức tạp. Hơn nữa, từ khi Ngôn ngữ mô hình hóa thống nhất (Unified Modeling Language – UML) được tổ chức OMG (*Object Management Group*) công nhận là chuẩn công nghiệp thì nó đã trở thành công cụ phổ dụng và hữu hiệu cho phương pháp mới này. Mục tiêu của tài liệu này nhằm giới thiệu các khái niệm cơ bản về tiếp cận hướng đối tượng và mô hình hóa hệ thống phần mềm theo phương pháp hướng đối tượng. Các khái niệm mới được mô tả, hướng dẫn thực hành thông qua ngôn ngữ chuẩn UML và phần mềm công cụ mô hình hóa nổi tiếng *Rational Rose* của *Raitonal Software Corporation*.

Phương pháp phân tích thiết kế hướng đối tượng được sử dụng rộng rãi tại các nước phát triển và bắt đầu được sử dụng tại một số đơn vị tin học tại Việt Nam. Tuy nhiên tài liệu bằng tiếng Việt về lĩnh vực này còn rất hiếm hoi, không đáp ứng nhu cầu hiện tại. Hơn nữa, nhận thức được tầm quan trọng của phương pháp mới này, một số trường đại học đã hình thành môn học liên quan đến vấn đề nói trên cho sinh viên, còn một số trường khác đang có kế hoạch đưa chủ đề này vào chương trình đào tạo chính khóa.

Chủ điểm của tài liệu được thể hiện dưới góc nhìn của người phát triển hệ thống phần mềm, không thể hiện dưới góc độ quan sát của nhà phương pháp luận. Lựa chọn này xuất phát từ thực tế là từ phương pháp luận hướng đối tượng dẫn đến việc ứng dụng nó vào xây dựng phần mềm cụ thể còn một khoảng cách xa vời và đầy khó khăn, đặc biệt với trình độ tin học hiện này nói chung còn chưa cao tại Việt Nam. Với quan điểm này, tài liệu được cấu trúc như sau:

Chương mở đầu trình bày khái quát về mô hình và mô hình hóa; các bước xây dưng hệ thống phần mềm và tầm quan trọng của phương pháp hướng đối tượng. Chương tiếp theo giới thiệu ngôn ngữ chuẩn công nghiệp UML, một công cụ hữu hiệu mô hình hóa hệ thống phần mềm. Trong các phần tiếp theo là trình bày kỹ thuật mô hình hóa, từ phân tích yêu cầu đến thiết kế hệ thống, kiến trúc hệ thống và cài đặt bằng ngôn ngữ lập trình. Chương cuối cùng là bài học thực nghiệm các kỹ thuật đã trình bày trong các chương trước vào bài toán cụ thể. Đặc biệt, trong mỗi chương tài liệu đều có phần thực hành trên phần mềm *Rational Rose* để độc giả có thể áp dụng ngày công cụ mới, kỹ thuật mới vào giải quyết vấn đề của riêng họ. Phần phụ lục trình bày một số mã trình trong một vài ngôn ngữ thông dụng tương ứng với các nhóm phần tử trong biểu đồ UML…

Hiện nay phần lớn các bạn sinh viên đại học năm cuối hoặc các kỹ sư tin học mới ra trường đều gặp khó khăn khi nhận nhiệm vụ xây dựng hệ thống phần mềm mới hay nâng cấp phần mềm có sẵn. Các bạn thường không biết bắt đầu từ đâu và làm như thế nào để có được phần mềm và phần mềm tốt, nói cách khác là còn thiếu phương pháp. Do vậy, quyển sách này có thể là tài liệu tham khảo tốt cho các bạn sinh viên và các kỹ sư tin học.

Quyển sách này được hình thành từ nội dung bài giảng của tác giả về chủ đề *Phát triển phần mềm hướng đối tượng băng UML* cho một số lớp sinh viên đại học. Trong quá trình biên soạn tác giả đã nhận được nhiều ý kiến đóng góp quí báu của các chuyên gia trong lĩnh vực này. Trước hết tác giả xin chân thành cảm ơn PGS. TSKH Nguyễn Xuân Huy, CN Ngô Trung Việt, TS Đặng Thành Phu, TS Đoàn Văn Ban, ThS Nguyễn Sơn Hải và các đồng nghiệp khác công tác tại Viện Công nghệ Thông tin, Trung tâm Khoa học Tự nhiên và Công nghệ Quốc gia đã đọc và cho ý kiến sửa chữa bản thảo.

Mặc dù đã rất cố gắng nhưng tài liệu này không tránh khỏi các sai sót. Tác giả xin chân thành cám ơn mọi ý kiến đóng góp của bạn đọc.

Địa chỉ liên lạc: Viện Công nghệ Thông tin, Trung tâm Khoa học Tự nhiên và Công nghệ Quốc gia. Email: dvduc@ioit.ncst.ac.vn

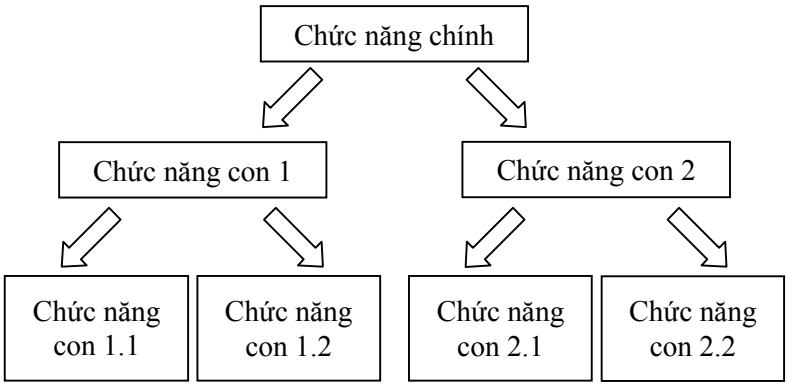
Hà nội, tháng 02 năm 2002

TÁC GIẢ

CHƯƠNG 1
MỞ ĐẦU

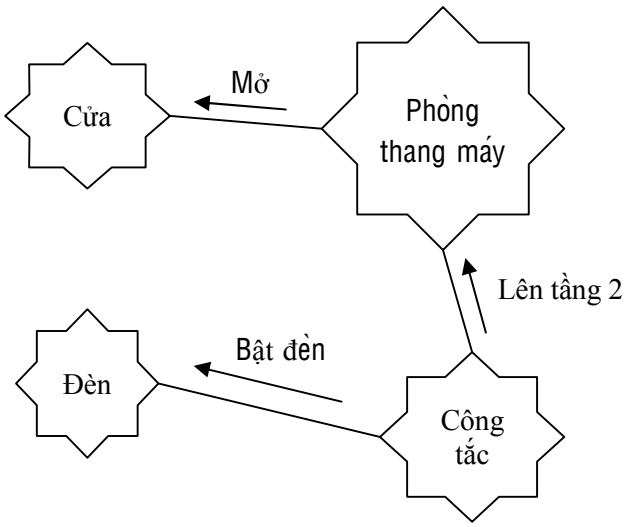
Phát triển phần mềm ngày càng trở nên phức tạp. Thay đổi giao diện từ các xâu ký tự sang giao diện đồ họa xu thế sự kiện; kiến trúc hệ thống đa tầng khách/chủ; cơ sở dữ liệu (CSDL) phân tán; Internet … làm tăng độ phức tạp của hệ thống phần mềm. Thách thức trong hai mươi năm tới của xây dựng hệ thống phần mềm không phải là tốc độ thực hiện chương trình, kinh phí hay sức mạnh của nó mà vấn đề sẽ là độ phức tạp (*Sun Microsystem*). Kẻ thù của chúng ta là độ phức tạp, ta phải loại bỏ chúng (*Jan Bean*). Vậy, loại bỏ độ phức tạp bằng cách nào? Các phương pháp tiếp cận hướng cấu trúc, tiệm cận hướng logic, tiếp cận hướng hướng đối tượng và tiếp cận hướng tác tử để có thể giải quyết vấn đề này nhưng ở mức độ khác nhau.

Tổng quát thì việc xây dựng phần mềm phải quan tâm đến tổ chức, các quan hệ và cấu trúc để hình thành được các hành vi phức tạp của hệ thống. Mọi việc khảo sát hệ thống phải được thực hiện với mức độ trừu tượng khác nhau, từ các chi tiết đến tổ chức tổng thể. Do vậy, xây dựng phần mềm là thực hiện dãy tương tác *chia nhỏ và hợp nhất*. Chia nhỏ để hiểu rõ vấn đề và hợp nhất để xây dựng hệ thống. Tiến trình chia nhỏ (tách) đã có truyền thống và tuân thủ các tiêu chí *chức năng*. Các chức năng của hệ thống được nhận diện, sau đó chúng được tách thành các chức năng con. Tiến trình này được thực hiện lặp đi lặp lại cho đến khi có được các thành phần đơn giản đến mức chúng được biểu diễn trực tiếp bằng các hàm hay thủ tục của ngôn ngữ lập trình (hình 1.1). Cách tiếp cận này được gọi là tiếp cận hướng chức năng (hay còn gọi là thủ tục, truyền thống). Người phát triển phần mềm sẽ tập trung vào các nhiệm vụ điều khiển và tách thuật toán lớn thành các thuật toán nhỏ. Khối chính để hình thành phần mềm ở đây là các hàm hay thủ tục.



Hình 1.1 Tiếp cận hướng chức năng

Kiến trúc phần mềm được cài đặt theo cách tiếp cận vừa mô tả trên sẽ phản ảnh các chức năng hệ thống. Tiếp cận trên cơ sở chức năng và cơ chế phân cấp chỉ cho lại kết quả mong muốn khi các chức năng được nhận biết đầy đủ và nó không được thay đổi theo thời gian. Thực tế lại không đúng như vậy vì trong rất nhiều trường hợp, phát triển phần mềm không bao giờ kết thúc hoàn toàn, luôn có cái gì đó phải sửa đổi, nâng cấp. Sửa đổi hay mở rộng hệ thống quá nhiều làm cho chương trình khác xa quan niệm ban đầu. Do đó cần phải có phương pháp mới cho khả năng làm chủ được độ phức tạp, giúp quản lý được chất lượng, độ tin cậy phần mềm ngày cả khi cấu trúc bị tách ra hay tiến hóa.



Hình 1.2 Tiếp cận hướng đối tượng

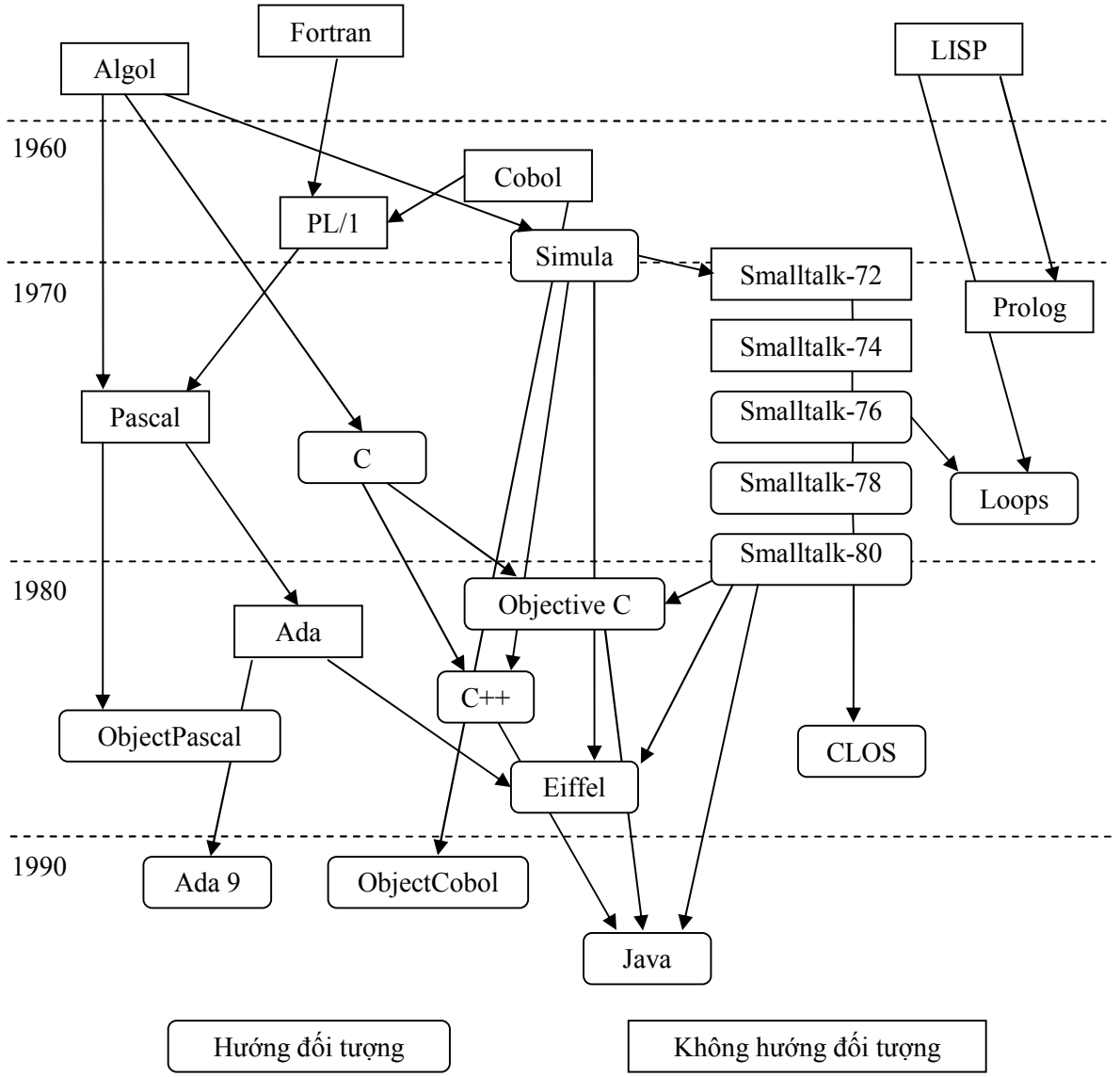
Quan điểm hướng đối tượng hình thành trên cơ sở tiếp cận hướng hệ thống, nó coi hệ thống như thực thể được tổ chức từ các thành phần mà chỉ được xác định khi nó thừa nhận và có quan hệ với các thành phần khác. Phương pháp tách vấn đề đang giải quyết để hiểu chúng ở đây không chỉ dựa trên cơ sở *cái hệ thống làm* mà còn dựa trên việc tích hợp *hệ thống là cái gì* với *hệ thống làm gì*. Thí dụ trên hình 1.2 mô tả các đối tượng và các quan hệ giữa các đối tượng của hệ thống thang máy. Theo cách tiếp cận này thì các chức năng hệ thống được biểu diễn thông qua cộng tác của các đối tượng; việc thay đổi, tiến hóa chức năng sẽ không ảnh hưởng đến cấu trúc tĩnh của phần mềm. Sức mạnh của tiếp cận hướng đối tượng là việc tách (chia) và nhập (thống nhất) được thực hiện nhờ tập phong phú các cơ chế tích hợp của chúng; khả năng thống nhất cao những cái nó đã được tách ra để xây dựng các thực thể phức tạp từ các thực thể đơn giản.

Tiếp cận hướng đối tượng đã tỏ rõ lợi thế khi lập trình các hệ thống phức tạp. Những người phát triển phần mềm nhận thấy rằng phát triển phần mềm hướng đối tượng sẽ cho lại phần mềm thương mại chất lượng cao: tin cậy, dễ mở rộng và dễ sử dụng lại, chạy trơn tru, phù hợp với yêu cầu người dùng đang mong đợi. Chúng còn cho khả năng hoàn thành phần mềm đúng kỳ hạn và không vượt quá khinh phí dự kiến ban đầu.

Ngoài phương pháp cấu trúc và phương pháp hướng đối tượng vừa đề cập trên đây, người ta còn thấy một số phương pháp khác được các nhà tin học đề xuất cho công nghệ phần mềm như tiếp cận hướng logic, tiếp cận hướng tác tử (agent)… Phương pháp tiếp cận hướng logic mô tả quan hệ "logic" giữa tính chất và thuộc tính của các sự kiện nhờ các cơ sở lập luận và luật suy diễn biểu diễn bởi những gợi ý trước. Tiếp cận này hình thành trên cơ sở ngôn ngữ lập trình *Prolog*. Phương pháp tiếp cận tác tử là kiểu mở rộng của tiếp cận hướng đối tượng. Gần đây nó được giới công nghiệp rất quan tâm. Các đơn vị cơ sở để phát triển hệ thống ở đây là các tác tử, nó là modun phần mềm. Đối tượng được khởi động khi nhận thông điệp từ bên ngoài và tác tử tích cực làm việc hay không phụ thuộc vào môi trường chứa nó. Tuy nhiên, phương pháp hướng đối tượng đang ngày cành được nhiều người sử dụng hơn cả.

1.1 LỊCH SỬ HƯỚNG ĐỐI TƯỢNG

Khái niệm hướng đối tượng hình thành từ ngôn ngữ lập trình *Simula*, nhưng nó chỉ trở nên quen thuộc khi xuất hiện ngôn ngữ *C++* và *SmallTalk* vào cuối những năm 80 của thế kỳ XX. Sơ đồ trong hình 1.3 chỉ ra thời gian xuất hiện và quan hện của các ngôn ngữ lập trình [OES00]. Trong hình vuông với góc tròn là tên các ngôn ngữ lập trình hướng đối tượng.



Hình 1.3 Các ngôn ngữ lập trình

Khi các ngôn ngữ hướng đối tượng được sử dụng rộng rãi thì nhu cầu có phương pháp phát triển phần mềm hướng đối tượng trở nên cấp bách. Vào đầu những năm 90 của thế kỷ XX đã xuất hiện các phương pháp hướng đối tượng sau đây: Phương pháp *Booch*, *OMT* (*Object Modeling Technique*), *OOSE* (*Object Oriented Software Engineering*)/*Objectory*, *Fusion* và *Coad/Yourdon*. Mỗi phương pháp có ký pháp, tiến trình và công cụ hỗ trợ riêng. Chúng đều có ưu điểm và nhược điểm riêng. Người sử dụng rất khó khăn để chọn cho mình một phương pháp phù hợp. Do nhận biết đươc các vấn đề này, vào năm 1994 các tác giả của các phương pháp này đã hợp tác nhằm tạo ra phương pháp mới. Bắt đầu là sự thống nhất phương pháp *Booch* với OMT-2 của *Rumbagh* để hình thành *Unified Method 0.8* tại *Rational Rose Corporation*. Tháng 6 năm 1995, *Ivar Jacobson* (tác giả của *OOSE/Objectory*) gia nhập với họ. Từ thời điểm này, nhóm phát triển phương pháp hướng đối tượng nói trên cho rằng nhiệm vụ của họ là tạo ra ngôn ngữ mô hình hóa thống nhất cho cộng đồng hướng đối tượng. Do vậy, họ đã đổi tên công việc của mình thành *Unified Modeling Language – UML* (Ngôn ngữ mô hình hóa thông nhất). *Booch*, *Rumbaugh* và *Jacobson* đã đưa ra nhiều phiên bản UML, trong đó phiên bản UML 0.9 xuất hiện năm 1995, UML xuất hiện vào năm 1997. Phần lớn UML được xây dựng trên nền tảng của các phương pháp *Booch*, OMT và OOSE, nhưng UML còn bao gồm cả các khái niệm có nguồn gốc từ các phương pháp khác như *David Harel*, *Gamma – Helm – Johnson – Vlissides* và *Fusion*. UML còn là kết quả của sự đóng góp từ các hãng lớn như Digital Equipment Corporation (DEC), Hewlett –

Packard (HP), I-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software Corporation, Texas Instrument, Taskon, ObjectTime và Unisys. Phiên bản UML 1.1 đã được đệ trình lên OMG (Object Management tháng 11-1997. Phiên bản UML 1.3 xuất hiện vào năm 1999. Phiên bản UML 1.4 xuất hiện vào tháng 2-2000.

1.2 MỘT SỐ KHÁI NIỆM CƠ BẢN

Phần này trình bày một số khái niệm cơ bản áp dụng trong phân tích và thiết kế hướng đối tượng.

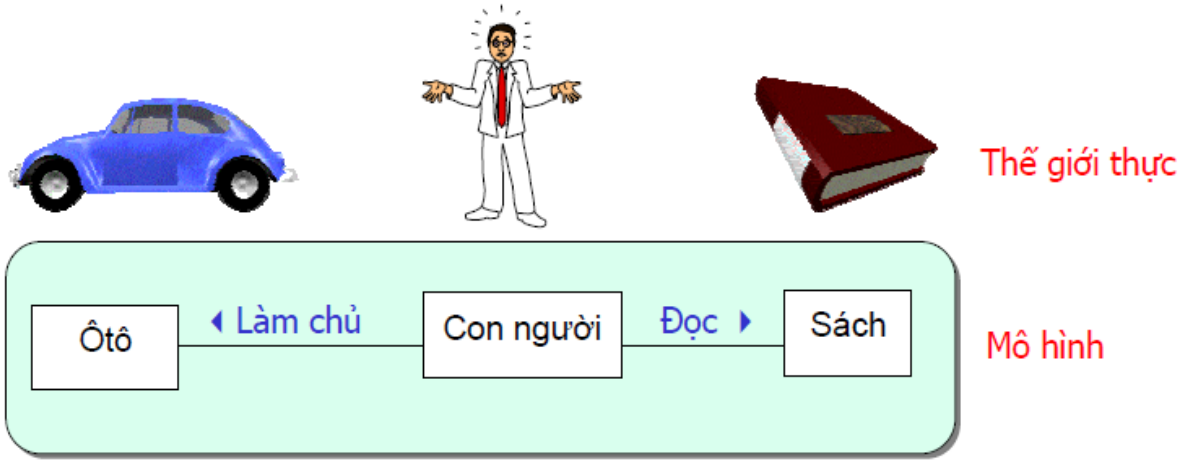
Phương pháp (*method*). Phương pháp (hay phương thức) là cách thức cấu trúc các suy nghĩ và hành động của con người. Nó cho biết chúng ta phải làm cái gì, làm như thế nào, làm khi nào và tại sao phải làm như vậy để hình thành hệ thông phần mềm.

Đối thương (*object*). Theo nghĩa thông thường thì đối tượng là người, vật hay hiện tượng mà con người nhằm vào trong suy nghĩ, trong hành động [PHE96]; là bất kỳ cái gì nhìn thấy và sờ mó được. Trong phương pháp hướng đối tượng thì đối tượng là trừu tượng cái gì đó trong lĩnh vực vấn đề hay trong cài đặt của nó; phản ảnh khả năng hệ thống lưu giữ thông tin về nó và tương tác với nó; gói các giá trị thuộc tính và các dịch vụ (phương thức, phương pháp) [OCAD91].

Lớp (*class*). Theo nghĩa thông thường thì lớp là nhóm của nhiều người hay vật có tính tương tự nhất định hay đặc điểm chung (từ điển Webster's). Trong phương pháp hướng đối tượng thì lớp là mô tả một hay nhiều đối tượng, mô tả tập thống nhất các thuộc tính và phương thức. Nó còn có thể mô tả cách tạo đối tượng mới trong lớp như thế nào [COAD91].

Trừu tượng (*abstract*). Trừu tượng là nguyên lý bỏ qua những khía cạnh của chủ thể (*subject*) không liên quan đến mục đích hiện tại để tập trung đầy đủ hơn vào các khía cạnh còn lại. Như vậy có thể nói rằng *trừu tượng là đơn giản hóa thế giới thực một cách thông minh*. Trừu tượng cho khả năng tổng quát hóa và ý tưởng hóa vấn đề đang xem xét. Chúng loại bỏ đi các chi tiết dư thừa mà chỉ tập chung và các điểm chính, cơ bản [LIBE98].

Mô hình (*model*). Nếu phải xây ngôi nhà thì chắc chắn không làm đơn giản là cứ mua gạch, sắt thép về lắp dần đến khi hình thành nhà ở, mà phải có kế hoạch chi tiết và thiết kế trước. Nói cách khác là phải xây dựng mô hình. Tương tự như vậy, trong lĩnh vực phần mềm, mô hình là kế hoạch chi tiết của hệ thống, nó giúp ta lập kế hoạch trước khi xây dựng hệ thống. Mô hình giúp ta khẳng định tính đúng đắn của thiết kế, phù hợp yêu cầu, hệ thống vẫn giữ vững khi yêu cầu người dùng thay đổi. Phương pháp hướng đối tượng xem một phần của thế giới thực như các đối tượng. Máy điện thoại, xe ô tô, con người … là các đối tượng. Các đối tượng này lại bao gồm nhiều đối tượng khác như xe ô tô có tay lái, bánh xe, … con người có tay, chân, mắt, mũi … Đối tượng thế giới thực có thế có cấu trúc rất phức tạp, làm cho con người khó nhận thức được chúng. Trong cuộc sống hàng ngày ta đơn giản hóa đối tượng khi suy nghĩ, hay nói cách khác ta làm việc với mô hình. Thí dụ, quả địa cầu là mô hình của trái đất. Mô hình chỉ lựa chọn một vài khía cạnh có ý nghĩa cho việc thực hiện công việc cụ thể nào đó. Chúng cho phép dự đoán, hiểu các khía cạnh của vấn đề đang quan tâm. Thí dụ khi làm việc với đối tượng con người: trong sổ lương có tên, số bảo hiểm xã hội và tiền lương. Nhưng trong sổ điện thoại lại chỉ có tên, số điện thoại và địa chỉ. Vậy, mô hình là bức tranh hay mô tả của vấn đề đang được cố gắng giải quyết hay biểu diễn. Mô hình còn có thể là mô tả chính giải pháp. Trong phát triển phần mềm, thay cho đối tượng thực, ta sẽ làm việc với biểu tượng (hình 1.4). Tiến trình phát triển phần mềm là làm giảm một số đặc trưng của đối tượng để hình thành mô hình: làm giảm độ phức tạp bằng mô hình trừu tượng.



Hình 1.4 Mô hình thế giời thực

Phương pháp luận (*methodology*). Phương pháp luận mô tả cách thức suy nghĩ về phần mềm và phát triển phần mềm. Nó bao gồm ngôn ngữ mô hình hóa *metamodel* (mô hình của mô hình) và tiến trình. Phương pháp luận khác với phương pháp. Phương pháp luận là nghiên cứu phương pháp. *Metamodel* mô tả hình thức các phần tử mô hình, cú pháp và ngữ nghĩa của các ký pháp trong mô hình.

Lĩnh vực vấn đề (*domain problem*). Mục tiêu của tiếp cận hướng đối tượng là mô hình hóa các đặc tính tĩnh và động của môi trường, nơi xác định yêu cầu phần mềm. Môi trường này được gọi là lĩnh vực vấn đề. Vấn đề là câu hỏi đặt ra để giải quyết hoặc xem xét. Lĩnh vực là không gian (vùng) của các hoạt động hoặc ảnh hưởng. Nó là vùng tác nghiệp hay kinh nghiệm của con người trong đó phần mềm được sử dụng. Vậy, lĩnh vực vấn đề là vùng mà ta đang cố gắng xem xét. Thí dụ của lĩnh vực vấn đề có thể là tài chính, giáo dục,…

Phân tích. Phân tích là tách, chia nhỏ tổng thể thành các phần để tìm ra đặc tính, chức năng, quan hệ… của chúng (từ điển Webster's). Khái niệm phân tích trong tiếp cận hướng đối tượng là thực hiện nghiên cứu lĩnh vực vấn đề, dẫn tới đặc tả hành vi quan sát từ ngoài và các thông báo nhất quán, hoàn chỉnh, khả thi của những cái cần [COAD91]. Phân tích hướng đối tượng tập trung vào tìm kiếm, mô tả các đối tượng (khái niệm) trong lĩnh vực vấn đề. Thí dụ hệ thống thư viện có khái niệm Sách, Thư viện …

Thiết kế. Là tập tài liệu kỹ thuật toàn bộ, gồm có bản tính toán, bản vẽ… để có thể theo đó mà xây dựng công trình, sản xuất thiết bị, làm sản phẩm… [PHE96]. Khái niệm phân tích trong tiếp cận hướng đối tượng là thực hiện đặc tả các hành vi bên ngoài, bổ sung chi tiết nếu cần thiết để cài đặt hệ thống trên máy tính, bao gồm tương tác người - máy, quản lý nhiệm vụ, quản lý dữ liệu [COAD91]. Thiết kế hướng đối tượng tập trung vào xác định đối tượng phần mềm logic sẽ được cài đặt bằng ngôn ngữ hướng đối tượng.

Xây dựng (lập trình) hướng đối tượng: là thiết kết các modun sẽ được cài đặt. Thí dụ lớp Book sẽ được cài đặt bằng C++, Java … như thế nào.

Mô hình hóa (*modeling*). Khái niệm *mô hình hóa* thường được sử dụng đồng nghĩa với phân tích, đó là việc thực hiện tách hệ thống thành các phần tử đơn giản để dễ hiểu. Trong khoa học máy tính, mô hình hóa bắt đầu từ mô tả vấn đề, sau đó là mô tả giải pháp vấn đề. Các hoạt động này còn được gọi là *phân tích và thiết kế*. Khi thu thập yêu cầu cho hệ thông, ta phải tìm ra nhu cầu tác nghiệp của người dùng và ánh xạ chúng thành yêu cầu phần mềm sao cho đội ngũ phát triển phần mềm hiểu và sử dụng được chúng. Tiếp theo là khả năng phát sinh mã trình từ các yêu cầu này, đồng thời đảm bảo rằng các yêu cầu phải phù hợp với mã trình vừa phát sinh và dễ dàng chuyển đổi mã trình ngược lại thành yêu cầu. Tiến trình này được gọi là mô hình hóa.

Mô hình hóa trực quan. Mô hình hóa trực quan là tiến trình lấy thông tin từ mô hình và hiển thị đồ họa bằng tập các phần từ đồ họa chuẩn. Tiêu chuẩn là cốt lõi để thực hiện một trong các lợi thế của mộ hình trực quan, đó là vấn đề giao tiếp. Giao tiếp giữa người dùng, người phát triển, phân tích viên, kiểm tra viên, người quản lý và những người khác tham gia dự án là mục tiêu quan trọng nhất của mô hình hóa trực quan. Tương tác này có thể được thực hiện bằng văn bản. Nhờ mô hình trực quan mà ta có thể chỉ ra các *tầng* mà hệ thống làm việc, bao gồm tương tác giữa người dùng với hệ thống, tương tác giữa các đối tượng trong hệ thống hay giữa các hệ thông với nhau. Sau khi tạo mô hình, ta có thể chỉ ra từng phần quan tâm. Thí dụ, người dùng có thể quan sát tương tác giữa họ với hệ thông từ mô hình, phân tích viên quan sát tương tác các đối tượng từ mô hình, người phát triển sẽ quan sát được đối tượng mà họ sẽ phát triển… Các nhà tin học đã rất cố gắng để hình thành ký pháp mô hình hóa trực quan. Một số ký pháp quen thuộc là của Booch, OMT và UML. Phần mềm công cụ *Rational Rose 2000* trợ giúp các ký pháp này. Tóm lại, lý do cơ bản để mô hình hóa là: xây dựng mô hình để hiểu sâu sắc hơn về hệ thống đang được xây dựng. Chúng ta xây dưng mô hình cho các hệ thống phức tạp vi ta không thể hiểu nó như tổng thể. Nhờ mô hình hóa ta sẽ đạt được các mục tiêu sau:

Mô hình giúp ta hiển thị hệ thống như chính nó hay như cách mà ta muốn nó hiển thị.

Mô hình cho phép ta đặc tả cấu trúc hay hành vi hệ thống.

Mô hình cho ta mẫu để hướng đẫn trong việc xây dựng hệ thống.

Mô hình giúp ta làm tài liệu cho các quyết định khi phân tích thiết kế hệ thống.

1.3 NGUYÊN TẮC QUẢN LÝ ĐỘ PHỨC TẠP

Như đã trình bày trên thì nhiệm vụ quan trọng nhất của người xây dựng hệ thống phần mềm ngày nay là quản lý được độ phức tạp. Theo *Coad* và *Yourdon* [COAD91] thì các nguyên tắc quản lý độ phức tạp của hệ thống trong phân tích và thiết kế hướng đối tượng bao gồm các vấn đề mô tả dưới đây.

Trừu tượng hóa. Sử dụng nguyên tắc trừu tượng hóa có nghĩa là thừa nhận thế giới thực là phức tạp; thay vì cố gắng hiểu biết *toàn bộ* bằng lựa chọn một phần của vấn đề. Trừu tượng bao gồm hai loại chính: trừu tượng thủ tục (*procedural*) và trừu tượng dữ liệu (*data*). Trừu tượng thủ tục thường được đặc trưng bởi trừu tượng *chức năng/chức năng con*. Chia nhỏ tiến trình xử lý thành các bước là phương pháp cơ bản để quản lý độ phức tạp. Tuy nhiên việc chia nhỏ để tổ chức thiết kế là khá tùy tiện và không ổn định. Trừu tượng thủ tục không phải là hình thức trừu tượng của phương pháp hướng đối tượng. Trừu tượng dữ liệu là cơ chế mạnh, dựa trên cơ sở tổ chức suy nghĩ và đặc tả về các nhiệm vụ của hệ thống. Trừu tượng dữ liệu là nguyên tắc xác định kiểu dữ liệu cho các thao tác áp dụng cho đối tượng, với ràng buộc là các giá trị lưu trữ trong đối tượng chỉ được sửa đổi hay quan sát thông qua các thao tác đó. Người thiết kế áp dụng trừu tượng dữ liệu để xác định thuộc tính và phương thức xử lý thuộc tính xâm nhập thuộc tính thông qua phương thức.

Bao bọc (*encapsulation*). Còn gọi là dấu thông tin. Nguyên tắc này dựa trên nền tảng là mỗi thành phần của chương trình được bao bọc hay dấu quyết định thiết kế đơn lẻ. Giao diện tời mỗi mođun được hình thành sao cho ít nhìn thấy các công việc bên trong. Bao bọc làm tăng tính sử dụng lại khi phát triển hệ thống mới. Bao bọc các nội dung liên quan với nhau làm giảm lưu lượng giữa các phần của hệ thông khi hoạt đông.

Kế thừa (*inheritance*). Cơ chế biểu diễn tính tương tự của các lớp, đơn giản hóa định nghĩa những lớp tương tự từ các lớp khác đã định nghĩa trước. Nó miêu tả tổng quát hóa và đặc biệt hóa, tạo ra các thuộc tính và phương thức chung cho các lớp phân cấp. Nguyên tắc này hình thành nền tảng của kỹ thuật biển diễn những cái chung của các lớp.

Kết hợp (*association*). Là hợp nhất hay liên kết các ý tưởng (từ điển Webster's). Sử dụng kết hợp để gắn một vài phần tử xảy ra vào thời điểm cụ thể hay dưới điều kiện tương tự nhau. Thí dụ, gắn xe cộ vào chủ xe…

Giao tiếp bằng thông điệp. Thông điệp là giao tiếp (viết nói) được trao đổi giữa người với người (từ điển Webster's). Giao tiếp bằng thông điệp liên quan đến tính bao bọc trong đó các chi tiết của hành động sẽ thực hiện được bao bọc trong nơi nhận thông điệp.

Đa hình (*polymorphism*). Đa hình là các thông điệp đồng âm được gửi đến các đối tượng của lớp khác nhau để khởi sự những hành vị khác nhau [OEST00].

Ảnh hưởng của phương pháp tổ chức. Bách khoa toàn thư *Britannica* cho rằng để hiểu thế giới thực, con người sử dụng ba phương pháp tổ chức suy nghĩ như sau: (1) Phân biệt đối tượng cụ thể với các thuộc tính của nó; thí dụ, phân biệt cây với kích thước của nó hoặc quan hệ không gian với các đối tượng khác. (2) Phân biệt giữa toàn bộ đối tượng với thành phần của nó; thí dụ, phân biệt cây với cành cây.(3) Phân biệt giữa các lớp đối tượng với nhau; thí dụ, phân biệt lớp cây với lớp đất đá. Cả ba phương pháp tổ chức này được áp dụng và chúng cho cái nhìn rõ ràng hơn trong lĩnh vực vấn đề và trách nhiệm của hệ thống khi tiếp cận hướng đối tượng.

Quy mô (*scale*). Nguyên tắc áp dụng qui tắc tổng thể-thành phần để quan sát cái gì đó rất lớn được gọi là qui mô.

Phân lớp hành vi. Sau khi đã tìm ra ảnh hưởng tổ chức suy nghĩ, ta phải hiểu rõ các hành vi của đối tượng. Hành vi là những phản ứng, cách cư xử biểu hiện ra ngoài. Phân lớp hành vi bao gồm ba loại sau: (a) trên cơ sở tạo ra kết quả tức thì; (b) sự tương tự của lịch sử tiến hóa (thay đổi theo thời gian) và (c) sự tương tự của các chức năng.

Mẫu (*pattern*). Năm 1977 Christopher Alexander [MULL97] đã đề xuất khái niệm mẫu khi thiết kế hệ thống theo quan điểm hướng đối tượng. Mẫu là tổ hợp đối tượng và lớp. Lợi thế của thiết kế theo mẫu cho phép xử lý các quan niệm về kiến trúc ở mức cao hơn đối tượng vì chúng là cụ thể trong lĩnh vực ứng dụng. Có thể xem mối tương ứng giữa mẫu và các đối tượng như mối tương ứng giữa chương trình con và các dòng lệnh chương trình. Mẫu là đơn vị sử dụng lại trong thiết kế hướng đối tượng.

1.4 NGUYÊN TẮC MÔ HÌNH HÓA

Khả năng của con người là có giới hạn khi khảo sát các vấn đề phức tạp như tổng thể. Thông qua mô hình hóa ta sẽ giới hạn vấn đề nghên cứu bằng cách chỉ tập trung vào một khía cạnh của vấn đề vào một thời điểm. Đó là quan điểm *chia để trị* và *Edsger Dijkstra* đã phát biểu từ vài năm trước đây: *Tấn công vào vấn đề khó bằng cách chia nhỏ nó thành dãy các vấn đề nhỏ hơn mà ta có thể giải quyết được*. Mô hình hóa sẽ làm tăng tri thức của con người. Việc chọn mô hình đúng cho khả năng mô hình làm việc ở mức trừu tượng cao. Mô hình hóa đã có lịch sử lâu đời trong mọi lĩnh vực kỹ nghệ. Người ta đã rút ra bốn nguyên tắc cơ bản sau [BRJ99]:

1. *Việc chọn mô hình nào để tạo lập có ảnh hưởng sâu sắc đến cách giải quyết vấn đề và cách hình thành các giải pháp.*

   Các mô hình đúng sẽ làm sáng tỏ vấn đề phát triển phức tạp nhất, cho cái nhìn thấu đáo vấn đề cần giải quyết. Mặt khác, mô hình tồi sẽ làm ta lạc lối, làm cho ta chỉ tập trung vào các nhiệm vụ không thích hợp. Việc chọn mô hình cho hệ thống phần mềm tác động mạnh đến cách quan sát thế giới. Nếu xây dựng hệ thống theo cách nhìn của người phát triển CSDL thì họ sẽ tập trung vào mô hình quan hệ thực thể, đẩy hành vi vào trigger (khởi sự hành động) và thủ tục lưu trữ. Dưới con mắt của người phân tích cấu trúc, mô hình tập trung vào thuật toán và luồng dữ liệu từ tiến trình này sang tiến trình khác. Dưới con mắt của người phát triển hướng đối tượng, hệ thống có kiến trúc tập trung vào vô số lớp và các

mẫu thử tương tác giữa các đối tượng lớp. Một cách tiếp cận trên có thể phù hợp cho mỗi lớp ứng dụng hay thói quen phát triển hệ thống. Tuy nhiên kinh nghiệm cho thấy rằng tiếp cận hướng đối tượng là ưu việt nhất cho mọi kiến trúc. Mỗi cách nhìn thế giới sẽ dẫn đến sự khác nhau về giá cả và lợi nhuận của hệ thống được xây dựng.

2. *Mỗi mô hình biểu diễn hệ thống với mức độ chính xác khác nhau.*

   Khi xây dựng nhà cao tầng, đôi khi ta phải đứng xa hàng trăm mét để quan sát tổng thể. Tương tự với các mô hình phát triển phần mềm, đôi khi ta chỉ cần có mô hình nhanh, đơn giản về giao diện người dùng; đôi khi lại phải quan sát sâu hơn đến mức các bit khi giải quyết vấn đề tắc nghẽn đường truyền hệ thống. Trong mọi trường hợp nói trên thì các loại mô hình tốt nhất là mô hình cho phép ta lựa chọn mức độ chi tiết khác nhau, phụ thuộc vào ai sẽ là người quan sát và tại sao họ lại cần quan sát nó. Người phân tích và người sử dụng cuối cùng muốn tập trung vào câu hỏi *cái gì?*, người phát triển sẽ tập trung trả lời câu hỏi *như thế nào?*. Cả hai muốn biểu diễn hệ thống ở mức độ chi tiết khác nhau và vào thời điểm khác nhau.

3. *Mô hình tốt nhất phải là mô hinh phù hợp với thế giới thực.*

   Mô hình càng gần với cách suy nghĩ của ta về thế giới thực thì càng dễ dàng quản lý độ phức tạp. Nếu mô hình vật lý của ngôi nhà mà không đáp ứng cách sử dụng của các vật liệu có trên thị trường thì giá trị của mô hình đó chỉ có hạn. Nếu giả thiết của mô hình toán học của con tàu vũ trụ là điều kiện lý tưởng và công nghệ hoàn hảo thì sẽ loại bỏ các đặc điểm nguy hiểm tiềm tàng của tàu vũ trụ thực. Tốt nhất là phải có mô hình kết nối rõ ràng với thế giới thực. Mọi mô hình đều đơn giản hóa thế giới thực, do vậy phải đảm bảo tiến trình đơn giản hóa sẽ không loại bỏ đi các chi tiết quan trọng. Với phần mềm, trong các kỹ thuật phân tích cấu trúc thì mô hình phân tích và mô hình thiết kế hệ thống sẽ được tách biệt nhau. Thiếu cầu nối giữa hại loại mô hình này, dẫn tới hệ thống sẽ được hiểu và xây dựng khác nhau vào các thời điểm khác nhau. Trong hệ thống hướng đối tượng, có khả năng liên kết mọi quan sát tương đối độc lập vào toàn thể.

4. *Không mô hình nào là đầy dủ. Mỗi hệ thống thường được tiếp cận thông qua tập mô hình gần như độc lập nhau.*

   Khi xây dựng tòa nhà, không có một bản thiết kế nào có thể bộc lộ toàn bộ chi tiết của nó. Ít nhất thì ta phải có thiết kế các tầng, thiết kế cầu thang, thiết kế hệ thông điện, nước, thiết kế hệ thống cứu hỏa,… Khái niệm *gần như độc lập nhau* ở đây có nghĩa rằng các mô hình này được hình thành và nghiên cứu tách biệt nhưng nó vẫn có quan hệ với nhau. Thí dụ, ta có thể thiết kế hệ thông điện, nước một cách độc lập, nhưng trong quá trình thiết kế này thì phải quan tâm đến thiết kế của các tầng nhà cho nó phù hợp. Tương tự trong hệ thống phần mềm hướng đối tượng, để hiểu kiến trúc hệ thống phần mềm ta cần một vài quan sát bổ trợ nhau: quan sát *trường hợp sử dụng* diễn tả yêu cầu hệ thống, quan sát thiết kế (thu thập từ vựng của không gian vấn đề và không gian giải pháp), quan sát tiến trình (mô hình hóa phân bổ tiến trình và luồng của hệ thống), quan sát cài đặt (tập trung vào hiện thực vật lý của hệ thống), quan sát triển khai (tập trung vào các nhiệm vụ kỹ nghệ của hệ thống). Mỗi quan sát đều có khía cạnh kiến trúc hay hành vi riêng. Tập hợp lại các quan sát này có khả năng biểu diễn được kế hoạch chi tiết của phát triển phần mềm.

Phục thuộc vào bản chất của hệ thống mà mỗi mô hình có tầm quan trọng khác nhau. Thí dụ, với hệ thống quản lý nhiều dữ liệu thì các mô hình quan sát thiết kế tĩnh sẽ quan trọng hơn. Nếu hệ thống phải có nhiều giao diện người dùng thì các quan sát *trường hợp sử dụng* tĩnh và động đều rất quan trọng. Hệ thống thời gian thực coi trọng quan sát tiến trình động. Cuối cùng, hệ thống phân tán (các ứng dụng Web) coi mô hình triển khai và cài đặt là quan trọng nhất.

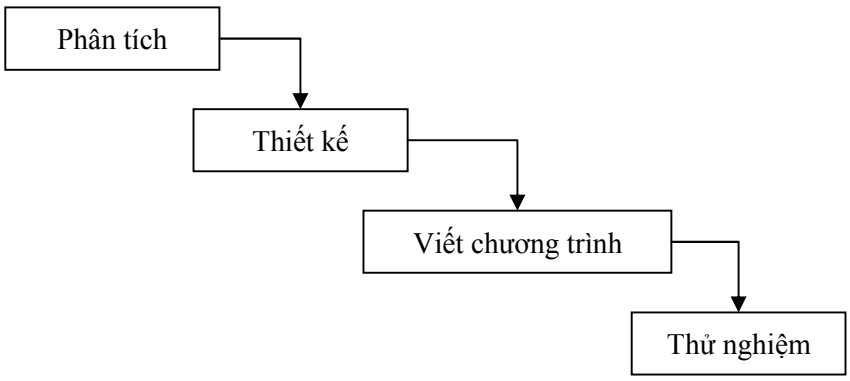
1.5 KHÁI QUÁT VỀ TIẾN TRÌNH PHÁT TRIỂN PHẦN MỀM

Hệ thống là tổ hợp phần cứng, phần mềm cung cấp giải pháp cho vấn đề cần giải quyết. Ngày nay, trong khi hệ thống quá phức tạp mà tri thức lại quá chuyên ngành cho nên một người không thể biết mọi khía cạnh tác nghiệp. Một người không thể hiểu đồng thời mọi vấn đề của hệ thống: từ thiết kế giải pháp, viết mã trình, triển khai trên nền phần cứng đến đảm bảo chắc chắn mọi thành phần phần cứng đều làm việc tốt với nhau. Tiến trình phát triển phần mềm phức tạp phải được nhiều người thực hiện. Trước hết là khách hàng, đó là người đưa ra vấn đề cần giải quyết. Phân tích viên làm tài liệu vấn đề của khách hàng và chuyển nó tới người phát triển, đó là những lập trình viên xây dựng phần mềm để giải quyết, kiểm tra và triển khai nó trên các phần cứng. Phát triển phần mềm có thể được thực hiện bằng nhiều con đường khác nhau. Các dự án có thể tuân thủ một trong các loại tiến trình lặp và tăng dần. Mỗi loại có ưu và nhược điểm riêng.

1.5.1 - Các phương pháp mô hình hóa hệ thống

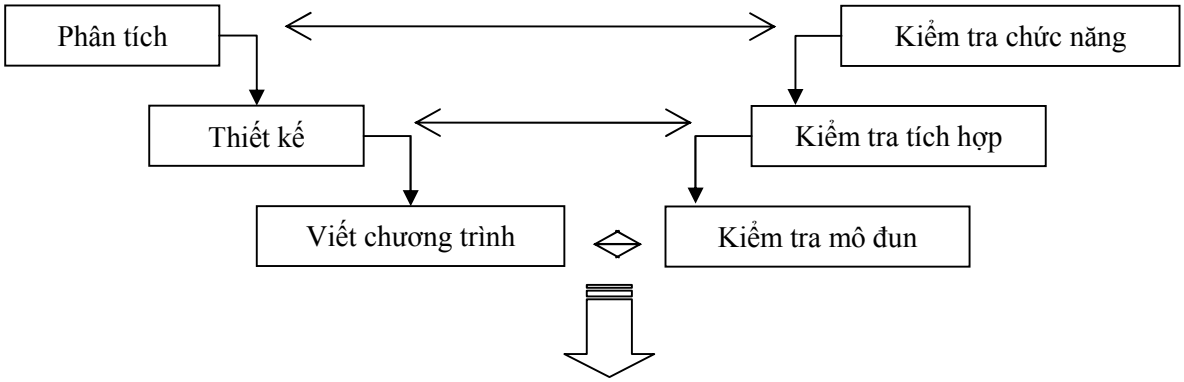
1.5.1.1 - Mô hình thác nước

Từ đã lâu, hệ thống phần mềm thường được mô hình hóa theo phương pháp *thác nước (waterfall)*. Phương pháp này được *Royce* mô tả từ năm 1970 (hình 1.5). Trong mô hình này phát triển phần mềm là dãy các pha liên tiếp từ phân tích yêu cầu, thiết kế hệ thống, phát triển hệ thống đến thử nghiệm và triển khai hệ thống. Pha sau chỉ được bắt đầu khi pha trước đã hoàn thành.



Hình 1.5 Mô hình thác nước

Để xây dựng được hệ thống phần mềm ta phải mô tả được vấn đề (*problem*) và yêu cầu (*requirement*) của khách hàng bằng trả lời các câu hỏi như *vấn đề của hệ thống là gì?* và *hệ thống cần phải làm gì?*. Pha phân tích của tiến trình tập trung vào việc điều tra vấn đề thay cho việc tìm ra giải pháp. Thí dụ, khi xây dựng hệ thống quản lý thư viện thì phân tích có nghĩa là tìm kiếm tiến trình tác nghiệp nào liên quan đến việc sử dụng nó. Để có tài liệu phân tích đầy đủ và đúng đắn thì phải phân tích *lĩnh vực vấn đề*. Lĩnh vực vấn đề là khu vực tác nghiệp của con người, trong đó phần mềm được xây dựng. Thí dụ, hệ thống phần mềm thư viện trong trường học lĩnh vực là giáo dục, sinh viên… Pha thiết kế tập trung vào giải pháp logic, thí dụ phải trả lời câu hỏi hệ *thống đang xây dựng thực hiện các yêu cầu và các ràng buộc như thế nào?*. Trong hệ thống quản lý thư viện thì pha này thực hiện trả lời câu hỏi *thu thập, ghi chép việc mượn, trả sách hay tạp chí như thế nào?*. Pha cài đặt (viết trương trình) tập trung vào mà hóa trương trình.



hình 1.6 Mô hình chữ "V"

Những người tham gia vào xây dựng hệ thống phần mềm như khách hàng, phân tích viên, lập trình viên… theo phương pháp *thác nước* rất ít khi cùng làm việc với nhau để chia sẽ các hiểu biết sâu sắc vấn đề đang giải quyết. Do vậy họ mất rất nhiều thời gian để xây dựng được hệ thống phần mềm.

Chu kỳ *thác nước* còn được biểu diễn dưới dạng chữ V (hình 1.6), trong đó pha kiểm tra được thực hiện đồng thời với các pha phát triển khác. Thí dụ, kiểm tra chức năng được thực hiện trong quá trình phân tích, kiểm tra tích hợp trong pha thiết kế, kiểm tra mođun trong pha mã hóa chương trình.

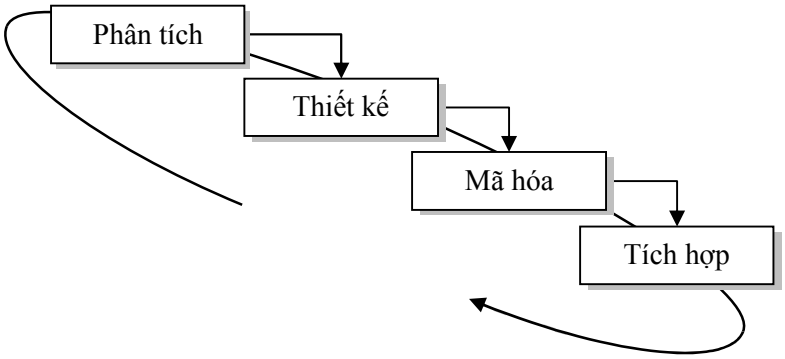
1.5.1.2 - Mô hình lặp và tăng dần

Mô hình *thác nước* không cho ta đi ngược lại chuỗi trình tự phát triển phần mềm như trình bày trên. Bắt đầu dự án, theo mô hình này thì ta phải xác định toàn bộ yêu cầu, nó được thực hiện thông qua bàn bạc với người sử dụng hệ thống và khảo sát chi tiết các tiến trình tác nghiệp. Thực tế thì khi kết thúc công việc, may mắn lắm chỉ 80% nhu cầu hệ thống là được thu thập trong pha phân tích. Tiếp theo là pha thiết kế, nơi kiến trúc hệ thống sẽ được xác định. Pha này tập trung vào những nhiệm vụ như đặt chương trình ở đâu, cần phần cứng nào… Trong khi thực hiện công việc này, ta có thể tìm ra một số nhiệm vụ mới (nhu cầu mới) của hệ thống. Do đó, xuất hiện nhu cầu đi trở lại người sử dụng để trao đổi, bàn bạc về nó; có nghĩa rằng chúng ta phải trở lại pha phân tích. Sau khi đi lại vài lần như vậy ta mới chuyển đến pha phát triển để bắt đầu lập trình hệ thống. Khi mã hóa chương trình, ta phát hiện ra rằng một vài quyết định khi thiết kế là không thể cài đặt. Vậy ta lại phải trở lại pha thiết kế xem xét lại các nhiệm vụ. Sau khi mã hóa xong, pha kiểm tra bắt đầu. Trong khi kiểm tra ta nhận thấy rằng một vài yêu cầu chưa đủ chi tiết, giải thích nhầm lẫn có thể xảy ra. Vậy ta phải quay trở lại pha phân tích để xem xét lại yêu cầu. Sau vài lần lặp ta mới có được hệ thống hoàn chỉnh và phân phát cho người sử dụng. Tác nghiệp có thể thay đổi theo thời gian khi xây dựng hệ thống. Người sử dụng có thể phàn nàn về sản phẩm ta làm ra không hoàn toàn như họ mong đợi. Nguyên nhân có thể là: vấn đề tác nghiệp (*bussiness*) thay đổi quá nhanh; người sử dụng không truyền đạt đúng cái họ muốn; đội ngũ dự án không tuân thủ tiến trình… Đội ngũ phát triển thường lập ra các biểu đồ và vô số tài liệu, văn bản, nhưng người dùng không phải lúc nào cũng hiểu cái mà đội ngũ phát triển cung cấp cho họ. Giải pháp nào để tránh các vấn đề này? Câu trả lời là mô hình hóa trực quan có thể giúp họ.

Phát triển phần mềm là tiến trình phức tạp. Nếu bỏ qua khả năng quay trở lại các bước thực hiện trước đó thì thiết kế hệ thống có thể sai làm và thiếu sót nhu cầu. Để có thể đi ngược lại các bước phát triển hệ thống phần mềm ta có phương pháp mới, phương pháp phát triển lặp (*iterative*). Phát triển lặp là làm đi làm lại việc gì đó. Trong phương pháp này ta sẽ đi qua các bước phân tích, thiết kế, phát triển, kiểm tra và triển khai phần mềm theo từng bước nhỏ nhiều

lần. Cần nhớ rằng không có khả năng thu thập đầy đủ mọi yêu cầu vào công đoạn đầu tiên của dự án. Các vấn đề có thể nảy sinh, vậy ta phải lập kế hoạch lặp trong dự án. Theo quan niệm này thì dự án được coi là dãy các thác nước nhỏ. Mỗi thác nước được thiết kế sao cho đủ lớn để hoàn thành thiện từng bộ phận quan trọng của dự án và đủ nhỏ để tối thiểu nhu cầu đi trở lại.

Hình 1.7 [MULL97] cho thấy mỗi chu kỳ lặp là một vòng đời thác nước nhỏ. Vòng lặp sau được hình thành trên cơ sở tiến hóa của vòng lặp trước đó. Như vậy, các pha truyền thống được lặp đi lặp lại và tăng dần. Trong phương pháp này, phân tích viên, người thiết kế, người lập trình… hợp tác làm việc để hiểu biết sâu sắc hệ thống, chi sẻ các ý tưởng mới dẫn tời xây dựng được hệ thông mạnh, phức tạp hơn.



Hình 1.7 Mô hình lặp và tăng dần

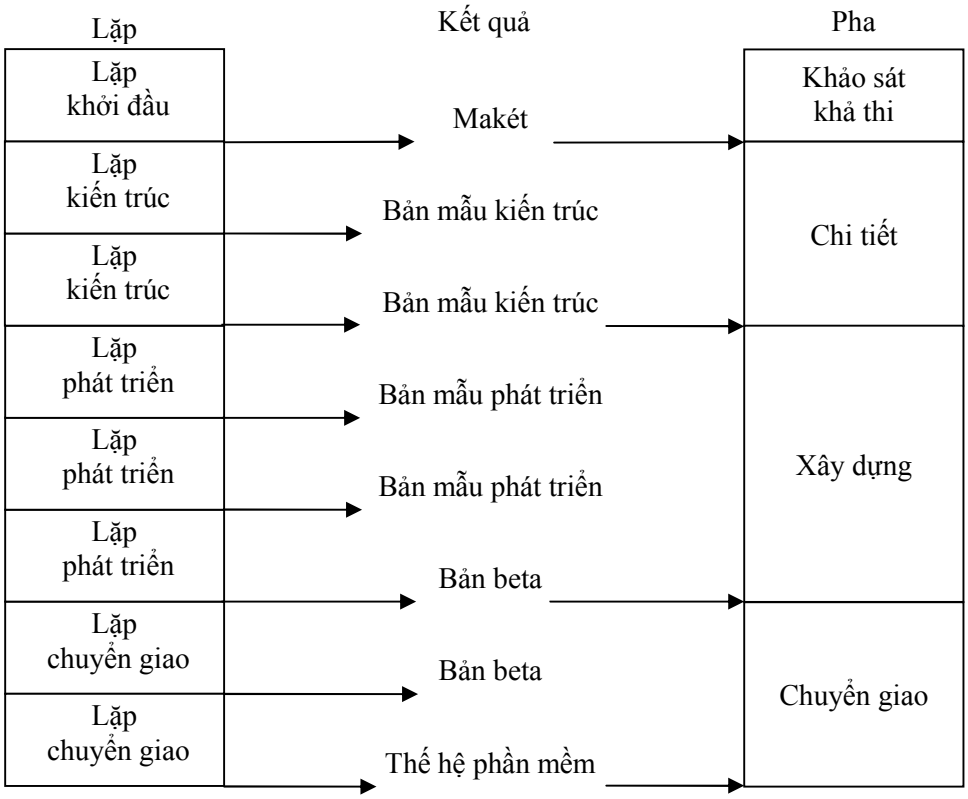
Có nhiều biết thể của chu kỳ lặp. Các chu kỳ lặp này được hình thành trên cơ sở phạm vi của dự án, độ phức tạp của vấn đề và các lựa chọn kiến trúc. Thí dụ, chu kỳ lặp hình chữ *b* của *Birrel N.D* và *Ould M.A*. (1985) tập trung vào pha bảo trì hệ thống, được áp dụng cho các dự án trung bình; chu kỳ lặp hình chữ *O* của *Boehm B.M* (1988) bao gồm lặp công việc phân tích và thiết kế. Mô hình này cho khả năng các nhóm thực hiện song song dự án.

1.5.2 - Các pha phát triển phần mềm

Không có tiến trình phát triển phần mềm nào là phù hợp cho mọi dự án, mọi lĩnh vực ứng dụng [MULL97]. Phần này mô tả tiến trình phát triển phần mềm tổng quát, mỗi dự án phải thích nghi chúng với các ràng buộc riêng. Phát triển phần mềm được quan sát từ hai góc độ bổ trợ nhau. Đó là, góc độ hỗ trợ bao gồm các khía cạnh tài chính, chiến lược, thị trường và con người và góc độ kỹ thuật bao gồm kỹ nghệ, kiểm tra chất lượng và phương pháp mô hình hóa.

1.5.2.1 - Khía cạnh kỹ thuật trong tiến trình phát triển phần mềm

Góc nhìn kỹ thuật tập trung vào triển khai và tổ chức các hoạt động kỹ thuật để dẫn tời sản sinh các thế hệ phần mềm khác nhau. Như trình bày trên, chu kỳ phát triển được xem như trình tự các lặp, thông qua nó mà phần mềm tiến triển dần. Kết quả của mỗi vòng lặp là chương trình có thể chạy được (khả thực). Nội dung của lặp phải có ích cho tiến trình phát triển và cho người sử dụng. Một số kết quả của lặp chỉ được sử dụng nội bộ tiến trình phát triển, một số khác được sử dụng để thể hiện trạng thái tiến triển. Từ lặp này đến lặp khác, chương trình sẽ được bổ sung các chức năng mới và chất lượng của nó đươc nâng cao cho đến một vòng lặp nào đó sẽ sản sinh ra thế hệ thứ nhất của phần mềm.



Hình 1.8 Tiến trình phát phần mềm

Các hoạt động truyền thống không được thực hiện trình tự như trong chu kỳ phát triển *thác nước*, nó được phân bổ vào các lặp khác nhau. Mỗi lặp bao gồm lập kế hoạch, phân tích, thiết kế, cài đặt, thử nghiệm và tích hợp. Chương trình khả thực là kết quả của các bước lặp được gọi là bản mẫu (*prototype*). Bản mẫu là tập con của một ứng dụng, nó được sử dụng để hình thành các yêu cầu hay đánh giá hiệu quả của công nghệ lựa chọn. Bản mẫu ngày càng phong phú thông qua mỗi bước lặp. Thông thường, bản mẫu thứ nhất chỉ có một mục đích là chưng minh quan niệm ứng dụng. Nó được hình thành càng nhanh càng tốt, đôi khi bỏ qua tiêu chuẩn chất lượng thông thường. Bản mẫu này thường được gọi là mô hình hay maket. Các bước lặp áp chót sẽ phát sinh các phiên bản beta, nó được gửi đến nhóm người sử dụng thử nghiệm. Phiên bản chính thức cũng là bản mẫu như các bản mẫu khác (hình 1.8) [MULL97]. Nó được xem như bản mẫu cuối cùng của thế hệ phần mềm đang phát triển và là bản mẫu đầu tiên của thế hệ phần mềm tiếp theo.

1.5.2.2 - <u>Quản lý rủi ro trong tiến trình phát triển lặp</u>

Như trong mọi hoạt động khác của con người, phát triển phần mềm phải tuân thủ luật chia sẻ rủi ro. Phân tích rủi ro bao gồm đánh giá dự án, công nghệ và tài nguyên để xác định, hiểu rõ bản chất của rủi ro và quan hệ giữa chúng phải được mô tả vắn tắt để mọi thành viên trong dự án biết chúng. Rủi ro phải được lượng hóa và phải biết rằng nó có thể hay không có thể loại bỏ. Không được mô tả thiếu rõ ràng trong tài liệu như "Hệ thống cần đủ nhanh" hay "Bộ nhớ cần đủ lớn"…

Rủi ro của dự án được chia thành bốn nhóm như sau đây:

Rùi ro về thương mại: Có thể thu thập đầy đủ thông tin cạnh tranh của sản phẩm trên thị trường?

Rủi ro về tài chính: Nhà đầu tư phát triển phần mềm có đủ kinh phí để hoàn thành dự án?

Rủi ro về kỹ thuật: Nền tảng công nghệ vững chắc và đã được thử thách?

Rủi ro về phát triển: Đội ngũ phát triển có đủ kinh nghiệm? Họ có làm chủ hoàn toàn công nghệ đang sử dụng?

Tuy nhiên, luôn nhớ rằng cái rủi ro lớn nhất là không biết được rủi ro xảy ra ở đâu. Nhiệm vụ quản lý rủi ro là phải lập kế hoạch làm giảm rủi ro, sau đó là thực hiện kế hoạch này.

1.5.2.3 - Khía cạnh hỗ trợ tiến trình phát triển phần mềm

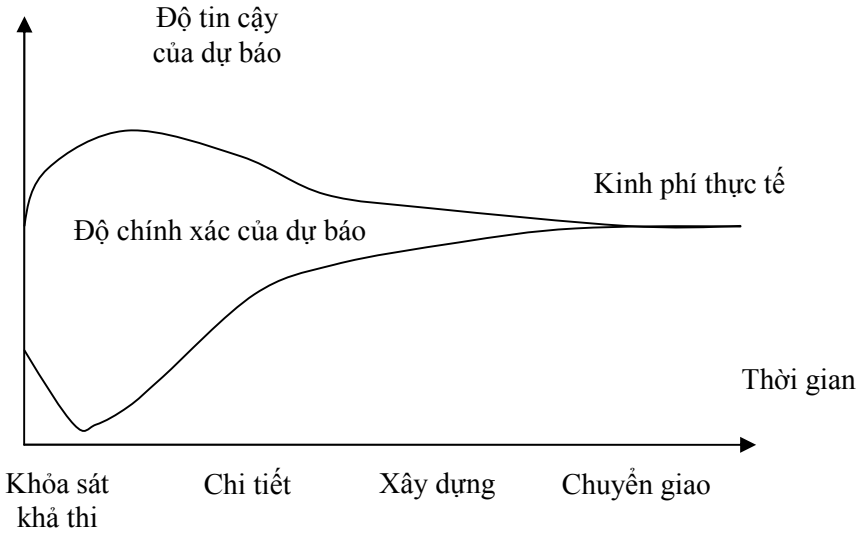
Từ góc nhìn hỗ trợ, phát triển phần mềm được thực hiện trong bố pha: Khảo sát khả thi hay khởi đâu (*inception*), chi tiết (*elaboration*), xây dựng (*construction*) và chuyển giao (*transition*). Chu kỳ phát triển phần mềm từ góc độ này được mô tả trên hình 1.8. Hình này còn cho thấy quan hệ giữa hai góc nhìn khác nhau: góc nhìn hỗ trợ và góc nhìn kỹ thuật. Dưới đây là mô tả các pha phát triển phần mềm từu góc nhìn hỗ trợ.

Pha khởi đầu (hay khảo sát khả thi). Pha này bao gồm khảo sát thị trường, đặc tả sản phẩm cuối cùng, xác định phạm vi dự án. Khởi đầu là bắt đầu dự án, từ khi ai đó cho rằng nếu có hệ thống phần mềm mới để giúp đỡ công việc của họ thì tốt hơn. Tiếp theo là ai đó nghiên cứu ý tưởng. Người quản lý (khách hàng) hỏi bao lâu thì có phần mềm? Kinh phí là phần mềm là bao nhiêu? Tính khả thi của dự án thế nào? Tiến trình tìm ra các câu trả lời cho các câu hỏi này thuộc pha khởi đầu. Khảo sát thị trường không phải là công việc của các kỹ sư phần mềm mà là công việc của chuyên gia khảo sát thị trường và phân tích cạnh tranh. Họ cố gắng đánh giá việc xây dựng hệ thống phần mềm mới hay nâng cấp phần mềm có sẵn có kinh tế không, giúp các công ty xác định các ưu, nhược điểm. Công việc đầu tiên ở đây là hình dung bức tranh tổng quát của sản phẩm để dễ dàng nhận ra và tách các thành phần cho pha chi tiết. Theo E. Morel thì bức tranh này được mô tả như sau:

Bức tranh sản phẩm = Cái gì + Cho ai + Giá bao nhiêu

trong đó, Cái gì muốn đề cập đến các đặc trưng tông thể về sản phẩm; *Cho ai* xác định khách hàng sử dụng sản phẩm và *Giá bao nhiêu* dự báo giá của mỗi sản phẩm mà người sử dụng có thể chấp nhận.

Thông thường phải xây dựng *bản mẫu khái niệm* cho pha khảo sát khả thi để đánh giá tính rủi ro của các chức năng phần mềm, sức mạnh và độ phức tạp của hệ thống, công nghệ mới sẽ áp dụng. Từ đó mà các quyết định về quan niệm sản phẩm được hình thành. Loại bản mẫu này không cần tuân thủ các quy luật phát triển phần mềm thông thường như độ tin cậy cao hay tốc độ xử lý phải nhanh. Đó chỉ là maket hệ thống, mã trình của nó sẽ không đóng vai trò gì trong sản phẩm cuối cùng. Từ đây, tính rủi ro của dự án được nhận biết và pha khảo sát thì sẽ tiếp tục cố gắng đánh giá kinh phí cho dự án và lợi nhuận sẽ đem lại. Dự báo kinh phí luôn là công việc khó khăn. Số liệu của dự báo ban đầu sẽ không bao giờ đúng, chỉ có được số liệu đúng khi kết thúc phát triển hệ thống. Do vậy, việc dự báo này thường được hiệu chỉnh dần dần trong nhiều bước khác.

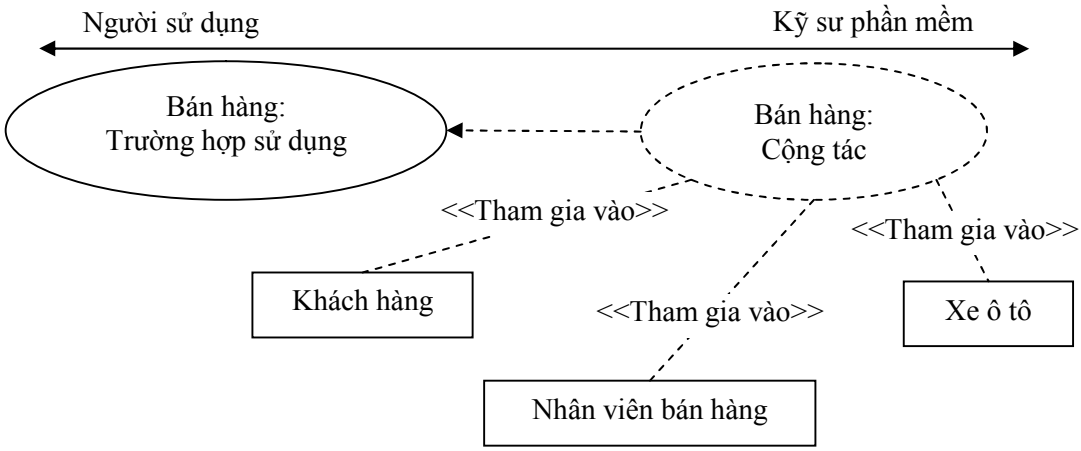


Hình 1.9 Độ tin cậy của dự báo kinh phí dự án

Với các dự án nhỏ, khảo sát khả thi thường chỉ giới hạn bởi danh sách yêu cầu phần mềm. Với các dự án trung bình (thực hiện khoảng một năm) thì pha khảo sát khả thi sẽ thực hiện trong khoảng một tháng, bao gồm cả việc xây dựng maket. Với các dự án lớn, việc xây dựng maket có thể là một dự án con và nó cũng có các bước thực hiện như mô tả trên.

Pha chi tiết. Pha chi tiết bắt đầu bằng phân tích yêu cầu và mô hình hóa lĩnh vực. Nó có nhiệm vụ lựa chọn kiến trúc, làm giảm mức độ rủi ro của dự án, cuối cùng là xác định được kế hoạch đầy đủ cho các nhiệm vụ phát triển hệ thống phần mềm. Tham gia vào pha này bao gồm kiến trúc sư hệ thống, chuyên gia lĩnh vực, người sử dụng, đại diện của nhóm kiểm tra chất lượng và thử nghiệm, người viết tài liệu, chuyên gia về các công cụ phát triển phần mềm.

Phân tích yêu cầu được thực hiện trên cơ sở khảo sát các trường hợp sử dụng. Các trường hợp sử dụng được mô tả theo khái niệm khách hàng, có thể khác xa với hình thức mô tả của kỹ nghệ phần mềm. Các phân tích viên có nhiệm vụ chuyển đổi chúng sang hình thức gần máy tính hơn, thí dụ, chuyển đổi trường hợp sử dụng sang cộng tác của các đối tượng trong lĩnh vực ứng dụng. Đó là các đối tượng trong thế giới thực, công tác với nhau để hình thành các chức năng trong trường hợp sử dụng. Người sử dụng vẫn có thể hiểu rõ trên hình thức biểu diễn này. Hình 1.10 là thí dụ về cài đặt trường hợp sử dụng Bán hàng bằng ba đối tượng hợp tác là khách hàng, người bán hàng và xe ô tô.



Hình 1.10 Cài đặt trường hợp sử dụng

Pha chi tiết cho lại các sản phẩm sau đây:

Mô tả hành vi hệ thống dưới hình thức *trường hợp sử dụng* (use case), ngữ cảnh hệ thống, các tác nhân (người sử dụng hệ thống), các kịch bản ứng dụng và mô hình các lớp ứng dụng. Với dự án trung bình thì có thể bao gồm hàng chục trường hợp sử dụng, khoảng 100 kịch bản chính, vài trăm kịch bản phụ và khoảng từ 50 đến 100 lớp lĩnh vực.

Kiến trúc, tài liệu mô tả kiến trúc, mô tả rủi ro.

Kế hoạch đầy đủ cho phát triển trong dự án.

Kế hoạch chi tiết cho các vòng lặp, tiêu chí đánh giá, danh sách yêu cầu cho mỗi vòng lặp và kết quả cảu kiểm tra chất lượng.

Có thể bao gồm các bản thảo của tài liệu hướng dẫn sử dụng.

Thời gian thực hiện pha chi tiết rất khác nhau trong từng dự án, nó phụ thuộc vào loại ứng dụng và cơ sở hạ tầng lựa chọn cho nó. Thí dụ, nếu dự án thực hiện trong một năm thì pha này sẽ kéo dài từ hai đến bốn tháng. Không nên đánh giá quá nhiều thời gian để có được phân tích hoàn hảo. Nên chuyển dần sang giai đoạn tiếp theo khi đã khảo sát khoảng 80% kịch bản chính. Không nên đi quá chi tiết khi khảo sát kịch bản để tránh việc mất đi tính trừu tượng trong giải pháp.

Pha xây dựng. Pha xây dựng đề cập đến tiến trình phát triển và kiểm tra phần mềm. Pha xây dựng tương ứng với triển khai các vòng lặp. Mỗi vòng lặp ở đây cho lại một mẫu khả thực ổn định, đầy đủ. Đó là những phiên bản đầu tiên của phần mềm. Việc cài đặt lặp bao gồm các hoạt động như sau: Nhận ra các kịch bản hoàn chỉnh hay sử dụng lại trong vòng lặp trên cơ sở khảo sát các rủi ro và kết quả của vòng lặp trước đó; giao nhiệm vụ cụ thể cho đội ngũ phát triển để hoàn thiện vòng lặp; xác định tiêu chí đánh giá, vị trí kiểm tra và hạn định; tập hợp lại tài liệu người sử dụng và tài liệu triển khai.

Pha xây dựng kết thúc khi hoàn thiện phần mềm và được kiểm nghiệm. Phải đảm bảo rằng phần mềm đồng bộ với mô hình. Với dự án nhỏ, pha xây dựng kéo dài từ hai đến ba tháng. Trong các nước phát triển, dự án hướng đối tượng trung bình được thực hiện trong khoảng một năm với năm hay sáu nhân viên thì pha xây dựng chiếm từ sáu đến chín tháng. Với dự án lớn thì mỗi tiểu hệ được xem như dự án con và chúng cũng có các vòng lặp riêng.

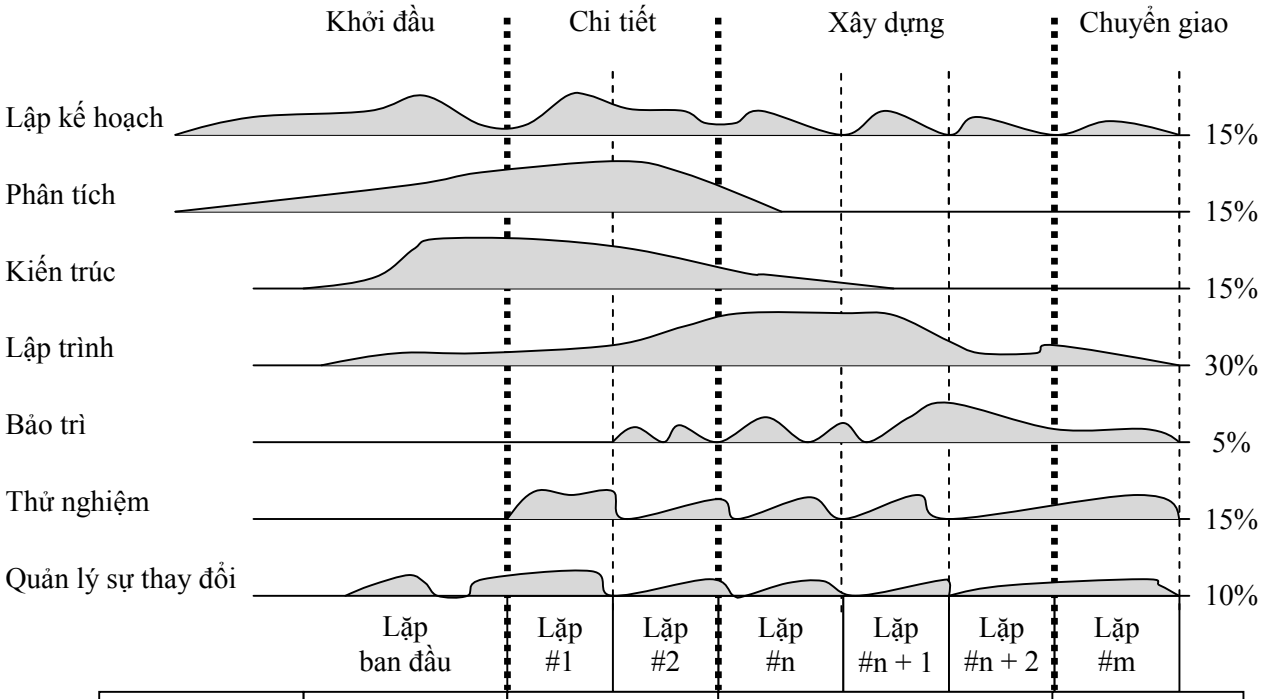
Pha chuyển giao. Pha chuyển giao bắt đầu khi sản phẩm phần mềm hoàn thiện được chuyển đến cộng đồng người sử dụng. Mức độ phức tạp của pha này phụ thuộc vào các ứng dụng cụ thể. Pha chuyển giao bao gồm sản xuất hàng loại, vận chuyển, cài đặt, huấn luyện, hỗ trợ kỹ thuật và bảo trì. Tham gia vào pha này bao gồm một vài người từ đội ngũ phát triển và thử nghiệm chương trình, kiến trúc sư hệ thống làm việc bán thời gian và người có trách nhiệm cập nhật tài liệu. Nhân viên hỗ trợ kỹ thuật, nhân viên bán hàng, quảng cáo sẽ thực hiện việc kiểm tra.

Trong pha chuyển giao, công việc phát triển phần mềm hầu như đã hoàn thành. Mọi sản phẩm đều đạt tời mức chín muồi để cung cấp rộng rãi đến người quản lý dự án và người sử dụng. Người sử dụng sẽ được cung cấp: phiên bản beta và phiên bản cuối cùng của chương trình khả thực; tài liệu hướng dẫn sử dụng, tài liệu triển khai và cài đặt. Người quản lý dự án được cung cấp: có mô hình hệ thống cuối cùng; tiêu chí đánh giá cho mỗi vòng lặp; mô tả việc phân phối sản phẩm; kết quả của kiểm tra chất lượng và các kinh nghiệm rút rra từ thực hiện dự án.

1.5.2.4 - Phân bổ hoạt động trong các pha phát triển phần mềm

Mỗi hoạt động phát triển phần mềm là khác nhau trong các pha phát triển. Không có sự tương ứng một – một giữa các pha dưới góc nhìn hỗ trợ với các pha cổ điển của chu kỳ *thác nước*. Các hoạt động như phân tích, triển khai trải dài trên nhiều pha từ góc nhìn hỗ trợ. Các hoạt động dược

phân bổ giữa các chu kỳ, điểm bắt đầu và kết thúc của chúng không tương ứng với các giới hạn các pha từ góc nhìn hỗ trợ (hình 1.11).



Công sức	5%	20%	65%	10%
Thời gian	10%	30%	50%	10%

Hình 1.11 Sơ đồ các pha phát triển phần mềm

Sơ đồ trên hình 1.11 cho ta thấy:

Kế hoạch thực hiện trong các pha phát triển.

Công việc phân tích được thực hiện chủ yếu trong pha chi tiết, một phần của nó được thực hiện trong pha khảo sát khả thi và pha xây dựng.

Phần lớn kiến trúc được xác định trong pha chi tiết.

Việc viết mã trình (bao gồm cả thử nghiệm các modun) bắt đầu từ pha chi tiết và được thực hiện trong toàn bộ pha cài đặt.

Việc bảo trì được xác định ngay từ phiên bản đầu tiên của phân mềm, thông thường nó bắt đầu trong pha chi tiết.

Thử nghiệm và kiểm tra chất lượng trả dài suốt toàn bộ các pha và áp dụng cho mọi bản mẫu chương trình.

Quản lý sự thay đổi (phiên bản và cấu hình) lưu trữ lịch sử toàn bộ dự án.

Dãy số bên phải hình 1.11 thể hiện phân bổ công sức thực hiện các hoạt động của một dự án phát triển phần mềm hướng đối tượng do năm nhân viên thực hiện trong khoảng một năm. Các hoạt động đó bao gồm:

Lập kế hoạch: bao gồm các hoạt động thí điểm dự án, kế hoạch phát triển quản lý các tài nguyên dự án.

Phân tích: Bao gồm phát triển mô hình đối tượng và mô hình người sử dụng, đặc tả tổng thể về dự án, mô tả các tiêu chí đánh giá.

Kiến trúc: bao gồm viết mã trình cơ sở, thiết kế tổng quan về ứng dụng và xây dựng cơ chế chung

Cài đặt: Nhóm toàn bộ thiết kế chi tiết, viết mã trình và kiểm tra các mođun chương trình.

Thử nghiệm và đánh giá: Bao gồm các hoạt động chứng minh rằng phần mềm phù hợp với mục tiêu ban đầu và các tiều chí đánh giá.

Quản lý thay đổi: Lưu trữ trạng thái kết quả của các giai đoạn phát triển.

Các dãy số phía dưới hình 1.11 chỉ ra phân bố công sức và thời gian cho các pha phát triển phần mềm của một dự án cỡ trung bình.

Chính bản thân *Ngôn ngữ mô hình hóa trực quan UML* không định nghĩa tiến trình phát triển phần mềm, nhưng UML và phần mềm công cụ *Rational Rose* (còn gọi tắt là *Rose*) được sử dụng có hiệu quả trong một số công đoạn của tiến trình phát triển phần mềm. Khi bắt đầu dự án, trong pha khảo sát khả thi, *Rose* được sử dụng để phát sinh mô hình trường hợp sử dụng. Trong pha chi tiết, *Rose* được sử dụng đễ xây dựng biểu đồ trình tự và biểu đồ cộng tác, chỉ ra các đối tượng tương tác với nhau như thế nào. Biểu đồ lớp được xây dựng trong *Rose* để thấy sự phụ thuộc vào nhau của các đối tượng. Trong giai đoạn đầu của pha xây dựng, biểu đổ thành phần được hình thành nhờ *Rose* để chỉ ra sự phụ thuộc của các thành phần trong hệ thống và cho ta khả năng phát sinh mã trình cơ bản. Trong suốt pha xây dựng, *Rose* cho ta khả năng chuyển đổi ngược mã trình thành mô hình để tích hợp mọi sự thay đổi trong quá trình phát triển. Trong pha chuyển giao, *Rose* được sử dụng để cập nhật các mô hình được tạo lập ra cho dự án.

CHƯƠNG 2
KHÁI QUÁT VỀ UML

UML là ngôn ngữ mô hình hóa, trước hết nó là mô tả ký pháp thống nhất, ngữ nghĩa và các định nghĩa về *metamodel* (mô tả định nghĩa chính ngôn ngữ mô hình hóa), nó không mô tả về phương pháp phát triển. UML được sử dụng để hiển thị, đặc tả, xây dựng và làm tài liệu các vật phẩm (*artifacts*) của phân tích quá trình xây dựng hệ thống phần mềm theo hướng đối tượng. UML được sử dụng cho mọi tiến trình phát triển phần mềm, xuyên suốt vòng đời phát triển và độc lập với các công nghệ cài đặt hệ thống.

2.1 GIỚI THIỆU UML

UML là ngôn ngữ chuẩn để viết kế hoạch chi tiết phần mềm. Nó phù hợp cho việc mô hình hóa các hệ thống như hệ thông tin doanh nghiệp, các ứng dụng phân tán trên nền Web, hệ thông nhúng thời gian thực… Các khung nhìn của ngôn ngữ được quan sát từ góc độ phát triển và triển khai hệ thống, nó không khó hiểu và dễ sử dụng. UML là ngôn ngữ mô hình được cả con người và máy sử dụng. Nhớ lại rằng phương pháp là cách cấu trúc rõ ràng suy nghĩ và hành động của ai đó. Phương pháp cho người sử dụng biết làm cái gì, làm như thế nào, khi nào làm việc đó và tại sao lại làm như vậy. Phương pháp chứa mô hình và các mô hình này đươc sử dụng để mô tả cái gì đó. Sự khác nhau chủ yếu của phương pháp và ngôn ngữ mô hình hóa là ngôn ngữ mô hình hóa thiếu tiến trình cho biết làm cái gì, làm như thế nào, khi nào làm việc đó và tại sao lại làm nhu vậy. Như mọi ngôn ngữ mô hình hóa khác, UML có ký pháp (các biểu tượng sử dụng trong mô hình) và tập các luật sử dụng nó. Các luật bao gồm cú pháp, ngữ nghĩa và *pragmatic* (luật hình thành câu có nghĩa).

Cần chú ý rằng ngay cả khi nắm chắc ngôn ngữ UML, không có gì đảm bảo sẽ có mô hình tốt. Tương tự nhà văn viết tiểu thuyết bằng ngôn ngữ tự nhiên, ngôn ngữ chỉ là công cụ, còn tiểu thuyết có hay hay không là phục thuộc hoàn toàn vào nhà văn.

Để sử dụng UML có hiệu quả, đòi hỏi phải hiểu được ba vấn đề chính sau [BRJ99]

Các phần tử cơ bản của mô hình UML

Các qui định liên kết các phần tử mô hình

Một số cơ chế chung áp dụng cho ngôn ngữ này.

UML là ngôn ngữ và nó chỉ là một phần của tiến trình phát triển phần mềm, độc lập với tiến trình. Tuy nhiên UML rất phù hợp với các tiến trình hướng *trường hợp sử dụng* (*Use case – UC*), lấy kiến trúc làm trung tâm, tương tác tăng dần.

2.1.1.1 - <u>UML là ngôn ngữ</u>

Ngôn ngữ phải có từ vựng và qui tắc tổ hợp các từ trong từ vựng để giao tiếp. Ngôn ngức mô hình là ngôn ngữ có từ vựng và qui tắc tập trung vào biểu diễn về mặt vật lý và khái niệm của hệ thống. UML là ngôn ngữ chuẩn công nghiệp để lập kế hoạch chi tiết phần mềm. Như đã trình bày trong chương trước, không có mô hình nào là thỏa mãn cho toàn bộ hệ thống. Thông thường ta phải xây dựng nhiều mô hình cho một hệ thống, do vậy ngôn ngữ phải cho phép biểu diễn nhiều khung nhìn (*views*) khác nhau của một kiến trúc hệ thống trong suốt quá trình phát triển phần mềm. Từ vựng và qui tắc ngôn ngữ UML cho ta cách thức xây dựng mô hình và đọc mô hình, nhưng không cho biết mô hình nào cần phải được lập và khi nào lập chúng. Nhiệm vụ đó được xác định nhờ qui trình phát triển phần mềm. Qui trình phát triển phần mềm sẽ giúp chúng ta đưa

ra quyết định hình thành vật phẩm nào, hoạt động nào và nhân viên nào sẽ tạo ra, sử dụng và quản lý chúng. Đồng thời chúng được sử dụng như thế nào vào việc quản lý toàn bộ dự án.

2.1.1.2 - UML là ngôn ngữ để hiển thị

Với nhiều lập trình viên, không có khoảng cách từ ý tưởng đến cài đặt mã trình. Họ suy nghĩ vấn đề và viết ngay mã trình cho nó. Thực tế là có thể trực tiếp viết mã trình cho một số việc, trực tiếp mô tả thuật toán và biểu thức bằng văn bản. Tuy nhiên, việc giao tiếp giữa mô hình khái niệm với những cái khác trong vòng đời phát triển phần mềm sẽ khó khăn khi mọi người không sử dụng chung một ngôn ngữ cho dự án. Nếu dự án hay tổ chức nào đó đưa ra ngôn ngữ riêng của họ thì người mới tham gia dự án sẽ gặp rất nhiều khó khăn. Hơn nữa, một số vấn đề của hệ thống phần mềm sẽ được hiểu rõ ràng hơn thông qua mô hình thay cho ngôn ngữ lập trình văn bản. Thí dụ, có thể suy luận ra ý nghĩa phân cấp lớp nhưng không thể hiểu thấu đáo nếu bắt đầu trực tiếp bằng mã trình của lớp trong phân cấp. Tương tự, ta không thể hiểu rõ hệ thống trên nền Web nếu không có mô hình. UML giúp xây dựng mô hình để dễ dàng giao tiếp. Một số công việc phù hợp với mô hình hóa băng văn bản, một số công việc khác lại phù hợp hơn với mô hình hóa bằng đồ họa. UML là ngôn ngữ đồ họa. Với nhiều hệ thống, mô hình trong ngôn ngữ đồ họa dễ hiểu hơn so với ngôn ngữ lập trình. Sau mỗi biểu tượng đồ họa của ngôn ngữ UML là ngữ nghĩa. Vậy, khi xây dựng mô hình trong UML thì người phát triển khác hay các công cụ hỗ trợ mô hình hóa có thể hiểu mô hình một cách rỗ ràng.

2.1.1.3 - UML làn ngôn ngữ đặc tả

Đặc tả là mô tả rõ ràng những điểm mấu chốt của vấn đề. UML cho phép mô tả mô hình chính xác, không nhập nhằng và hoàn thiện. UML hướng tới đặc tả thiết kế, phân tích và quyết dịnh cái đặt trong quá trình phát triển và triển khai hệ thống phần mềm.

2.1.1.4 - UML là ngôn ngữ để xây dựng

UML là không phải là ngôn ngữ lập trình trực quan, nhưng mô hình của nó có thể kết nối trực tiếp tới các ngôn ngữ lập trình khác nhau. Có nghĩa rằng có thể ánh xạ mô hình trong UML tới các ngôn ngữ lập trình khác nhau như Java, C++ hay các bảng CSDL quan hệ, CSDL hướng đối tượng. Ánh xạ này cho khả năng biến đổi thuận từ mô hình UML sang ngôn ngữ lập trình. Đồng thời, cho khả năng biến đổi ngược lại từ cài đặt về mô hình UML; có nghĩa rằng nó cho khả năng làm việc với văn bản hay đồ họa một cách nhất quán.

2.1.1.5 - UML là ngôn ngữ tài liêu

UML hướng tới làm tài liệu kiến trúc hệ thống và các chi tiết của nó. UML cho khả năng biểu diễn yêu cầu, thử nghiệm, mô hình hóa các hoạt động lập kế hoạch và quản lý sản phẩm.

UML cho biết giới hạn của hệ thống và các chức năng chính của nó thông qua UC và tác nhân.

Trong UML, các UC được mô tả bằng biểu đồ logic

Biểu diễn cấu trúc tĩnh của hệ thống nhờ biểu đồ lớp

Mô hình hóa các hành vi đối tượng bằng biểu đồ chuyển trạng thái

Phản ảnh kiến trúc cài đặt vật lý bằng biểu đồ thành phần và biểu đồ triển khai

Mở rộng các chức năng bằng *stereotypes*

2.2 MÔ HÌNH KHÁI NIỆM CỦA UML

Để hiểu được UML ta phải hình dung được mô hình khái niệm của ngôn ngữ. Nó đòi hỏi nắm được ba vấn đề chính, bao gồm các phần tử cơ bản để xây dựng mô hình, quy tắc liên kết các phần tử mô hình và một số cơ chế chung sử dụng cho ngôn ngữ. Khi nắm vững được các vấn đề này thì ta có thể đọc được mô hình UML và tạo ra một vài mô hình cơ bản.

2.2.1 - Phần tử mô hình trong UML

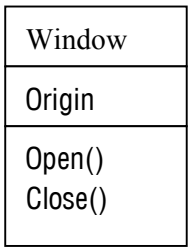
Các khối hình thành mô hình UML gồm ba loại như sau: phần từ, quan hệ và biểu đồ. Phần tử là trừu tượng căn bản trong mô hình, các quan hệ gắn các phần tử này lại với nhau; còn biểu đò nhóm tập hợp các phần tử. Trong UML có bốn loại phần tử mô hình, đó là cấu trúc, hành vi, nhóm và chú thích. Các phần tử này là các khối để xây dựng hướng đối tượng cơ bản của UML.

2.2.1.1 - Phần tử cấu trúc

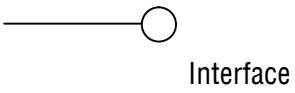
Phần từ cấu trúc là các danh từ trong mô hình UML. Chúng là bộ phận tĩnh của mô hình để biểu diễn các thành phần khái niệm hay vật lý. Có bảy loại phần tử cấu trúc như mô tả sau đây:

Lớp. Lớp mô tả tập các đối tượng cùng chung thuộc tinh, thao tác, quan hệ và ngữ nghĩa. Một lớp cài đặt một hay nhiều ghép nối. Hình 2.1 biểu diễn đồ họa lớp bằng hình chữ nhật, thông thường chú có tên, thuộc tính và thao tác.

Giao diện. Giao diện là tập hợp các thao tác làm dịch vụ của lớp hay thành phần. Giao diện mô tả hành vi thấy được từ ngoài của thành phần. Giao diện biểu diễn toàn bộ hay một phần hành vi của lớp. Giao diện định nghĩa tập đặc tả thao tác chứ không định nghĩa cài đặt của chúng. Ký pháp đồ họa của nó được mô tả trên hình 2.2. Giao diện thường không đứng một mình mà được gắn vào lớp hay thành phần thực hiện giao diện.



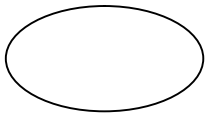
Hình 2.1 Lớp



Hình 2.2 Giao diện

Phần tử cộng tác. Phần tử cộng tác là mô tả ngữ cảnh của tương tác. Ký pháp đồ họa của nó là hình elíp với đường vẽ nét đứt, kèm theo tên như trên hình 2.3. Phần tử cộng tác thể hiện một giải pháp thi hành bên trong hệ thống, bao gồm các lớp, quan hệ và tương tác giữa chúng để đạt được một chức năng mong đợi của UC.

Hình 2.3 Cộng tác



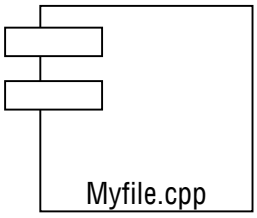
Hình 2.4 Use case

Trường hợp sử dụng (*Use case*). Mô tả chình tự các hành động mà hệ thống sẽ thực hiện để đạt được một kết quả cho tác nhân nào đó. Tác nhân là những gì bên ngoài tương tác với hệ thống. Tập hợp các UC của hệ thống sẽ hình thành các trường hợp mà hệ thống được sử dụng. Sử dụng UC để cấu trúc các phần tử có tính hành vi trong mô hình. Nó được hiện thực hóa bởi phần tử cộng tác như vừa mô tả trên. Hình 2.4 là ký pháp đồ họa của UC.

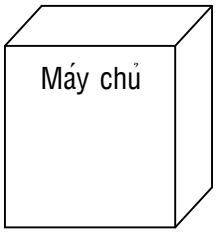
Lớp tích cực (*active class*). Lớp tích cực là lớp có đối tượng làm chủ một hay nhiều lớp tiến trình hay luồng. Lớp tích cực được xem như lớp thông thường nhưng đối tượng của nó biểu diễn các thành phần có hành vi đang tương tranh với các thành phần khác. Ký pháp đồ họa của nó tương tự lớp thông thường nhưng biên chữ nhật được tô đậm. Thông thường chúng cũng có tên, thuộc tính và thao tác.

Thành phần. Thành phần biểu diễn vật lý mã nguồn, các tệp nhị phân trong quá trình phát triển hệ thống. Ký pháp đồ họa của nó đươc biểu diễn trên hình 2.5

Nút (node). Nút là thể hiện thành phần vật lý, tồn tại khi chương trình chạy và biểu diễn các tài nguyên tính toán. Có thể đặt tập các thành phần trên nút chuyển từ nút này sang nút khác. Nút có thể là máy tính, thiết bị phần cứng. Ký pháp đồ họa của chúng được mô tả trên hình 2.6.



Hình 2.5 Cộng tác

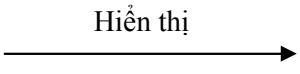


Hình 2.6 Use case

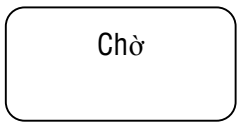
2.2.1.2 - <u>Phần tử hành vi</u>

Phần tử hành vi là bộ phận động của mô hình UML. Chúng là các động từ của mô hình, biểu diễn hành vi theo thời gian và không gian. Có hai loại chính là tương tác và trạng thái.

Tương tác. Tương tác là hành vi bao gồm tập các thông điệp trao đổi giữa các đối tượng trong ngữ cảnh cụ thể để thực hiện mục đích cụ thể. Hành vi của nhóm đối tượng hay của mỗi thao tác có thể được chỉ ra bằng tương tác. Biểu diễ đồ họa của thông điệp được thể hiện như trên hình 2.7, bao gồm mũi tên và tên thao tác của nó.



Hình 2.7 Thông điệp



Hình 2.8 Trạng thái

Máy trạng thái. Máy trạng thái là hành vi chỉ ra trật tự các trạng thái mà đối tượng hay tương tác sẽ đi qua để đáp ứng sự kiện. Hành vi của lớp hay cộng tác của lớp có thể được xác định bằng

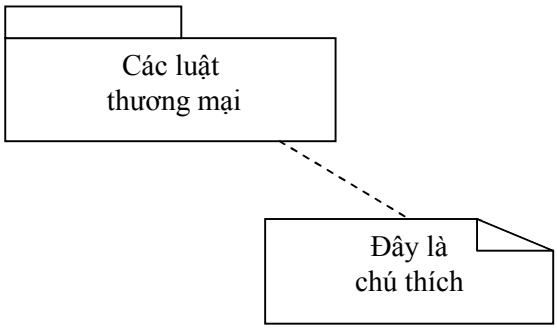
máy trạng thái. Máy trạng thái kích hoạt nhiều phần tử, bao gồm trạng thái, chuyển tiếp (từ trạng thái này sang trạng thái khác), sự kiện và các hoạt động (đáp ứng sự kiện). Ký pháp đồ họa của trạng thái được thể hiện trên hình 2.8, nó chứa tên và trạng thái con nếu có.

2.2.1.3 - Phần tử nhóm

Phần tử nhóm là bộ phận tổ chức của mô hình UML. Chỉ có một phần tử thuộc nhóm này có tên là gói (*package*). Gói là cơ chế đa năng để tổ chức các phần tử vào nhóm. Các phần tử cấu trúc, hành vi và ngay cả phần tử nhóm có thể cho vào gói. Không giống thành phần (*component*), phần tử nhóm hoàn toàn là khái niệm, có nghĩa rằng chúng chỉ tồn tại vào thời điểm phát triển hệ thống chứ không tồn tại vào thời gian chạy chương trình. Ký pháp đồ họa của nhóm được biểu diễn trên hình 2.9. gói giúp ta quan sát hệ thống ở mức tổng quan hơn.

Chú thích (*annotaitonal*)

Phần tử chú thích là bộ phận chú giải của mô hình UML. Đó là lời giải thích áp dụng để mô tả các phần tử khác trong mô hình. Phần tử chú thích được gọi là ghi chú (*note*). Ký pháp đồ họa của chúng thể hiện trên hình 2.9

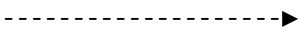


Hình 2.9 Nhóm và lời ghi chú

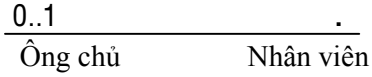
2.2.2 - Các quan hệ trong UML

Có bốn loại quan hệ trong UML, bao gồm quan hệ phụ thuộc, kết hợp, khai quát và hiện thực hóa. Chúng là cơ sở để xây dựng mọi quan hệ trong UML.

Phụ thuộc (*dependency*). Phục thuộc là quan hệ ngữ nghĩa hai phần tử trong đó thay đổi phần tử độc lập sẽ tác động đến ngữ nghĩa của phần tử phục thuộc. Ký pháp đồ họa của nó được thể hiện trên hình 2.10.



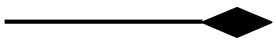
Hình 2.10 Cộng tác



Hình 2.11 Use case

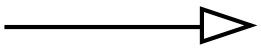
Kết hợp (*association*). Kết hợp là quan hệ cấu trúc để mô tả tập liên kết (một liên kết là kết nối giữa các đối tượng). Khi đối tượng của lớp này gửi/nhận thông điệp đến/từ đối tượng của lớp kia thì ta gọi chúng là có quan hệ kết hợp. Ký pháp đồ họa của kết hợp được mô tả trên hình 2.11 chúng có thể chứa tên nhiệm vụ và tính nhiều (*multiplicity*). Tụ hợp (*aggregation*) là dạng đặc biệt của kết hợp, nó biểu diễn quan hệ cấu trúc giữa toàn thể và bộ phận. Ký pháp đồ họa của nó trên hình 2.12. một dạng dặc biệt của tập hợp là quan hệ hợp thành (*composition*), trong đó nếu như đối tượng toàn thể bị hủy bỏ thì các đối tượng bộ phận của nó cũng bị hủy bỏ theo. Biểu diễn đồ họa của tập hợp như trên hình 2.13.

Hình 2.12 Tụ hợp

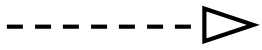


Hình 2.13 Use case

Khái quát hóa (*generalization*). Khái quát hóa là quan hệ đặc biệt hóa/ khái quát hóa mà trong đó đối tượng cụ thể sẽ kế thừa các thuộc tính và phương pháp của đối tượng tổng quát. Ký pháp đồ họa của khái quát hóa được mô tả trên hình 2.14.



Hình 2.14 Khái quát hóa



Hình 2.4 Hiện thực

Hiện thực hóa (*realization*). Hiện thực hóa là quan hệ ngữ nghĩa giữa giao diện và lớp (hay thành phần) hiện thực lớp; giữa UC và hợp tác hiện thực UC. Biểu diễn đồ họa của nó dược mô tả trên hình 2.15.

2.2.3 - Kiểu dữ liệu

Kiểu dữ liệu không phải là phần tử mô hình trong UML. Kiểu dữ liệu cơ sở là kiểu dữ liệu không có cấu trúc. UML có các kiểu dữ liệu sau:

Boolean: là kiểu đếm với hai giá trị True và False

Biểu thức (Expression): là xâu ký tự có cú pháp

Tính nhiều (Multiplicity): là tập không rỗng của các số nguyên dương và ký tự * (để biểu thị tính nhiều vô hạn).

Tên (Name): là xâu ký tự cho khả năng đặc tả phần tử.

Số nguyên (Integer): là kiểu cơ bản và là phần tử của tập vô hạn các sô nguyên âm và dương.

Xâu (String): là trật tự cảu các ký tự, được sử dụng là tên.

Thời gian (Time): xâu ký tự biểu dirn giá trị tuyệt đối hay khoảng tương tương đối.

Không lý giải (Uninterpreted): là 'cái gì đó' mà ý nghĩa của nó phụ thuộc và lĩnh vực.

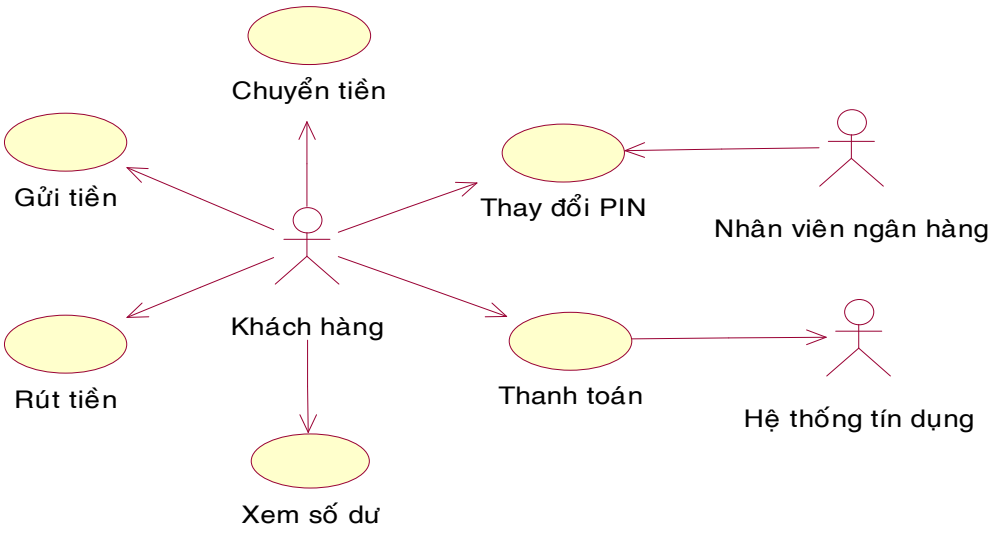
2.2.4 - Biểu đồ UML

Biểu đồ là biểu diễn đồ họa tập hợp các phần tử mô hình. Vẽ biểu đồ để biểu diễn hệ thống đang xây dựng dưới các góc độ quan sát khác nhau. Có thể hiểu biểu đồ là ánh xạ của hệ thống. Một phần tử có thể xuất hiện trong một hay nhiều biểu đồ. Về lý thuyết thì biểu đồ có thể bao gồm tổ hợp vô số phần từ đồ họa và quan hệ vừa mô tả trên. UML cho khả năng xây dựng một vài kiểu biểu đồ trực quan để biểu diễn các khía cạnh khác nhau của hệ thống, bao gồm biểu đồ trường hợp sử dụng (*use case diagram*), biểu đồ trình tự (*sequence diagram*), biểu đồ cộng tác (*collaboration diagram*), biểu đồ lớp (*class diagram*), biểu đồ biến đổi trạng thái (*state transition diagram*), biểu đồ thành phần (*component diagram*) và biểu đồ triển khai (*deployment diagram*). Các loại biểu đồ nay sẽ được giới thiệu tóm tắt dưới đây thông qua ví dụ về xây dựng phần mềm cho hệ thống máy rút tiền tự động (*Automated Teller Machine – ATM*). Giả sử rằng ta có các máy rút tiền tự động ATM đặt tại các đường phố khác nhau trong thành phố. Chúng được nối với máy tính trung tâm tại trụ sở ngân hàng thông qua hệ thống mạng máy tính. Máy tính trung tâm lưu trữ và quản trị CSDL khách hàng, xử lý một số công việc chuyên ngành và yêu cầu ATM trả tiền. Máy rút tiền tự động bao gồm máy đọc thẻ từ, màn hình và bàn phím để tương tác với người sử dụng.

2.2.4.1 - ***Biểu đồ trường hợp sử dụng (Use case – UC)***

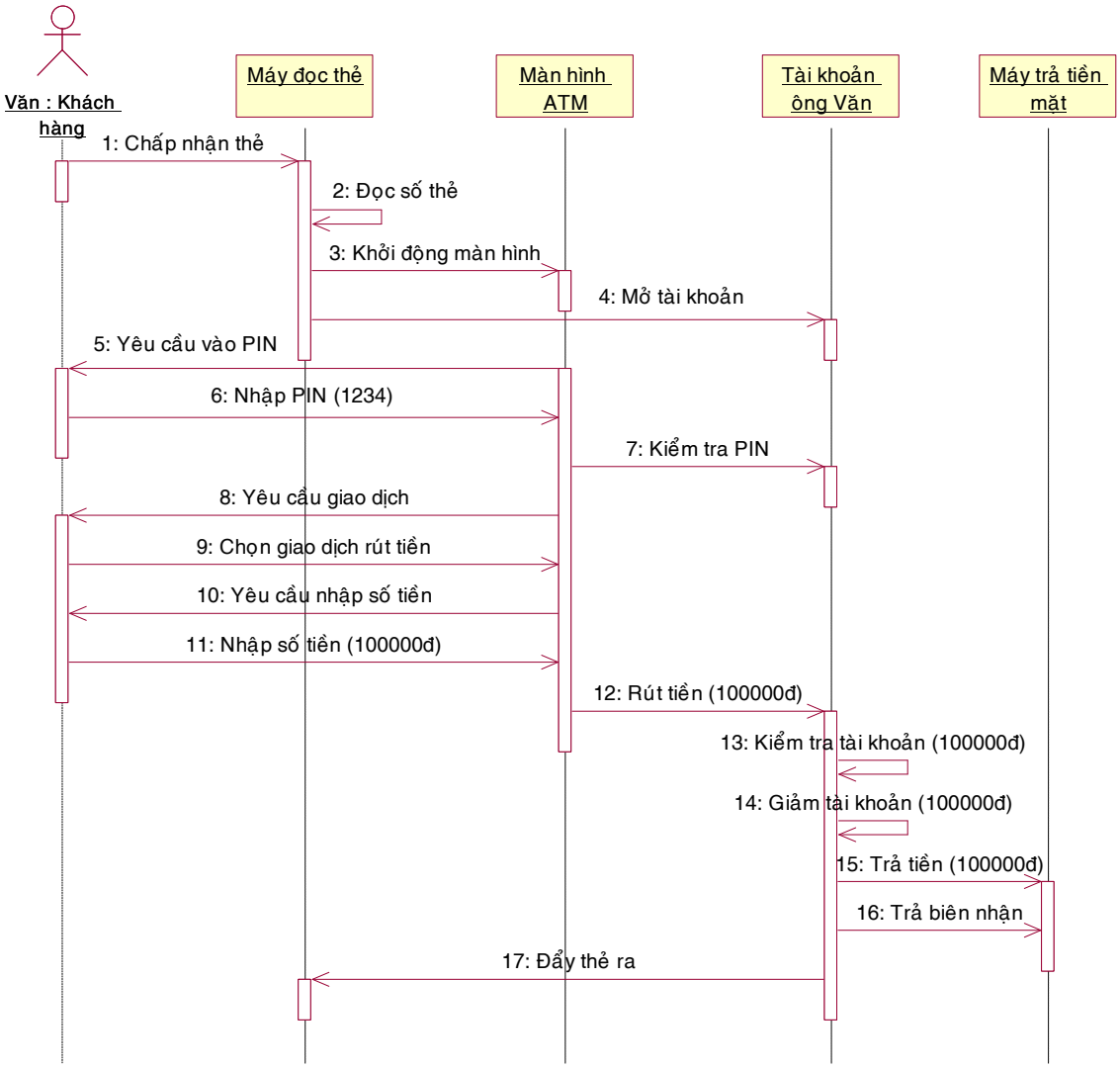
Biểu đồ này chỉ ra tương tác giữa các UC và tác nhân. UC biểu diễn các chức năng hệ thống. Tác nhân là con người hay hệ thống khác cung cấp hay thu nhận thông tin từ hệ thống đang được xây dựng. Biểu đồ UC tập trung vào quan sát trạng thái tĩnh của các UC trong hệ thống. Vì UC biểu diễn yêu cầu hệ thống từ góc độ người dùng, cho nên UC là chức năng mà hệ thống phải có. Biểu đồ loại này chi ra tác nhân nào khởi động UC và khi nào tác nhân nhận thông tin từ hệ thống.

Biểu đồ trên hình 2.16 chỉ ra tương tác giữa các UC và tác nhân của hệ thống rút tiền tự động ATM. *Khách hàng* (là tác nhân) có khả năng khởi động một số UC như *Rút tiền*, *Gửi tiền*, *Chuyển tiền*, *Xem số dư tài khoản*, *Thay đổi số căn cước cá nhân (PIN)* và *Thanh toán*. *Nhân viên ngân hàng* (là tác nhân khác) có khả năng khởi động UC với *Thay đổi số căn cước cá nhân*. Trường hợp sử dụng *Thanh toán* có mũi tên đi đến *Hệ thống tín dụng*. Hệ thống ngoài có thể là tác nhân, thí dụ *Hệ thống tín dụng* là hệ thống ở bên ngoài hệ thống ATM, do vậy nó là tác nhân. Mũi tên đi từ UC đến tác nhân cho biết UC phát sinh thông tin để tác nhân sử dụng (UC trả lại thông tin cho tác nhân).



Hình 2.16 Biểu đồ UC của ATM

Biểu đồ UC chỉ ra chức năng tổng thể của hệ thống đang phát triển. Khách hàng, quản lý dự án, phân tích viên, lập trình viên, kỹ sư kiểm tra chất lượng và những ai quan tâm tổng thể đến hệ thống đều có thể biết được hệ thống sẽ hỗ trợ gì thông qua biểu đồ này.



Hình 2.17 Biểu đồ trình tự của hệ thông ATM

2.2.4.2 - Biểu đồ trình tự (sequence)

Biểu đồ trình tự chỉ ra luồng chức năng xuyên qua các UC, nó là biểu đồ mô tả tương tác giữa các đối tượng và tập trung vào mô tả trật tự các thông điệp theo thời gian. Thí dụ trường hợp sử dụng *Rút tiền* có thể có nhiều trình tự như khách hàng yêu cầu rút tiền khi đã hết tiền trong tài khoản, khi khách hàng nhập nhầm PIN hay trường hợp hoạt động bình thường. Hình 2.17 là thí dụ kịch bản bình thường (nhập dúng PIN, số tiền trong tài khoản đủ số lượng yêu cầu rút). Tác nhân tác động UC này được hiển thị trên đỉnh biểu đồ, thí dụ tác nhân khách hàng trên hình 2.16.

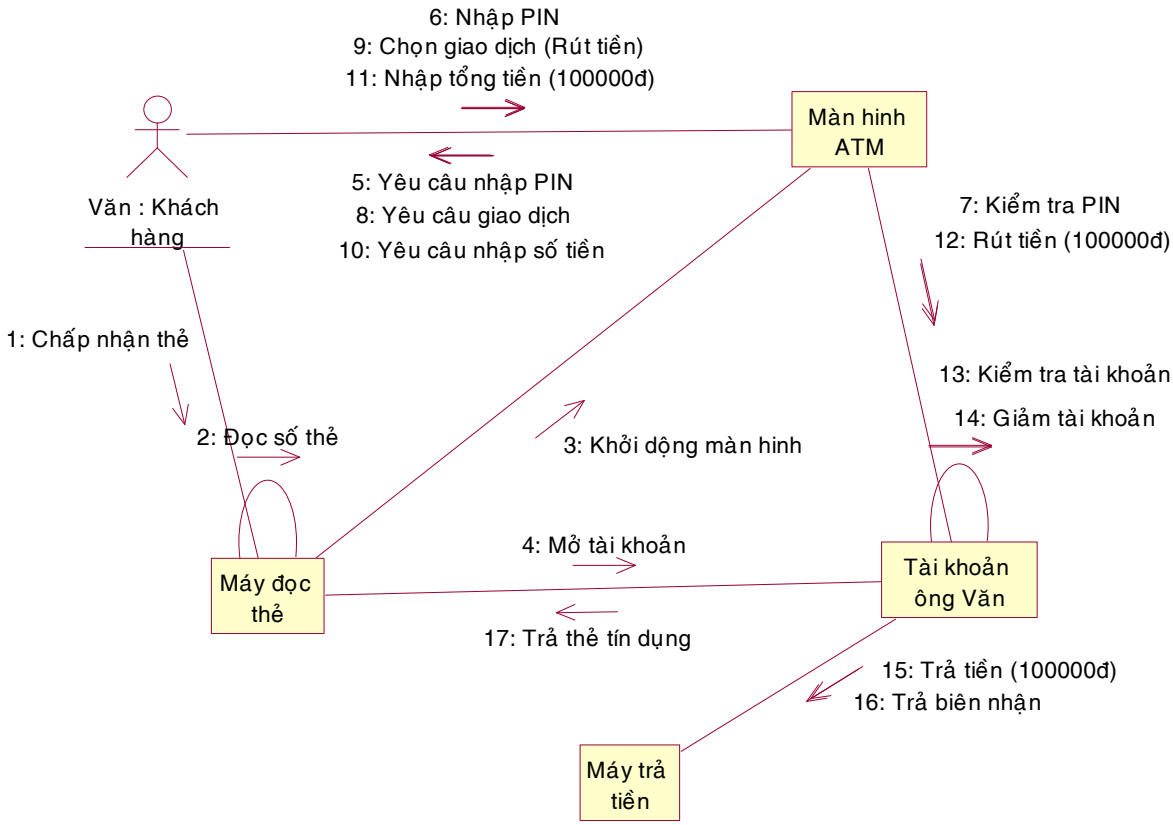
Các đối tượng mà hệ thống cần để thực hiện UC *Rút tiền* cũng được đặt trên đỉnh biểu đồ. Mỗi mũi tên trong biểu đồ thể hiện thông điệp truyền giữa tác nhân và đối tượng hay đối tượng với đối tượng để thực hiện chức năng cụ thể. Chú ý rằng biểu đồ trình tự hiển thị phần tử mô hình đối tượng chứ không phải lớp. Thí dụ, biểu đồ này hiển thị tên khách hàng cụ thể (ông Văn) thay cho hiển thị lớp *Khách hàng*.

UC rút tiền bắt đầu khi khách hàng bỏ thẻ tín dụng vào máy đọc thẻ. Máy đọc thẻ là đối tượng được chỉ ra trong hình chữ nhật trên đỉnh biểu đồ. Sau đó máy đọc thẻ sẽ đọc số thẻ tín dụng, mở đối tượng tài khoản của ông Văn và khởi động màn hình của hệ thống ATM. Màn hình hiển thị dấu nhắc để nhập số hiệu khách hàng (PIN). Thí dụ, khách hàng nhập số PIN 1234. Màn hình kiểm tra PIN với đối tượng tài khoản và thấy phù hợp. Màn hình hiển thị các chức năng để ông Văn lựa chọn. Ông ta chọn chức năng *Rút tiền*. Màn hình hiển thị dấu nhắc để ông Văn nhập số tiền sẽ rút. Ông ta nhập số tiền rút 100000đ. Màn hình thực hiện rút tiền từ tài khoản có đủ 100000đ? Giảm số dư trong tài khoản đi 100000đ. Yêu cầu máy trả tiền chi trả 100000đ tiền mặt và in ra biên nhân. Cuối cùng yêu cầu máy đọc thẻ trả lại thẻ tín dụng cho ông Văn.

Biểu đồ trình tự trên đây mô tả toàn bộ luồng sử lý cho UC rút tiền thông qua thí dụ trường hợp ông Văn rút 100000đ. Khách hàng có thể thấy được tiến trình tác nghiệp cụ thể của họ thông qua biểu đồ. Phân tích viên thấy được luồng tiến trình, người phát triển thấy được các đối tượng cần xây dựng và các thao tác cho các đối tượng này, kỹ sư kiểm tra chất lượng có thể thấy chi tiết của tiến trình để xây dựng qui trình thử nghiệm, kiểm tra. Biểu đồ trình tự có ích cho mọi người tham gia dự án.

2.2.4.3 - Biểu đồ cộng tác (Collabaration)

Biểu đồ cộng tác chỉ ra các thông tin như biểu đồ trình tự theo cách khác, nó tập trung vào tổ chức cấu trúc của các đối tượng gửi và nhận thông điệp. Hình 2.18 là thí dụ biểu đồ cộng tác của biểu đồ trình tự tương ứng mô tả trên hình 2.17. Biểu đồ cộng tác và biểu đồ trình tự thuộc loại biểu đồ tương tác và chúng có thể biến đổi qua lại. Trong biểu đồ cộng tác, đối tượng đặt trong chữ nhật, tác nhân là người hình cây như trong biểu đồ trình tự. Trong khi biểu đồ trình tự biểu diễn tương tác đối tượng và tác nhân theo thời gian thì biểu đồ cộng tác lại không quan tâm đến thời gian. Thí dụ trên hình 2.18 cho thấy máy đọc thẻ ra lệnh mở tài khoản ông Văn, đối tượng tài khoản của ông Văn ra lệnh máy đọc trả lại thẻ tín dụng. Các đối tượng giao tiếp trực tiếp với nhau được thể hiện bằng đường nối.

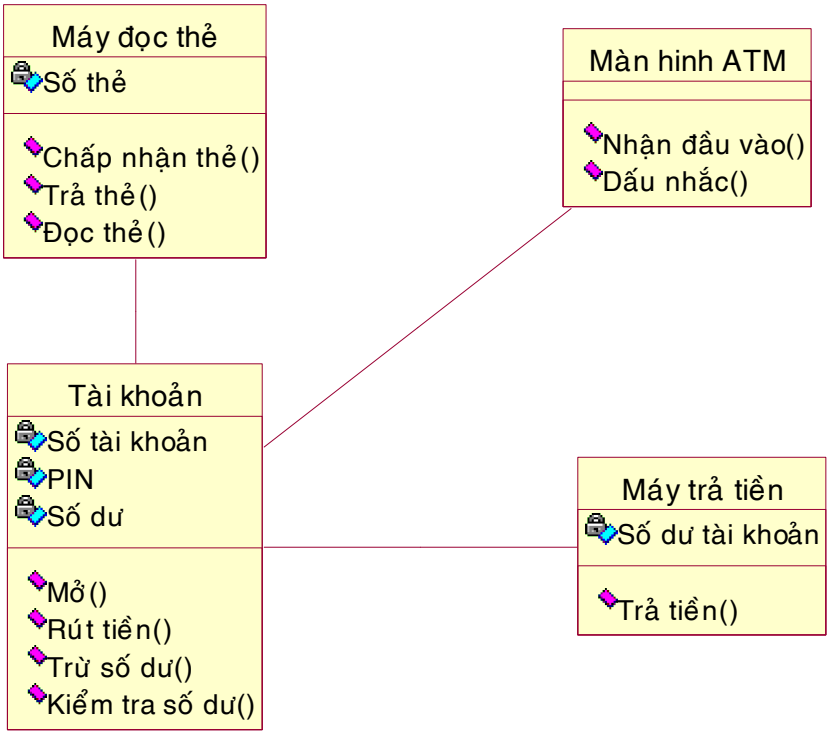


Hình 2.18 Sơ đồ cộng tác ông Văn rút 100000đ

Tuy rằng biểu đồ cộng tác cùng chỉ ra các thông tin như biểu đồ trình tự, nhưng biểu đồ cộng tác được sử dụng vì lý do khác. Kỹ sư kiểm tra chất lượng và kiến trúc sư hệ thống thấy được việc phân bổ tiến trình giữa các đối tượng thông qua biểu đồ loại này. Thí dụ, nếu biểu đồ cộng tác có hình dạng như ngôi sao, với nhiều đối tượng giao tiếp với đối tượng trung tâm thì kiến trúc sư có thể kết luận rằng hệ thống quá phục thuộc vào một đối tượng và họ sẽ đi thiết kế lại phân bổ tiến trình để nâng cao hiệu suất hệ thống.

2.2.4.4 - Biểu đồ lớp (class)

Biểu đồ lớp chỉ ra tương tác giữa các lớp trong hệ thống. Các lớp được xem như kế hoạch chi tiết của các đối tượng. Tài khoản ông Văn là đối tượng; *Tài khoản* là lớp. Lớp *Tài khoản* chứa *PIN* khách hàng và hành vi *Kiểm tra PIN*. Mỗi lớp trong biểu đồ lớp được tạo ra cho mỗi loại đối tượng trong biểu đồ trình tự và cộng tác. Thí dụ biểu đồ lớp cho UC *Rút tiền* được mô tả trên hình 2.19



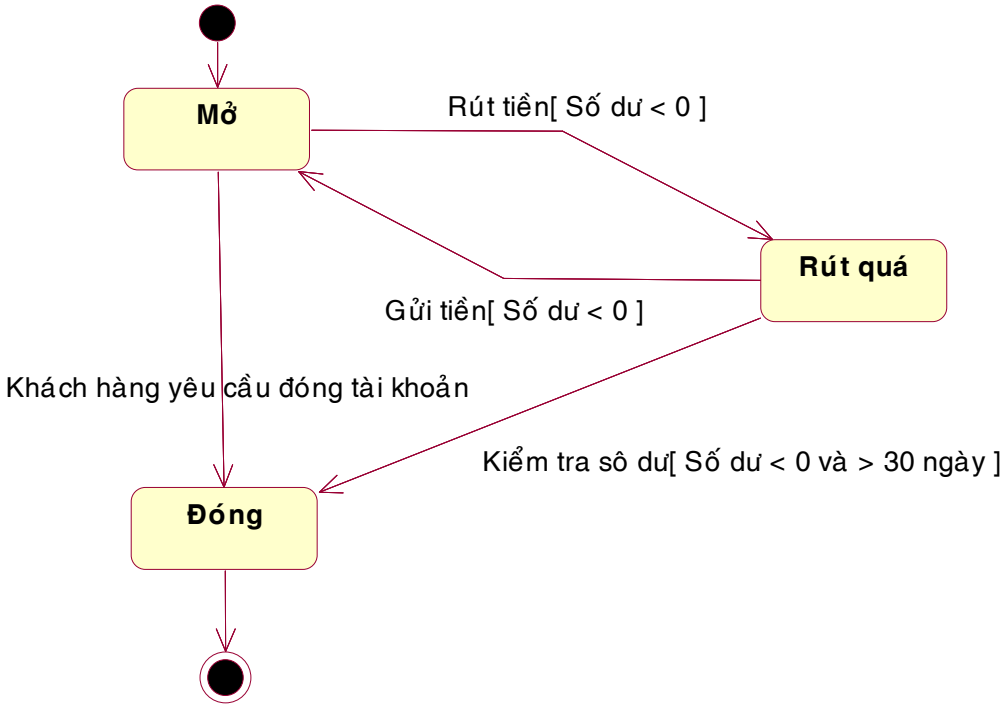
Hình 2.19 Biểu đồ lớp của UC Rút tiền

Biểu đồ lớp trên hình 2.19 chỉ ra quan hệ giữa các lớp hình thành nên UC *Rút tiền*. Biểu đồ này bao gồm bốn lớp, đó là *Máy đọc thẻ, Tài khoản, Màn hình ATM*, và *Máy trả tiền*. Mỗi lớp trong biểu đồ được biểu diễn bằng hình chữ nhật chia làm ba phần: tên lớp (thí dụ tên lớp *Tài khoản*), thuộc tính (thí dụ lớp *Tài khoản* chứa ba thuộc tính: *Số tài khoản, Số căn cước cá nhân – PIN* và *Cân đối tài khoản*) và thao tác (thí dụ lớp *Account* có bốn thao tác: *Mở tài khoản, Rút tiền, Trừ tiền trong tài khoản* và *Kiểm tra số tiền trong tài khoản*. Đường nối giữa các phần tử biểu đồ lớp là quan hệ giao tiếp giữa chúng. Phía trái của một số thuộc tính và thao tác có gắn biểu tượng khóa; có nghĩa rằng đó là các thuộc tính và thao tác riêng.

Người phát triển sử dụng biểu đồ lớp để xây dựng các lớp. Các công cụ phần mềm như *Rose* phát sinh mã trình xương sống cho các lớp, sau đó người phát triển phải chi tiết hóa nó bằng ngôn ngữ lập trình. Kiến trúc sư quan sát thiết kế hệ thống thông qua biểu đồ lớp. Nếu trên biểu đồ thấy một lớp chứa quá nhiều chức năng thì phải chia chúng ra nhiều lớp khác nhau. Kiến trúc sư cũng dễ phát hiện ra nếu giữa các lớp thiếu giao tiếp.

2.2.4.5 - Biểu đồ chuyển trạng thái (state transition)

Biểu đồ chuyển trạng thái mô tả vòng đời của đối tượng, từ khi nó được sinh ra đến khi bi phá hủy. Biểu đồ chuyển trạng thái cung cấp cách thức mô hình hóa các trạng thái khác nhau của đối tượng. Trong khi biểu đồ lớp cung cấp bức tranh tĩnh về các lớp và quan hệ của chúng thì biểu đồ chuyển trạng thái được sử dụng để mô hình hóa các hành vi động của hệ thống.



Hình 2.20 Biểu đồ biến đổi trạng thái của lớp tài khoản

Biểu đồ chuyển trạng thái chỉ ra hành vi của đối tượng. Thí dụ, đối tượng *Tài khoản* trong ngân hàng có thể ở trong một vài trạng thái như mở, đóng hay rút quá mức. Tài khoản sẽ ứng xử khác nhau với mỗi trạng thái khác nhau. Hình 2.20 là thí dụ biểu đồ chuyển trạng thái của lớp Tài khoản.

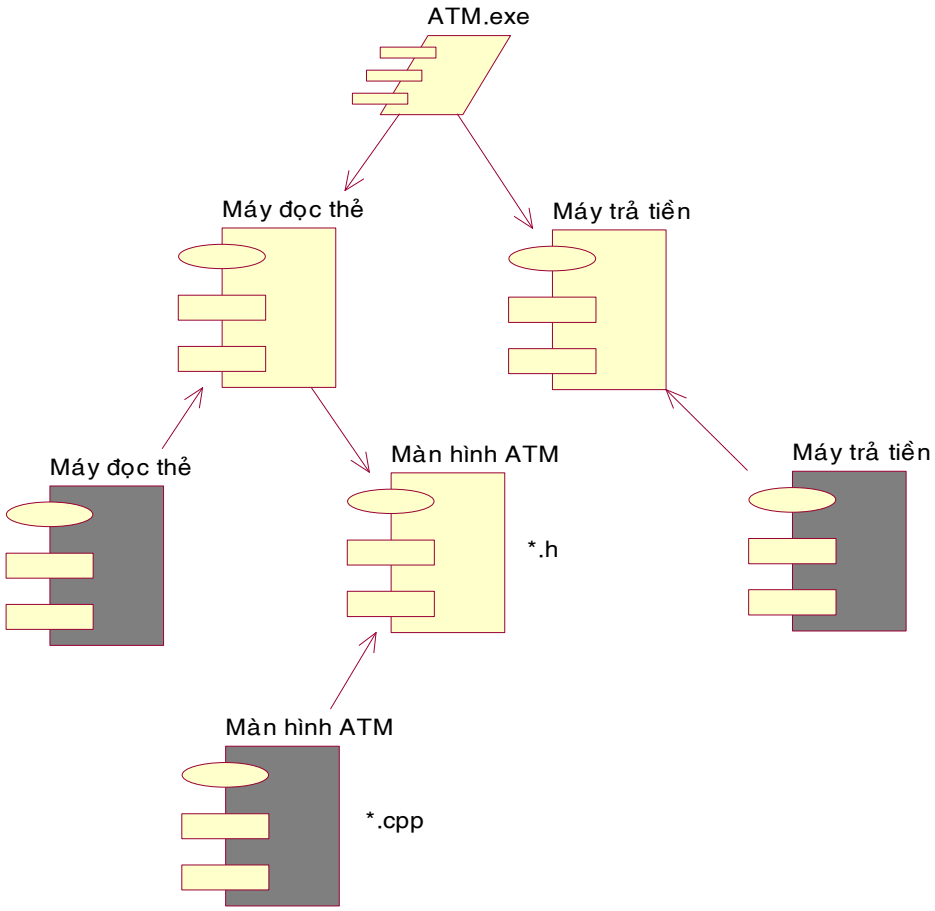
Biểu đồ trên cho thấy các trạng thái và quá trình chuyển trạng thái của Tài khoản. Thí dụ, khi *Tài khoản* đang mở và *Khách hàng* yêu cầu đóng *Tài khoản* thì nó chuyển sang trạng thái đóng. Yêu cầu của *Khách hàng* được gọi là sự kiện. Sự kiện là cái gây ra biến đổi từ trạng thái này sang trạng thái khác. Nếu *Tài khoản* mở và khách hàng rút tiền thì có thể dẫn tới trạng thái rút quá. Trạng thái này xảy ra khi khách hàng con nợ ngân hàng hay *Tài khoản* < 0. Điều kiện này có thể được nêu trên biểu đồ trong đấu ngoặc vuông và được gọi là *điều kiện giác*. Điều kiện gác điều khiển việc xảy ra hay không xảy ra biến đổi trạng thái. Biểu đồ biến đổi trạng thái có hai trạng thái đặc biệt, đó là *Khởi đầu (Start)* và *Dừng (Stop)*. Trên biểu đồ trạng thái chỉ có duy nhất một trạng thái *Start* và có thể có nhiều hay không có trạng thái *Dừng*. Các tiến trình xảy ra khi đối tượng đang trong trạng thái nào đó thì được gọi là *hành động (action)*. Thí dụ, khi *Tài khoản* bị rút quá thì thông báo được gửi tới khách hàng; việc gửi thông báo là hành động.

Thông thường, không tạo lập biểu đồ chuyển trạng thái cho mọi lớp mà chỉ sử dụng cho các lớp phức tạp. Nếu đối tượng của lớp tồn tại trong nhiều trạng thái và có ứng xử khác nhau trong mỗi trạng thái thì nên xây dựng biểu đồ chuyển trạng thái cho chúng. Nhiều dự án không quan tâm đến loại biểu đồ này. Biểu đồ chuyển trạng thái chỉ dành cho việc làm tài liệu *Rose* không phát sinh mã trình từ biểu đồ này.

2.2.4.6 - Biểu đồ thành phần (component)

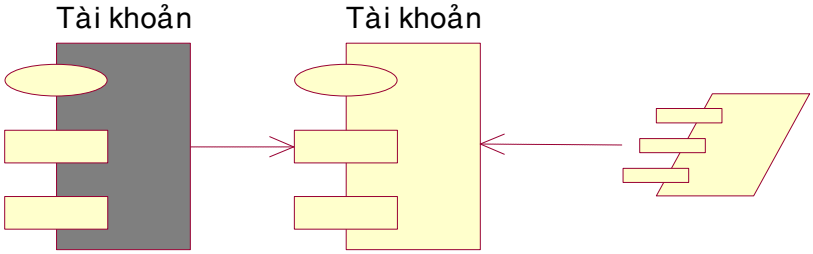
Biểu đồ thành phần cho ta cái nhìn vật lý của mô hình. Biểu đồ thành phần cho ta thấy được các thành phần phần mềm trong hệ thống và quan hệ giữa chúng. Hai loại thành phần trong biểu đồ, đó là thành phần khả thực và thành phần thư viện. Trong *Rose*, mỗi lớp mô hình được ánh xạ đến một thành phần mã nguồn.

Hình 2.21 mô tả biểu đồ thành phần của hệ thông ATM. Biểu đồ trên hình 2.21 là các thành phần phía máy trạm trong hệ thống ATM. Nếu quyết định cài đặt hệ thống bằng ngôn ngữ C++ thì trong mỗi lớp sẽ có tiệp .cpp và tiệp .h riêng biệt, vậy mỗi lớp được ánh xạ đến hai thành phần riêng trong biểu đồ. Thí dụ, lớp Màn hình ATM ánh xạ đến hai thành phần để thể hiện tiệp *.h (header)* và tiệp *.cpp (thân lớp)* của lớp. Thành phần tô đen là đặc tả cảu gói (*package specification*); biểu diễn tệp *cpp* của lớp *Màn hình ATM*. Thành phần không tô đen cũng được gọi là đặc tả gói, biểu diễn tiệp *.h* của lớp. Thành phần *ATM.exe* trên biểu dồ là đặc tả nhiệm vụ và biểu diễn luồng (*thread*) xử lý. Các thành phần nối với nhau bằng đường gạch – gạch để cho biết chúng có quan hệ phụ thuộc. Thí dụ, lớp *Máy đọc thẻ* phụ thuộc vào lớp *Màn hình ATM*. Có nghĩa rằng phải có lớp *Màn hình ATM* thì lớp *Máy đọc thẻ* mới dịch được. Khi toàn bộ các lớp dịch được thì ta mới có thành phần *ATMClient.exe*.



Hình 2.21 Biểu đồ thành phần của ATM client

Thí dụ máy rút tiền tự động ATM có hai luồng xử lý cho nên chúng có hai trình khả thực. Một trình là *Máy trạm ATM*, bao gồm các thành phần *Máy trả tiền, Máy đọc thẻ* và *Màn hình ATM*. Trình thứ hai là *Máy chủ ATM*, nó chỉ có thành phần *Tài khoản*. Biểu đồ thành phần của *Máy chủ ATM* trên hình 2.22.



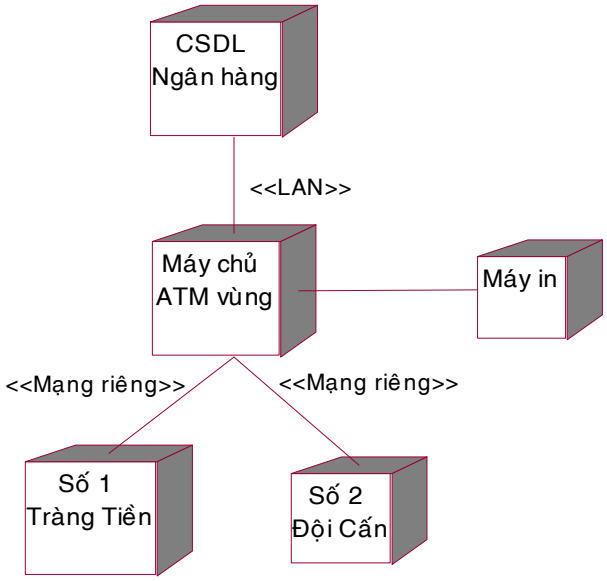
Hình 2.22 Biểu đồ thành phần của máy chủ ATM

Có thể có nhiều biểu đồ thành phần cho một hệ thống, số lượng này phụ thuộc vào các hệ thống con của chúng. Mỗi hệ thống con là gói thành phần. Tổng quát, gói là tập hợp các đối tượng. Thí dụ này có hai gói, đó là gói *Máy trạm ATM* và gói *Máy chủ ATM*.

Bất kỳ ai có trách nhiệm dịch chương trình đều quan tâm đến biểu đồ loại này. Biểu đồ cho ta thấy trình tự dịch của các modun trong hệ thống. Đồng thời nó cũng cho biết rõ thành phần nào được tạo ra khi chạy chương trình. Biểu đồ thành phần chỉ ra ánh xạ của lớp và các thành phần cài đặt.

2.2.4.7 - Biểu đồ triển khai (deployment)

Biểu đồ triển khai chỉ ra bố trí vật lý của mạng và các thành phần hệ thống sẽ đặt ở đâu. Trong thí dụ hệ thống ATM thì ATM bao gồm nhiều hệ thống con chạy tách biệt trên các thiết bị vật lý khác nhau hay còn gọi là nút. Biểu đồ triển khai của hệ ATM được thể hiện trên hình 2.23. Biểu đồ trên hình này cho thấy *Máy trạm ATM* sẽ chạy trên nhiều địa điểm khác nhau. Chúng giao tiếp với *Máy chủ ATM* qua mạng riêng. *Máy chủ ATM* sẽ giao tiếp với các *Máy chủ CSDL* thông qua mạng LAN. Như vậy, hệ thống ATM này có kiến trúc ba tầng: một tầng là CSDL, một tầng là máy chủ, tầng còn lại là máy trạm.



Hình 2.23 Biểu đồ triển khai của hệ thống ATM

Thông qua biểu đồ triển khai mà người quản lý dự án, người sử dụng, kiến trúc sư và đội ngũ triển khai hiểu phân bổ vật lý của hệ thống và các hệ thống con sẽ được đặt ở đâu.

2.3 KIẾN TRÚC HỆ THỐNG

Kiến trúc là trừu tượng hóa các khía cạnh quan trọng nhất của hệ thống. Nó cung cấp khung trong đó thiết kế sẽ được xây dựng. Nó mô tả tầm cỡ, sức mạnh của hệ thống, thu thập các UC quan trọng nhất và các yêu cầu ứng dụng. Nó thể hiện phần mềm sẽ được tổ chức như thế nào và cung cấp các giao thức trao đổi dữ liệu và giao tiếp giữa các mođun. Việc hiển thị, đặc tả, xây dựng và làm tài liệu hệ thống phần mềm đòi hỏi hệ thống phải được xem xét từ nhiều khía cạnh khác nhau. Người sử dụng khác nhau (người sử dụng cuối cùng, nhà phân tích, người phát triển, người tích hợp hệ thống, kiểm tra viên, người quản lý dự án …) có cách quan sát hệ thống khác nhau vào các thời điểm khác nhau. Kiến trúc hệ thống là *vật phẩm* quan trọng nhất, được sử dụng để quản lý các điểm nhìn khác nhau để điều khiển phát triển hệ thống tăng dần và lặp trong suốt chu kỳ sống. Kiến trúc là tập các quyết định về:

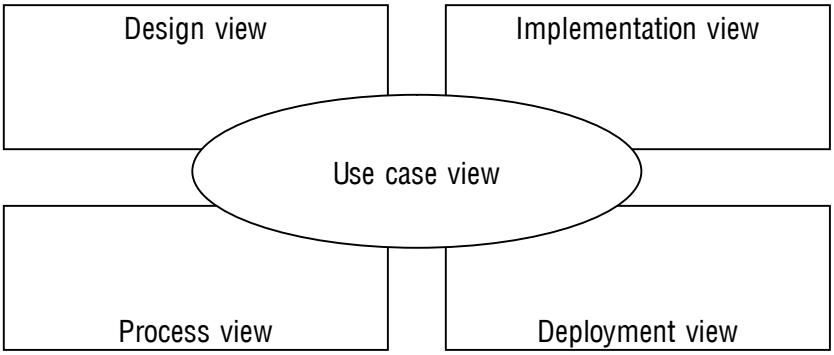
Tổ chức của hệ thống phần mềm

Lựa chọn các phần tử cấu trúc và giao diện cho hệ thống

Hành vi của chúng thể hiện trong hợp tác giữa các phần tử

Tổ hợp các phần tử cấu trúc và hành vị vaò hệ thống con lớn hơn

Kiến trúc phần mềm không chỉ liên quan đến cấu trúc và hành vi mà cả chức năng, tính sử dụng lại, dễ hiểu, ràng buộc công nghệ …



Hình 2.24 Mô hình hóa kiến trúc hệ thống

Kiến trúc hệ thống phần mềm được mô tả bằng các khung nhìn. Các khung nhìn ánh xạ vào tổ chức và cấu trúc hệ thống, mỗi khung nhìn tập trung vào khía cạnh cụ thể của hệ thống (hình 2.24).

Theo biểu đồ của *Philippe Krutchen* [LIBE98] ta có năm khung nhìn như sau: khung nhìn trường hợp sử dụng (*Use case view*), khung nhìn logic (*Logical view*), khung nhìn cài đặt (*Implementation view*), khung nhìn triển khai (*Deployment view*) và khung nhìn tiến trình (*Process view*). Mỗi khung nhìn phục vụ mục đích riêng.

2.3.1 - Khung nhìn UC

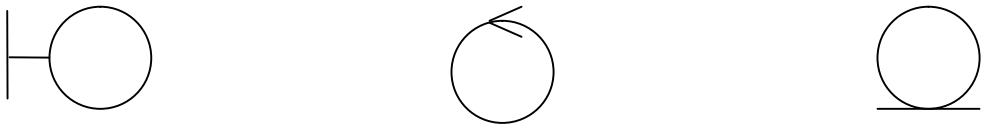
Khung nhìn này đứng trước mọi khung nhìn khác (hình 2.24). Nó được hình thành từ giai đoạn phân tích yêu cầu và được sử dụng để điều khiển và thúc đẩy phần việc còn lại của thiết kế. Nó mô tả các hành vi hệ thống theo cách nhìn của khách hàng, phân tích viên và kỹ sư kiểm tra, thử nghiệm. Khung nhìn UC chứa các tác nhân, UC, biểu đồ UC (khía cạnh tĩnh của khung nhìn) trong hệ thống. Chúng cũng có thể bao gồm vài biểu đồ trình tự, biểu đồ cộng tác (khía cạnh động của khung nhìn) và gói. Khung nhìn UC tập trung vào mức độ cao của cái hệ thông sẽ làm, không quan tâm đến hệ thống làm như thế nào. Chúng không xác định tổ chức của hệ thống.

Khi dự án bắt đầu, khách hàng, phân tích viên và người quản lý dự án làm việc với UC, biểu đồ UC và tài liệu UC để thống nhất về hệ thống. Một khi khách hàng đã nhất trí về các UC và tác nhân thì họ sẽ nhất trí về phạm vi hệ thống. Hệ thống được tiếp tục phát triển băng khung nhìn logic.

2.3.2 - Khung nhìn thiết kế

Rose gọi khung nhìn này là khung nhìn logic (*logical view*). Khung nhìn logic biểu diễn tổ chức của các lớp có ý nghĩa nhất và các quan hệ của chúng với nhau. Khung nhìn logic tập trung vào hệ thống cài đặt hành vi trong UC như thế nào. Nó bao gồm các lớp, biểu đồ lớp, biểu đồ đối tượng (khía cạnh tĩnh của khung nhìn), biểu đồ tương tác, biểu đồ biến đổi trạng thái (khía cạnh động của khung nhìn) và các gói. Hầu hết mọi người trong dự án đều quan tâm đến khung nhìn logic.

Thông thường đội ngũ phát triển phần mềm tiệm cận khung nhìn logic theo hai bước. Bước thứ nhất là nhận ra các lớp phân tích (*analysis class*). Các lớp này độc lập với ngôn ngữ. Trong UML các lớp này được biểu diễn bằng các biểu tượng sau:



Lớp phân tích có thể xuất hiện cả ở trong biểu đồ tương tác của khung nhìn UC. Một khi đã nhận ra các lớp phân tích thì đội ngũ phát triển phần mềm chuyển chúng sang lớp thiết kế (*design class*). Đó là những lớp phụ thuộc ngôn ngữ.

Khung nhìn logic tập trung vào cấu trúc logic của hệ thống. Trong khung nhìn này ta sẽ nhận ra các *bộ phận* hệ thống, khảo sát thông tin và hành vi, khảo sát quan hệ giữa các bộ phận. Cần cẩn thận khi gán thông tin và hành vi cho lớp, nhóm các lớp, khảo sát quan hệ giữa các lớp và gói để đảm bảo khả năng sử dụng lại.

2.3.3 - Khung nhìn cài đặt

Rose gọi khung nhìn này là khung nhìn thành phần (*component view*). Thành phần là mođun vật lý hay tiệp mã trình để lắp ráp thành hệ thống vật lý. Khung nhìn thành phần bao gồm: thành phần, biểu đồ thành phần và gói.

Người quan tâm nhất đến khung nhìn thành phần là người có trách nhiệm quản lý mã trình, dích chương trình và triển khai ứng dụng. Một vài thành phần là thư viện, một số khác là mã trình khả thực (*exe*) và thư viện (*dll*).

2.3.4 - Khung nhìn triển khai

Khung nhìn này tập trung vào phân bổ vật lý của tài nguyên và phân bổ nhiệm vụ giữa các tài nguyên. Khung nhìn triển khai liên quan đến triển khai vật lý của hệ thống, khác với kiến trục logic. Thí dụ, hệ thống có kiến trúc ba tầng logic, bao gồm giao diện, logic và tác nghiệp và logic CSDL tách biệt nhau. Nhưng triển khai có thể chỉ có hai tầng, trong đó logic tác nghiệp và logic CSDL trên cùng một máy. Khung nhìn triển khai bao gồm tiến trình (luông thực hiện trong vùng nhớ riêng), bộ xử lý và thiết bị.

Khung nhìn triển khai chỉ ra các tiến trình và thiết bị trên mạng và các kết nối vật lý giữa chúng. Biểu đồ triển khai cũng hiển thị tiến trình và chỉ ra tiến trình nào chạy trên máy nào.

2.3.5 - Khung nhìn tiến trình

Khung nhìn tiến trình biểu diễn phân tách các luồng thực hiện chương trình (tiến trình – *process*, luồng – *thread*, nhiệm vụ – *task*,…), đồng bộ giữa các luồng, phân bổ các đối tượng và lớp cho các luồng thực hiện khác nhau. Khung nhìn tiến trình tập trung vào các nhiệm vụ tương tranh tương tác với nhau như thế nào trong hệ thông đa nhiệm. Trong biểu đồ cảu phần mềm công cụ *Rose 2000* không có khung này.

2.3.6 - Cần bao nhiêu khung nhìn

Không phải tất cả các hệ thống đều đòi hỏi đầy đủ các khung nhìn mô tả trên. Hệ thống trên máy riêng lẻ có thể bỏ khung nhìn triển khai, nếu hệ đơn xử lý thì bỏ khung nhìn tiến trình, nếu chương trình nhỏ thì bỏ khung nhìn cài đặt.

2.4 RATIONAL ROSE LÀ GÌ?

Rational Rose là phần mềm công cụ mạnh hỗ trợ phân tích, thiết kế hệ thống phần mềm theo hướng đối tượng. Nó giúp ta mô hình hóa hệ thống trước khi viết mã trình, nó đảm bảo tính đúng đắn, hợp lý của kiến trúc hệ thống từ khi khởi đầu dự án. Mô hình *Rose* là bức tranh hệ thống, nó bao gồm toàn bộ biểu đồ UML, tác nhân, trường hợp sử dụng, đối tượng, lớp, thành phần và các nút triển khai trong hệ thống. Nó mô tả chi tiết hệ thống bao gồm cài gì và chúng làm việc ra sao để người phát triển hệ thống có thể sử dụng mô hình như kế hoạch chi tiết cho việc xây dựng hệ thống. Rose hỗ trợ giải quyết vấn đề muôn thủa là đội ngũ dự án giao tiếp với khách hàng và làm tài liệu yêu cầu.

Theo phong cách lập trình truyền thống thì sau khi đã xác định yêu cầu hệ thống, người phát triển sẽ lấy một vài yêu cầu, quyết định thiết kế và viết mã trình. Một số người phát triển khác cũng làm như vậy với yêu cầu khác, thiết kế khác. Cách làm này dẫn tời nhiều khó khăn cho ai muốn hiểu và quản trị toàn bộ hệ thống, họ khó thấy được quyết định thiết kế đã được làm trước đó. Nếu không có tài liệu thiết kế thì khó đảm bảo rằng hệ thống được xây dựng đúng là hệ thống mà người sử dụng nghĩ tới. Tuy rằng các yêu cầu được làm tài liệu đầy đủ, nhưng thiết kế chỉ tồn tại trong đầu người phát triển nào đó, người khác sẽ không có ý tưởng gì về cấu trúc hệ thống. Nếu người phát triển chuyển đi nơi khác thì dự án sẽ gặp khó khăn. Phong cách khác phát triển hệ thống là sau khi xác định yêu cầu, các thiết kế phải được làm tài liệu chi tiết. Mọi người tham gia phát triển cùng trao đổi quyết định thiết kế trước khi viết mã trình. Do vậy, dự án không còn phải lo lắng khi ai đó rời bỏ dự án. Ngoài người phát triển hệ thông quan tâm đến mô hình, ta còn thấy mọi thành viên khác của dự án đều có thể thu nhận các thông tin cần thiết từ mô hình.

Khách hàng và quản lý dự án sử dụng các biểu đồ UC để có cài nhìn bao quát về hệ thống và thống nhất với nhau về phạm vi dự án.

Quản lý dự án sử dụng biểu đồ UC và tài liệu để chia nhỏ dự án thành tiểu dự án có thể quản lý được.

Thông qua tài liệu UC, các phân tích viên và khách hàng thấy được các chức năng hệ thống sẽ cung cấp

Các phân tích viên và người phát triển, thông qua các biểu đồ trình tự và biểu đồ công tác, thấy được logic mà hệ thống tuân thủ, các đối tượng trong hệ thống và các thông điệp giữa các đối tượng.

Đội ngũ kiểm tra chất lượng thu thập thông tin thông qua tài liệu UC và các biểu đồ tương tác để viết mô tả kiểm tra hệ thống.

Người phát triển sử dụng biểu đồ lớp, biểu đồ biến đổi trạng thái để có cái nhìn chi tiết về các phần hệ thống và chúng có quan hệ với nhau như thế nào.

Đội ngũ triển khai sử dụng các biểu đồ thành phần và biểu đồ triển khai để thấy được các tệp khả thực (*exe*) nào, tệp *DLL* nào và các thành phần khác cần được tạo lập; các thành phần này được triển khai trên mạng như thế nào.

Toàn bộ đội ngũ dự án sử dụng mô hình để đảm bảo rằng các yêu cầu có thể được chuyển sang mã trình và ngược lại, mã trình có thể được chuyển trở lại yêu cầu hệ thống.

Hơn nữa, *Rational Rose* còn hỗ trợ phát sinh mã khung chương trình trong nhiều ngôn ngữ khác nhau như *C++, Java, Visual Basic, Oracle8 ...*

2.5 KHẢ NĂNG SỬ DỤNG UML

Mục tiêu của UML là để mô tả bất kỳ loại hệ thống nào bằng biểu đồ hướng đối tượng. Đặc tính của hệ thống đó được tóm tắt như sau:

Các hệ thống thông tin: Lưu trữ, truy vấn, biểu diễn thông tin. Quản lý số lượng thông tin lớn có quan hệ phức tạp trong CSDL quan hệ hay CSDL hướng đối tượng.

Các hệ thống kỹ thuật: Quản lý và điều khiển các thiết bị kỹ thuật như truyền tin, hệ thống quân sự, dây chuyền công nghiệp. Thông thường phải quản lý các giao diện người sử dụng đặc biệt, không có phần mềm chuẩn. Các hệ thống này phần lớn là hệ thống thời gian thực.

Các hệ thống nhúng thời gian thực: Thực hiện trên phần cứng đơn giản nhúng trong các thiết bị khác như điện thoại di động, xe ô tô, các dụng cụ gia đình… Các hệ thống này thường thiếu thiết bị như màn hình, ổ đĩa…

Các hệ thông phân tán: Phân tán trên nhiều máy. Đòi hỏi cơ chế giao tiếp đồng bộ để đảm bảo toàn vẹ dự liệu. Thường được xây dựng trên cơ chế đối tượng như CORBA, COM/DCOM…

Các phần mềm hệ thống: Xác dịnh hạ tầng kỹ thuật để phần mềm khác sử dụng như hệ điều hành, CSDL…

Các hệ thống thương mại: Mô tả mục tiêu, tài nguyên (con người, máy tính…) và các quy luật (chiến lược thương mại, luật …) và qui trình thương mại.

Tuy nhiên ngày nay một hệ thống thường thuộc nhiều loại hoặc tổ hợp của các loại hệ thống kể trên, nhưng ta vẫn có khả năng sử dụng UML để mô hinh hóa chúng.

2.6 THỰC HÀNH

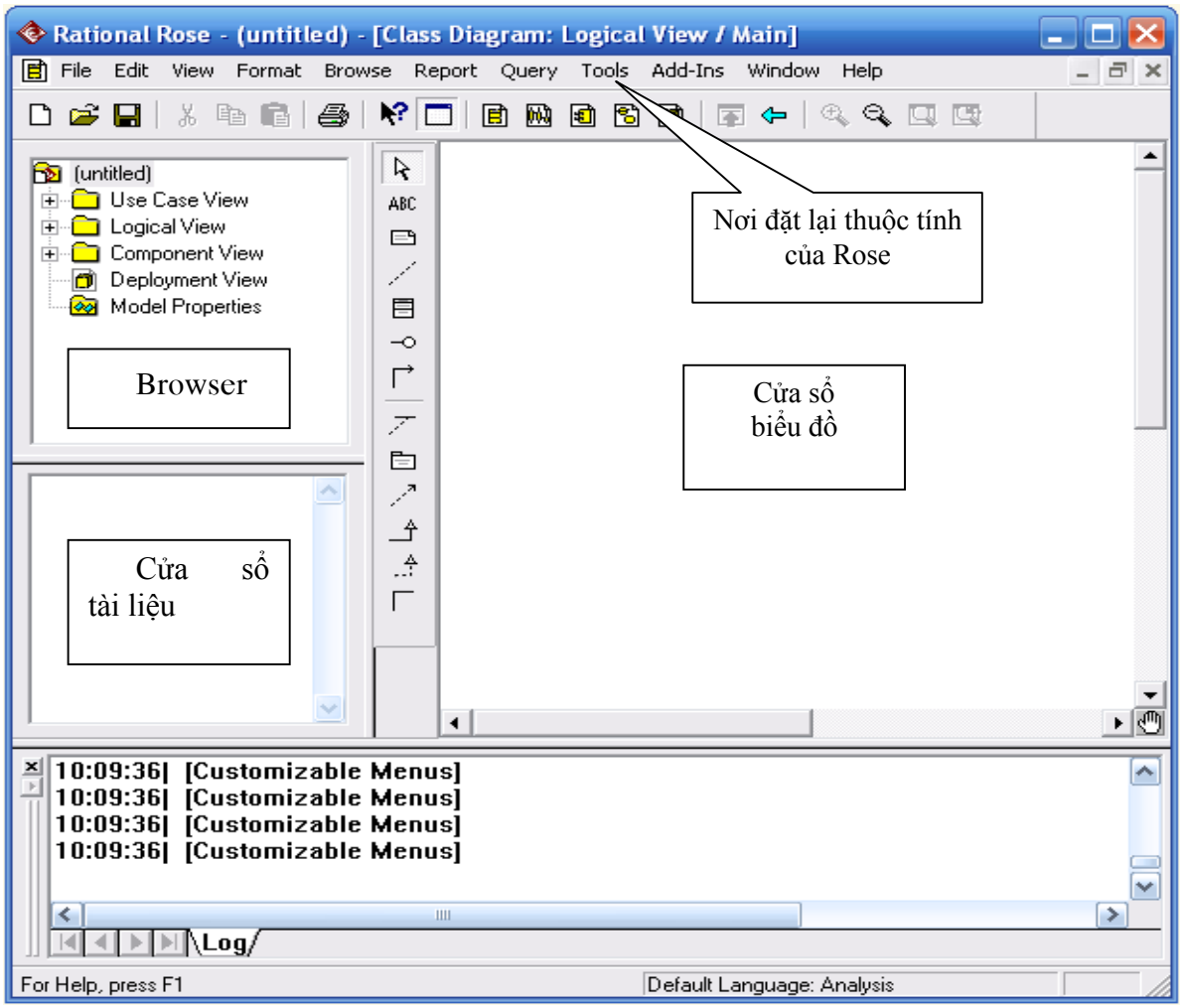
Phần mềm *Rose* có nhiều phiên bản khác nhau. Độc giả có thể sử dụng phần mềm *Rose 98, Rose 2000, Rose v2002* hay các phiên bản thử nghiệm để thực hành các bài tập trong tài liệu này. Độc giả có thể tìm kiếm các phiên bản thử nghiệm hay phiên bản hạn chế (dành cho huấn luyện, đào tạo) của Rose trên trang Web sau:

http://www.rational.com/tryit/rose

Mục tiêu của bài thực hành này là làm quen với các màn hình của phần mềm công cụ *Rose 2000*, một phiên bản của phần mềm công cụ *Rational Rose*. Hãy thự hiện theo các bước như mô tả dưới đây để làm quen với Rose:

1. Khởi động *Rose 2000*. Trên màn hình có các thành phần như hình 2.25.

2. Có thể tắt/mở các cửa sổ duyệt (*Brower Window*) và cửa sổ tài liệu (*Document Window*) bằng cách chọn thực đơn *View>Brower*.



Hình 2.25 Giao diện của Rational Rose

3. Cửa sổ duyệt chứa danh sách toàn bộ phần tử mô hình trong mô hình hiện hành. Browser có thể trôi nổi hay bám dính (*docked*) bằng cách nhấp đúp chuột trên biên cửa sổ. Các phần tử mô hình hiển thì trong *Browser* dưới dạng cây. Các thông tin nén đươc thể hiện bằng dấu +. Nếu nhấn chuột trên dấu + ta sẽ có thông tín không nén.

   Thự hành: tạo lập tệp mô hình mới bằng cách chọn thực đơn *File>New*, *File>Save as* với tên *tutorial.mdl*

4. Cửa sổ tài liệu 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 và nút). Để tạo tài liệu cho phần tử mô hình ta làm như sau: chọn phần tử (nhấn chuột lên phần tử), nhập tài liệu vào cửa sổ tài liệu. Cửa sổ tài liệu cũng có thể tắt/mở, trôi nổi hay bám dính như cửa sổ Browse.

   Thự hành: Nhập tài liệu cho Use case View, Logical View và Component View.

   Use Case View chứa thông tin về tác nhân và UC cho hệ thống đang phát triển.

   Logical View chứa thông tin về lớp và quan hệ giữa chúng

   Component View chứa thông tin về phần mềm và các phần tử khả thực và thư viện.

5. Lưu trữ mô hình vào đĩa từ

6. Cửa sổ biểu đồ là nơi cho phép ta tạo lập và sử đổi khung nhìn đồ họa mô hình hiện hành. Mỗi biểu tượng trong biểu đồ biểu diễn một thành phần mô hình hóa. Mỗi phần tử mô hình có thể hiển thị trong nhiều biểu đồ mô hình khác nhau. Cửa sổ biểu đồ xuất hiện khi nhấn đúp chuột trên cửa sổ biểu đồ trong cửa sổ duyệt.

CHƯƠNG 3
MÔ HÌNH HÓA
TRƯỜNG HỢP SỬ DỤNG

Phân tích và thiết kế bao gồm mô hình hóa vấn đề và giải pháp từ các góc nhìn khác nhau. Trong pha phân tích ta thường quan tâm đến ba loại mô hình,đó là mô hình trường hợp sử dụng, mô hình lĩnh vự và mô hình giao diện người sử dụng. Mô hình trường hợp sử dụng mô tả hệ thống sẽ được sử dụng như thế nào. Mô hình lĩnh vực sẽ mở rộng mô hình trường hợp sử dụng bằng cách đặt hệ thống vào ngữ cảnh. Mô hình giao diện người sử dụng mô tả người sử dụng tương tác với hệ thống như thế nào. Chương này mô tả cách phân tích tìm kiếm trường hợp sử dụng *(Use case – UC)* , tác nhân *(actor)* và quan hệ giữa chúng. Trường hợp sử dụng và tác nhân xác định phạm vi hệ thống. UC là những gì bên trong hệ thống còn tác nhân là những gì bên ngoài hệ thống. Tiếp theo là trình bày về cách tạo lập biểu đồ UC. Cuối chương là phần giới thiệu về công cụ phần mềm *Rational Rose 2000* và ứng dụng nó vào các công việc nói trên.

3.1 PHÂN TÍCH TRƯỜNG HỢP SỬ DỤNG (USE CASE – UC)

3.1.1 - UC là gì?

Khái niệm UC được *Ivar Jacobson* đề xuất vào năm 1994 khi làm việc cho hãng *Eriction*. UC mô tả ai đó sử dụng hệ thống như thế nào, mô tả tương tác giữa người sử dụng với hệ thống phần mềm để thực hiện các thao tác giải quyết công việc cụ thể nào đó. UC không cho biết hệ thống làm việc bên trong như thế nào. Nó không phải là thiết kế, cũng không phải là kế hoạch cài đặt, nó là một phần của vấn đề cần giải quyết. tiến trình của hệ thống được chia nhỏ thành các UC để có thể nhận ra từng bộ phận của nó một cách rõ ràng và để nhiều người có thể cùng xử lý.

UC là nền tảng của phân tích hệ thống. Việc tìm ra đầy đủ các UC đảm bảo rằng hệ thống sẽ xây dựng đáp ứng mọi nhu cầu của người sử dụng. Mỗi UC là tập hành động. mỗi hành động là cái gì đó mà hệ thống làm, nó là hạt nhân được hệ thống thực hiện hoàn toàn hay không được thực hiện phần nào.

3.1.2 - Xây dựng UC để làm gì?

Mục tiêu xây dựng UC trong tiến trình phát triển hệ thống phần mềm được tóm tắt như sau:

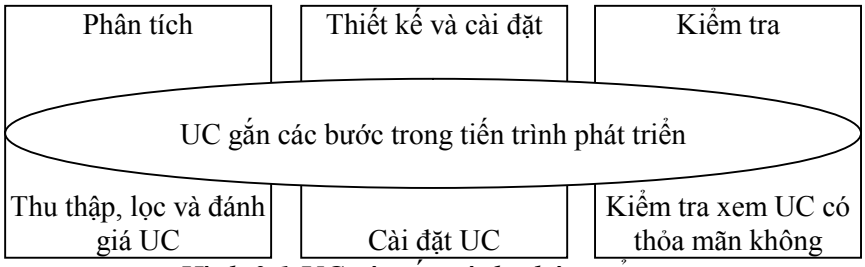
Hình thành quyết định và mô tả yêu cầu chức năng hệ thống. Là kết quả của thỏa thuận giữa khách hàng và người phát triển hệ thống phần mềm.

Cho phép mô tả rõ ràng và nhất quán cái hệ thống sẽ làm, sao cho mô hình có khả năng được sử dụng xuyên suốt quá trình phát triển.

Cung cấp cơ sở để kiểm tra, thử nghiệm hệ thống.

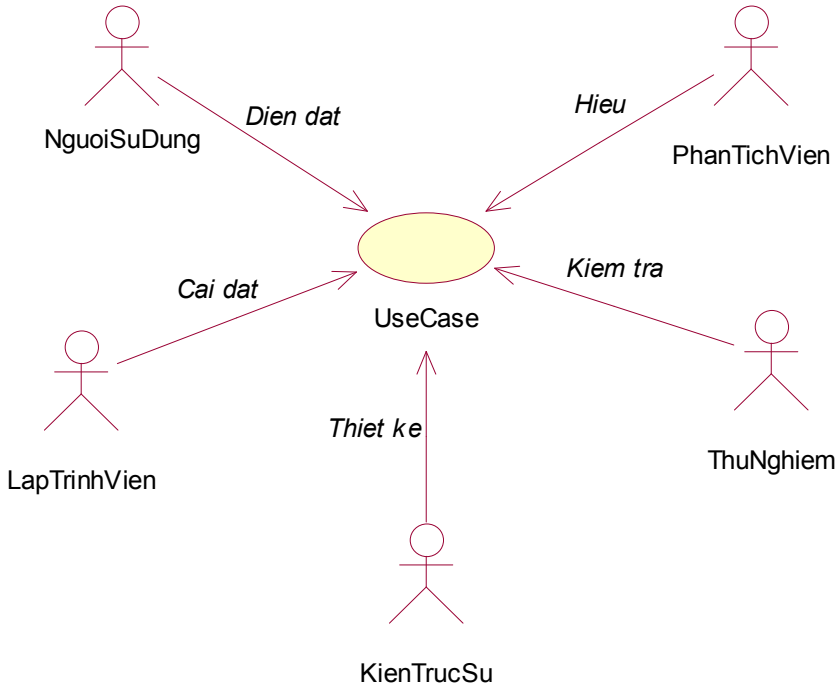
Cho khả năng dễ thay đổi hay mở rộng yêu cầu hệ thống.

Hình 3.1 thể hiện tiến trình phân tích phần mềm theo quan điểm hướng đối tượng được hình thành trên cơ sở UC. Các hành động trong các pha phân tích, thiết kế và cài đặt, kiểm tra và thử nghiệm chương trình đều liên quan đến UC.



Hình 3.1 UC và tiến trình phát triển

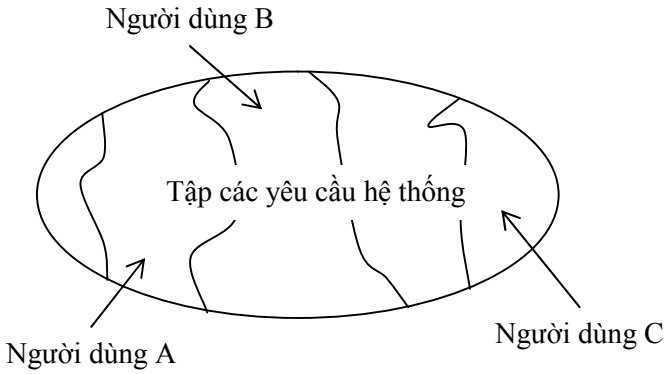
Từ mô tả trên ta thấy UC cung cấp cách thức để người phát triển, người sử dụng cuối cùng, các chuyên gia lĩnh vực hiểu nhau. Hình 3.2 cho chúng ta thấy ai sẽ cần UC và cần để làm gì. Người sử dụng diễn đạt UC, kiến trúc sư có nhiệm vụ tiết kế UC, phân tích viên hiểu UC, lập trình viên hiểu UC và nhân viên kiểm tra chất lượng kiểm tra UC



Hình 3.2 Ai quan tâm đến UC?

3.1.3 - Tìm kiếm UC như thế nào ?

Thông thường thì việc xác định và hiểu rõ các yêu cầu hệ thống là rất khó vì khối lượng thông tin liên quan khổng lồ. Do vậy, các yêu cầu sẽ được mô tả một cách lộn xộn không cấu trúc, mâu thuẫn với nhau, thiếu nhiều vấn đề quan trọng, không chính xác… Kết quả của nó có được là tập yêu cầu mờ, rất khó thay đổi. khái niệm UC được đưa ra để tập trung vào biểu thị các yêu cầu từ phía người ứng dụng, xuất phát từ quan điểm đơn giản là hệ thống được xây dựng trước hết là cho người sử dụng chúng. Phân hoạch tập yêu cầu là giảm thiểu đáng kể độ phức tạp của việc xác định yêu cầu (hình 3.3)



Hình 3.3 Phân hoạch yêu cầu bằng UC

Trong phương pháp hướng đối tượng, câu hỏi đặt ra khi bắt đầu thực hiện dự án thường là: *Tìm kiếm UC như thế nào?* Không thể thiếu sự tham gia của khách hàng vào xây dựng UC. Khách hàng hiểu rõ hệ thống sẽ được sử dụng như thế nào. Cách tốt nhất để tìm kiếm UC là phỏng vấn người sử dụng và khảo sát tài liệu của họ. Việc phỏng vấn người sử dụng phải bằng khái niệm, ngôn từ của lĩnh vực vấn đề và của chính người sử dụng. Khi phân tích lĩnh vực ứng dụng, phân tích viên còn phải hợp tác chặt chẽ với với các chuyên gia lĩnh vực và người quản lý dự án. Các chuyên gia lĩnh vực có nhiều kinh nghiệm sử dụng và thiết kế các sản phẩm tương tự, họ có thể giúp chúng ta hiểu chi tiết về người sử dụng tương tác với hệ thống như thế nào. Chất lượng phân tích bị ảnh hưởng nhiều từ các chuyên gia lĩnh vực, do vậy, phải ưu tiên việc giao tiếp, trao đổi kỹ lưỡng với họ. Việc trao đổi với người sử dụng và các chuyên gia lĩnh vực thường được thực hiện độc lập để có thể so sánh cách nhận thức và cách diễn đạt khác nhau của họ. nếu có sự khác biệt thì phải có cuộc họp thảo luận chung để đạt được sự nhất trí hay làm rõ các vấn đề còn mâu thuẫn. kết quả ban đầu của hoạt động này là tác nhân và UC mức cao (mô tả yêu cầu chức năng hệ thống) được bộc lộ. trong thực tế, không phải người sử dụng nào cũng có khả năng mô tả rõ ràng cách mà họ muốn sử dụng hệ thống. những cái họ biết thường nhiều hơn cái họ đang diễn giải. chính UC và biểu đồ UC là công cụ tốt nhất để giúp người sử dụng nói về hệ thống từ góc nhìn của họ.

Thông thường tiến trình tìm kiếm UC sau khi tiềm kiếm tác nhân. tác nhân là thực thể bên ngoài tương tác với hệ thống. Chúng có thể là con người nhưng cũng có thế là hệ thống hay thiết bị phần cứng khác cần tương tác và thời gian. Tương tác là sự trao đổi thông tin. tác nhân con người (người sử dụng) là tác nhân điển hình của mỗi hệ thống. thí dụ trong hệ thống ATM thì khách hàng và người bảo trì hệ thống là tác nhân, trong quản lý thư viện thì độc giả và thủ thư là tác nhân. Tác nhân biểu diễn nhiệm vụ (nó là lớp) chứ không phải là người dùng cụ thể của hệ thống. Thí dụ, độc giả là tác nhân, nhưng *Anh Văn*, *Chị Đào* không phải là tác nhân. Ngân hàng có hệ thống tín dụng để quản trị các thông tin về tài khoản tín dụng của mỗi khách hàng. Hệ thống rút tiền tự động của ATM phải có khả năng giao tiếp với hệ thống tín dụng, vậy hệ thống tín dụng trở thành tác nhân. Thời gian là tác nhân khi nó xác định thời điểm xảy ra sự kiện trong hệ thống.

Một trong các kỹ thuật tìm kiếm tác nhân là trả lời các câu hỏi sau đây:

- Ai sẽ sử dụng các chức năng chính của hệ thống?
- Ai giúp hệ thống làm việc hằng ngày?
- Ai quản trị, bảo dưỡng để hệ thống làm việc liên tục?
- Hệ thống quản lý thiết bị phần cứng nào?
- Hệ thống đang xây dựng tương tác với hệ thống khác nào?

- Ai hay cái gì quan tâm đến kết quả hệ thống cho lại?

Sau khi có tác nhân, hãy trả lời các câu hỏi sau đây để tìm ra các UC:

- tác nhân yêu cầu hệ thống thực hiện chức năng gì?
- Tác nhân cần đọc, tạo lập, bãi bỏ, lưu trữ, sửa đổi các thông tin nào trong hệ thống?
- Có cần thông báo cho tác nhân về sự kiện xảy ra trong hệ thống? có cần tác nhân thông báo cái gì đó cho hệ thống?
- Hệ thống cần vào / ra nào? Vào / ra đi đến đâu hay từ đâu?

Sau khi tìm được UC thì phải gán tên cho chúng. Đặt tên UC theo khái niệm tác nghiệp (không sử dụng khái niệm chuyên môn, kỹ thuật), sử dụng động từ, câu ngắn. UC độc lập với cài đặt (độc lập với ngôn ngữ lập trình), nó có thể được mô tả bằng ngôn ngữ lập trình Java, C++, văn bản trên giấy hay bằng công cụ phần mềm nào đó (*Rational Rose*, *Microsoft Modeler*). Mô tả UC bao gồm các phần tử sau đây:

- Khởi đầu UC - sự kiện khởi động UC. Nó phải được mô tả rõ ràng, thí dụ, "*UC bắt đầu khi X xảy ra*".
- Kết thúc UC – sự kiện dừng UC. Nó phải được nhận diện và làm tài liệu rõ ràng, thí dụ, "*Khi Y xảy ra thì UC kết thúc*".
- Tương tác giữa UC và tác nhân – mô tả rõ ràng cái bên trong hệ thống và cái bên ngoài hệ thống.
- Trao đổi thông tin – tương ứng với các tham số của tương tác giữa hệ thống và tác nhân. nên mô tả theo phong cách không phải phần mềm, thí dụ, "*Người sử dụng làm việc với hệ thống và nhập tên, mật khẩu*".
- Niên đại và nguồn gốc của thông tin – mô tả khi nào hệ thống đòi hỏi thông tin bên trong và bên ngoài, khi nào hệ thống lưu trữ chúng.
- Lặp hành vi trong UC – có thể được mô tả bằng mã giả (*pseudo – code*).
- Tình thế phụ - phải được biểu diễn theo cách thống nhất giữa các UC.

Mỗi một hệ thống thường có khoảng từ 20-50 UC. Mỗi UC cần phải biểu diễn trọn vẹn một giao dịch giữa người sử dụng và hệ thống để cho lại kết quả nào đó. Sau khi xác định xong UC cho hệ thống thì phải kiểm tra xem liệu chúng đã được phát hiện hết chưa. Công việc này được thực hiện bằng cách trả lời các câu hỏi sau đây:

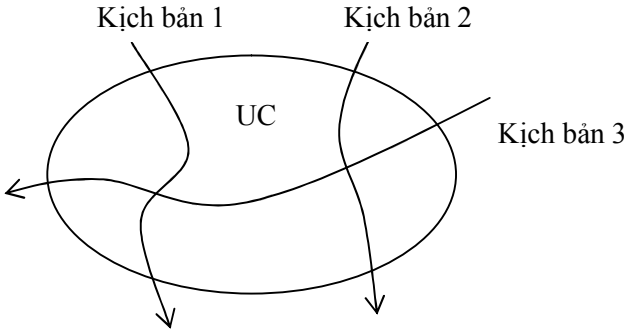
- Mỗi yêu cầu chức năng ở trong ít nhất một UC? Nếu yêu cầu chức năng không ở trong UC nào thì nó sẽ không được cài đặt sau này.
- Đã xem xét mọi tác nhân tương tác với hệ thống?
- Tác nhân cung cấp cho hệ thống thông tin nào?
- Tác nhân nhận thông tin nào từ hệ thống?
- Đã nhận biết mọi hệ thống bên ngoài mà hệ thống này tương tác?
- Thông tin nào hệ thống bên ngoài nhận và gửi cho hệ thống này?
- Các UC vừa nhận ra từ hệ thống có đặc trưng chính như sau:
- UC luôn được tác nhân kích hoạt, tác nhân ra lệnh trực tiếp hay gián tiếp hệ thống thực hiện UC. UC nối với tác nhân bằng quan hệ kết hợp giao tiếp.

- UC cung cấp giá trị trở lại cho tác nhân; giá trị là cái gì đó mà tác nhân muốn nhận được từ UC.
- UC là hoàn chỉnh, nó được mô tả đầy đủ, không chia UC thành các UC nhỏ hơn.

3.1.4 - Luồng sự kiện trong UC

Như trình bày trên đây, UC bắt đầu mô tả cái hệ thống sẽ làm (không chỉ ra nó sẽ làm như thế nào). Nhưng khi xây dựng hệ thống ta cần mô nó tả chi tiết hơn, các chi tiết này (hành vi của UC) được viết trong tài liệu văn bản luồng sự kiện (*flow of events*) để dễ hiểu. mục đích của luồng sự kiện là làm tài liệu luồng logic đi qua các UC. Tài liệu luồng sự kiện mô tả chi tiết người sử dụng sẽ làm gì và hệ thống sẽ làm gì. Tuy là chi tiết nhưng luồng sự kiện vẫn độc lập ngôn ngữ. Thí dụ, ở đây không đề cập đến loại ngôn ngữ lập trình cụ thể nào sẽ sử dụng để cài đặt. Mỗi UC có nhiều luồng sự kiện (luồng chính, luồng phụ); không thể thể hiện mọi chi tiết của UC trong một luồng sự kiện.

Kịch bản (*scenario*) chỉ ra luồng sự kiện trong một thể hiện (*instance*) cụ thể của UC (hình 3.4). nó là trình tự hành động cụ thể để mô tả hành vi. Kịch bản đi xuyên suốt UC theo nhánh chính hay các nhánh phụ hoặc nhánh đặc biệt của đường đi. Kịch bản là một thể hiện của UC, tương tự đối tượng là thể hiện của lớp, nó cho biết cách sử dụng của hệ thống. khách hàng hiểu hệ thống phức tạp thông qua tập kịch bản mô tả hành vi của nó. Thí dụ, trong một hệ thống có UC tuyển nhân viên. Chức năng tác nghiệp này có nhiều biến thể: tuyển nhân viên mới, thuyên chuyển nhân viên từ cơ quan khác, tuyển nhân viên nước ngoài. Mỗi biến thế biểu diễn trình tự khác nhau, mỗi trình tự đó là kịch bản.



Hình 3.4 Kịch bản trong UC

3.1.4.1 - Làm tài liệu luồng sự kiện

Thông thường tài liệu luồng sự kiện bao gồm: mô tả vắn tắt UC; tiền điều kiện; luồng sự kiện chínhl; luồng sự kiện rẽ nhánh và hậu điều kiện UC. Chúng sẽ được xem xét kỹ lưỡng hơn như dưới đây.

Mô tả. Mỗi UC cần có mô tả ngắn gọn là nó sẽ làm gì. Thí dụ, UC chuyển tiền của hệ thống ATM sẽ có mô tả như sau: UC chuyển tiền cho phép khách hàng hay nhân viên nhà băng chuyển số tiền từ tài khoản này sang tài khoản khác.

Mô tả UC phải ngắn gọn nhưng phải đầy đủ về loại người sử dụng sẽ sử dụng UC và kết quả cuối cùng mà UC cho lại. các định nghĩa UC này sẽ giúp toàn bộ đội ngũ của dự án thấy rõ tại sao dự án lại có UC này và UC này định làm cái gì.

Tiền điều kiện (*Pe –conditions*). Tiền điều kiện của một UC liệt kê các điều kiện cần được thực hiện trước khi UC khởi động. không phải UC nào cũng có tiền điều kiện. Thí dụ, tiền điều

kiện có thể là UC nào đó được thực hiện trước UC này hay tiền điều kiện là người sử dụng phải có quyền xâm nhập để thực hiện UC này.

Luồng sự kiện chính và luồng rẽ nhánh. Mọi chi tiết của UC được mô tả trong các luồng sự kiện chính và luồng rẽ nhánh. Nó mô tả từng bước cái gì sẽ xảy ra để thực hiện các chức năng của UC. Luồng sự kiện tập trung vào cái hệ thống sẽ làm chứ không tập trung vào nó làm như thế nào, và được mô tả từ góc nhìn của người sử dụng. Luồng sự kiện chính và luồng rẽ nhánh bao gồm:

- UC khởi động như thế nào
- Các đường đi xuyên qua các UC
- Luồng chính thông qua UC
- Luồng rẽ nhánh thông qua UC
- Các luồng lỗi
- UC kết thúc thế nào.

Khi làm tài liệu luồng sự kiện có thể sử dụng các hình thức khác nhau như mô tả bằng văn bản, biểu đồ luồng (*flow chart*)… bằng cách nào đi nữa thì luồng sự kiện phải phù hợp với yêu cầu hệ thống. khách hàng có tài liệu này để khẳng định tính chính xác cái mà họ mong đợi.

Sau đây là thí dụ làm tài liệu luồng sự kiện của UC rút tiền trong hệ thống rút tiền tự động.

Luồng chính

1. UC bắt đầu khi khách hàng đặt thẻ vào máy ATM
2. ATM hiển thị thông báo và chờ khách hàng số căn cước cá nhân (*PIN*)
3. Khách hàng nhập PIN
4. ATM khẳng định PIN có hợp lệ? Nếu PIN không hợp lệ thì thực hiện luồng A1
5. ATM hiển thị các lựa chọn
    a. Gửi tiền vào tài khoản
    b. Rút tiền
    c. Chuyển tiền sang tài khoản khác
6. Khách hàng chọn *Rút tiền*
7. ATM hiển thị câu hỏi số tiền sẽ rút
8. Khách hàng nhập số tiền muốn rút lần này.
9. ATM xác định số dư còn đủ? Nếu không đủ thực hiện luồng A2. Nếu phát hiện lỗi khi kiểm tra số dư thì thực hiện luồng E1
10. ATM trừ số tiền trong tài khoản của khách hàng
11. ATM chuyển tiền cho khách hàng
12. ATM in biên nhận
13. ATM trả lại thẻ tín dụng
14. UC kết thúc.

Luồng nhánh A1: số căn cước cá nhân *(PIN) không hợp lệ*

1. ATM thông báo căn cước cá nhân (*PIN*) không hợp lệ

2. ATM trả lại thẻ tín dụng
3. Kết thúc UC.

Luồng nhánh A2: không đủ tiền trong tài khoản để rút

1. ATM thông báo không đủ tiền
2. ATM trả lại thẻ tín dụng
3. Kết thúc UC.

Luồng E1: lỗi xảy ra khi kiểm tra số dư không đủ

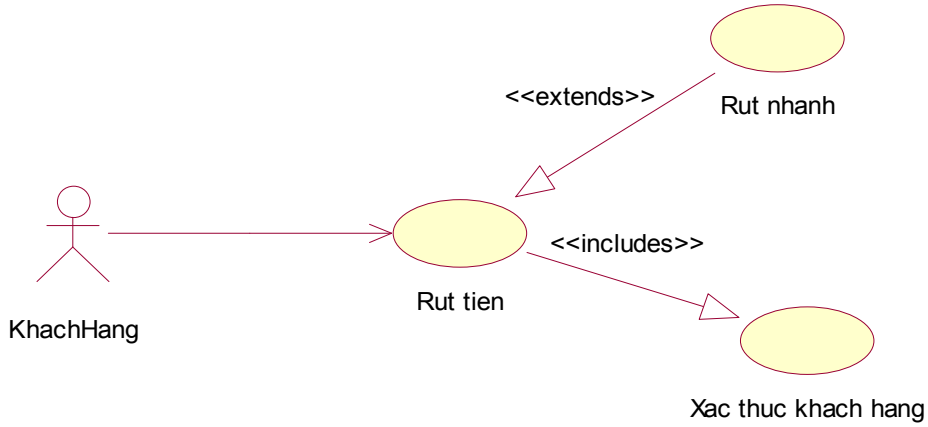
1. ATM thông báo lỗi
2. ATM ghi thông báo này vào tệp chứa lỗi (*error log*)
3. ATM trả lại thẻ tín dụng
4. Kết thúc UC

Hậu điều kiện (post–conditions). Hậu điều kiện là điều kiện được thực hiện ngay sau khi kết thúc UC. Nó mô tả trạng thái hệ thống hay tác nhân sau khi hoàn thành UC. Thí dụ, phải đặt cờ báo hiệu UC đã hoàn thành. Tương tự tiền điều kiện, hậu điều kiện có thể chứa thông tin ra lệnh UC khác chạy. không phải mọi UC trong hệ thống đều có hậu điều kiện.

3.2 BIỂU ĐỒ TRƯỜNG HỢP SỬ DỤNG

Mô hình UC trong UML được mô tả bằng một hay nhiều biểu đồ UC. Các biểu đồ UC là công cụ mạnh để thu thập yêu cầu hệ thống. chúng hiển thị các UC, làm dễ dàng các giao tiếp giữa các phân tích viên hệ thống, người sử dụng và giữa phân tích viên hệ thống vời khách hàng. Biểu đồ UC chỉ ra quan hệ giữa các UC và tác nhân. thông thường phải tạo ra vài biểu đồ UC cho một hệ thống. biểu đồ mức cao, *Rational Rose* gọi nó là biểu đồ chính (*Main*), chỉ ra gói hay nhóm UC. Các biểu đồ khác chỉ ra tập các UC và tác nhân. số lượng biểu đồ UC trong một dự án là tùy ý (tốt nhất là khoảng 50 cho dự án tương đối lớn). số lượng biểu đồ đảm bảo đầy đủ thông tin, nhưng cũng không quá nhiều vì sẽ dẫn tới rối loạn. mục đích chính của biểu đồ UC là làm tài liệu tác nhân (mọi thứ bên ngoài phạm vi hệ thống) UC mọi thứ bên trong phạm vi hệ thống) và các quan hệ giữa chúng. Một vài điều cần chú ý khi tạo lập UC như sau:

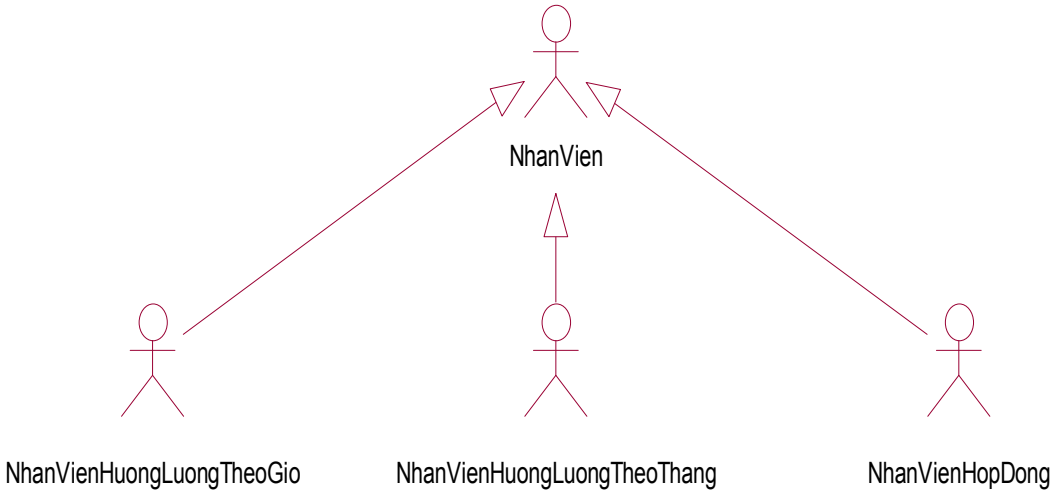
- Không nên mô hình hóa giao tiếp tác nhân – tác nhân. theo như định nghĩa thì tác nhân nằm ngoài hệ thống, do vậy giao tiếp giữa các tác nhân cũng nằm ngoài phạm vi hệ thống đang xây dựng. Chỉ nên sử dụng biểu đồ luồng công việc (*workflow*) để khảo sát quan hệ này.
- Không hình thành quan hệ trực tiếp giữa hai UC, trừ trường hợp chúng có quan hệ sử dụng (*Uses*) hay quan hệ mở rộng (*extend*s). Biểu đồ UC cho biết có những UC nào trong hệ thống nhưng không cho biết trật tự thực hiện các UC. Biểu đồ hoạt động sẽ cho biết trật tự này.
- Mỗi UC phải được tác nhân khởi động, trừ trường hợp đặt biệt khi UC có quan hệ sử dụng hay mở rộng
- CSDL được xem như một lớp dưới toàn bộ biểu đồ UC. Có thể nhập dữ liệu vào CSDL bằng một UC và sử dụng UC khác để xâm nhập chúng. Không hình thành luồng thông tin giữa các UC.



Hình 3.5 Các UC trừu tượng

Một UC không được tác nhân khởi động thì gọi là UC trừu tượng. UC trừu tượng cung cấp một số chức năng cho UC khác sử dụng. Chúng là những UC tham gia vào quan hệ *Uses* hay *extends*. Hình 3.5 là thí dụ phần biểu đồ có sử dụng UC trừu tượng.

Tác nhân mà không có hiện thực (*isntance*) thì gọi là tác nhân trừu tượng. Thí dụ của tác nhân trừu tượng như sau: chia nhân viên làm việc trong một cơ quan thành ba nhóm. Bao gồm nhân viên hưởng lương tháng, nhân viên hưởng lương theo giờ và nhân viên hợp đồng. Ba loại nhân viên này được xem như các tác nhân trong hệ thống quản lý nhân sự. như vậy, trong cơ quan không có ai được gọi là *Nhân viên* mà phải là *Nhân viên hưởng lương tháng*, *Nhân viên hưởng lương giờ* hay *Nhân viên hợp đồng*. Tuy nhiên ta vẫn có thể tạo ra tác nhân *Nhân viên* mang các tính chất chung của ba loại nhân viên nói trên. Nhưng không có hiện thực nào cho tác nhân nhân viên, vậy nó là tác nhân trừu tượng. hình 3.6 mô tả tác nhân trừu tượng và các quan hệ của nó với các tác nhân

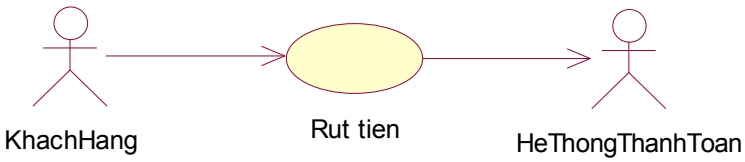


hình 3.6 Tác nhân trừu tượng

Các kiểu quan hệ giữa UC và tác nhân trong UML bao gồm quan hệ giao tiếp, quan hệ sử dụng (*Uses*) quan hệ mở rộng (*extends*) và quan hệ tổng quát hóa (*generallization*) tác nhân.

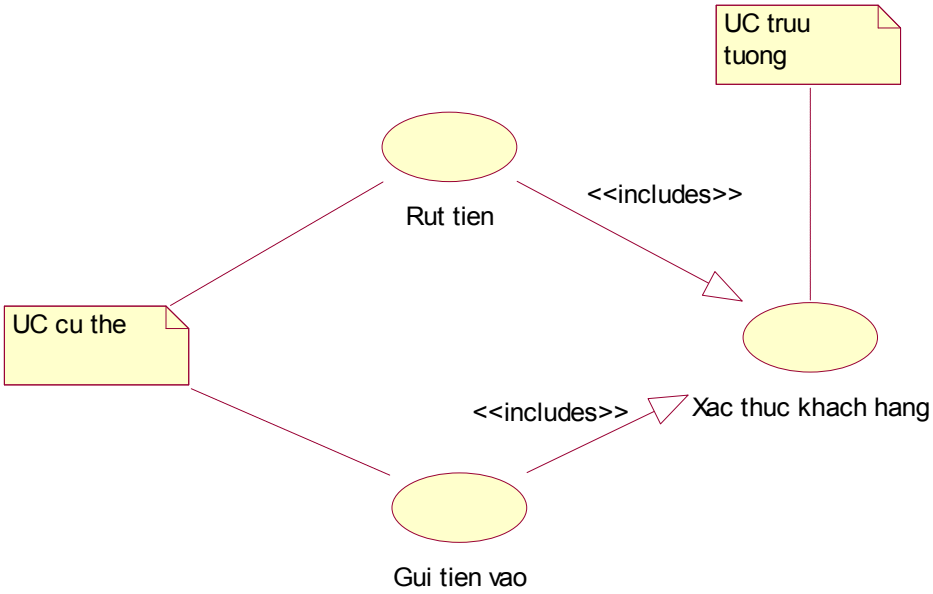
Quan hệ giao tiếp. trong UML. Quan hệ được thể hiện bằng mũi tên. Thí dụ hình 3.7 mô tả tác nhân *Khách hàng* có quan hệ với hệ thống để chạy chức năng *Rút tiền*. Hình này còn cho thấy UC *Rút tiền* kích hoạt tác nhân *Hệ thống thanh toán.*

Quan hệ sử dụng (*Uses*). Trong phiên bản UML 1.3, quan hệ *Uses* được gọi là quan hệ gộp (*include*). Quan hệ *Uses* cho phép một UC sử dụng chức năng của UC khác. Quan hệ này thường được sử dụng để mô hình hóa một vài chức năng sử dụng lại, dùng chung cho hai hay nhiều UC. Trong thí dụ máy rút tiền tự động ATM, hai UC *Rút tiền* và *Gửi tiền vào ngân hàng* cần xác định khách hàng và *số căn cước cá nhân* (PIN) trước khi thực hiện giao dịch. Do vậy chức năng xác nhận này có thể đặt trong một UC riêng, gọi là *Xác nhận khách hàng*. Vào bất kỳ thời điểm nào, UC nào cần xác định khách hàng thì sử dụng UC này.



Hình 3.7 Quan hệ giao tiếp

Trong UML quan hệ *Uses* được mô tả bằng mũi tên và từ *<<includes>>* như trên hình 3.8. trên hình vẽ này, hai UC *Rút tiền* và *Gửi tiền vào* sử dụng chức năng trong UC *Xác nhận khách hàng*. UC *Xác nhận khách hàng* là UC trừu tượng . Còn các UC *Rút tiền* và *Gửi tiền vào* là các UC cụ thể. Các hình chữ nhật gấp góc là các chú thích của biểu đồ

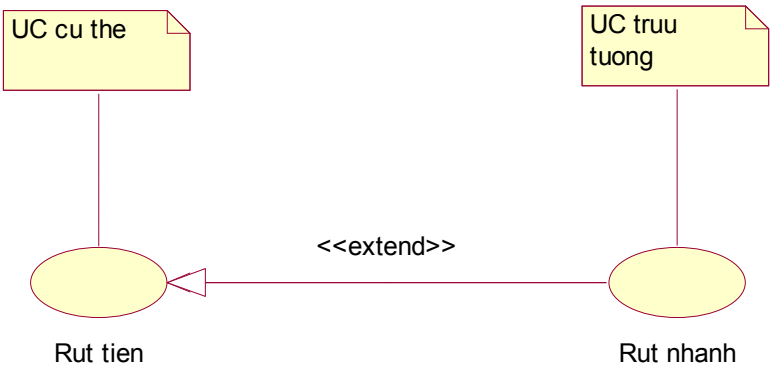


Hình 3.8 Quan hệ include

Quan hệ mở rộng (*extends*). Quan hệ mở rộng cho phép UC mở rộng tùy ý chức năng do UC khác cấp. nó chỉ ra rằng trong một điều kiện nào đó một UC được mở rộng bằng UC khác (mở rộng là gộp vài hành vi của UC tổng quát hơn để sử dụng lại). nó tương tự như quan hệ *Uses* ở chỗ cả hai quan hệ đều tách phần chức năng chung ra một UC mới, đó là UC trừu tượng. Trong UML, quan hệ mở rộng được vẽ bằng mũi tên và có từ *<<extends>>* như trên hình 3.9.

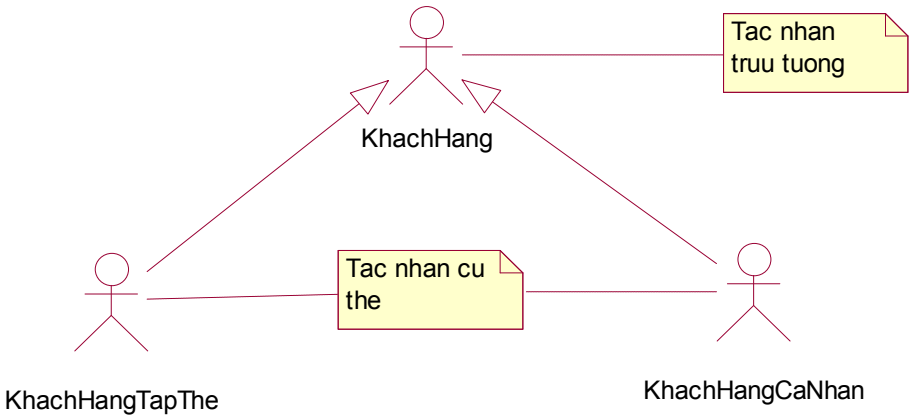
Thí dụ trên hình 3.9 mô tả UC *Rút tiền* đôi khi sử dụng chức năng trong UC *Rút tiền nhanh*. Chỉ và chỉ khi khách hàng chọn thuộc tính rút nhanh dành cho rút số tiền không lớn (thí dụ không rút quá 100.000đ) thì UC này mới chạy. Nó có thể bỏ qua một số thao tác ít quan trọng. UC *Rút nhanh* cung cấp chức năng mở rộng. Nó là UC trừu tượng còn *Rút tiền* là UC cụ thể.

Quan hệ tổng quát hóa (*generalization*) tác nhân. Quan hệ tổng quát hóa tác nhân



Hình 3.9 Quan hệ mở rộng

được sử dụng để chỉ ra một vài tác nhân có một số cái chung. Thí dụ hệ thống có hai loại khách hàng, khách hàng cá nhân và khách hàng tập thể. Biểu đồ trên hình 3.10 mô tả hai loại khách hàng này. Khách hàng cá nhân và tập thể là tác nhân cụ thể. ta có thể định nghĩa loại tác nhân tổng quát (*Khách hàng*) và đặc biệt hóa chúng nhờ quan hệ tổng quát hóa (để có tác nhân *Khách hàng cá nhân* và *Khách hàng tập thể*). Tác nhân *Khách hàng* là tác nhân trừu tượng. ta có thể tiếp tục chia nhỏ, thí dụ chia *Khách hàng tập thể* thành các *Công ty tư nhân* và *Tổ chức chính phủ*… để xây dựng quan hệ tổng quát hóa.



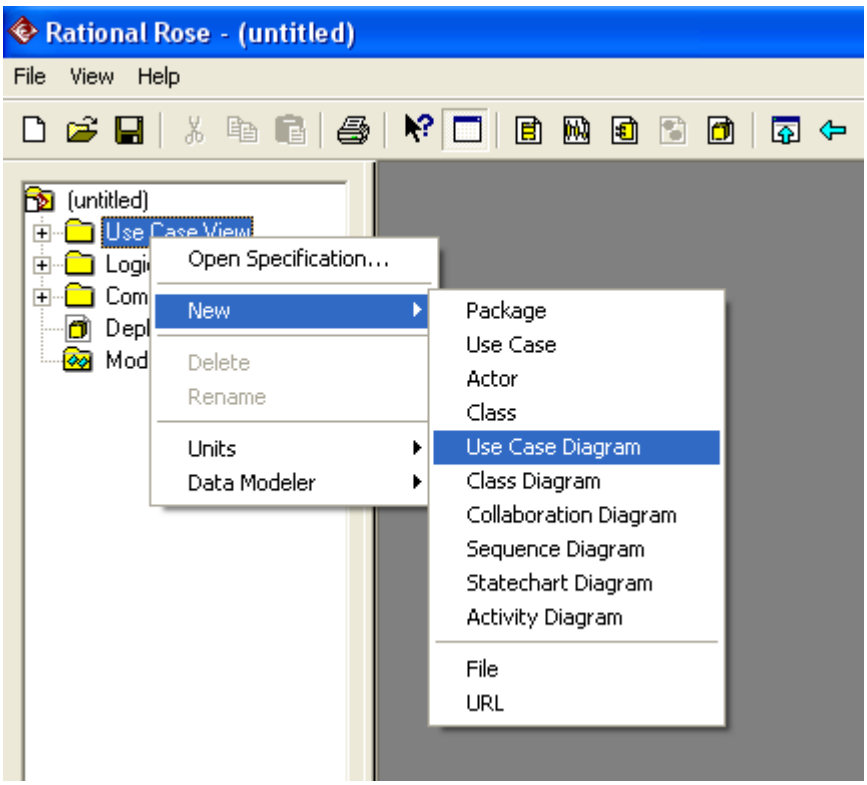
Hình 3.10 Quan hệ tổng quát hóa tác nhân

Không nhất thiết lúc nào cũng tạo ra những kiểu quan hệ như trình bày trên đây. Nếu khách hàng là tập thể tác động vài UC mà khách hàng cá nhân không tác động thì nên có tổng quát hóa tác nhân. Nếu cả hai sử dụng cùng UC thì không cần tổng quát hóa tác nhân. thí dụ biểu đồ UC của hệ thống ATM đã được mô tả trên hình 2.16 của chương 2.

3.3 THỰC HÀNH

3.3.1 - Sử dụng Rational Rose.

Khởi động phần mềm *Rational Rose 2000*.



Hình 3.11 Bổ sung UC mới

3.3.1.1 - Tạo lập biểu đồ trường hợp sử dụng (UC)

Trong Rose các biểu đồ UC được tạo lập trong khung nhìn *USE case*. Rose cung cấp biểu đồ UC mặc định có tên *Main*. Số lượng biểu đồ UC có thể được tạo là tùy ý. Để xâm nhập vào biểu đồ *Main USE case* ta làm như sau (hình 3.11):

1. Nhấn chuột vào dấu + cạnh *USE Case View* trong Browser để mở chúng.
2. Nhìn thấy được biểu đồ *Main USE Case*.
3. Nhấp đúp trong *Main diagram* để mở chúng

Tạo lập biểu đồ USE case mới:

1. Nhấn phím phải trong gói *USE Case View* trong Browser
2. Chọn *New* > *USE Case Diagram* từ thực đơn
3. Đặt tên cho biểu đồ mới.
4. Nhấp đúp trên tên của biểu đồ mới để mở chúng.

Sử dụng các phím trên thanh công cụ để bổ sung các phần tử biểu đồ vào biểu đồ UC mới.

3.3.1.2 - Bãi bỏ biểu đồ UC

Có thể bãi bỏ biểu đồ UC trong Browser, một khi biểu đồ UC bị xóa thì không thể lấy lại nó. Trình tự xóa biểu đồ UC như sau:

1. Nhấn phím phải trên biểu đồ UC trong Browser
2. Chọn *Delete* từ thực đơn.

3.3.1.3 - Bổ sung UC

Có hai cách bổ sung USE Case vào mô hình: bổ sung UC vào biểu đồ USE case tích cực hay bổ sung trực tiếp USE case mới vào Browser. Bổ sung USE case mới vào biểu đồ USE case được thực hiện như sau

1. Chọn phím *USE case* trên thanh công cụ.
2. Nhấn bất kỳ đâu trên biểu đồ Use case. Use case mới có tên là *NewUSECase*.
3. Nhập tên cho USE case mới.
4. Use case mới tự động cập nhật vào cửa sổ BROWSER, dưới khung nhìn UC.

Bổ sung USE case tồn tại trong mô hính vào biểu đồ USE case:

1. Di USE case từ BROWSER vào biểu đồ USE case đang mở

Bổ sung USE case vào BROWSER được thực hiện như sau:

Nhấn phím chuột phải trên gói khung nhìn Use case trong Browser

2. Chọn *New> USE Case*
3. Use case mới sẽ xuất hiện trong BROWSER
4. Đặt tên cho USE case mới
5. Để gắn USE case vào biểu đồ, di USE case mới từ BROWSER vào biểu đồ.

3.3.1.4 - Bãi bỏ USE case

Có khả năng bãi bỏ UC trong một biểu đồ hay trong toàn bộ các biểu đồ của mô hình. Việc bỏ UC khỏi biểu đồ UC như sau:

1. Chọn USE case trong biểu đồ
2. Nhấn phím *Delete*
3. Use case đã chọn sẽ biến mất khỏi biểu đồ USE case nhưng nó còn trong BROWSER và trong các biểu đồ khác

Việc bãi bỏ USE case trong toàn bộ mô hình được thực hiện như sau:

1. Chọn USE case trong biểu đồ
2. Chọn *Edit> Delete from Model* hay nhấn các phím *Ctrl*+D
3. Use case vừa chọn bị loại bỏ trong toàn bộ mô hình và BROWSER.

3.3.1.5 - Đặc tả USE case

Rose cho khả năng đặc tả chi tiết từng USE case, cho khả năng làm tài liệu thuộc tính như tên, mức ưu tiên, *stereotype* của USE case.

Mở đặc tả Use case theo các bước sau:

1. Nhấn phím chuột phải trên Use case biểu đồ Use case
2. Chọn thực đơn *Open Specification*.

Hoặc

1. Nhấn phím chuột phải trên Use case trong Browser
2. Chọn thực đơn *Open Specification*.

3.3.1.6 - Đặt tên USE case

Mỗi Use case trong mô hình cần có tên duy nhất, độc lập với cài đặt. Thông thường sử dụng động từ hay các câu ngắn làm tên UC, đặt tên Use case như sau:

1. Chọn Use case trong Browser hay trong biểu đồ Use case
2. Nhập tên Use case.

Bổ sung tài liệu cho Use case:

1. Chọn Use case trong Browser
2. Nhập mô tả Use case trong cửa sổ tài liệu.

3.3.1.7 - Quan sát thành viên của USE case

Rose cho khả năng liệt kê toàn bộ các lớp và thao tác trong một UC cụ thể. Chức năng này cho ta dễ theo dõi sự ảnh hưởng đến các lớp khi thay đổi yêu cầu.

Để xem các lớp và thao tác tham gia trong Use case, ta làm như sau:

1. Chọn Use case trong biểu đồ Use case
2. Chọn *Report> Show Participants* trong UC
3. Cửa sổ *Participants* sẽ hiển thị

3.3.1.8 - Gán Stereotype cho Use case

Trong UML, sử dụng *stereotype* để phân nhóm các thành phần mô hình. Thí dụ, ta có hai *Use case A* và *Use case B*. Ta có thể tạo ra hai *stereotype* cho *Use case A* và *B*. *Stereotype* không được sử dụng thường xuyên cho Use case, nó thường sử dụng cho lớp hay cho quan hệ

Gán *stereotype* cho Use case như sau:

1. Nhấn phím chuột phải trên Use case trong Browser hay trong biểu đồ UC
2. Chọn *Open Specification*
3. Nhập *stereotype* trong vùng *stereotype.*

3.3.1.9 - Gán mức ưu tiên cho Use case

Có thể gán mức ưu tiên cho Use case để biết trật tự làm việc vơi các Use case. Gán mức ưu tiên thông qua *Open Specification* tương tự như trường hợp trên.

3.3.1.10 - Tạo lập USE case trừu tượng

Use case trừu tượng là Use case không được kích hoạt trực tiếp từ tác nhân. Nó cung cấp thêm một số chức năng để Use case khác sử dụng. tạo Use case trừu tượng như sau:

1. Tạo Use case trong Browser hay biểu đồ Use case
2. Nhấn phím chuột phải trên biểu đồ Use case hay trong Browser
3. Chọn *Open Specification*
4. Đánh dấu hộp *Abstract*

3.3.1.11 - Gắn tệp vào UC

Sau khi chọn thực đơn đặc tả UC (*Open Specification*), ta có thể quan sát các biểu đồ định nghĩa trong UC, các quan hệ giữa các UC và giữa UC với tác nhân. đồng thời, bảng *Files* trong cửa sổ này còn cho khả năng gán tệp vào UC. Nội dung tệp có thể là tài liệu về luồng sự kiện cho UC, đặc tả các yêu cầu, các kịch bản… Gán tệp vào UC theo các bước sau:

1. Nhấn phím chuột phải vào vùng trắng của *Files tab*
2. Chọn thực đơn *Insert File* để xen tệp
3. Sử dụng hộp thoại *Open* để tìm tên tệp sẽ gán
4. Chọn *Open* để gắn tệp vào UC.

3.3.1.12 - Bổ sung tác nhân

Tương tự như Use case, có thể bổ sung tác nhân vào biểu đồ UC hay trực tiếp vào browser. Một tác nhân trong browser có thể được bổ sung vào một hay nhiều biểu đồ UC.

Trình tự bổ sung tác nhân vào biểu đồ UC như sau:

1. Chọn phím *Actor* trên thanh công cụ
2. Nhấn bất kỳ đâu trong biểu đồ UC. Tác nhân mới sẽ được gán tên mặc định là *NewClass*
3. Nhập tên cho tác nhân mới, nó được tự động gán vào browser.

3.3.1.13 - Hủy bỏ tác nhân

Tương tự như Use case, nếu muốn hủy bỏ tác nhân khỏi biểu đồ UC thì chọn chúng rồi nhấn phím *Delete*. Nếu muốn loại bỏ tác nhân khỏi mô hình thì nhấn phím *Ctrl+D* sau khi đã chọn nó.

3.3.1.14 - Đặc tả tác nhân

Tương tự như Use case, tác nhân cũng có cửa sổ đặc tả để đặt tên, *stereotype*, số yếu tố (*cardinality*) và các chi tiết khác

Sau khi nhấn phím phải chuột trên tác nhân trong biểu đồ UC hay browser ta có thể chọn thực đơn *Open Specification* để gán tên, *stereotype*… cho tác nhân cách làm này tương tự như gán đặc tả cho UC vừa mô tả trên.

3.3.1.15 - Đặt số yếu tố (cardinality) tác nhân

Số yếu tố chỉ ra tổng số hiện thực (*instance*) của tác nhân. thí dụ số yếu tố được sử dụng để thể hiện nhiều người là khách hàng, một người làm quản trị

Số yếu tố	*Ý nghĩa*
n (mặc định)	Nhiều
0..0	zero
0..1	0 hay 1
0..nhiều	0 hay nhiều
1..1	chỉ một
1..n	một hay nhiều

Nhập số yếu tố trong *Detail tab* sau khi chọn thực đơn *Open Specification*.

3.3.1.16 - Tạo lập tác nhân trừu tượng

Tạo tác nhân trừu tượng là tác nhân không có hiện thực (instance), số yếu tố của nó bằng 0. tác nhân trừu tượng được tạo lập như sau:

1. Tạo tác nhân trong Browser hay biểu đồ UC
2. Nhấn phím phải chuột trên tác nhân
3. Chọn thực đơn *Open Specification*
4. Chọn *Detail tab*
5. Đánh dấu hộp thoại *Abstract*

3.3.1.17 - Các quan hệ trong biểu đồ

Bổ sung quan hệ uses vào biểu đồ UC theo các bước sau

1. Chọn phím *Generalization* trên thanh công cụ
2. Nối UC cụ thể để UC trừu tượng
3. Nhấn phím phải chuột trên đường quan hệ, chọn thực đơn *Open Specification*
4. Trong cửa sổ *Stereotype*: nhập uses
5. Mở cửa sổ *UC specification* của UC trừu tượng
6. Đánh dấu vào hộp thoại *Abstract*

Bãi bỏ quan hệ uses trong biểu đồ UC theo các bước sau:

1. Chọn quan hệ trên biểu đồ UC
2. Chọn *Edit> Delete from ModelI* hay nhấn phím *Ctrl+D.*

Việc bổ sung và bãi bỏ quan hệ *extends* trong biểu đồ UC được thực hiện tương tự như bổ sung hay bãi bỏ quan hệ *uses* như vừa mô tả trên đây.

Bổ sung quan hệ khái quát hóa tác nhân (*actor generalization*) như sau:

1. Chọn quan hệ trên biểu đồ UC
2. Chọn phím *Generalization* trên thanh công cụ
3. Vẽ từ tác nhân cụ thể sang tác nhân trừu tượng
4. Mở cửa sổ đặc tả tác nhân của tác nhân trừu tượng
5. Chọn *Detail tab*
6. Đánh dấu *Abstract*.

Quan hệ khái quát hóa tác nhân được hủy bỏ bằng cách nhấn phím *Ctrl+D* sau khi đã chọn quan hệ trên biểu đồ UC

3.3.1.18 - Làm việc với ghi chú (notes)

Khi tạo biểu đồ, nên gắn ghi chú vào UC hay tác nhân. thí dụ ghi chú làm rõ tại sao tác nhân lại tương tác với UC, tại sao UC lại có quan hệ *uses* hay *extends*, tại sao UC hay tác nhân này lại kế thừa từ UC hay tác nhân khác. *Rational Rose* có công cụ (phím trên thanh công cụ) để làm việc này. Bổ sung ghi chú vào biểu đồ như sau:

1. Chọn phím *Note* trên thanh công cụ
2. Nhấn chuột vào nơi muồn đặt ghi chú trên biểu đồ
3. Nhập văn bản cho nó.

Gắn ghi chú vào phần tử mô hình sau:

1. Chọn phím *Anchor Note to Item* trên thanh công cụ
2. Di từ ghi chú đến tác nhân liên quan. *Rose* sẽ vẽ đường nét đứt giữa chúng

Bãi bỏ ghi chú bằng cách thực hiện các bước sau:

1. Chọn ghi chú hay hộp văn bản trong biểu đồ
2. Nhấn phím *Delete* trên bàn phím.

3.3.1.19 - Làm việc với gói (packages)

Khi hiển thị, đặc tả, xây dựng và làm tài liệu hệ thống lớn thì phải quản lý số lượng rất lớn các phần tử mô hình. Do vậy phải có cơ chế tổ chức chúng lại thành gói. Trong UML, gói là cơ chế đa năng để tổ chức các phần tử mô hình vào nhóm. Các tác nhân, UC, lớp, thành phần đều có thể nhóm trong gói. Trong khung nhìn UC ta có thể nhóm UC và tác nhân vào gói. Các phần tử có ngữ nghĩa gần nhau thì được nhóm vào cùng gói.

Bổ sung gói vào mô hình trong *Rose 2000* như sau:

1. Nhấn phím phải chuột trên khung nhìn UC trong browser
2. Chọn *New> Package*

3. Nhập tên của gói mới

Gói trong biểu đồ UC bị hủy bỏ bằng cách nhấn các phím *Ctrl+D* sau khi nó đã được chọn, hoặc nhấn phím phải chuột trên gói định hủy bỏ trong browser rồi chọn thực đơn *Delete.*

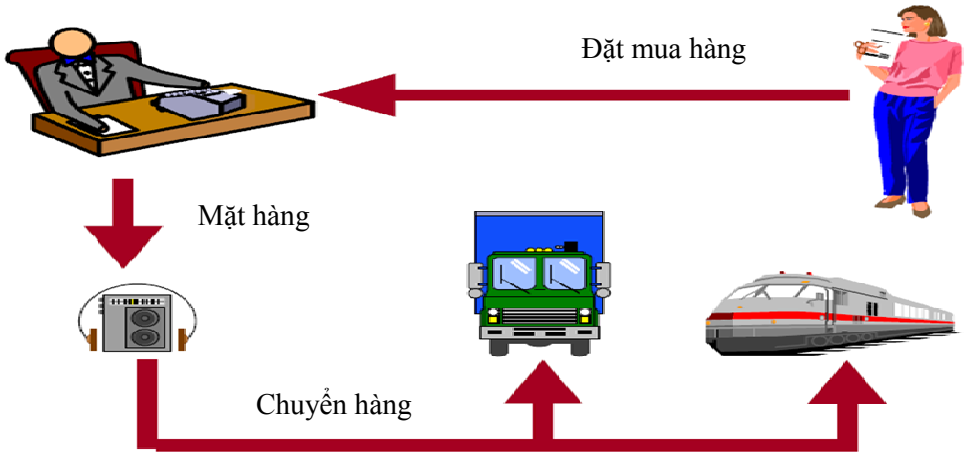
3.3.2 - Thí dụ: hệ thống bán hàng

3.3.2.1 - Vấn đề cần giải quyết

Thí dụ này liên quan tới xây dựng hệ thống theo dõi bán hàng. Mô tả công việc của người sử dụng hệ thống như sau (hình 3.13):

Khi nhận được điện thoại đặt hàng, họ sẽ điền vào mẫu yêu cầu mua hàng (*form*). Sau đó gửi yêu cầu này tới thủ kho. Thủ kho sẽ điền các mặt hàng vào hóa đơn mua hàng (*order*) và chuẩn bị chuyển hàng đến khách hàng. Một bản sao đơn hàng sẽ được gửi cho kế toán. Kế toán sẽ nhập vào hệ thống kế toán để trong báo giá.

Từ mô tả công việc bán hàng như trên, ta có thể xác định rằng hệ thống cần cho khả năng nhập các đơn đặt hàng mới (*Enter New Order*), sửa đổi đơn đăt hàng có sẵn (*Modify Existing Order*), điền đơn hàng (*Fill Order*), trong báo cáo kiểm kê kho (*Print Inventory Report*) và nhập hàng vào kho (*Restock Inventory*). Khi bổ sung đơn hàng mới, hệ thống cần thông báo cho hệ thống kế toán (*Accounting System*) để trong báo giá. Nếu hết mặt hàng nào đó trong kho, hệ thống sẽ từ chối (*Backorder*) bán mặt hàng này. Hình 3.14 là biểu đồ kết quả, nó được xây dựng bằng phần mềm công cụ *Rational Rose 2000* theo những bước mô tả dưới đây.



Hình 3.13 Tiến trình mua bán hàng

3.3.2.2 - Xây dựng biểu đồ Use case

Tạo lập biểu đồ UC cho hệ thống xủ lý đơn hàng:

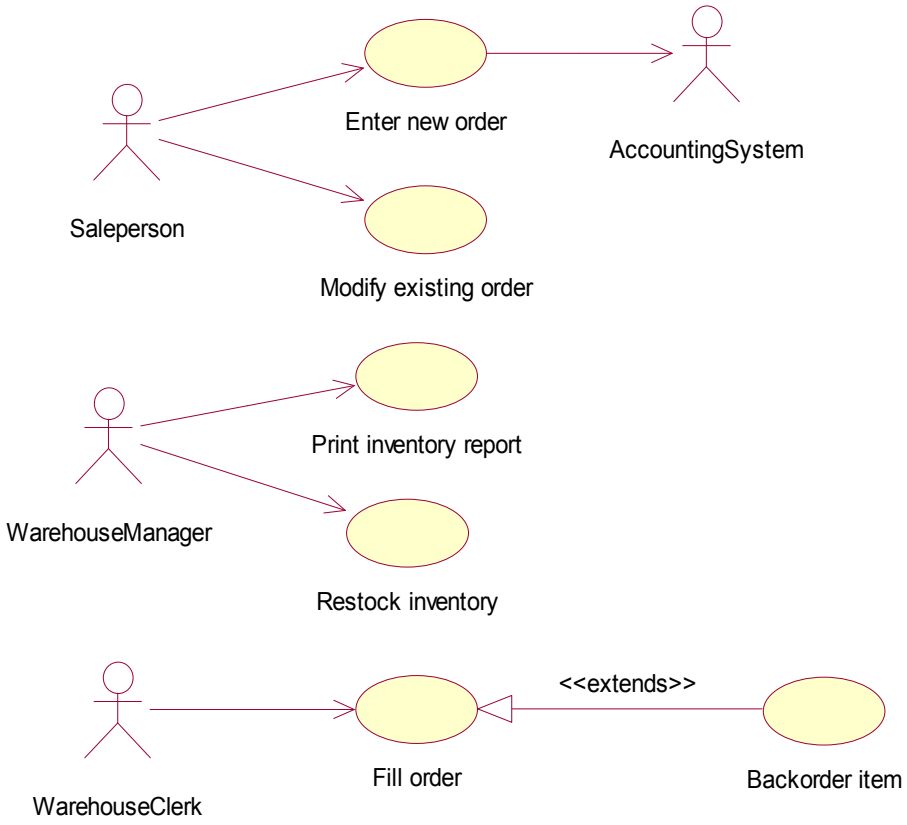
1. Nhấn đúp chuột trên biểu đồ *Main Use Case* trong browser để vẽ nó
2. Sử dụng phím UC để vẽ UC mới
3. Nhập tên cho UC: *Enter New Order*
4. Lặp lại để vẽ UC khác như sau
    a. *Modify Existing Order*
    b. *Print Inventory Report*

c. *Restock Inventory*

d. *Fill Order*

e. *Backorder Item*

5. Sử dụng phím *Actor* để vẽ tác nhân
6. Đặt tên tác nhân mới là *Salesperson*.



Hình 3.14 Biểu đồ UC của hệ thống bán hàng

7. Lặp lại để vẽ các tác nhân sau
    a. Warehourse Manager
    b. Warehourse Clerk
    c. Accounnting System

Đánh dấu UC trừu tượng

8. Nhấn phím phải chuột trên UC Backorder Item trong biểu đồ UC
9. Chọn thực đơn *Open Specification*
10. Đánh dấu hộp thoại *Abstract*

Bổ sung quan hệ

1. Sử dụng phím *Unidirectional Associaton* trên thanh công cụ để vẽ quan hệ giữa *Salesperson* với *Enter New Order*

2. Lặp lại cho các quan hệ còn lại

Bổ sung quan hệ extends

1. Sử dụng phím *Generalization* trên thanh công cụ để vẽ vào biểu đồ
2. Nhấn phím phải chuột trên quan hệ vừa vẽ
3. Chọn thực đơn *Open Specification*
4. Nhập từ ẽ trong hộp danh sách Stereotype

Bổ sung mô tả UC

1. Chọn UC *Enter New Order* trong browser
2. Nhập mô tả vào cửa sổ tài liệu. thí dụ, mô tả có thể là: trường hợp sử dụng này cho người sử dụng khả năng bổ sung đơn hàng mới vào hệ thống
3. Tương tự, nhập mô tả vào các cửa sổ tài liệu cho các UC khác

Bổ sung mô tả tác nhân

1. Chọn tác nhân *Salesperson*
2. Nhập mô tả vào cửa sổ tài liệu. thí dụ, mô tả có thể là: *Nhân viên bán hàng là người thực hiện nhiệm vụ buôn bán sản phẩm*.
3. Tương tự, bổ sung mô tả cho các tác nhân khác.

Gắn tệp vào UC

1. Gắn tệp mô tả luồng sự kiện vào UC *Enter New Order*. Trước khi gắn phải tạo tệp văn bản mô tả luồng sự kiện. Thí dụ, tạo tệp *OrderFlow.doc* với mô tả như sau:
    a. Nhân viên bán hàng chọn thực đơn *Create New Order* để thực hiện
    b. Hệ thống hiển thị mẫu khai *Order Detail*
    c. Nhân viên bán hàng nhập số đơn hàng, tên khách hàng và các mặt hàng mà khách yêu cầu

Nhân viên bán hàng lưu lại đơn hàng

    d. Hệ thống tạo ra đơn hàng mới và lưu nó vào cơ sở dữ liệu
2. Nhấn phím phải chuột trên UC *Enter New Order*
3. Chọn thực đơn *Open Specification*
4. Chọn bảng *Files*
5. Chọn tên tệp để gắn vào UC (nhấn phím phải vào vùng *Filename*)

CHƯƠNG 4
MÔ HÌNH HÓA
TƯƠNG TÁC ĐỐI TƯỢNG

Chương này trình bày về mô hình hóa quan hệ giữa các đối tượng trong hệ thống. Hai loại biểu đồ sẽ được xem xét ở đây, đó là biểu đồ trình tự và biểu đồ cộng tác. Hai loại biểu đồ này được gọi một tên chung, đó là biểu đồ tương tác.

4.1 ĐỐI TƯỢNG VÀ TÌM KIẾM ĐỐI TƯỢNG

Đối tượng là cái để gói thông tin và hành vi. Thí dụ, hệ thống ATM có các đối tượng như màn hình ATM, máy đọc thẻ, giấy biên nhận, tài khoản của khách hàng… Mỗi đối tượng gói một vài thông tin và một vài hành vi. Thí dụ, đối tượng *Tài khoản của ông A* chứa thông tin như số tài khoản (*1234*), tên chủ tài khoản (*ông A*), số dư tài khoản (*20 triệu đồng*)… đối tượng này còn chứa các hành vi như cộng tiền vào tài khoản khi ông A gửi vào, trừ tiền trong tài khoản khi ông A rút tiền ra hay thông báo khi ông A rút quá số tiền trong tài khoản… Các vùng lưu giữ thông tin trong đối tượng được gọi là thuộc tính (*attribute*). Giá trị của thuộc tính thay đổi được (thí dụ số tiền trong tài khoản của ông A có thể tăng trong tuần tới) nhưng thuộc tính thì không. Hành vi đối tượng được gọi là thao tác (*operation*). Các thao tác trong thí dụ này thực hiện hiệu chỉnh cân đối tài khoản khi gửi hay rút tiền.

Cách tốt nhất để tìm ra đối tượng là khảo sát *danh từ* trong luồng sự kiện, hay tìm trong tài liệu kịch bản. Kịch bản là phiên bản cụ thể của luồng sự kiện. Thí dụ, một luồng sự kiện của UC *Rút tiền*. Một kịch bản của nó có thể là ông A rút 100.000đ. Kịch bản khác có thể là chị B rút 100.000đ nhưng chị ta nhập nhầm mật mã (*PIN*). Kịch bản khác có thể là ông C rút 100.000đ nhưng số dư tài khoản của ông ta chỉ là 90.000đ. Thông thường có nhiều kịch bản cho một luồng sự kiện. Mỗi biểu đồ trình tự hay biểu đồ cộng tác sẽ mô tả một trong các kịch bản này. Nếu tìm ra danh từ trong kịch bản thì một trong số đó có thể là tác nhân và một số khác có thể là đối tượng hay thuộc tính của đối tượng. Khi xây dựng biểu đồ tương tác thì danh từ cho biết cái nào là đối tượng. Nếu còn do dự rằng đó là đối tượng hay thuộc tính thì hãy hỏi xem nó có hành vi hay không. Nếu nó chỉ có thuộc tính thì đó có thể là thuộc tính, nếu có hành vi thì nó phải là đối tượng. Không nhất thiết mọi đối tượng là ở trong luồng sự kiện. Thí dụ mẫu báo cáo (*form*s) không trong luồng sự kiện nhưng nó xuất hiện trong biểu đồ để tác nhân có thể nhập hay xem dữ liệu. Các đối tượng điều khiển cũng không ở trong luồng sự kiện. Đối tượng điều khiển là đối tượng điều khiển trình tự luồng xuyên qua UC.

Hơn nữa có rất nhiều khái niệm trong khi phân tích sẽ được chuyển sang đối tượng khi thiết kế. Một số đối tượng trong thiết kế được ủy quyền từ các tác nhân hay khái niệm trừu tượng trong lĩnh vực vấn đề. Thí dụ, trong hệ thống quản lý thư viện, ta phải tách khái niệm tên sách và chính quyển sách. Tên sách là trừu tượng không tương ứng với đối tượng nào trong thế giới thực.

4.2 BIỂU ĐỒ TƯƠNG TÁC

Biểu đồ tương tác (*interaction diagram)* được sử dụng trong UML để mô hình hóa khía cạnh động của hệ thống (hệ thống đang chạy). Biểu đồ tương tác chỉ ra một tương tác, bao gồm tập đối tượng, quan hệ và các thông điệp trao đổi giữa chúng. Biểu đồ này chỉ ra từng bước của một luồng điều khiển cụ thể trong UC. UC *Rút tiền* trong hệ thống rút tiền tự động ATM có nhiều luồng sự kiện. Do vậy ta sẽ có nhiều biểu đồ tương tác cho UC này. UML có hai loại biểu đồ

tương tác, đó là biểu đồ trình tự và biểu đồ cộng tác. Biểu đồ trình tự theo trật tự thời gian, biểu đồ cộng tác chỉ ra cùng loại thông tin nhưng có cách tổ chức khác. Trong khi biểu đồ trình tự tập trung vào điều khiển thì biểu đồ cộng tác tập trung vào luồng dữ liệu. Các biểu đồ này đều liên quan đến các đối tượng cài đặt chức năng thực hiện trong UC. Hai biểu đồ này được xây dựng cho đối tượng lớp hay cả hai.

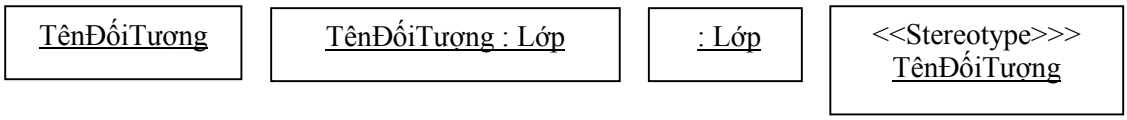
Khi xây dựng biểu đồ tương tác có nghĩa là ta gán trách nhiệm cho đối tượng. Thực hiện bổ sung thông điệp vào biểu đồ có nghĩa là gán trách nhiệm cho đối tượng nhận thông điệp. hãy đảm bảo rằng phải gán trách nhiệm phù hợp cho đối tượng tương ứng. Trong phần lớn các ứng dụng, đối tượng *màn hình* và đối tượng *mẫu báo cáo* thường không thực hiện tác nghiệp nào. Chúng chỉ được sử dụng để người dùng nhập liệu và quan sát thông tin. Do vậy, nếu logic tác nghiệp thay đổi sẽ không ảnh hưởng gì đến giao diện hệ thống và ngược lại, thay đổi giao diện sẽ không ảnh hưởng để tác nghiệp. Thí dụ, nếu cần in báo cáo về cân đối tài khoản, đối tượng *Tài khoản của ông A* không có trách nhiệm làm việc này, hãy gán trách nhiệm cho đối tượng khác, nó có khả năng tìm kiếm trong mọi tài khoản để hình thành báo cáo.

Từ biểu đồ tương tác, người thiết kế và người phát triển có thể xác định các lớp sẽ xây dựng, quan hệ giữa các lớp, thao tác và trách nhiệm của mỗi lớp. Biểu đồ tương tác trở thành nền tảng cho các công việc còn lại khi thiết kế. Biểu đồ trình tự là trật tự theo thời gian của các thông điệp, nó có ích khi ai đó muốn quan sát luồng logic trong kịch bản. Biểu đồ cộng tác có ích khi ta muốn quan sát giao tiếp giữa các đối tượng. Tổng quát thì biểu đồ tương tác chứa các thành phần sau đây:

- Đối tượng: biểu đồ tương tác sử dụng tên đối tượng, tên lớp hay cả hai.
- Liên kết: liên kết là hiện thực của kết hợp.
- Thông điệp: thông qua thông điệp lớp hay đối tượng có thể yêu cầu lớp hay đối tượng khác thực hiện chức năng cụ thể. thí dụ, form có thể yêu cầu chức năng Report tự in.
- Chú thích (notes) và ràng buộc như mọi biểu đồ khác.

4.2.1 - Biểu đồ trình tự

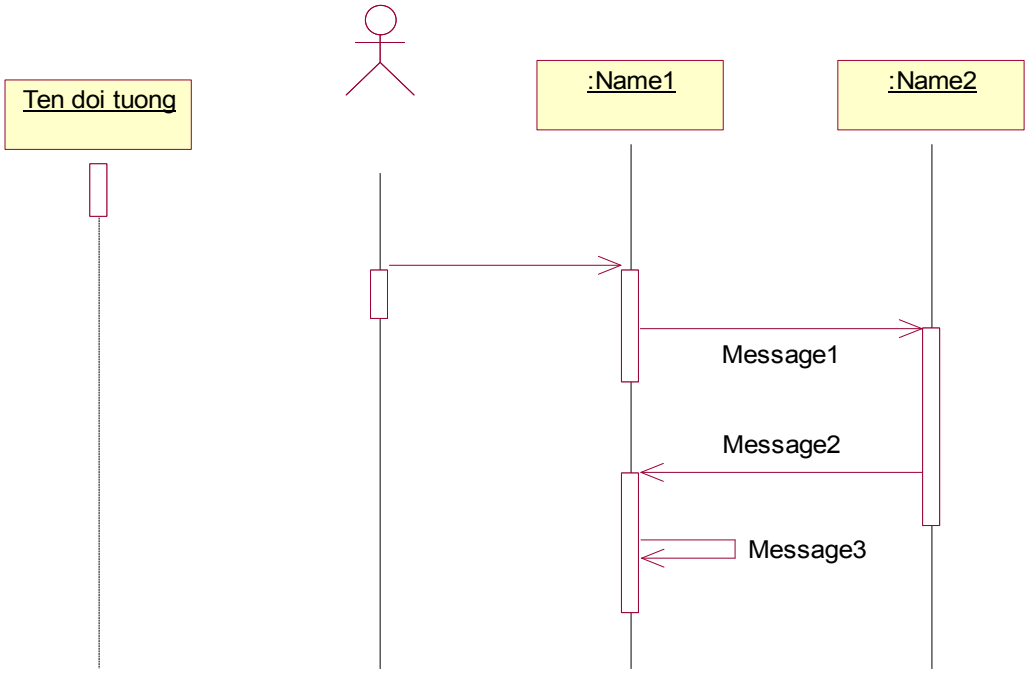
Biểu đồ trình tự (*sequence diagram*) là biểu đồ tương tác theo trật tự thời gian của các giao tiếp bằng thông điệp giữa các đối tượng; biểu đồ được đọc từ đỉnh xuống đáy. Mỗi UC có nhiều luồng dữ liệu. Mỗi biểu đồ trình tự biểu diễn một luồng dữ liệu.



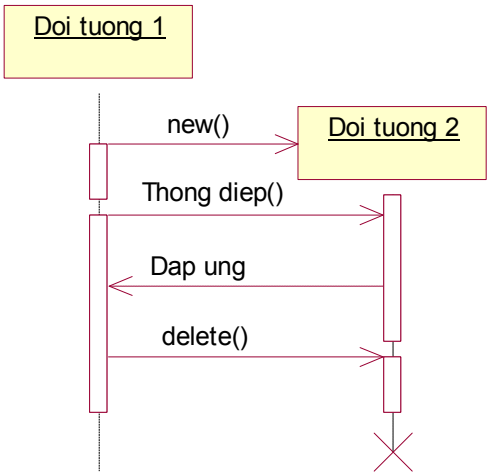
Hình 4.1 Biểu diễn đối tượng

Đọc biểu đồ trình tự bằng cách quan sát các đối tượng và thông điệp. Biểu đồ trình tự bao gồm các phần tử để biểu đồ đối tượng, thông điệp và thời gian. Đối tượng trong biểu đồ được biểu diễn bằng hình chữ nhật, trong hình chữ nhật là tên của nó (hình 4.1). Đối tượng được vẽ trong đỉnh biểu đồ. Thứ tự của các đối tượng trong biểu đồ được tổ chức sao cho biểu đồ được quan sát dễ dàng. Thời gian được biểu diễn bằng đường gạch gạch theo phương thẳng đứng (gọi là chu kỳ sống – *lifeline* – của đối tượng), bắt đầu từ đỉnh và kết thúc ở đáy biểu đồ. Hình chữ nhật hẹp dọc theo chu kỳ sống được gọi là *hoạt động*. Hoạt động biểu đồ thời gian thực thi một hành động tương ứng (hình 4.2a). Chú ý rằng trong biểu đồ trình tự do *Rational Rose* tạo ra, đôi khi hình chữ nhật này bị ẩn là chỉ để cho dễ quan sát biểu đồ.

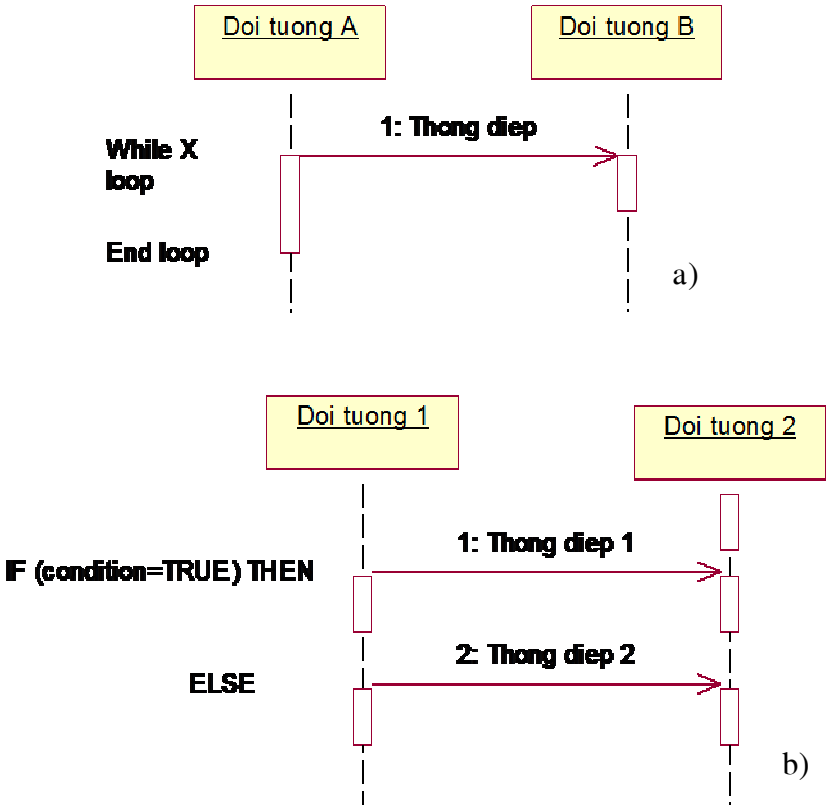
Thông điệp được vẽ bằng mũi tên đóng đi từ chu kỳ sống của đối tượng này đến chu kỳ sống của đối tượng khác. Trên mũi tên là thông điệp. mỗi thông điệp là biểu diễn một đối tượng gọi hàm của đối tượng khác. Khi định nghĩa thao tác cho lớp sau này thì mỗi thông điệp sẽ trở thành thao tác. Một đối tượng còn có thể gửi thông điệp đến chính nó. Các thông điệp này gọi là phản thân, nó chỉ ra rằng đối tượng gọi chính thao tác của mình. Hình 4.2b mũi tên tập phần tử biểu đồ cơ bản của biểu đồ trình tự. Tuy rằng biểu tượng tác nhân kích hoạt biểu đồ trình tự nhưng nó không phải là phần tử của biểu đồ loạiại này [KALA01]. Trong quá trình tương tác, đối tượng mới có thể được lập hay đối tượng cũ bị hủy bỏ. Việc tạo lập hay hủy bỏ đối tượng có thể được biểu đồ trong biểu đồ trình tự (hình 4.3). Trong thí dụ này, thông điệp *new* tạo lập đối tượng 2 và thông điệp *delete* hủy bỏ nó. Hình 4.3 còn cho thấy đáp ứng do thông điệp cung cấp với tên *response*, nó có thể làm tham số cho thông điệp khác. Trên biểu đồ trình tự, đáp ứng được biểu diễn bởi đường nét đứt và mũi tên mở.



Hình 4.2 Thành phần cơ bản của biểu đồ trình tự



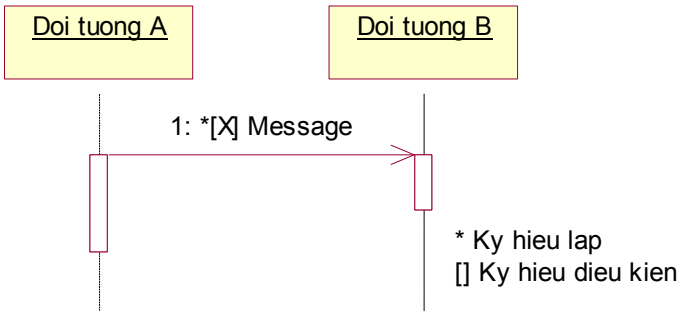
Hình 4.3 Tạo lập và bãi bỏ đối tượng



Hình 4.4 Scripts trong biểu đồ trình tự

Trong khi *Notes* được sử dụng để chú thích cho đối tượng thì *Scripts* được sử dụng để chú thích cho thông điệp. Chúng được đặt tại phía trái của biểu đồ trình tự, ngang với mức thông điệp cần chú thích. Có thể sử dụng *Scripts* để mô tả các logíc điều kiện trong biểu đồ, thí dụ lệnh *IF* hay trong vòng lặp *while*... Mã trình không được phát sinh cho *Scrips* nhưng nó cho người phát triển hệ thống biết được logic cần tuân thủ. Thí dụ bổ sung *Scripts* cho biểu đồ trình tự được mô tả trên hình 4.4.

Năm 1996, *Buschman* và cộng sự đề nghị bổ sung điều kiện vào thông điệp theo ký pháp *[condition] message()*. Biểu đồ trên hình 4.5 sử dụng ký pháp này. Vào thời gian đó, UML đã đưa ra biểu đồ hoạt động (*activity diagram*) phù hợp hơn cho thủ tục đa nhánh. Do vậy, việc sử dụng điều kiện trong biểu đồ trình tự không còn được khuyến cáo nữa [OEST00].



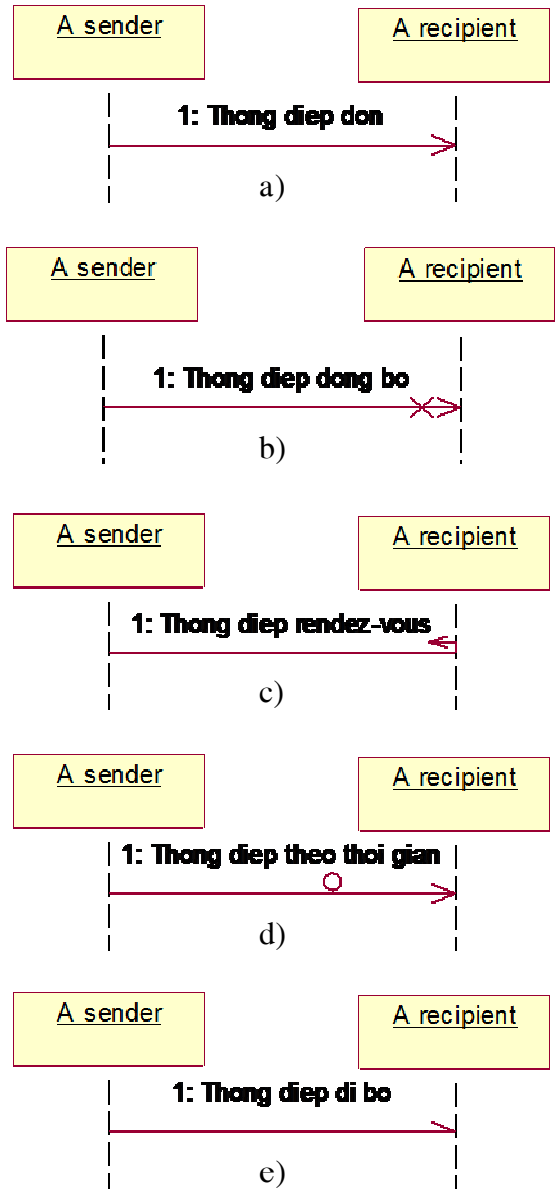
Hình 4.5 Thông điệp có điều kiện

Thông điệp là phương tiện giao tiếp giữa các đối tượng. Nó được biểu diễn trong biểu đồ hợp tác bởi mũi tên giữa nơi gửi và nơi nhận. Kiểu mũi tên chỉ ra loại thông điệp, nói cách khác là loại đồng bộ của thông điệp. UML có các loại đồng bộ thông điệp như sau đây:

Đơn: là giá trị mặc định của thông điệp. Phần tử biểu đồ này chỉ ra rằng thông điệp sẽ chạy trong luồng đơn của điều khiển. Cả biểu đồ trình tự và biểu đồ cộng tác đều sử dụng biểu tượng thông điệp đơn như trên hình 4.6a.

Đồng bộ: thuộc tính này được sử dụng khi đối tượng gửi thông điệp và chờ đến khi nó được xử lý xong. Cả biểu đồ trình tự và biểu đồ cộng tác đều sử dụng biểu tượng thông điệp đồng bộ như trên hình 4.6b.

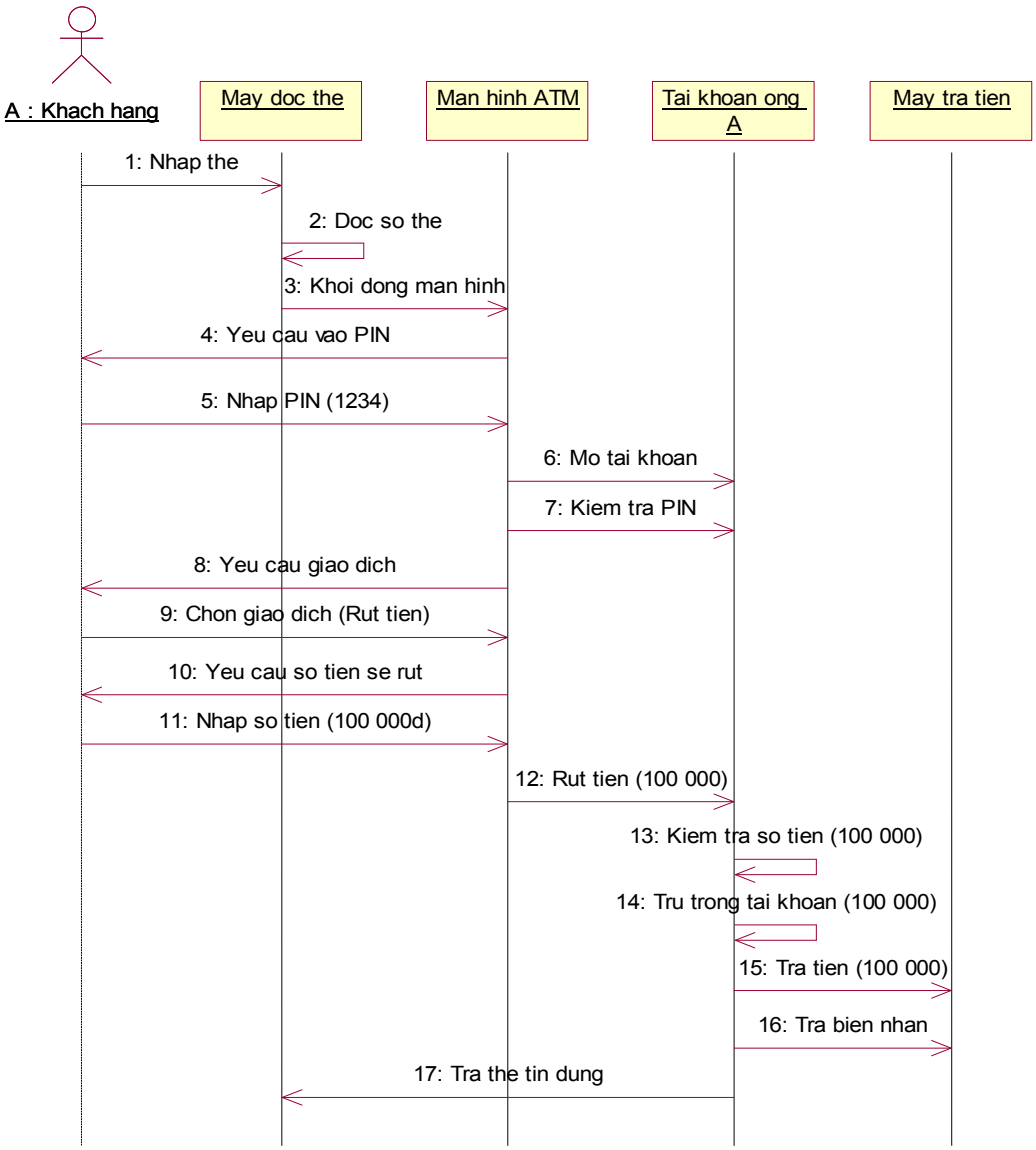
Cản trở (Balking, Rendez-vous): đối tượng gửi thông điệp đến nơi nhận. Nếu nơi nhận chưa xử lý ngay thì đối tượng gửi bãi bỏ thông điệp. Cả biểu đồ trình tự và biểu đồ cộng tác đều sử dụng biểu tượng thông điệp cản trở như trên hình 4.6c.



Hình 4.6 Các loại thông điệp

Hết hạn (time out): Đối tượng gửi thông điệp đến đối tượng nhận và chờ một khoảng thời gian. Nếu sau thời gian đó mà thông điệp chưa được xử lý thì đối tượng gửi bãi bỏ chúng. Cả biểu đồ trình tự và biểu đồ cộng tác đều sử dụng biểu tượng thông điệp hết hạn như trên hình 4.6d.

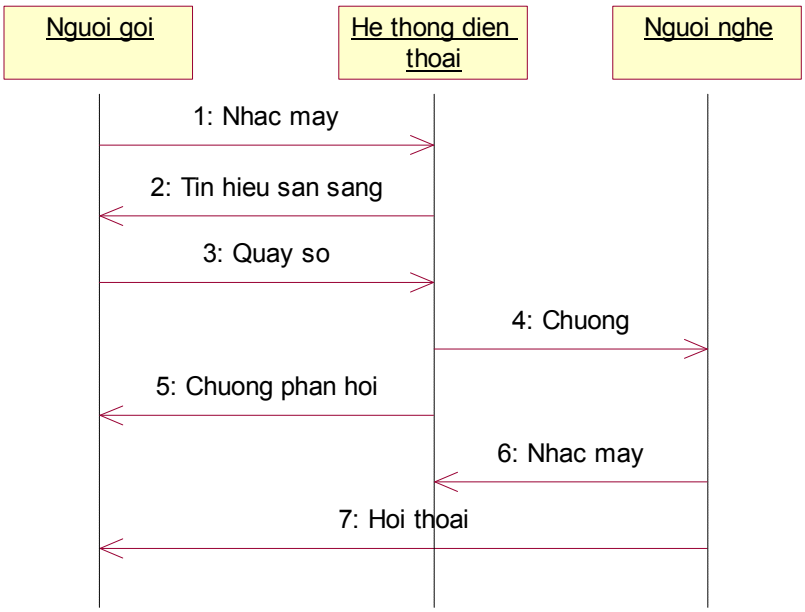
Dị bộ: Đối tượng gửi thông điệp đến đối tượng nhận. Đối tượng gửi tiếp tục làm việc khác không quan tâm đến thông điệp đó có được xử lý hay không. Cả biểu đồ trình tự và biểu đồ cộng tác đều sử dụng biểu tượng thông điệp dị bộ như trên hình 4.6e.



Hình 4.7 Biểu đồ trình tự ông A rút 100.000đ từ ATM

Hình 4.7 là thí dụ biểu đồ trình tự mô tả ông A rút 100.000đ từ máy rút tiền tự động ATM. Biểu đồ này có 5 đối tượng: Ông A (khách hàng), máy đọc thẻ tín dụng, màn hình ATM, tài khoản của ông A, máy chi trả. Đối tượng tác nhân (ông A) khởi động UC được vẽ tại góc bên trái của biểu đồ. Tiến trình rút tiền được bắt đầu khi ông A đặt thẻ tín dụng vào máy đọc thẻ. máy đọc thẻ đọc số hiệu thẻ và yêu cầu màn hình ATM tự khởi động. Màn hình ATM nhắc ông A nhập số căn cước cá nhân PIN, thí dụ nhập số 1234. ATM mở tài khoản của ông A. Số PIN của ông A được kiểm tra và màn hình ATM nhắc ông A thực hiện giao dịch. Ông A chọn *Rút tiền*. ATM nhắc ông A nhập số tiền muốn rút ra. Ông A nhập 100.000đ ATM kiểm tra xem số dư tài khoản của ông A có đủ số tiền muốn rút hay không. Rồi trừ số tiền trong tài khoản đi 100.000đ. ATM trả cho ông A 100.000đ và đẩy thẻ tín dụng ra khỏi máy đọc. Chú ý rằng trên biểu đồ này hình chữ nhật con dọc theo chu kỳ sống của đối tượng không được vẽ là để dễ quan sát biểu đồ. Các thông điệp số 2, 13 và 14 là các thông điệp phản thân.

Thí dụ về biểu đồ trình tự của hệ thống điện thoại được mô tả trên hình 4.8. biểu đồ này có 3 đối tượng: người gọi, hệ thống điện thoại và người nghe. Tiến trình gọi điện thoại bắt đầu khi người gọi *Nhấc máy gọi*. Hệ thống điện thoại gửi lại *Tín hiệu được phép gọi*. Người gọi *quay số*. Hệ thống điện thoại làm *Reo chuông* máy người bị gọi và *Gửi tín hiệu chuông trở lại người gọi*. Người bị gọi *Nhắc máy* để nghe. Người bị gọi *Hội thoại* với người gọi.

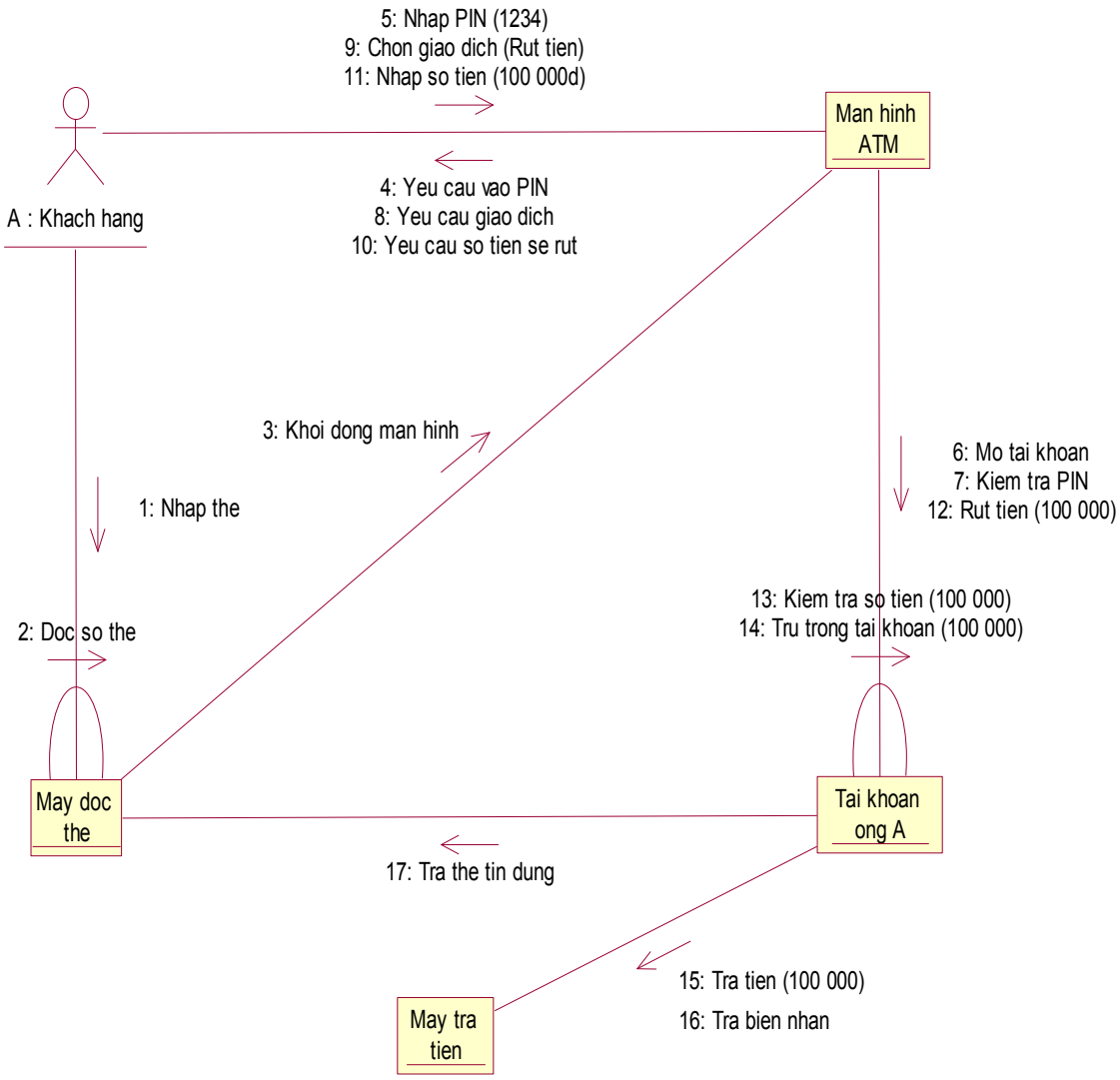


Hình 4.8 Biểu đồ trình tự gọi điện thoại

4.2.2 - Biểu đồ cộng tác

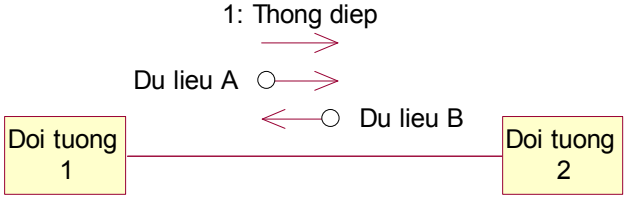
Tương tự như biểu đồ trình tự, biểu đồ cộng tác (*collaboration diagram*) chỉ ra luồng sự kiện xuyên qua kịch bản của UC. Trong khi biểu đồ trình tự có trật tự theo thời gian, thì biểu đồ cộng tác tập trung nhiều hơn vào quan hệ giữa các đối tượng, tập trung vào tổ chức cấu trúc của các đối tượng gửi hay nhận thông điệp.

Thí dụ trên hình 4.9 là biểu đồ cộng tác thể hiện luồng sự kiện ông A rút 100.000đ của hệ thống ATM. Hình này cho thấy các thông tin trên biểu đồ trình tự cũng có trên biểu đồ cộng tác, nhưng nó cho ta cách nhìn khác về luồng sự kiện. Biểu đồ này cho khả năng dễ quan sát quan hệ giữa các đối tượng nhưng khó quan sát hơn về trình tự thông tin. Do vậy, thông thường phải tạo cả hai biểu đồ cho cùng kịch bản.



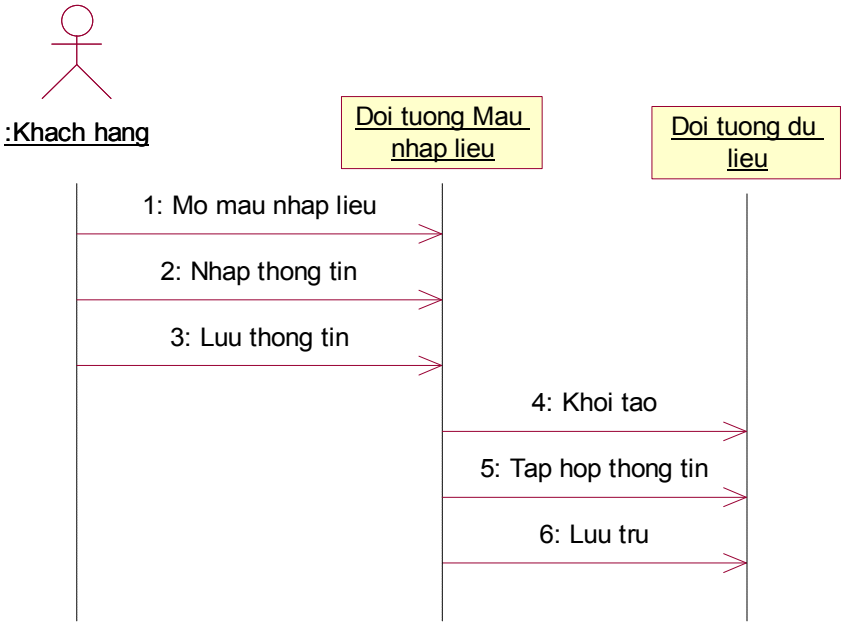
Hình 4.9 Biểu đồ cộng tác ông A rút 100.000đ

Một sự khác nhau giữa biểu đồ cộng tác và biểu đồ trình tự đã đề cập trên là biểu đồ trình tự tập trung vào luồng điều khiển còn biểu đồ cộng tác thì không. Sự khác nhau giữa chúng là biểu đồ cộng tác hiển thị luồng dữ liệu (*Data flows*) còn biểu đồ trình tự thì không. Luồng dữ liệu được sử dụng để chỉ ra thông tin trở về khi đối tượng gửi thông điệp đến đối tượng khác (hình 4.10). Tổng quát thì không nên gán luồng dữ liệu cho mọi thông điệp trong biểu đồ cộng tác ví nó sẽ làm rối biểu đồ. Khi thông điệp cho lại cấu trúc dữ liệu thì nên gắn luồng dữ liệu. Khi ánh xạ thông điệp thành thao tác lớp, thông tin trong luồng dữ liệu sẽ được gắn vào thao tác. Tổng quát thì không nên quá tập trung vào luồng dữ liệu. Bổ sung chúng vào biểu đồ khi cho nó là có ý nghĩa thực sự cho việc phát triển hệ thống.



Hình 4.10 Luồng dữ liệu trong biểu đồ cộng tác

Biểu đồ trình tự và biểu đồ cộng tác chỉ phù hợp cho việc mô tả từng biến thể của thủ tục. Nó không phù hợp cho việc xác định đầy đủ các hành vi trên một biểu đồ. Các biểu đồ hoạt động (*activity diagram*) và biểu đồ trạng thái (*state diagram*) sẽ được đề cập trong các chương sau.

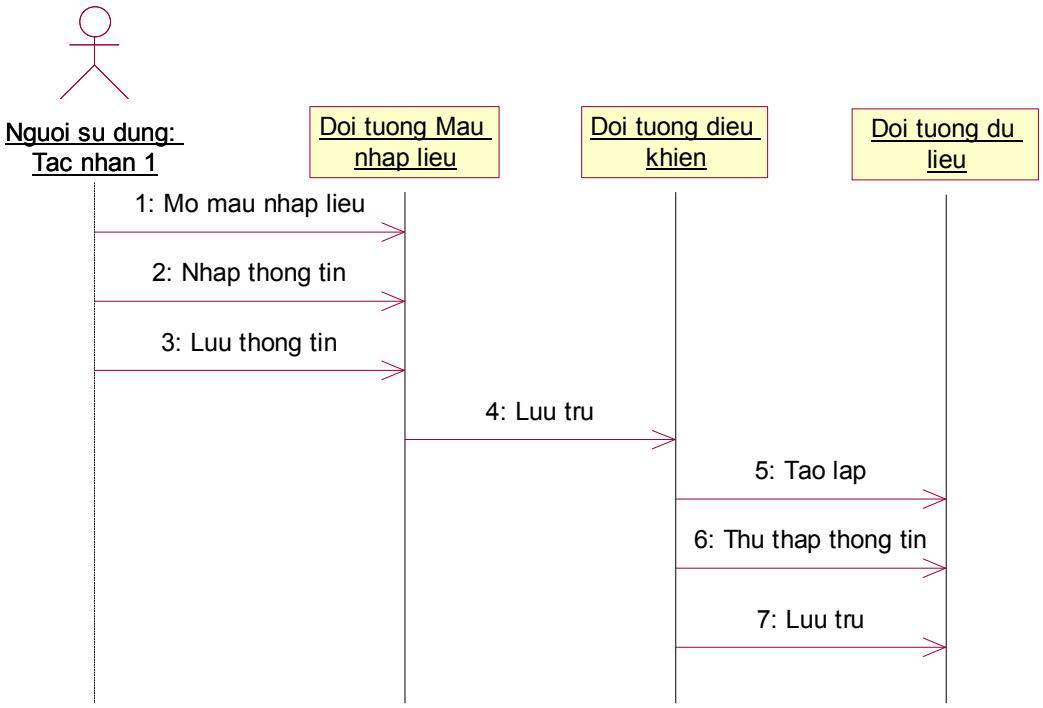


Hình 4.11 Biểu đồ trình tự bước thứ nhất

4.3 KỸ THUẬT XÂY DỰNG BIỂU ĐỒ TƯƠNG TÁC

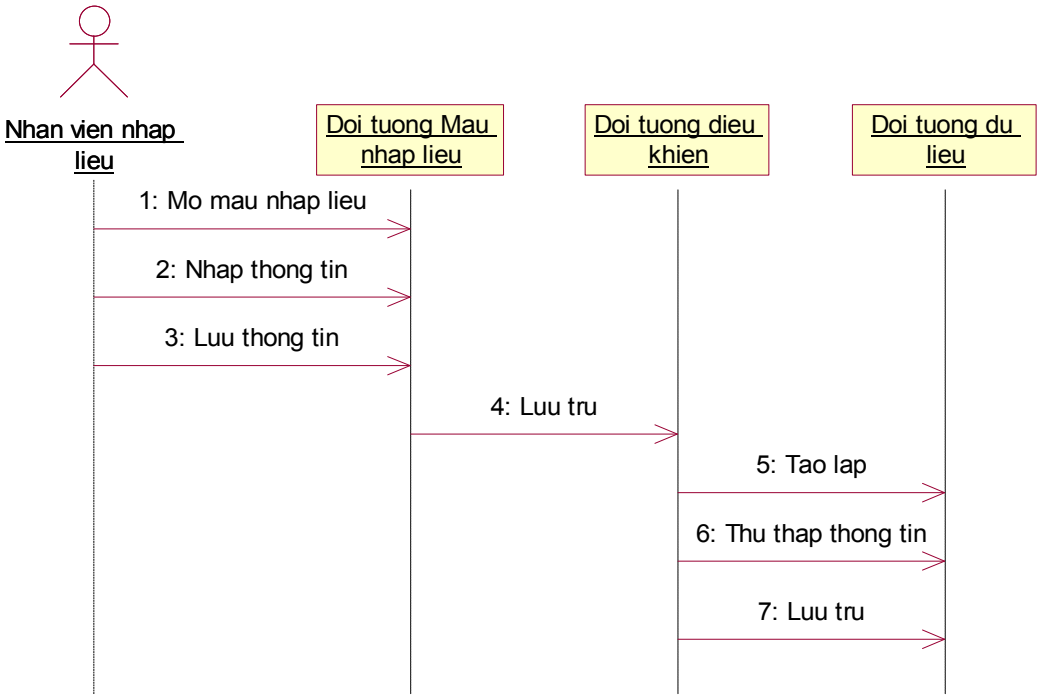
Có nhiều phương pháp xây dựng biểu đồ tương tác, nhưng tiệm cận hai bước là hay được sử dụng hơn cả để lập ra biểu đồ tương tác đầy đủ cho hệ thống liên quan đến CSDL. Bước thứ nhất chỉ tập trung vào các thông tin liên quan đến khách hàng. Không ánh xạ ngay thông điệp thành thao tác và không ánh xạ ngay đối tượng thành lớp. Biểu đồ tạo ra trong bước này dành cho nhà phân tích, khách hàng và những ai chỉ quan tâm đến luồng tác nghiệp, để biết luồng logic chạy trong hệ thống như thế nào. Biểu đồ trình tự điển hình sau bước thứ nhất được thể hiện trên hình 4.11.

Trong bước hai, khi khách hàng đã đồng ý với biểu đồ tạo ra trong bước một thì nó sẽ được bổ sung chi tiết hơn. Vào thời điểm này biểu đồ sẽ ít tác dụng cho khách hàng nhưng lại rất có ích với người phát triển hệ thống, kiểm tra chất lượng và các nhân viên khác của đội ngũ dự án. Một vài đối tượng mới có thể được thêm vào biểu đồ. Thông thường, mỗi biểu đồ tương tác đều có đối tượng điều khiển để điều khiển trình tự xuyên qua kịch bản. mọi biểu đồ tương tác của một uc đều chia sẻ cùng đối tượng điều khiển. sau khi bổ sung đối tượng điều khiển biểu đồ trình tự sẽ như trên hình 4.12.



Hình 4.12 biểu đồ trình tự với đối tượng điều khiển

Đối tượng điều khiển không xử lý tác nghiệp nào, chúng chỉ gửi thông điệp đến đối tượng khác. Đối tượng điều khiển có trách nhiệm điều phối công việc của đối tượng khác và được coi là đối tượng quản trị. Lợi thế của sử dụng đối tượng điều khiển là khả năng tách logic tác nghiệp ra khỏi logic trình tự. Nếu trình tự thay đổi thì chỉ đối tượng điều khiển thay đổi. Ta có thể bổ sung thêm đối tượng quản lý an toàn, quản lý hay kết nối CSDL. Rất nhiều đối tượng chỉ xây dựng một lần và có thể sử dụng lại trong các ứng dụng khác nhau. Sau đây là thí dụ về ứng dụng liên quan đến CSDL.

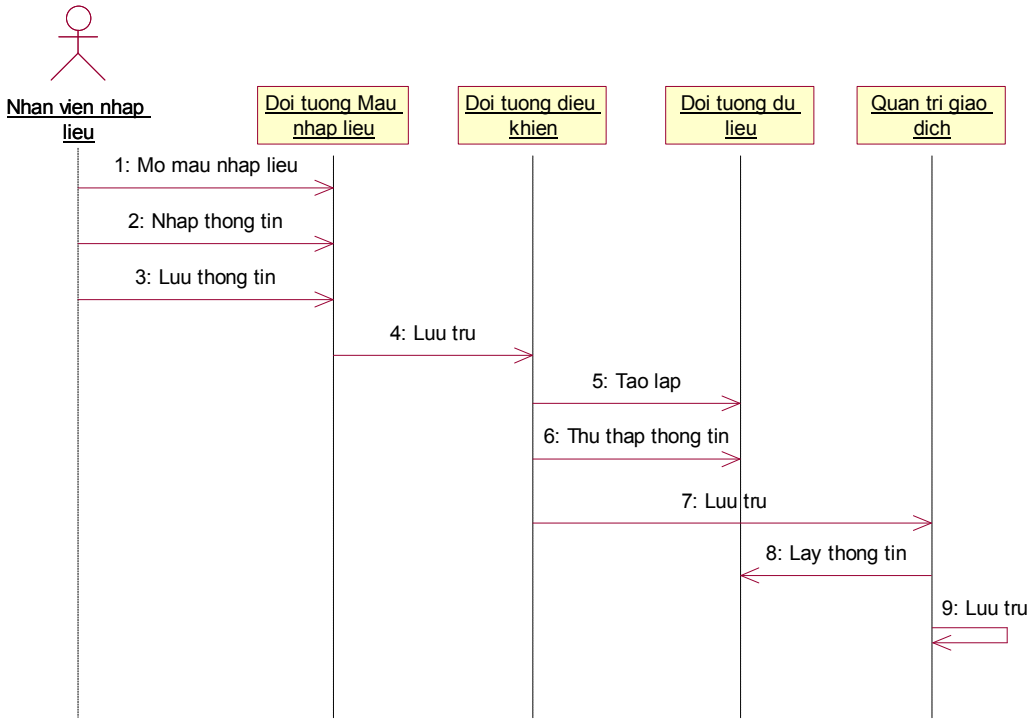


Hình 4.13 Logic ứng dụng tích hợp với logic CSDL

Có hai chức năng thường được sử dụng trong CSDL, đó là lưu trữ và khai thác thông tin. Giả sử rằng phải lưu trữ nhân viên mới có tên *Ông A* vào CSDL. Nếu đối tượng *Ông A* biết về CSDL thì nó tự lưu trữ vào CSDL. Cách khác là tách *Ông A* ra khỏi logic CSDL để đối tượng khác có trách nhiệm lưu trữ *Ông A* vào CSDL. Hình 4.13 là biểu đồ trình tự thể hiện trường hợp *Ông A* biết về CSDL. Logic ứng dụng và logic CSDL trong thí dụ này không tách rời nhau. Đối tượng *Ông A* vừa phải quan tâm đến logic ứng dụng và logic CSDL (lưu trữ *Ông A* vào CSDL và truy cập *Ông A* từ CSDL). Lợi thế của tiệm cận này là dễ mô hình hóa và cài đặt. tuy nhiên khi CSDL thay đổi nó sẽ kéo theo sự thay đổi tại rất nhiều nơi trong ứng dụng vì rất nhiều đối tượng chứa cả logic CSDL. Trong tiệm cận tách logic ứng dụng ra khỏi logic CSDL ta phải tạo ra đối tượng mới có trách nhiệm với CSDL. Các đối tượng này được gọi là đối tượng *Quản trị phiên giao dịch* (*Transaction Manager*). Tại đây, trong khi đối tượng *Ông A* vẫn chứa logic tác nghiệp thì đối tượng *Quản trị phiên giao dịch* sẽ biết cách truy vấn *Ông A* từ CSDL và lưu trữ *Ông A* vào CSDL. Biểu đồ trình tự của cách tiếp cận này được trình bày trên hình 4.14.

Lợi thế của giải pháp này là dễ dàng sử dụng lại đối tượng *Ông A* cho ứng dụng với các CSDL khác nhau hay không có CSDL. Nó cho khả năng giảm thiểu các thay đổi, khi CSDL thay đổi sẽ không làm thay đổi logic ứng dụng, và ngược lại. Bất lợi của tiệm cận này là ta phải thêm thời gian để mô hình hóa và cài đặt hệ thống. công việc tiếp theo phía nhiệm vụ CSDL là bổ sung thêm các đối tượng quản lý lỗi, an toàn dữ liệu, giao tiếp giữa các tiến trình… Các chi tiết này không quan trọng với người sử dụng nhưng lại rất quan trọng với người phát triển hệ thống.

Sau khi bổ sung đầy đủ các đối tượng, bước tiếp theo là ánh xạ từng đối tượng vào lớp có sẵn hay lớp mới, ánh xạ từng thông điệp trong biểu đồ thành thao tác. Cuối cùng là thực hiện đặc tả đối tượng và đặc tả thông điệp như gán đồng bộ, gán tần xuất thông điệp và cách thức lưu trữ đối tượng…



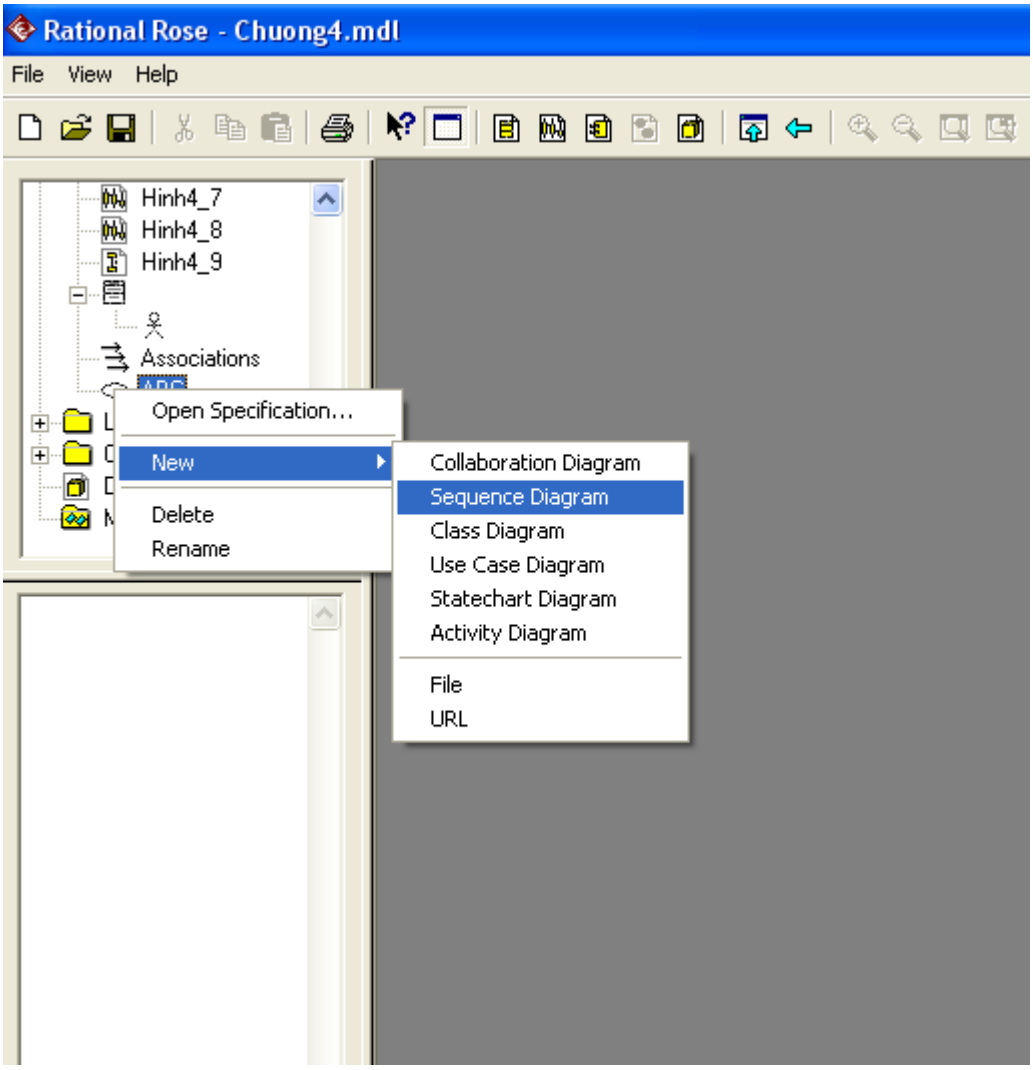
Hình 4.14 Logic ứng dụng tách khỏi logic CSDL

4.4 THỰC HÀNH

4.4.1 - Sử dụng Rational Rose

4.4.1.1 - Tạo lập biểu đồ trình tự

Các biểu đồ trình tự được tạo lập trong khung nhìn UC (*Use Case View*) hay khung nhìn logic (*Logical view*) của browser. Biểu đồ trình tự phải được vẽ trực tiếp trong UC như trên hình 4.15 hay trực tiếp trong gói.



Hình 4.15 Tạo lập biểu đồ trình tự mới

Tạo biểu đồ trình tự mới như sau:

1. Nhấn phím phải chuột trên gói hay UC trong browser
2. Chọn *New> Sequence diagram*
3. Đặt tên cho biểu đồ trình tự mới
4. Nhấn đúp trên biểu đồ trình tự mới trong browser để mở.

Mơ biểu đồ trình tự có sẵn:

1. Tìm biểu đồ trình tự trong khung nhìn UC của browser
2. Nhấn đúp chuột vào biểu đồ trình tự để mở nó.

Sử dụng các phím trên thanh công cụ để bổ sung các đối tượng, thông điệp vào biểu đồ hoặc di tác nhân, lớp từ browser vào biểu đồ trình tự. Chú ý rằng khi bỏ đối tượng khỏi biểu đồ. Lớp trong mô hình không bị xóa.

4.4.1.2 - Bãi bỏ biểu đồ trình tự

Trong quá trình làm việc có thể một số biểu đồ trình tự là quá cũ, không còn chính xác hay dư thừa. Để cho mô hình nhất quán, ta có thể hủy bỏ các biểu đồ trình tự đó đi. Trong Rose, việc hủy bỏ biểu đồ trình tự được thực hiện như sau:

1. Nhấn phím phải chuột trên biểu đồ trình tự trong browser
2. Chọn delete từ thực đơn

4.4.1.3 - Gắn tệp vào biểu đồ trình tự

Trong Rose, ta có thể gán tệp vào biểu đồ trình tự. Nội dung tệp có thể là mô tả kịch bản mà biểu đồ trình tự mô hình hóa, hoặc một vài dòng mã lệnh cài đặt logic trong biểu đồ hay mô tả các nhu cầu liên quan. Gắn tệp vào biểu đồ trình tự theo các bước sau đây:

1. Nhấn phím phải chuột trên biểu đồ trình tự trong browser
2. Chọn *New>File*
3. Sử dụng hộp thoại *Open* để chọn tên tệp sẽ gắn vào biểu đồ
4. Chọn *Open*

Bãi bỏ tệp gắn vào biểu đồ trình tự như sau:

1. Nhấn phím phải trên tệp trong browser
2. Chọn *Delete* từ thực đơn

4.4.1.4 - Tạo lập và hủy bỏ biểu đồ cộng tác

Tương tự biểu đồ trình tự, biểu đồ cộng tác được tạo lập trong browser, trực tiếp trong UC hay trong gói. Cách tạo lập khác là chuyển đổi tự động biểu đồ trình tự có sẵn sang biểu đồ cộng tác nhờ **nhấn phím F5**. Trình tự hủy bỏ biểu đồ cộng tác được thực hiện tương tự như hủy bỏ biểu đồ trình tự vừa mô tả trên đây.

4.4.1.5 - Bổ sung và hủy bỏ tác nhân trong biểu đồ tương tác

Mỗi biểu đồ trình tự và biểu đồ cộng tác đều có một tác nhân. Đối tượng tác nhân lá kích thích ngoài để hệ thống thực hiện một vài chức năng. Đối tượng tác nhân của biểu đồ tương tác bao gồm các tác nhân thao tác với UC trong biểu đồ UC

Tạo lập đối tượng tác nhân trên biểu đồ tương tác như sau:

1. Mở biểu đồ tương tác
2. Chọn tác nhân trong browser
3. Di tác nhân từ Browser đến biểu đồ đang mở

Hủy bỏ đối tượng tác nhân trên biểu đồ thao tác theo trình tự sau:

1. Chọn tác nhân trên biểu đồ tương tác
2. Chọn *Edit>Delete from Model*, hay nhấn phím *Ctrl+D*

Chú ý rằng việc hủy bỏ đối tượng khỏi biểu đồ sẽ không hủy bỏ lớp trong mô hình.

4.4.1.6 - Bổ sung đối tượng vào biểu đồ tương tác

Công việc tiếp sau khi biểu đồ tương tác được tạo lập là bổ sung đối tượng khác vào biểu đồ. Ta phải tìm các đối tượng tham gia vào biểu đồ trình tự hay biểu đồ cộng tác cụ thể bằng cách tìm các danh từ trong tài liệu về luồng sự kiện và kịch bản. bổ sung đối tượng vào biểu đồ trình tự theo các bước sau đây:

1. Chọn phím *Object* trên thanh công cụ
2. Nhấn phím trái chuột trong vùng biểu đồ nơi sẽ đăt đối tượng. Trong biểu đồ trình tự, các đối tượng đặt tại hàng đỉnh
3. Nhập tên cho đối tượng mới
4. Sau khi vẽ các đối tượng trong biểu đồ, ta có thể sắp xếp lại chúng bằng nhấn-di-nhả đối tượng, bổ sung đối tượng vào giữa các đối tượng đã vẽ trước đó.

Trong biểu đồ cộng tác, bổ sung đối tượng theo trình tự sau:

1. Chọn phím *Object* trên thanh công cụ
2. Nhấn phím chuột trong vùng biểu đồ nơi sẽ đặt đối tượng. Trong biểu đồ cộng tác, các đối tượng đặt tại bất kỳ đâu.
3. Nhập tên đối tượng mới

4.4.1.7 - Bãi bỏ đối tượng trong biểu đồ tương tác

Biểu đồ tương tác được huỷ bỏ theo trình tự sau:

1. Chọn đối tượng
2. Chọn *Edit* > *Delete from Model*, hay *Ctrl+D*

4.4.1.8 - Đặc tả đối tượng

Cửa sổ đặc tả đối tượng trong Rose cho nhập các thông tin sau: tên đối tượng, lớp của đối tượng, lưu trữ, đa bản, tài liệu. Mở cửa sổ đặc tả đối tượng như sau:

1. Nhấn phím phải chuột vào biểu đồ trình tự và cộng tác
2. Chọn thực đơn *Open Specification*

4.4.1.9 - Đặt tên đối tượng

Mỗi đối tượng trên biểu đồ phải có tên duy nhất. Trong khi tên lớp là rất khái quát như *Nhân viên*, *Công ty* thì tên đối tượng lại rất cụ thể như *Ông A*, *Microsoft*…

Đặt tên đối tượng như sau

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ
2. Chọn *Open Specification*
3. Nhập tên vào cửa sổ *Name*. Mỗi đối tượng trên biểu đồ có tên duy nhất. Thực hiện tương tự cho việc nhập tài liệu của đối tượng.

4.4.1.10 - Ánh xạ đối tượng vào lớp

Mỗi đối tượng trong biểu đồ trình tự và biểu đồ hợp tác có thể ánh xạ vào một lớp. Thí dụ đối tượng tài khoản *Ông A* ánh xạ vào lớp *Tài khoản*. Việc này được thực hiện trong cửa sổ đặc tả đối tượng (*Object Specification*). Khi chọn lớp cho đối tượng, ta có thể chọn lớp có sẵn hay lập lớp mới.

Ánh xạ đối tượng vào lớp có sẵn như sau:

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ tương tác
2. Chọn *Open Specification*
3. Nhập tên lớp hay chọn tên lớp trong hộp *Class*
4. Khi đã ánh xạ đối tượng vào lớp, tên lớp sẽ xuất hiện cùng tên đối tượng trên biểu đồ.

Bãi bỏ ánh xạ lớp cho đối tượng theo trình tự sau đây:

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ tương tác
2. Chọn thực đơn *Open Specification*
3. Chọn (*Unspecified*) trong hộp danh sách *Class*

Tạo lập lớp mới cho đối tượng:

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ tương tác
2. Chọn thực đơn *Open Specification*
3. Chọn <*New*> trong hộp danh sách *Class*. Nhập thông tin cho lớp mới trong cửa sổ đặc tả.

Chỉ hiển thị tên lớp trên biểu đồ:

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ tương tác
2. chọn thực đơn *Open Specification*
3. nhập tên đối tượng vào cửa sổ *Name*
4. chọn (*Unspecified*) trong hộp danh sách *Class*

Chỉ hiển thị tên lớp trên biểu đồ:

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ tương tác
2. Chọn thực đơn *Open Specification*
3. Xóa tên đối tượng trong cửa sổ *Name*

4.4.1.11 - Thiết lập lưu trữ đối tượng

Rose cho khả năng thiết lập thuộc tính lưu trữ cho mỗi đối tượng trong biểu đồ. Các thuộc tính đó là:

Lưu trữ (Persistent): đối tượng lưu trữ là đối tượng sẽ được lưu trữ trong CSDL, hay lưu trữ dưới khuôn dạng khác.

Tĩnh (Static): là đối tượng tồn tại trong bộ nhớ cho đến khi chương trình kết thúc

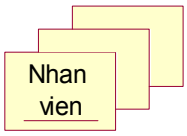
Tạm thời (Transient): là đối tượng chỉ tồn tại trong bộ nhớ trong khoảng thời gian ngắn

Thiết lập thuộc tính lưu trữ đối tượng như sau:

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ tương tác
2. Chọn thực đơn *Open Specification*
3. Chọn phím radio tương ứng trong vùng *Persistance*

4.4.1.12 - Sử dụng nhiều bản đối tượng

Rational Rose có khả năng sử dụng một biểu tượng để biểu diễn nhiều hiện thực của một lớp. Thí dụ, để biểu diễn danh sách nhân viên trong biểu đồ tương tác ta có thể sử dụng biểu tượng đa hiện thực (*multiple instances*). Ký pháp của UML như sau:



Gán thuộc tính đa hiện thực cho đối tượng theo các bước sau:

1. Nhấn phím phải chuột trên đối tượng trong biểu đồ tương tác
2. Chọn thực đơn *Open Specification*
3. Đánh dấu *on* hay *off* trong hộp *Multiple Instances*. Rose sẽ sử dụng bên trong phù hợp (đơn hay đa hiện thực) trong biểu đồ cộng tác, biểu tượng đơn hiện thực trong biểu đồ trình tự

4.4.1.13 - Bổ sung thông điệp vào biểu đồ trình tự

Trong biểu đồ trình tự, thông điệp được vẽ giữa các đường *lifelines* của đối tượng hay đường *lifelines* của chính đối tượng. thông điệp sẽ hiển thị từ trên xuống đáy biểu đồ

Trình tự bổ sung thông điệp như sau:

1. Chọn phím *Object Message* trên thanh công cụ.
2. Di chuột từ *lifeline* của đối tượng hay tác nhân gửi thông điệp đến đối tượng hay tác nhân nhận thông điệp
3. Nhập tên thông điệp

4.4.1.14 - Thứ tự thông điệp trong biểu đồ trình tự

Thực hiện sắp xếp lại bằng nhấn, di, thả. Thứ tự thông điệp được tự động sắp xếp

4.4.1.15 - Đánh số thông điệp trong biểu đồ trình tự

Mặc định, số thứ tự thông điệp được gán trên biểu đồ tương tác và là ẩn. Để hiển thị / ẩn thứ tự thông điệp ta làm như sau:

1. Chọn thực đơn Tool > Options
2. Chọn DiagramTab
3. Đặt thuộc tính on / off cho Sequence Numbering

4.4.1.16 - Quan sát Forcus of Control

Để tắt mở các hình chữ nhật con trên chu kỳ sống của biểu đồ trình tự hãy thực hiện như sau :

1. Chọn thực đơn *Tool > Options*
2. Chọn *Diagram Tab*
3. Đánh dấu hay bỏ *Forcus of Control*

4.4.1.17 - Bổ sung thông điệp vào biểu đồ cộng tác

Trước khi bổ sung thông điệp vào biểu đồ cộng tác ta phải vẽ đường dẫn giao tiếp giữa hai đối tượng. đường dẫn này được gọi là liên kết (*link*). Để bổ sung thông điệp vào sơ đồ cộng tác ta làm như sau:

1. Chọn phím *Object Link* từ thanh công cụ
2. Di từ đối tượng này đến đối tượng khác để vẽ liên kết
3. Chọn phím *Link Messages* hay *Reverse Link Messages* trên thanh công cụ
4. Nhấn phím phải chuột trên liên kết giữa hai đối tượng. Rose sẽ vẽ mũi tên thông điệp
5. Nhập tên cho thông điệp mới

Việc bổ sung thông điệp phản thân được thực hiện tương tự

4.4.1.18 - Bổ sung Data Flows vào biểu đồ cộng tác

Vẽ luồng dữ liệu (*data flows*) trong biểu đồ cộng tác như sau:

1. Chọn phím *Data Flow* hay *Reverse Data Flow* trên thanh công cụ.
2. Nhấn trên thông điệp sẽ trả lại dữ liệu. Rose tự động gán mũi tên luồng dữ liệu vào biểu đồ.
3. Nhập data sẽ trả lại vào luồng dữ liệu mới tạo ra.

4.4.1.19 - Đặc tả thông điệp

Trong Rose có thể đặt nhiều thuộc tính cho mỗi thông điệp. Thí dụ, với UC và tác nhân có thể được gán tên, tài liệu cho thông điệp. Còn có thể đặt thuộc tính đồng bộ hay tần xuất. Để mở đặc tả thông điệp ta nhấn đúp phím trái chuột trên thông điệp của biểu đồ.

4.4.1.20 - Đặt tên thông điệp

Đặt tên và tài liệu thông điệp trong cửa sổ đặc tả. Tên thông điệp thường là mục đích của thông điệp. Để đặt tên ta thực hiện như sau:

1. Nhấn đúp trên thông điệp của biểu đồ tương tác
2. Nếu đã ánh xạ thông điệp nhận vào lớp, thì thao tác của lớp này xuất hiện trong hộp *Name*. Hãy chọn một trong chúng làm tên thông điệp hay nhập tên mới cho thông điệp

4.4.1.21 - Ánh xạ thông điệp thành thao tác

Trước khi phát sinh mã trình, mọi thông điệp trên biểu đồ tương tác phải được ánh xạ thành thao tác lớp. Thực hiện ánh xạ thông điệp theo các bước như sau:

1. Đảm bảo đối tượng nhận phải được ánh xạ thành lớp
2. Nhấn phím phải chuột trên thông điệp trong biểu đồ
3. Danh sách thao tác xuất hiện
4. Chọn thao tác từ danh sách

Tạo lập thao tác mới cho thông điệp theo các bước như sau:

1. Đảm bảo đối tượng nhận phải được ánh xạ thành lớp
2. Nhấn phím phải chuột trên thông điệp trong biểu đồ
3. Chọn <*new operation*>
4. Nhập tên và chi tiết cho tên thao tác mới
5. Nhấn phím *OK* để đóng cửa sổ đặc tả và bổ sung thao tác mới
6. Nhấn phím phải chuột trên thông điệp
7. Chọn thông điệp mới từ danh sách

4.4.1.22 - Chọn thuộc tính đồng bộ và tần xuất thông điệp

Trên *Detail Tab* của cửa sổ đặc tả thông điệp ta có thể gán thuộc tính đồng bộ các thông điệp đang gửi. Cũng trên bảng này ta có thể đặt tần xuất thông điệp. Tần xuất thông điệp bao gồm thuộc tính gợi ý thông điệp gửi đều đặn (*Periodic*) và không gửi đều đặn (*Aperiodic*).

Đặt đồng bộ và tần xuất thông điệp trong Rose như sau:

1. Nhấn đúp trên thông điệp của biểu đồ tương tác
2. Chọn *Detail Tab* trong cửa sổ đặc tả
3. Chọn thuộc tính đồng bộ mong muốn bằng đánh dấu vào phím radio đồng bộ (*synchronzation*) hay tần xuất (*frequency*)

4.4.1.23 - Chuyển đổi giữa biểu đồ trình tự và biểu đồ cộng tác

Thông thường, ta lập biểu đồ trình tự hay biểu đồ cộng tác cho một kịch bản cụ thể. nếu không có các như *Rose* thì công việc lập biểu đồ còn lại là khá khó khăn. *Rose* cho khả năng tự động chuyển đổi giữa chúng.

Tạo lập biểu đồ cộng tác từ biểu đồ trình tự như sau:

1. Mở biểu đồ trình tự
2. Chọn *Browser* > *Create Collabaration diagram*, hay nhấn phím F5
3. Biểu đồ cộng tác được tự động lập từ biểu đồ trình tự.

Tạo lập biểu đồ trình tự từ cộng tác như sau:

1. Mở biểu đồ cộng tác
2. Chọn *Browser* > *Create Collabaration diagram*, hay nhấn phím F5

3. Biểu đồ trình tự được tự động lập từ biểu đồ cộng tác

4.4.2 - Thí dụ: hệ thống bán hàng (tiếp theo)

Sau khi mọi người tham gia dự án đã đồng ý với biểu đồ uc đã tạo ra từ các chương trước đây công việc tiếp theo là pha xây dựng biểu đồ tương tác. Hãy khảo sát UC *Enter New Order* làm thí dụ. Người thiết kế đã làm tài liệu cho UC như sau

- Người bán hàng bổ sung đơn hàng mới.
- Người bán hàng thử bổ sung đơn bán hàng nhưng một mặt hàng nào đó đã hết.
- Người bán hàng thử bổ sung đơn bán hàng nhưng không ghi được vào CSDL vì lỗi.

Thí dụ này chỉ vẽ một trong các biểu đồ tương tác cho UC của mô hình. Cần bổ sung các biểu đồ khác để mô hình hóa các trường hợp xảy ra như người sử dụng chọn các nhánh thực hiện khác hay chọn sai thuộc tính…

4.4.2.1 - Xây dựng biểu đồ trình tự

Setup

1. Chọn *Tool \ Options*
2. Chọn bảng *Diagram*
3. Đánh dấu *Sequence Numbering*, *Collaboration Nubering* và *Focus of Control*
4. Nhấn phím *OK*

Tạo biểu đồ trình tự

1. Nhấn phím phải chuột trên *Logical View* trong browser
2. Chọn *New\Sequence Diagram*
3. Đặt tên cho biểu đồ là *Bổ sung đơn hàng*
4. Nhấn đúp trên biểu đồ để mở chúng

Bổ sung tác nhân và đối tượng vào biểu đồ

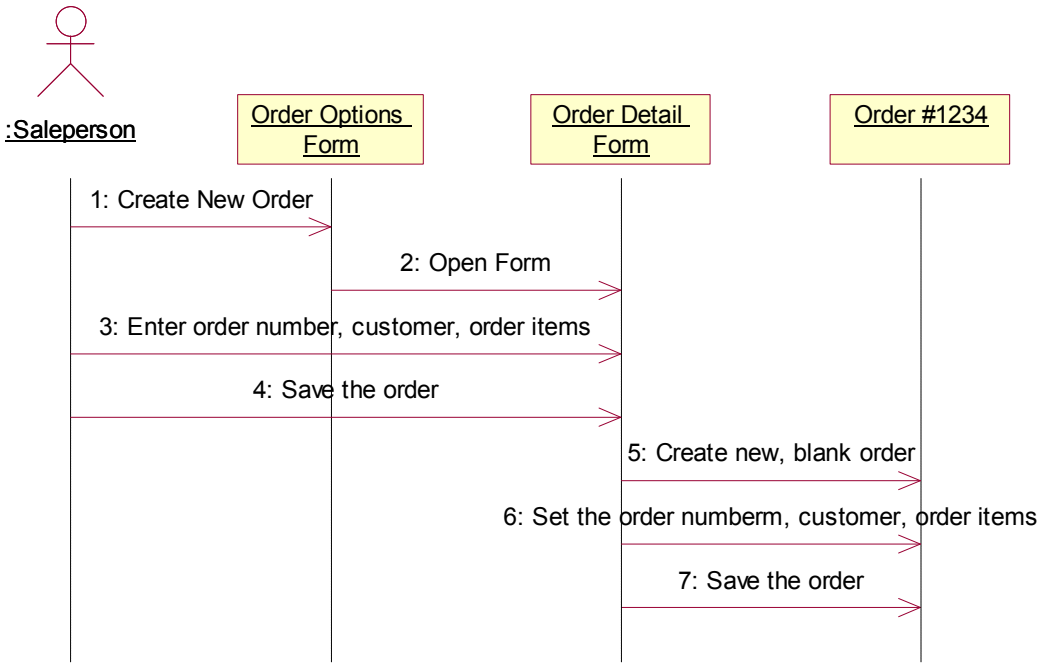
1. Di và thả tác nhân *Salesperson* từ browser tới biểu đồ
2. Sử dụng phím công cụ để bổ sung đối tượng *Order Option Form* (mẫu lựa chọn đặt hàng)
3. Lặp lại bước hai để bổ sung các đối tượng *Order Detail Form* (mẫu đặt hàng chi tiết) và *Order #1234*.

Bổ sung thông điệp vào biểu đồ

1. Sử dụng phím công cụ *Object Message*
2. Vẽ từ tác nhân *Salesperson* để *lifetime* của đối tượng *Order Options Form* (mẫu lựa chọn đặt hàng).
3. Nhập tên thông điệp *Create New Order* (tạo đơn hàng mới)

4. Lặp lại bước hai và bước ba để bổ sung các thông điệp *Open form* (giữa *Order Option Form* và *Order Detail Form*), *Enter order number*, *customer*, *order item*s (giữa *Salesperson* và *Order Detail Form*), *Save the order* (giữa *Salesperson* và *Order Detail Form*), *Create new*, *blank order* (giữa *Order Detail Form* và *Order #1234*), *Set the order number*, *customer*, *order items* (giữa *Order Detail Form* và *Order #1234*) và *Save the order* (giữa *Order Detail Form* và *Order #1234*).

Biểu đồ kết quả trong hình 4.16 là biểu đồ trình tự bước một để bổ sung đơn hàng mới. tiếp theo ta phải quan tâm đến đối tượng điều khiển và kết nối CSDL. Biểu đồ cho thấy đối tượng *Order Detail Form* có quá nhiều trách nhiệm. Do vậy, tốt hơn là chia sẻ bớt nhiệm vụ cho đối tượng điều khiển. Việc lưu trữ dữ liệu vào CSDL cũng được chuyển cho đối tượng khác.



Hình 4.16 biểu đồ trình tự bước một

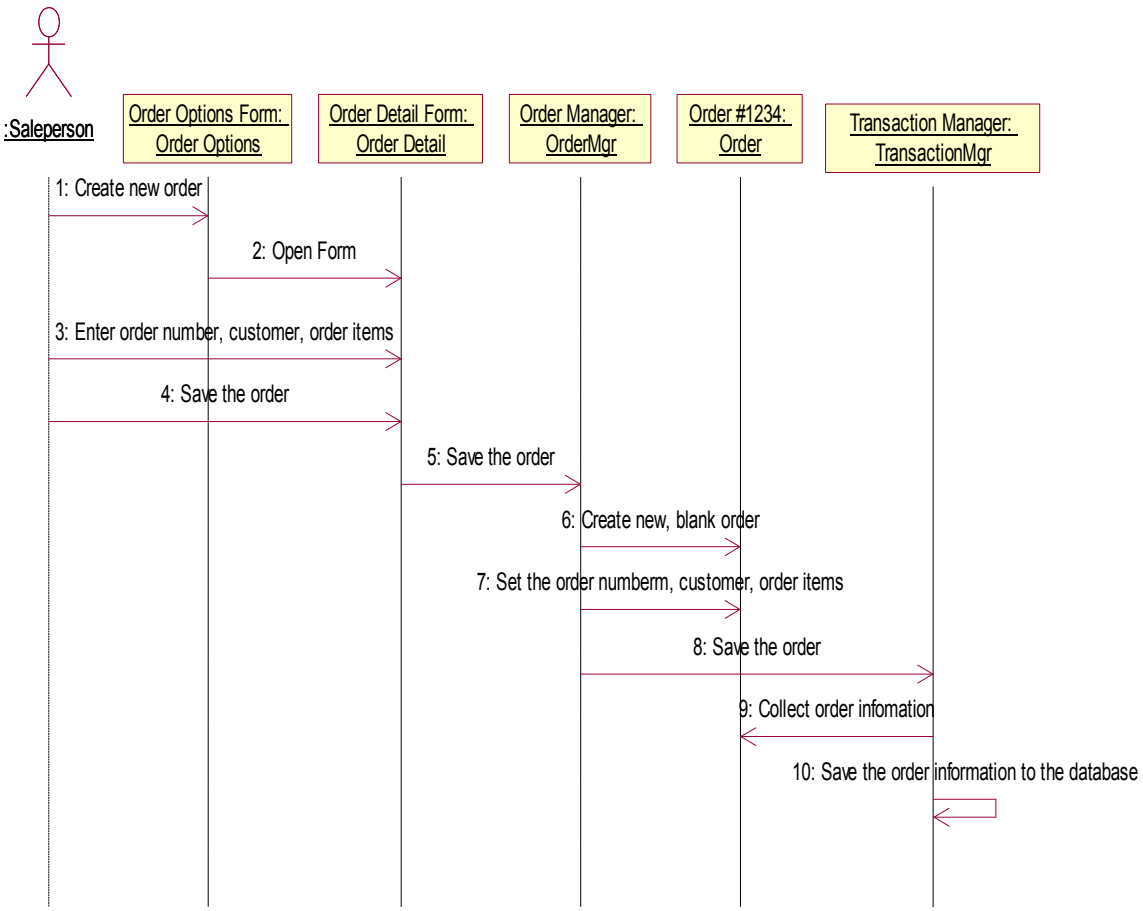
Bổ sung các đối tượng vào biểu đồ

Bổ sung đối tượng *Order Manager* vào giữa các đối tượng *Mẫu chi tiết đơn hàng* và *Đơn hàng #1234* nhờ phím công cụ.

Bổ sung đối tượng *Transaction Manager* vào phía phải đối tượng *Order #1234.*

Gán trách nhiệm cho các đối tượng mới

Sửa đổi các thông điệp trên biểu đồ trình tự bước một (hình 4.16) để có biểu đồ trình tự mới như trên hình 4.17.



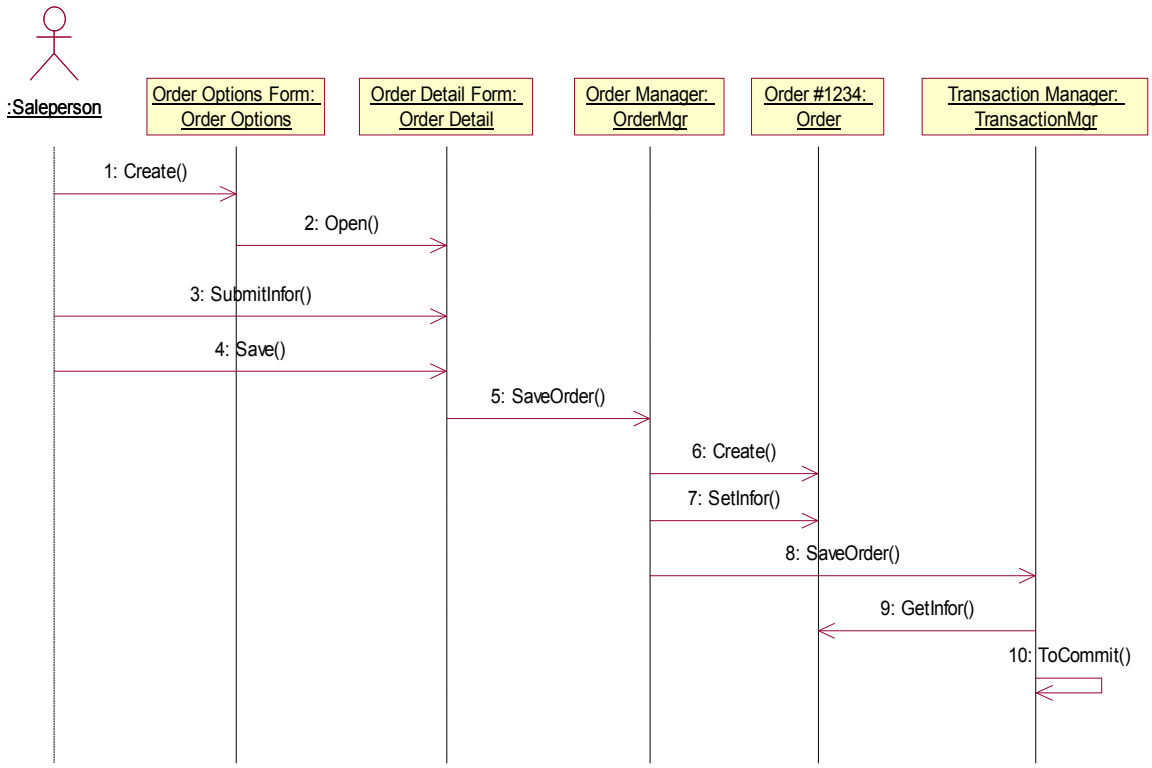
Hình 4.17 Biểu đồ trình tự bước hai

Ánh xạ đối tượng vào lớp

1. Nhấn phím phải chuột trên đối tượng *Order Detail Form* để chọn thực đơn *Open Specification*.
2. Chọn <*New*> trong hộp danh sách *Class*.
1. Nhập *OrderOptions* vào vùng *Name*.
2. Nhấn phím *OK*.
3. Nhập *OrderOptions* trong hộp danh sách cho vùng *Class*.
4. Nhấn phím *OK*.
5. Lặp lại để ánh xạ các đối tượng còn lại cho lớp.
    - Tạo lớp *OrderDetail* cho đối tượng *Order Detail Form*
    - Tạo lớp *OrderMgr* cho đối tượng *Order Manager*
    - Tạo lớp *Order* cho đối tượng *Order #1234*
    - Tạo lớp *TransactionMgr* cho đối tượng *Transaction Manager*

Ánh xạ thông điệp vào thao tác

1. Nhấn đúp vào thông điệp *Create new order*
2. Chọn <*new operation*>.
3. Nhập tên *Create* vào vùng *Name*
4. Nhấn *OK*
5. Nhấn đúp lên thông điệp lần nữa
6. Chọn thao tác mới : *Create()*
7. Lặp lại để có các thao tác cho các thông điệp còn lại (hình 4.18)



Hình 4.18 Biểu đồ trình tự với các thao tác

4.4.2.2 - Tạo dựng biểu đồ cộng tác (collaboration)

Tạo biểu đồ cộng tác

1. Nhấn phím phải chuột trên khung nhìn logic trong browser
2. Chọn *New\Collaboration diagram*
3. Đặt tên *Add order* cho biểu đồ mới
4. Nhấn đúp chuột trên biểu đồ mới để mở chúng

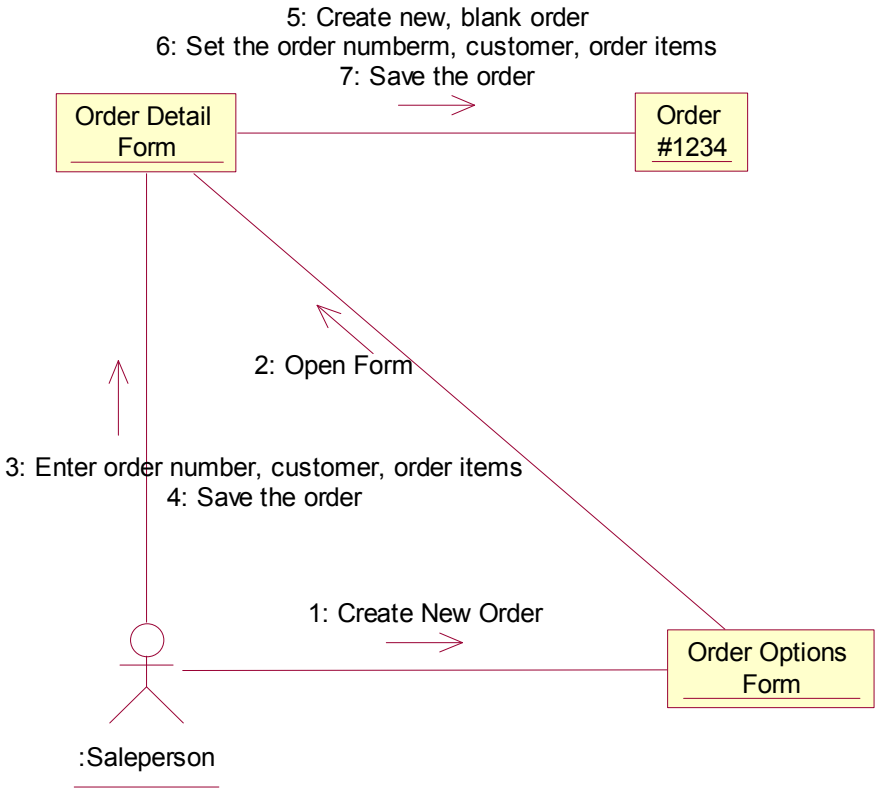
Bổ sung tác nhân và đối tượng

1. Di tác nhân *Salesperson* từ browser vào biểu đồ
2. Sử dụng phím *Object* trên thanh công cụ để bổ sung đối tượng vào biểu đồ
3. Đặt tên *Order Options Form* cho đối tượng mới.
4. Lặp lại để thêm các đối tượng *Order Detail Form* và *Order #1234*

Bổ sung thông điệp vào biểu đồ

1. Chọn phím công cụ *ObjectLink*
2. Vẽ từ tác nhân *Salesperson* đến đối tượng *Order Options Form*
3. Lặp để vẽ giữa
   - *Salesperson* và *Order Detail Form*
   - *Order Options Form* và *Order Detail Form*
   - *Order Detail Form* và *Order #1234*
4. Chọn phím *Link Message* để nhấn trên liên kết giữa *Salesperson* và *Order Detail Form*
5. Nhập *Create new order*
6. lặp từ bước 4 đến bước 6 để bổ sung thông điệp sau:
   - *Open form* (giữa *Order Options Form* và *Order Detail Form*)
   - *Enter order number, customer, order items* (giữa *Salesperson* và *Order Detail Form*)
   - *Save the order* (giữa *Salesperson* và *Order Detail Form*)
   - *Create new, blank order* (giữa *Order Detail Form* và *Order #1234*)
   - *Set the order number, customer, order items* (giữa *Order Detail Form* và *Order #1234*)
   - *Save the order* (giữa *Order Detail Form* và *Order #1234*)

Biểu đồ cộng tác của bước một như trong hình 4.19. Tương tự như vẽ biểu đồ trình tự, ta có thể chi tiết biểu đồ cộng tác bước một bằng cách bổ sung các đối tượng mới.



Hình 4.19 Biểu đồ cộng tác bước một

Bổ sung các đối tượng vào biểu đồ

Bổ sung hai đối tượng *Order Manager* và *Transaction Manager* như sau:

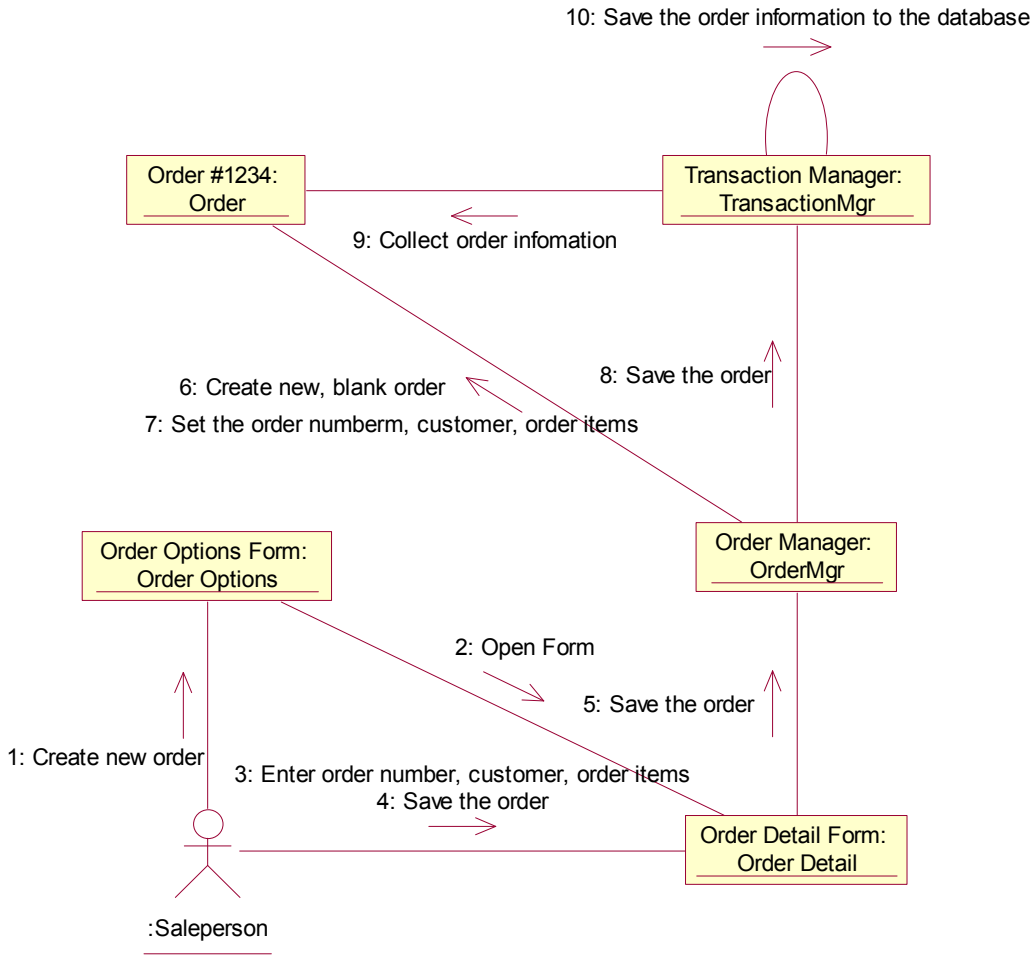
1. Chọn phím *Object* từ thanh công cụ
2. Nhấn phím chuột giữa đối tượng *Order Detail Form* và đối tượng *Order #1234* để bổ sung đối tượng mới
3. Đặt tên *Order Manager* cho đối tượng mới
4. Chọn lại phím công cụ *Object*
5. Bổ sung đối tượng mới vào phía phải đối tượng *Order #1234*
6. Đặt tên *Transaction Manager* cho đối tượng mới.

Gán trách nhiệm cho đối tượng mới

1. Chọn thông điệp *Create new*, *blank order* (chọn từ, không chọn mũi tên)
2. Nhấn *Ctrl+D* để xóa thông điệp
3. Lặp để xóa thông điệp 6 và 7 (*Set the order number, customer, order items,* và *Save the order*)
4. Chọn liên kết giữa *Order Detail Form* và *Order #1234* rồi nhấn *Ctrl+D*
5. Chọn phím công cụ *Object link* để vẽ giữa các đối tượng *Order Manager* và *Order #1234*

6. Chọn phím công cụ *Object link* để vẽ giữa các đối tượng *Order #1234* và *Transaction Manager*
7. Chọn phím công cụ *Object link* để vẽ giữa các đối tượng *Order Manager* và *Transaction Manager*
8. Chọn phím công cụ *Link Message* để bổ sung các thông điệp mới:
   - *Save the order* (giữa *Order Detail Form* và *Order Manager*)
   - *Create new blank order* (giữa *Order Manager* và *Order #1234*)
   - *Set the order number, customer, order items* (giữa *Order Manager* và *Order #1234*)
   - *Save the order* (giữa *Order Manager* và *Transaction Manager*)
   - *Collect order information* (giữa *Order Manager* và *Order #1234*)
9. Chọn phím *Link to Self* để bổ sung tự liên kết trên đối tượng Order Manager
10. Chọn phím *Link Message* để bổ sung thông điệp *Save the order information to the database* cho no.

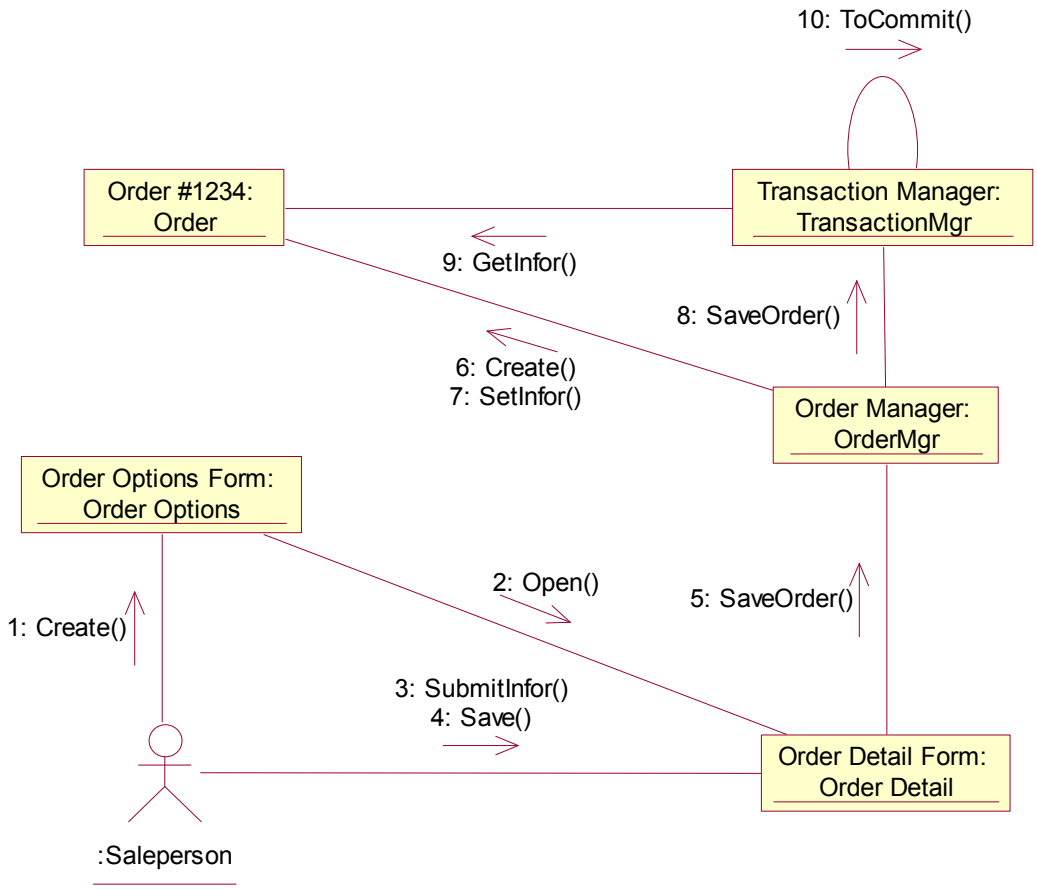
Biểu đồ kết quả như hình 4.20



Hình 4.20 Biểu đồ cộng tác bước hai

Ánh xạ đối tượng vào lớp

1. Tìm lớp *OrderOptions* trong Browser
2. Di nó vào đối tượng *Order Detail Form* trong biểu đồ
3. Lặp cho các cặp lớp và đối tượng sau:
   - Lớp *OrderDetail* cho đối tượng *Order Detail Form*
   - Lớp *OrderMgr* cho đối tượng *Order Manager*
   - Lớp *Order* cho đối tượng *Order #1234*
   - Lớp *TransactionMgr* cho đối tượng *Transaction Manager*.



Hình 4.21 Biểu đồ cộng tác với thao tác

Ánh xạ thông điệp và thao tác

1. Nhấn phím phải chuột trên thông điệp 1: *Create new order*
2. Chọn *Open Specification*
3. Nhập tên *Create* trong hộp danh sách *Name*
4. Nhấn *OK*
5. Lặp cho các ánh xạ sau đây:

- Thông điệp 2: *Open form* vào thao tác *Open*
- Thông điệp 3: *Enter order number, customer, order items* vào thao tác *SubmitInfo*
- Thông điệp 4: *Save the order* vào thao tác *Save*
- Thông điệp 5: *Save the order* vào thao tác *SaveOrder*
- Thông điệp 6: *Create new, blank order* vào thao tác *Create*
- Thông điệp 7: *Set the order number,customer, order items* vào thao tác *SetInfo*
- Thông điệp 8: *Save the order* vào thao tác *SaveOrder*
- Thông điệp 9: *Collect order information* vào thao tác *GetInfo*
- Thông điệp 10: *Save the order information to database* vào thao tác *Commit*

Biểu đồ kết quả như trên hình 4.21

Trong phần mềm công cụ *Rational Rose*, ta có thể tạo lập biểu đồ trình tự từ biểu đồ cộng tác hay ngược lại bằng **nhấn phím F5** hay chọn thực đơn *Browser > Create (Sequence\Collaboration) Diagram*.

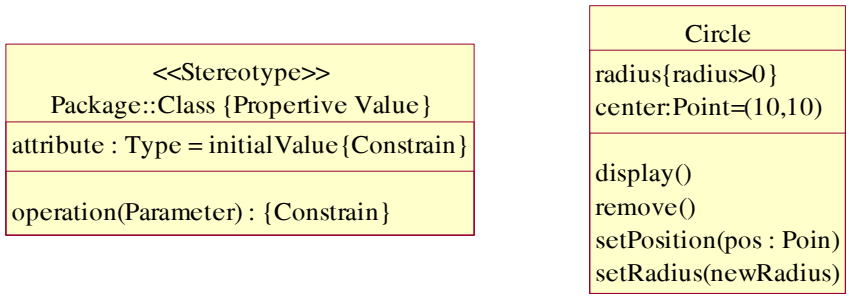
CHƯƠNG 5
BIỂU ĐỒ LỚP VÀ GÓI

Sau khi đã phân tích lĩnh vực vấn đề, ta sẽ tìm hiểu sâu hơn về các đối tượng trong đó để xây dựng mô hình đối tượng. Chương bốn đã đề cập đến một trong các kỹ thuật thiết kế hệ thống là xây dựng biểu đồ cộng tác. Chương này sẽ khảo sát một kỹ thuật thiết kế khác, đó là biểu đồ lớp. Phần cuối của chương là một số vấn đề liên quan đến ánh xạ đối tượng vào CSDL.

5.1 LỚP VÀ TIỀM KIẾM LỚP

Như đã trình bày trong các chương trước, đối tượng là cái gì đó tồn tại trong thế giới thực. Nó có thể là một phần của hệ thống như máy móc, tổ chức… Nhưng cũng có đối tượng không tồn tại trực tiếp, mà được phát sinh từ khảo sát cấu trúc hay hành vi của đối tượng trong thế giới thực. Do vậy, các đối tượng này cũng như đối tượng trong thế giới thực đều liên quan đến hiểu biết của chúng ta về thế giới thực. Lớp là mô tả thuộc tính, hành vi và ngữ nghĩa của một kiểu (một tập) đối tượng. Đối tượng là hiện thực của lớp. Quan hệ của đối tượng với lớp tương tự như quan hệ của biến với kiểu biến trong ngôn ngữ lập trình thông thường. Chúng ta sử dụng khái niệm lớp để phân tích các đối tượng nhận ra trong thế giới thực. Nhà bác học *Darwin* đã sử dụng khái niệm lớp để mô tả nòi giống loài người. Ông ta đã tổ hợp các lớp thông qua kế thừa để mô tả thuyết tiến hóa của mình. Tương tự, kỹ thuật kế thừa giữa các lớp cũng được sử dụng trong thiết kế hệ thống phần mềm theo phương pháp hướng đôi tượng. Không có lý do gì để xây dựng lớp trước khi tìm ra đối tượng. Biết rằng mỗi đối tượng là hiện thực lớp, nhưng lớp phải được đặc tả đầy đủ, rõ ràng sau khi đặc tả đối tượng. Thế giới quanh ta bao gồm các đối tượng chứ không phải lớp, do vập tìm đối tượng trước khi trừu tượng lớp từ chúng. Tóm lại, lớp là cái gì đó cung cấp kế hoạch chi tiết cho đối tượng. Nói cách khác lớp xác định thông tin nào đối tượng sẽ lưu giữ và hành vi nào đối tượng có thể có. Thí dụ lớp tài khoản của ông A là *Account*, lớp của ngôi nhà tại số 1 Tràng Tiền là *House*… Lớp *House* cho biết ngôi nhà co độ cao, chiều rộng, tổng số buồng ở. Lớp là khái niệm tổng quát, jo1 cho ta mẫu về đối tượng.

Ký pháp của lớp trong UML được biểu diễn bởi hình chữ nhật có ba phần dành cho tên lớp, các thuộc tính và các thao tác (hình 5.1). Tên lớp thường là danh từ bắt đầu bằng chữ hoa. Thuộc tính có tên của nó, kiểu đặc tả, giá trị khởi đầu, kiểu giá trị và ràng buộc. Thao tác cũng có tên, danh sách kiểu giá trị của tham số (có thể có giá trị khởi đầu) và ràng buộc. Phía trên tên lớp của hình 5.1a là tên *stereotype* của lớp trong dấu <<>>. Dưới tên lớp trong dấu ngoặc nhọn là mở rộng ngữ nghĩa của phần tử mô hình (*tagged values*) [OEST00], thí dụ chúng có thể là xâu ký tự {*abtract*}. Tên lớp có thể có tiền tố, thí dụ tiền tố là tên gói trước hai dấu chấm. HÌnh 5.1b là ví dụ ký pháp của lớp hình tròn *Circle*. Chúng có thuộc tính bán kính (*radius*), vị trí (*position*) và các thao tác như *display(), remove(), setPosition(pos) và setRadius(newRadius)*. Trong thí dụ này không có ký pháp gói còn ràng buộc {*radius>0*} cho biết thuộc tính đòi hỏi giá trị *radius* luôn lớn hơn 0. Giá trị khởi đầu (10,10) của thuộc tính *center* có nghĩa rằng khi đối tượng của lớp *Circle* được tạo lập thì thuộc tính này được gán bởi giá trị (10,10).



Hình 5.1 Ký pháp đồ họa của lớp

Vấn đề khó khăn nhất của thiết kế theo quan điểm hướng đối tượng là tìm ra đầy đủ các lớp cho hệ thống. Mặc dù vậy, đôi khi cũng dễ dàng tìm ra một số lớp. Nhiều tài liệu có lời khuyên là nơi tốt nhất để bắt đầu tìm lớp là *luồng sự kiện* của UC. Tìn ra *danh từ* trong luồn sự kiện sẽ cho ta biết về lớp. *Động từ* trong đó là phương pháp. Hãy chú ý rằng mọi danh từ tìm ra không nhất thiế là lớp mà nó có thể là một trong bốn loại sau: tác nhân, lớp, thuộc tính lớp và biểu thức không phải các loại trên.

Lớp còn có thể tìm thấy trong *biểu đồ tương tác*. Tìm những cái chung của đối tượng để hình thành lớp. Thí dụ có thể tạo lập biểu đồ trình tự chỉ sự tiến trình chi trả cho khách hàng trong ứng dụng ATM. Biểu đồ này có hai đối tượng *Khách hàng A* và *Khách hang B* rút tiền. Cả *Khách hàng A* và *Khách hang B* đều có chung một vài thuộc tính, đó là tên khách hàng, địa chỉ, số diện thoại. Hơn nữa, cả hai đều có một số thao tác tương tự nhau. Vậy lớp mới có thể được hình thành (thí dụ lớp *Employee*) cho đối tượng *Khách hàng A* và *Khách hang B*. Mỗi đối tượng trong biểu đồ tương tác sẽ được ánh xạ vào lớp tương ứng.

Tuy nhiên một số lớp không thể tìm thấy trong luồng sự kiện và biểu đồ tương tác. Trong pha phân tích yêu cầu thường nhận thấy nhu cầu *quan sát dữ liệu* trong các báo các khác nhau. Mỗi loại báo cáo này được biểu diễn như một lớp. Nếu có quá nhiều loại báo cáo thì thông thường những báo cáo tương tự được tập hợp để biểu diễn bởi một lớp, đồng thời sử dụng các biến thành phần để quản lý các biến thể khác nhau của báo cáo. Nhóm lớp khác được hình thành để *biểu diễn giao diện* giữa các thành phần khác nhau của hệ thống, đặc biệt để biểu diễn giao diện giữa lớp ta tạo ra với hệ thống khác mà nó tương tác. Giao diện là nơi một đối tượng tương tác với một số đối tượng khác (thí dụ trình điều khiển máy in hay lớp nào đó của thư viện). Các lớp giao diện hình thành tầng ảo, nó bảo vệ thiết kế của ta trước sự thay đổi của cả hai phía giao diện. Một tập lớp khác được hình thành để biểu diễn *thiết bị phần cứng* khác nhau mà hệ thống phần mềm sẽ tương tác. Thí dụ, hệ thống rút tiền tự động của ATM tương tác với máy đọc thẻ từ và mày in, mỗi thiết bị phần cứng này đươc biểu diễn bởi một lớp. Việc biểu diễn thiết bị và giao diện như các lớp cho khả năng bao gói các yêu cầu riêng của chúng. Nếu phải thay thế thiết bị sau này, ta không thể viết lại toàn bộ mã trình đã có.

Tốt nhất việc tìm kiếm lớp phải được cùng thưc hiện với chuyên gia lĩnh vực vấn đề. Sau đây là một sô câu hỏi giúp ta tìm ra lớp [ERIK98]:

- Có thông in nào cần lưu trữ hay phân tích? Nếu có nó là ứng viên của lớp.
- Có hệ thống ngoài không? Nếu có thì nó được xem như những lớp chứa trong hệ thống của ta hay hệ thống của ta tương tác với chúng.
- Có mẫu, thư viện lớp, thành phần…? Nếu có, thông thường chúng chứa các ứng viên lớp.

- Hệ thống cần quản lý các thiết bị ngoại vi nào? Mọi thiết bị kỹ thuật nối với hệ thống đều là ứng viên lớp.
- Tác nhân đóng vai trò tác nghiệp nào? Các nhiệm vụ này có thể là lớp; Thí dụ người sử dụng, thao tác viên hệ thống, khách hàng…

5.2 BIỂU ĐỒ LỚP

Biểu đồ lớp và biểu đồ đối tượng thuộc hai góc nhìn mô hình bổ sung cho nhau. Biểu đồ lớp chỉ ra trừu trượng thế giới thực, tập trung vào giải thích cấu trúc tĩnh từ góc nhìn tổng quát. Biểu đồ đối tượng biểu diễn trường hợp đặc biệt, cụ thể vào một thời điểm, nó thể hiện cấu trúc tĩnh và hành vi. Thông thường ta xây dựng đồng thời biểu đồ lớp và biểu đồ đối tượng.

Khung nhìn logic có thể chứa các thành phần đồ họa sau: biểu đồ tương tác, các lớp, biểu đồ lớp, biểu đồ UC, thuộc tính, thao tác, kết hợp (*association*) và biểu đồ chuyển trạng thái. Trong biểu đồ lớp được sử dụng để hiển thị lớp và gói của chúng trong hệ thống. Biểu đồ lớp cho hình ảnh tĩnh của các bộ phận hệ thống và các quan hệ giữ chúng.

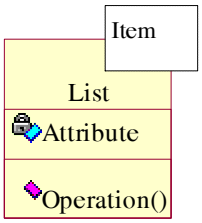
Thông thường mỗi hệ thống có vài biểu đồ lớp. Một số biểu đồ lớp trong số đó hiển thị lớp và quan hệ giữa các lớp, một vài biểu đồ lớp khác chỉ hiển thị gói lớp và quan hệ giữa các gói. Có thể tạo rất nhiều biểu đồ lớp để mô tả toàn bộ bức tranh hệ thống. Các biểu đồ lớp giúp người phát triển phần mềm quan sát và lập kế hoạch cấu trúc hệ thống trước khi viết mã trình. Nó đảm bảo rằng hệ thống được thiết kế tốt ngay từ ban đầu.

5.2.1 - Các loại lớp trong biểu đồ

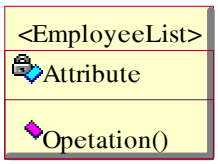
Biểu đồ lớp có thể chứa nhiều loại lớp khác nhau, chúng có thể là lớp thông thường, lớp tham số, lớp hiện thực, lớp tiện ích, lớp tiện ích tham số, lớp tiện ích hiện thực và *metaclass*

Lớp tham số (*parameterized class*). Lớp tham số là lớp được sử dụng để tạo ra họ các lớp khác. Nó còn có tên là lớp mẫu (*template*). Ngôn ngữ C++ hỗ trợ đầy đủ lớp này, nhưng *Java* lại không hoàn toàn như vậy. Thí dụ từ lớp tham số *List* có thể tạo ra các lớp hiện thực *Employeelist, AccountList*... Trong UML, lớp tham số được biểu diễn như trên hình 5.2.

Lớp hiện thực (*instantiated class*). Lớp hiện thực là lớp tham số mà đối số của nó có giá trị. Thí dụ, lớp tham số trên đây chứa danh sách phần tử. Bây giờ ta có thể cung cấp giá trị cho đối số của phần tử, kết quả là ta có danh sách các nhân viên. Trong UML, ký pháp lớp hiện thực là lớp có tên đối số trong ngoặc <> như trên hình 5.3.

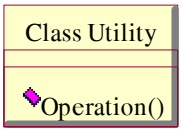


Hình 5.2 Lớp tham số

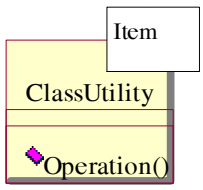


Hình 5.3 Lớp hiện thực tham số

Lớp tiện ích (*class utility*). Lớp tiện ích là tập hợp các thao tác. Thí dụ có tập hàm toán học như *squareroot(), cuberoot()*... sẽ được sử dụng nhiều nơi trong hệ thống, chúng có thể được gói vào lớp tiện ích để lớp khác trong hệ thống cùng sử dụng. Trong biểu đồ lớp, lớp tiện ích được thể hiện bằng lớp đường viền bóng như trên hình 5.4.

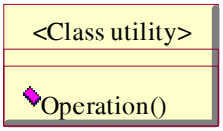


Hình 5.4 Lớp tiện ích



Hình 5.5 Lớp tiện ích tham số

Lớp tiện ích tham số (*parameterized class utility*). Lớp tiện ích tham số là lớp tham số chứa tập các thao tác. Đó là mẫu để tạo lập ra lớp tiện ích. Lớp này được vẽ trong biểu đồ lớp bằng biểu tượng như trên hình 5.5.



Hình 5.6 Lớp tiện ích hiện thực

Hình 5.7 Metaclass

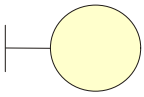
Lớp tiện ích hiện thực (*instantiated class utility*). Lớp tiện ích hiện thực là lớp ham số mà đối của chúng có giá trị. Lớp này được vẽ trong biểu đồ lớp bằng biểu tượng như trên hình 5.6.

Metaclass. *Metaclass* là lớp mà hiện thực của nó là lớp chức không phải là đối tượng. Lớp tham số và lớp tiện ích là những thí dụ của *metaclass*. Lớp này được vẽ trong biểu đồ lớp bằng biểu tượng trên hình 5.7

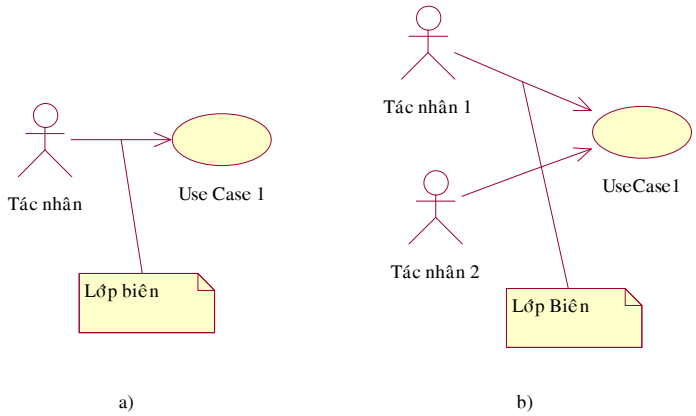
5.2.2 - Stereotype của lớp

Trong biểu đồ lớp, *stereotype* là cơ chế để phân nhóm lớp. Thí dụ, để nhanh chóng tìm kiếm biểu mẫu trong mô hình, ta tạo *stereotype* với tên biểu mẫu (*form*) rồi gán nó cho mọi lớp tương ứng. UML có ba loại *stereotype* mặc định để gán cho ba loại lớp, đó là lớp biên (*Boundary*), lớp thực thể (*Entity*) và lớp điều khiển (*Control*).

Lớp biên (*boundary class*). Lớp biên là lớp nằm trên biên hệ thống và pah62n thế giới còn lại. Chúng có thể là biểu mẫu (*form*), báo cáo (*report*), giao diện với phần cứng như máy in, máy quyét… và là giao diện với các hệ thống khác. Ký pháp trong UML của lớp biên như sau:



Để tìm lớp biên hãy khảo sát biểu đồ UC. Với mỗi tác nhân ta có ít nhất một lớp biên (hình 5.8a). Lớp biên là cái cho pháp tác nhân tương tác với hệ thống. Không nhất thiếp phải tạo ra một lớp biên cho mỗi tác nhân. Nếu hai tác nhân cùng kích hoạt một UC thì chỉ cần tạo ra một lớp biên cho cả hai (hình 5.8b).



Hình 5.8 Lớp biên

Lớp thực thể (*entity class*). Lớp thực thể lưu trữ thông tin mà nó sẽ được ghi vào bộ nhớ ngoài. Thí dụ lớp *Employee* là lớp thực thể. Các lớp thực thể có thể tìm thấy trong luồn sự kiện và biểu đồ tương tác. Nó là lớp có ý nghĩa nhất đối với người sử dụng. Trong UML, ký pháp của lớp thực thể như sau:

Thông thường ta phải tạo ra một bảng trong CSDL cho mỗi lớp loại này. Thay cho xác định cấu trúc CSDL trước, ta sẽ phát triển cấu trúc CSDL từ thông tin của mô hình đối tượng. Các yêu cầu hệ thống xác định luồng sự kiện. Luồng sự kiện xác định đối tượng và lớp. Mỗi thuộc tính của lớp thực thể trở thành trường torng CSDL.

Lớp điều khiển (*control class*). Lớp điều khiển có trách nhiệm điều phối hoạt động của các lớp khác. Thông thường mỗi UC có một lớp điều khiển để điều khiển trình tự các sự kiện trong nó. Chú ý rằng lớp điều khiển không tự thực hiện chức năng nào (lớp khác không gửi nhiều thông điiệp đến chúng), nhưng chúng lại gửi nhiều thông điệp đi đến lớp khác. Do vậy, lớp điều khiển còn gọi là lớp quản lý. Trong UML, lớp điều khiển được ký hiệu như sau:

Một loại lớp điều khiển khác là lớp cùng chia sẽ một vài UC. Thí dụ lớp SecurityManager có trách nhiệm điều khiển các sự kiện liên quan đến an toàn. Lớp TransactionManager có trách nhiệm điều phối các thông điệp liên quan đến giao dịch CSDL. Sử dụng loại lớp điều khiển này là cách tốt nhất để tách các chức năng trong hệ thống. Nếu thay đổi trình tự logic của chứng năng nào thì chỉ lớp điều khiển bị ảnh hưởng.

Ngoài các *stereotype* mặc định của UML trình bày trên đây, ta còn có thể bổ sung *stereotype* mới vào mô hình để sử dụng cho mục đích riêng.

5.3 GÓI

Gói được sử dụng để nhóm các lớp có cái chung với nhau. Có nhiều quan điểm khi hình thành gói lớp như dựa trên các chứ năng hay dựa trên đặt tính kỹ thuật. Quan điểm chung là ta có thể tùy ý gói các lớp lại với nhau. Thí dụ, nếu chọn tiêu chí là *stereotype* thì sẽ có gói của các lớp *Entity*, gói của các lớp *Boundary*, gói của các lớp *Control*... Quan điểm khác là gói lớp theo chức

năng. Thí dụ, có thể gói *Security* chứa lớp liên quan đến an toàn hệ thống, gói khá là *Reporting* hay *Error Handling*… Lợi thế của gói là cho khả năng dễ sử dụng lại. Thí dụ, có thể lấy gói *Sercurity* để sử dụng lại trong các ứng dụng khác. Sau đây là ví dụ về các gói trong hệ thống quản lý thư viện.

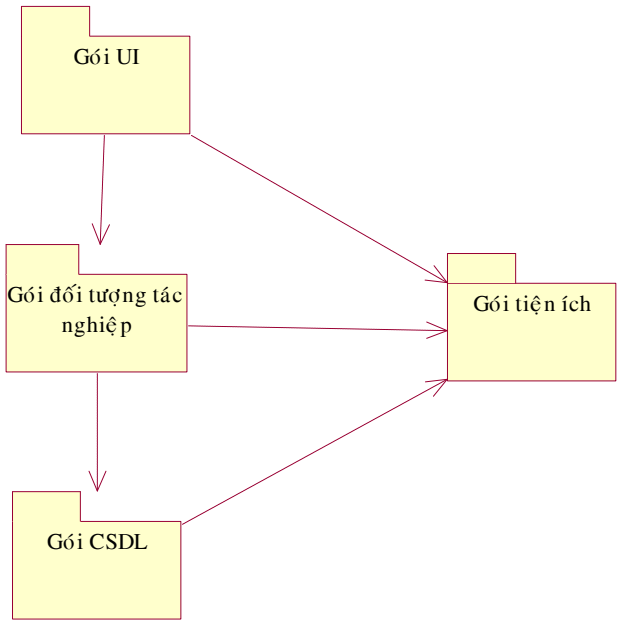
Gói giao diện (UI): bao gồm các lớp giao diện người dùng, cho khả năng quan sát dữ liệu và nhập dữ liệu mới. Các lớp này xây dựng trên cơ sở thư viện chuẩn của ngôn ngữ lập trình như *MFC* của *Visual C++*. Gói này sẽ kết hợp với gói các đối tượng tác nghiệp. Gói UI gọi các thao tác trên đối tượng tác nghiệp để truy vấn hay nhập dữ liệu.

Gói đối tượng tác nghiệp: bao gồm các lĩnh vực từ mô hình phân tích. Chúng sẽ được chi tiết khi thiết kế để bổ sung các thao tác và hỗ trợ lưu trữ. Gói các đối tượng tác nghiệp hợp tác với gói CSDL để các lớp đối tượng tác nghiệp kế thừa từ lớp lưu trữ trong gói CSDL.

Gói CSDL: gói này cung cấp dịch vụ cho các lớp khác trong gói tác nghiệp để nó có thể lưu trữ (thí dụ ghi lên tập đĩa).

Gói tiện ích: gói này chứa các dch vụ để các gói khác trong hệ thống sử dụng. Thí dụ gói chứa lớp lưu trữ để lưu trữ đối tượng trong toàn bộ hệ thống và nó được sử dụng trong các gói UI, tác nghiệp và CSDL.

Hình 5.9 là thiết kế các gói vừa mô tả trên.



Hình 5.9 Gói các lớp

5.4 THUỘC TÍNH LỚP

Thuộc tính là bộ phận thông tin liên kết với lớp, thí dụ lớp *Công ty* chứa các thuộc tính như *Tên, địa chỉ, số nhân viên*. Mỗi lớp trong mô hình chứa một hay nhiều thuộc tính.

5.4.1 - Tìm kiếm thuộc tính

Có nhiều nơi để tìm ra thuộc tính lớp. Trước hết, tìm thuộc tính trong tài liệu UC. Tìm các *danh từ* trong luồng sự kiện. Chú ý rằng không phải mọi danh từ là thuộc tính mà một số danh từ trong đó có thể là lớp, đối tượng… Thí dụ khi xem xét luồng sự kiện: *"Người sử dụng nhập tên*

nhân viên, địa chỉ, số bảo hiểm xã hội và số điện thoại", ta nhận ra lớp *Nhân viên* và các thuộc tính là *Tên, Địa chỉ, Số điện thoại và Số chứng minh thư.* Sau khi tìm ra các danh từ thì phải thận trọng khi quyết định nó là lớp hay thuộc tính, điều này phụ thuộc vào ứng dụng cụ thể. Thí dụ, danh từ công ty sẽ gợi ý lớp có tên là *Công ty* hay nó được xem là thuộc tính của lớp khác, lớp *Nhân viên*? Khi muốn lưu trữ thông tin về công ty và nó có hành vi liên quan đến công ty thì hãy cho nó là lớp. Thí dụ, hệ thống bán hàng, ta cần lưu trữ thông tin về các công ty có mua sản phẩm hay có nhu cầu về dịch vụ, thì công ty là lớp với các thuộc tính số lượng nhân viên của công ty, tên, địa chỉ công ty… Ngược lại, khi không cần biết thông tin chi tiết về công ty, thí dụ trong hệ thống có chức năng in thư gửi khách hàng thuộc các công ty khác nhau thì hãy coi tên công ty là thuộc tính của lớp nào đó, thí dụ lớp *Giao tiếp*. Mặc khác, cần phải xem xét thông tin tư sanh từ đó có hành vi hay không. Nếu công ty có hành vi trong ứng dụng thì nó được mô hình hóa như lớp, ngược lại nếu công ty không có hành vi thì mô hình hóa nó như thuộc tính. Nơi khác để có thể tìm ra thuộc tính là trong tài liệu yêu cầu hệ thống. Nếu có yêu cầu loại thông tin mà hệ thống sẽ thu thập thì các thông tin này đều là thuộc tính trong lớp nào đó. Cuối cùng, thuộc tính có thể được tìm thấy trong cấu trúc CSDL. Nếu đã hình thành cấu trức CSDL trước đó thì các trường trong bảng là nơi tốt nhất để xác định thuộc tính. Thí dụ, bảng *Nhân viên* có các trường *Tên, Địa chỉ, Số điện thoại và Số chứng minh thư* thì các trường này là ứng viên thuộc tính. Thông thường thực hiện ánh xạ một-một giữa bảng CSDL với lớp thực thể. Nhưng cần chú ý rằng không phải lúc nào cũng ánh xạ được như vậy mà phải xem xét kỹ lưỡng khi thiết kế CSDL và các lớp vì bảng CSDL quan hệ không hỗ trợ trực tiếp kế thừa.

Khi đã nhận ra thuộc tính thì phải đảm bảo chắc chắn rằng nó là cần thiết cho yêu cầu hệ thống nào đó, có nghĩa rằng nó cần thiết cho khách hàng. Điều này sẽ tránh được vấn đề muôn thuở là vô số thông tin được tìm ra nhưng không ai cần đến chúng. Tiếp theo là phải thận trọng gán thuộc tính cho lớp phù hợp, thí dụ lớp *Nhân viên* có tên và địa chỉ, nhưng không cần có thông tin về loại sản phẩm mà công ty của nhân viên này sản xuất. Thông tin sản phẩm nên được đặt trong lớp khác, lớp *Sản phẩm*. Không nên xây dựng lớp có quá nhiều hay quá ít thuộc tính (tốt nhất mỗi lớp có ít hơn 10-15 thuộc tính). Phải đảm bảo chắc chắn rằng các thuộc tính trong lớp phải thực sự cần thiết và thuộc lớp đó. Nếu lớp có quá nhiều thuộc tính thì có thể chia ra nhiều lớp nhỏ. Ngược lại, không nên xây dựng lớp với một hay hai thuộc tính.

5.4.2 - Đặc tả thuộc tính

Các thuộc tính của lớp được gán đặc tả chi tiết, chúng bao gồm kiểu dữ liệu thuộc tính, *stereotype,* giá trị khởi đầu, phạm vi…

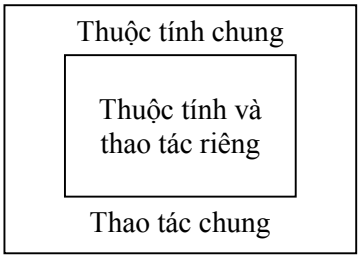
Kiểu dữ liệu thuộc tính. Kiểu dữ liệu là kiểu phụ thuộc ngôn ngữ lập trình như *string, integer, long* hay *boolean*. Trước khi phát sinh mã trình từ biểu đồ ta phải gán kiểu dữ liện cho mỗi thuộc tính. Có thể sử dụng kiểu dữ liệu mặc định cho ngôn ngữ lập trình hay kiểu dữ liệu mới (tên lớp) do ta xác lập.

***Stereotype* của thuộc tính**. Tương tự tác nhân, UC và lớp, ta cũng có thể gán *stereotype* cho thuộc tính (việc này không bắt buộc). Đây cũng chỉ là cách phân lớp thuộc tính để dễ quan sát và dễ hiểu biểu đồ.

Giá trị khởi đầu thuộc tính. Rất nhiều thuộc tính có giá trị mặc định. Thí dụ, lớp *Order* chứa các thông tin và hành vi về đơn mua hàng của công ty. Lớp này có thể có thuộc tính với tên *TaxRate* để lưu trữ tỷ lệ thuế mua hàng. Nếu thực tế tỷ lệ thuế mua hàng là 10% thì có thể nạp giá trị ban đầu 0,08 cho thuộc tính *TaxRate*. Việc nạp giá trị khởi đầu thuộc tính cũng có thể là tùy ý, không bắt buộc..

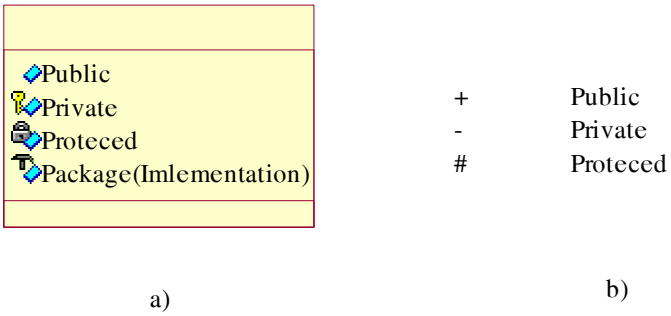
Phạm vi của thuộc tính. Một rong những tính chất quan trọng của lập trình hướng đối tượng là tính bao gói. Mỗi lớp bao gói một ít thông tin và một vài hành vi nhờ các thuộc tính và thao tác của nó. Thuộc tính nằm trong lớp cho nên nó ẩn với các lớp khác. Hình 5.10 mô tả quan sát thuộc

tính và thao tác của lớp. Vì thuộc tính được bao gói trong lớp cho nên ta phải định nghĩa lớp nào có thể xâm nhập và làm thay đổi được nó. Việc làm này được gọi là xác định phạm vi thuộc tính (*attribute visibility*).



Hình 5.10 Phạm vi thuộc tính và thao tác

UML hỗ trợ bốn tùy chọn phạm vi thuộc tính, đó là chung (*public*), riêng (*private*), bảo vệ (*protected*), gói (*package*) hay cài đặt (*implemetation*). Ký pháp đồ họa của chúng trong biểu đồ như trên hình 5.11. Hình 5.11a là ký pháp chuẩn của UML, UML không có biểu tượng cho thuộc tính có phạm vi *package* hay *implemetation.* Hình 5.11b là ký pháp phạm vi trong phần mềm công ty *Rational Rose.*



Hình 5.11 Biểu tượng phạm vi

- *Public*. Các thuộc tính có tính chất này sẽ được xâm nhập từ mọi lớp khác. Bất kỳ lớp nào rong hệ thống đều có thể quan sát và sửa đổi giá trị của thuộc tính. Thí dụ ta có hai lớp với tên *Employee* và *Company*. Lớp *Employee* có thuộc tính *Address*. Nếu *Address* có tính chất *public* thì lớp *Company* có thể quan sát và sửa đổi giá trị của nó.
- *Private*. Thuộc tính được gán tính chất này thì sẽ không *nhìn thấy* được từ lớp khác. Nếu *Address* được gán *Private* thì chỉ lớp *Employee* mới làm thay đổi được nội dung của thuộc tính này còn lớp *Company* thì không. Nếu lớp *Company* muốn quan sát hay sửa đổi giá trị của *Address* thì nó phải yêu câu *Employee* làm việc này thông qua các thao tác *public*.
- *Protected*. Thuộc tính có tính chất này được chính lớp của nó và các lớp phân cấp dưới nó (lớp kế thừa) có thể xâm nhập. Giả sử rằng ta có lớp nhân viên hưởng lương theo giờ với tên *HourlyEmp* và lớp nhân viên hưởng lương tgheo tháng *SalariedEmp* đều kế thừa từ lớp *Employee.* Nếu thuộc tính *Address* được gán tính chất *protected* thì lớp *Employee, HourlyEmp, SalariedEmp* có thể xâm nhập nó còn có lớp *Company* thì không.
- *Package Implemetation.* Thuộc tính được gán tính chất này cho thấy nó là *public*, nhưng chỉ các lớp trong gói mới xâm nhập được. Nếu *Address* được gán thuộc tính *package* thì lớp *Company* có thể xâm nhập nếu lớp *Company* và lớp *Employee* ở trong cùng gói.

Tính chất lưu trữ. Tính chất này mô tả các thuộc tính sẽ được lưu trữ trong lớp như thế nào. Có ba loại lưu trữ như mô tả dưới đây:

- *Giá trị (by value)*. Thuộc tính được gán tính chất này cho biết nó sẽ được chứa trong lớp. Thí dụ, nếu có thuộc tính kiểu *string* thì xâu sẽ được lưu trong lớp.
- *Tham chiếu (by reference)*. Thuộc tính được gán tính chất này cho biết nó được lưu trữ ở bên ngoài lớp, nhưng lớp có con trỏ trỏ đến nó. Thí dụ, có thuộc tính kiểu *Date* trong lớp *Person*. Đối tượng *Date* được đặt ngoài lớp *Person.* Thuộc tính trong *Person* chỉ là con trỏ trỏ đến đối tượng ngoài.
- *Chưa xác định (Unspecified)*. Thuộc tính được gán tính chất này cho biết cách lưu trữ là chưa xác định, nhưng khi phát sinh mã trình *Rational Rose* coi là *By value.*

Thuộc tính tĩnh. Khi gán thuộc tính thông thường cho lớp thì mỗi hiện thực của lớp sẽ có riêng bản sao thuộc tính. Thí dụ, lớp *Account* với thuộc tính *Rate* được hiện thực hóa thành hai đối tượng *Tài khoản ông A* và *Tài khoản ông B*. Mỗi đối tượng này đều có bản sao thuộc tính *Rate*. Nhưng thuộc tính *static* là thuộc tính chia sẽ cho mọi hiện thực lớp. Nếu *Rate* được gán tính chất *static* thì nó sẽ chia sẽ cho *Tài khoản ông A* và *Tài khoản ông B.* Có nghĩa rằng chỉ có một thuộc tính *Rate* và dùng chung cho các đối tượng của lớp *Account.* UML biểu diễn thuộc tính tĩnh bằng dấu $ trước tên thuộc tính. Thí dụ lớp *Account* có thuộc tính tỷ lệ lãi xuất như sau: *#$Rate:float*

Thuộc tính suy diễn. Là thuộc tính được hình thành từ một hay nhiều thuộc tính khác. Thí dụ, lớp *Chữ nhật* có các thuộc tính *Độ rộng* và *Chiều cao.* Nó cũng có thể có thuộc tính *Diện tích* được tính từ độ rộng và chiều cao. Vì *Diện tích* được suy diễn từ *Độ rộng* và *Chiều cao* cho nên nó được gọi là thuộc tính suy diễn. Trong UML thuộc tính suy diễn được ký hiệu bằng dấu / trước tên thuộc tính. Thí dụ thuộc tính suy diễn *Diện tích* của lớp *Chữ nhật* như sau: */Diện tích.*

5.5 THAO TÁC CỦA LỚP

Thao tác là hành vi kết hợp với lớp. Mỗi thao tác có ba phần như sau: tên thao tác, tham số thao tác và kiểu cho lại của thao tác. Tham số là đối tượng mà thao tác nhận ở đầu vào. Kiểu cho lại là đầu ra của thao tác. Trên biểu đồ lớp có thể nhìn thấy tên thao tác hay tên thao tác, sau đó là tham số và giá trị cho lại. Ký pháp của thao tác trên UML như sau:

Operation Name (arg1: arg1 data type, arg2: arg2 data type…): return type.

Thao tác xác định trách nhiệm của lớp. Sau khi đã nhận biết thao tác thì xem xét lại lớp chứa chùng, chú ý rằng:

- Nếu lớp chỉ có một hay hai thao tác thì nên gộp vào lớp khác.
- Nếu lớp không có thao tác thì nên mô hình nó như thuộc tính.
- Nếu lớp có quá nhiều thao tác thì khó quản lý, cho nên nên chia sẽ chúng ra các lớp khác.

Việc lập biểu đồ tương tác có nghĩa là đã thực hiện phần lớp các công việc cần thiết cho việc tìm kiếm thao tác. Có bốn loại thao tác cần xem xét, bao gồm:

Thao tác cài đặt (*implemetor*). Thao tác này cài đặt một vài chức năng tác nghiệp. Chúng được tìm ra từ các biểu đồ tương tác. Biểu đồ tượng tác tập trung vào các chức năng tác nghiệp, hầu như mỗi thông điệp trên biểu đồ được ánh xạ thành thao tác cài đặt. Mỗi thao tác này có thể được chuyển ngược lại về nhu cầu hệ thống. Việc này được thực hiện như sau: Mỗi thao tác được hình thành từ biểu đồ tương tác, biểu đồ thao tác được hình thành từ các chi tiết trong luồng sự kiện, luồng sự kiện được hình thành từ UC và cuối cùng là UC được hình thành từ yêu cầu hệ thống. Điều này khẳng định rằng mỗi yêu cầu đều được cài đặt bằng mã trình và mỗi đoạn mã trình phản ánh ngược lại yêu cầu.

Thao tác quản lý *(manager*). Thao tác *manager* quản lý việc tạo lập và hủy bỏ các đối tượng. Thí dụ, cấu tử và hủy tử của lớp thuộc nhóm này.

Thao tác xâm nhập (*access*). Thông thường thuộc tính có tính chất *private* hay *protected*. Tuy nhiều lớp trong hệ thống lại có nhu cầu xâm nhập thuộc tính của lớp khác. Việc này được thực hiện hiện nhờ thao tác xâm nhập. Thí dụ, lớp *Employee* có thuộc tính *salary* và ta không muốn các lớp khác xâm nhập trực tiếp thuộc tính này. Hãy gán đặc tính *private* cho nó và bổ sung hai tháo tác xâm nhập vào lớp *Employee*, đó là *GetSalary* và *SetSalary*. Hãy gán tính chất *public* cho cả hai thao tác này để các lớp khác có thể gọi chúng được. Thao tác *GetSalary* có khả năng lấy giá trị trong thuộc tính *Salary* và trả lại giá trị cho lớp gọi nó. Thao tác *SetSalary* hỗ trợ lớp khác gán giá trị vào thuộc tính *Salary*. Cách tiếp cận này cho khả năng các thuộc tính được bao gói an toàn trong lớp và được bảo vệt rước lớp khác, nhưng vẫn cho các lớp khác xâm nhập chúng. Thông thường thao tác *Get* và *Set* được hình thành cho mỗi thuộc tính trong lớp.

Thao tác trợ giúp (*helper*). Thao tác *Helper* là các thao tác mà chính lớp chứa nó cần đến để thực hiện trách nhiệm, nhưng các lớp khác không cần biết gì về chúng. Đó là các thao tác có tính chất *private* và *proteted* của lớp. Tương tự như thao tác cài đặt, thao tác loại này cũng được tìm ra khi khảo sát biểu đồ tương tác. Thông thường nó được hình thành từ các thông điệp phản thân trong biểu đồ trình tự và biểu đồ cộng tác.

Tóm lại, để nhận ra các thao tác của lớp ta phải:

- Khảo sát biểu đồ tương tác: PHần lớn các thông điệp của biểu đồ này sẽ trở thành thao tác *implementor*. Các thông điệp phản thân trở thành thành *helper.*
- Các thao tác quản lý: Bổ sung cấu tử và hủy tử.
- Các thao tác xâm nhập: Tạo lập các thao tác *Get, Set* cho các thuộc tính cần được xem xét trong lớp khác hay lớp khác cần làm thay đổi giá trị của chúng.

5.6 QUAN HỆ

Quan hệ là kết nối ngữ nghĩa giữa các lớp, nó cho phép một lớp biết về các thuộc tính, thao tác và quan hệ của lớp khác. Các quan ệh được thể hiện trên biểu đồ lớp. Giữa các lớp có bốn kiểu quan hệ chính, đó là: kết hợp (*association*), phụ thuộc (*dependencies*), tập hợp (*aggregation*) và khái quát hóa (*generalization*).

Để tìm kiếm quan hệ ta phải khảo sát mọi phần tử mô hình. Thông tin về quan hệ kếp hợp và quan hệ phụ thuộc được dễ dàng tìm thấy trong các biểu đồ tương tác. Các quan hệ tập hợp và khái quát hóa có thể được tìm thấy khi khảo sát các lớp. Vậy, để tìm ra quan hệ giữa các lớp ta thực hiện các công việc sau đây:

Khởi đầu bằng khảo sát biểu đồ trình tự và biểu đồ cộng tác. Nếu *Lớp A* gửi thông điệp đến *Lớp B* trên biểu đồ tương tác, thì giữa chúng phải có quan hệ. Thông thường, quan hệ tìm ra ở đây là quan hệ kết hợp hay phụ thuộc.

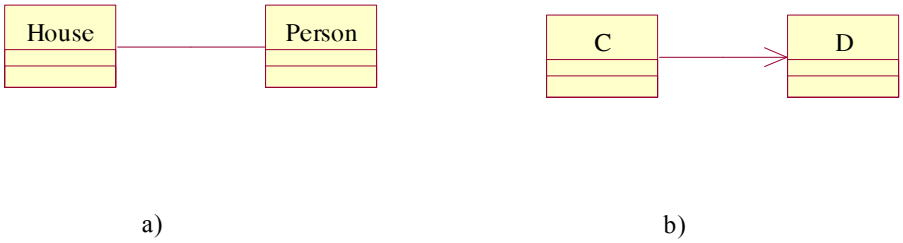
Khảo sát các lớp để tìm ra quan hệ tổng thể - thành phần. Bất kỳ lớp nào được hình thành từ các lớp khác thì chúng có quan hệ tập hợp.

Khảo sát các lớp để tìm ra quan hệ khái quát hóa. Nếu tìm ra các lớp hoàn toàn khác nhau cùng kế thừa từ một lớp thứ ba thì giữa chúng với lớp thứ ba có quan hệ khái quát hóa.

5.6.1 - Quan hệ kết hợp

Quan hệ kết hợp là kết nối ngữ nghĩa giữa hai lớp. Quan hệ này được vẽ bằng đường đơn rong biểu đồ lớp. Khi có quan hệ kết hợp, mỗi lớp có thể gửi thông điệp đến lớp khác trong biểu

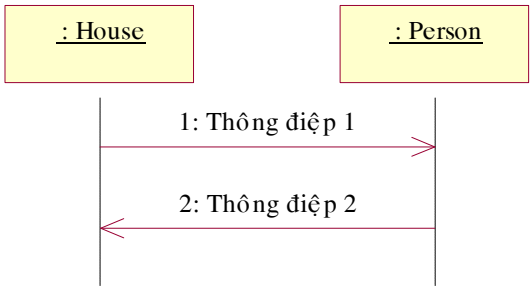
đồ tương tác. Kết hợp có thể một chiều hay hai chiều. Cú pháp UML biểu diễn kết hợp hai chiều bằng đường vẽ không mũi tên (hình 5.12a) hay có mũi tên trên cả hai đầu, còn kết hợp một chiều thể hiện bằng một mũi tên (hình 5.12b).



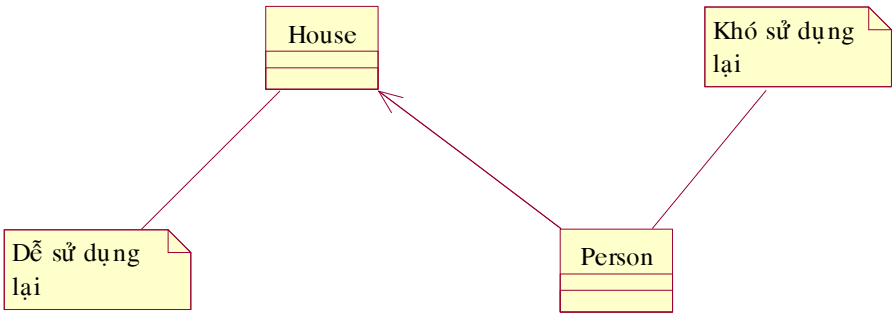
Hình 5.12 Quan hệ kết hợp

Quan hệ kết hợp hai chiều trên hình 5.11a giữa lớp *House* và lớp *Person* thể hiện rằng lớp *Person* biết thuộc tính và thao tác *public* của lớp *House,* và ngược lại lớp *House* biết các thuộc tính và thao tác có tính chất *public* của lớp *Person*. Trong biểu đồ trình tự *House* gửi thông điệp đến *Person* và *Person* gửi thông điệp đến *House* (hình 5.13).

Quan hệ kết hợp trên hình 5.13 là quan hệ hai chiều. Thực tế, ta thường phải chuyển đổi quan hệ kết hợp hai chiều thành quan hệ một chiều vì quan hệ một chiều dễ quản lý, dễ xây dựng hơn và nó cho khả năng tìm ra lớp tái sử dụng. Trên hình 5.14 là thí dụ quan hệ một chiều. Trong thí dụ này, lớp *Person* biết thuộc tính và thao tác *public* của lớp *House*, nhưng lớp *House* không biết gì về lớp *Person.* Trên biểu đồ tương tác, *Person* có thể gữi thông điệp để *House* nhận, nhưng *House* không thể gữi thông điệp.



Hình 5.13 Quan hệ kết hợp hai chiều



Hình 5.14 Quan hệ kết hợp một chiều

Cài đặt quan hệ kết hợp hai chiều bằng cách đặt đối tượng lớp này làm thuộc tính của lớp kia. Từ thí dụ quan hệ kết hợp giữa lớp *House* và lớp *Person*, Rose sẽ đặc thuộc tính *person* vào lớp *House* và thuộc tính *house* vào lớp *Person*. Hình 5.15 là cài đặt quan hệ kết hợp hai chiều bằng ngôn ngữ C++. Hình 5.16 là cài đặt quan hệ kết hợp một chiều.

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

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

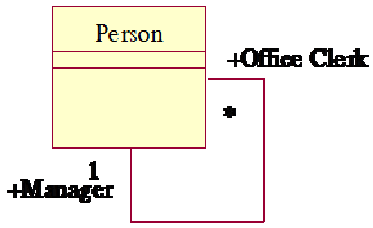
Hình 5.15 Cài đặt quan hệ kết hợp hai chiều

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

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

Hình 5.16 Cài đặt quan hệ kết hợp một chiều

Quan hệ kết hợp có thể đệ qui. Kết hợp đệ qui cho thấy một hiện thực của lớp có quan hệ với một hiện thực khác của cùng lớp đó. Thí dụ, với lớp *Person*, một người có thể là cha của nhiều người hay một người là quản lý của nhiều nhân viên như hình 5.17. *UnBoungedSetByReference* trong cài đặt trên hình 5.17 có thể là tập (*set*) hay danh sách (*list*).




```
   class Person
   {
   public:
      Person();
      ~Person();
   private:
      UnboundedSetByReference      <Person>
person;
   };
```

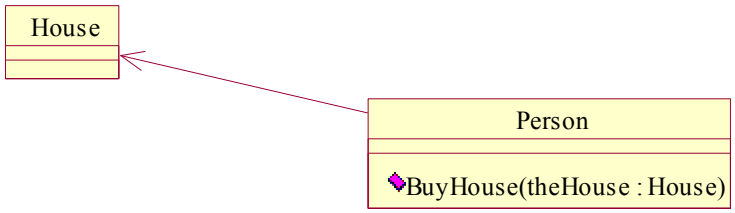
Hình 5.17 Quan hệ đệ qui

5.6.2 - Quan hệ phụ thuộc

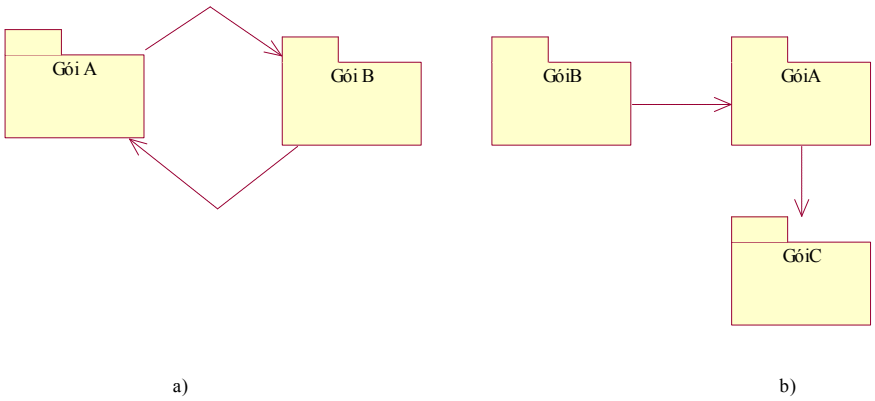
Phụ thuộc cũng là quan hệ kết nối giữa hai lớp, nhưng nó khác nhau chút ít so với quan hệ kết hợp. Quan hệ phụ thuộc luôn là quan hệ một chiều, chỉ ra một lớp phụ thuộc vào lớp khác. Thể hiện quan hệ phụ thuộc trên biểu đồ bằng mũi tên gạch gạch. Hình 5.18a là thí dụ lớp Person phụ thuộc vào lớp độc lập *House*. Hình 5.18b mô tả quan hệ phụ thuộc giữa hai gói, trong đó gói A phụ thuộc vào gói B.

Hình 5.18 Quan hệ phụ thuộc

Quan hệ phụ thuộc chỉ ra rằng một lớp tham chiếu đến lớp khác. Do vậy, nếu lớp tham chiếu thay đổi sẽ ảnh hưởng đến lớp sử dụng nó. Khi phát sinh mã trình cho hai lớp này, quan hệ này không đòi hỏi bổ sung thuộc tính nào cho lớp. Tuy nhiên các lệnh của ngôn ngữ sẽ trợ giúp quan hệ khi phát sinh mã trình cài đặt. Thí dụ, ngôn ngữ C++ sẽ bổ sung lệnh gộp *#include* vào mã trình. Trong quan hệ kết hợp giữa hai lớp như trên hình 5.14, *Person* có thuộc tính *House*. Trong qun hệ phụ thuộc như hình 5.18a, vì *Person* không có thuộc tính *House* nên chúng phải có cách khác để biết về *House*, đó là (a) *House* phải là tổng thể; (b) hiện thực của lớp *House* được coi là biến riêng của thao tác trong *Person;* (c) *House* được chuyển đến thao tác của *Person* như tham số (hình 5.19).



Hình 5.19 Mở rộng tham số của thao tác



Hình 5.20 Loại bỏ quan hệ phụ thuộc vòng

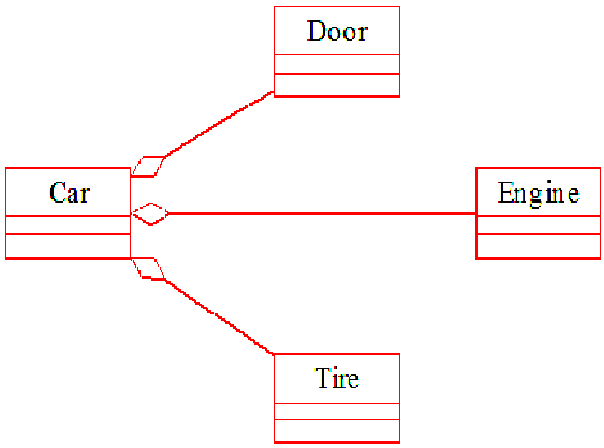
Phụ thuộc gói. Phụ thuộc còn được vẽ giữa các gói như vẽ cho các lớp (hình 5.18b). Tồn tại phụ thuộc gói từ gói A đến gói B nếu một vài lớp trong gói A có quan hệ một chiều tới gói B. Nói cách khác một vài lớp trong gói A cần biết về một vài lớp trong gói B. Gói B có thể được sử dụng lại. Khi tạo phụ thuộc gói cần tránh phụ thuộc vòng như vài lớp trong B cần thiết vài lớp trong A và ngược lại (hình 5.20a). Trong trường hợp này cả A và B không sử dụng lại được. Phụ thuộc vòng được loại bỏ bằng cách chia một gói làm hai. Thí dụ, có thể lấy các lớp trong B mà A phụ thuộc ra để lập gói mới có tên là C như hình 5.20b.

5.6.3 - Phụ thuộc tụ hợp

Tự hợp (*aggregation*) là hình thức mạnh của kết hợp. Quan hệ kết hợp giữa hai lớp có nghĩa là chúng có cùng mức, không lớp nào quan trọng hơn. Tụ hợp là qau hệ giữa toàn thể và bộ phận (*whole-part*), trong đó một lớp biểu diễn cái lớn hơn (tổng thể) còn lớp kia biểu diễn cái nhỏ hơn (bộ phận). Tụ hợp biểu diễn quan hệ *has-a*, có nghĩa rằng một đối tượng của lớp tổng hể có nhiều đối tượng của lớp thành phần. Thí dụ xe ô tô đượ lắp ráp từ bánh xe, tay lái, động cơ… Thí dụ khác, một hay nhiều căn phòng có một bức tường. Tụ hợp được thể hiện trên biểu đồ UML bằng đoạn thẳng kết thúc hạt kim cương tắng gần lớp toàn thể. Hình 5.21 cho thấy *Company* hình thành từ các *Depatment*. Hình 5.22 thể hiện quan hệ tụ hợp của xe ô tô với các phụ kiện của chúng và cái đặt trong C++.

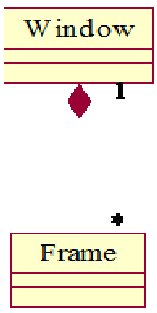
Hình 5.21 Quan hệ phụ thuộc tập hợp

Cài đặt được mô tả trên hình 5.22 cho thấy tổng thể và bộ phận được hình thành và hủy bỏ vào thời điểm khác nhau. Do vậy, quan hệ tụ hợp này còn được gọi là tự hợp bởi tham chiếu (*by refrence*). Một dạng đặc biệt của quan hệ tụ hợp nói trên là quan hệ gộp (*composition).* Đây là hình thức mạnh hơn của quan hệ tụ hợp. Trong quan hệ này thì tổng thể và thành phần được hình thành và hủy bỏ cùng thời điểm. Thí dụ quan hệ giữa cửa sổ và bàn phím trên màn hình là quan hệ gộp. Phím bấm và cửa sổ cùng được tạo lập và hủy bỏ vào cùng thời điểm. Quan hệ gộp còn được gọi là quan hệ tụ hợp bởi giá trị (*by value).* Trong UML, quan hệ gộp được biểu diễn bởi viên kim cương đen như hình 5.23.




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

Hình 5.22 Quan hệ tập hợp: tổng thể - thành phần




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

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

Hình 5.23 Quan hệ gộp

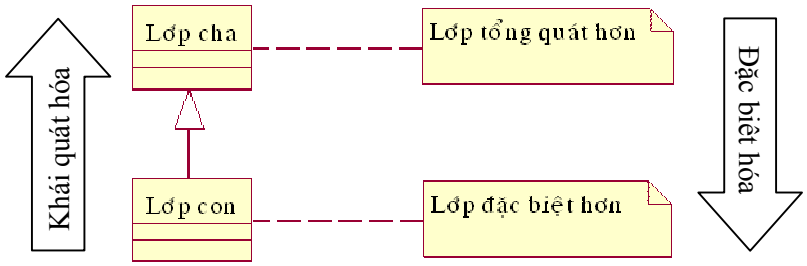
5.6.4 - Quan hệ khái quát hóa

Khái quát hóa (*gereralization*) và đặt biệt hóa (*specialization*) là hai cách nhìn về phân cấp lớp. Phân cấp lớp (*hierarchy* hay *classification*) cho khả năng quản lý độ phức tạp bằng cách xếp đặt các đối tượng vào cây lớp để làm tăng mức trừu tượng hóa.

Khái quát hó là tiến trình khá khó khăn, nó đòi hỏi khả năng trừu tượng cao để có được phân cấp tối ưu. Cây của lớp không phát triển từ gốc mà được hình thành từ lá của nó vì lá thuộc về thế giới thực. Khái quát hóa gộp các thành phần chung của tập lớp để hình thành lớp tổng quát hơn và nó được gọi là lớp cha.

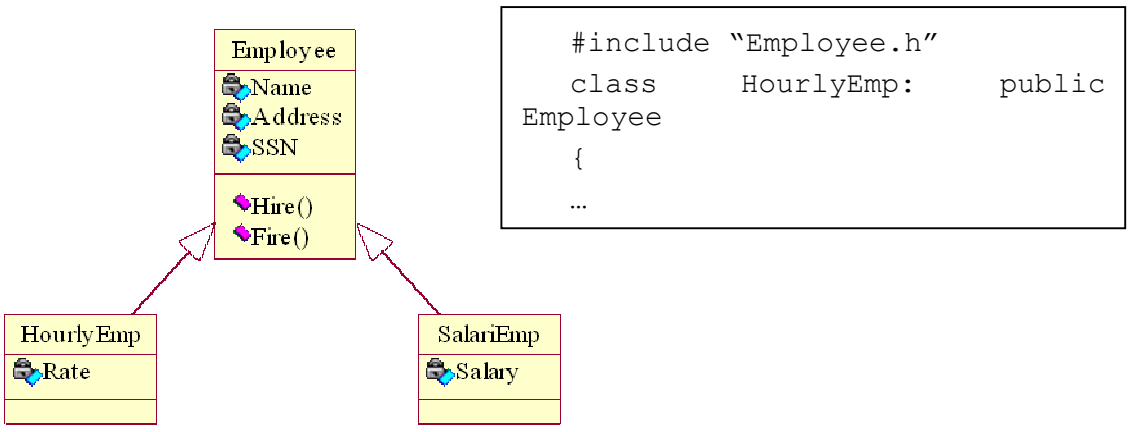
Đặt biệt hóa cho phép thu thập các đặt trưng cụ thể của tập đối tượng chưa được lớp nào nhận ra. Các đặc tính mới được biểu diễn bởi lớp mới, nó là lớp con của lớp đang tồn tại.

Tổng quát hóa và đặt biệt hóa là hai điểm nhìn ngược nhau về quan niệm phân cấp lớp, nó biểu diễn hướng mở rộng phân cấp lớp (hình 5.24).



Hình 5.24 Khái quát hóa và đặc biệt hóa

Trong UML khái quát hóa là quan hệ kế thừa giữa hai lớp. NÓ cho phép một lớp kế thừa các thuộc tính và thao tác *public* và *protected* của lớp khác.

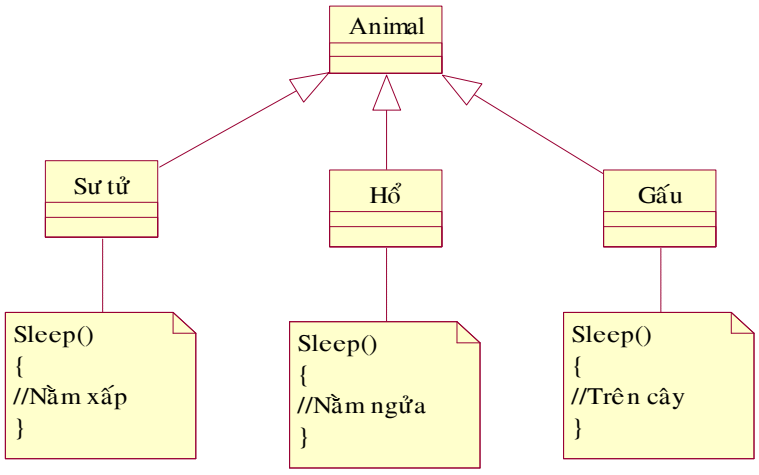


Hình 5.25 Quan hệ khái quát hóa

Thí dụ trên hình 5.25 cho thấy hai loại nhân viên trong cơ quan: nhân viên hưởng lương theo giờ và nhân viên hưởng lương theo tháng, cả hai đều kế thừa từ lớp *Employee*. Lớp *Employee* chứa các phần tử chung của hai lớp *HourlyEmp* và *SalariedEmp*. Hai lớp này kế thừa thuộc tính *Name, Adrress, SNN* và các thao tác *Hire()* và *Fire()* của lớp *Employee*. Mỗi lớp cấp thấp có thể có thuộc tính thao tác và quan hệ riêng để bổ sung vào các thành phần mà nó kế thừa. Thí dụ, lớp *HourlyEmp* có thuộc tính *Rate* và lớp *SalariedEmp* có thuộc tính *salary*.

Khi xác định khái quát hóa, ta có thể xây dựng cấu trúc kế thừa từ dưới lên hay từ trên xuống. Để xây dựng cấu trúc từ trên xuống, hãy kháo sát lớp có các kiểu khác nhau. Thí dụ bắt đầu từ lớp nhân viên (*Employee*) à nhận ra rằng cơ quan có nhiều loại nhân viên khác nhau. Do

vậy, phải hình thành các lớp (*HourlyEmp* và *SalariedEmp*) cho các loại nhân viên này. Xây dựng cấu trúc từ dưới lên là khảo sát các phần giống nhau của các lớp có sẵn để hình thành lớp mới cho các phần tử chung. Bắt đầu từ các lớp *HourlyEmp* và *SalariedEmp*, và nhận thấy rằng chúng có nhiều phần tử tương tự. Do vậy, phải hình thành lớp mới (*Employee*) để lưu trữ các thành phần chung này. Trên hình 5.25 còn có cài đặt quan hệ kế thừa giữa các lớp *Employee, HourlyEmp* và *SalariedEmp* trong ngôn ngữ C++.



Hình 5.26 Đa hình

Lớp mô tả các thuộc tính và hành vi chung cho các lớp khác trong quan hệ kế thừa là lớp trừu tượng. Lớp trừu tượng là lớp không có đối tượng. Lớp *Vihicle* trên hình 5.25 là thí dụ về lớp trừu tượng. Lớp này không có đối tượng và có ký hiệu {*abstract*} dưới tên lớp hay tên nghiêng trong biểu đồ UML. Thông thường, lớp trừu tượng có thao tác trừu tượng, đó là thao tác không có cài đặt trong lớp nơi nó được khai báo. Nếu các lớp kế thừa từ lớp trừu tượng có cài đặt các thao tá các thao tác này thì nó là lớp cụ thể (lớp có đối tượng), ngược lại nó cũng là lớp trừu tượng. Thí dụ thao tác *drive()* được cài đặt khác nhau trong lớp *Car* và *Boat,* hai lớp này là lớp cụ thể. Khi đã xây dựng phân cấp kế thừa, tiếp cận hướng đối tượng có khả năng mạnh xử lý các kiểu suy diễn khác nhau, hay ta gọi nó có khả năng đa hình (*polymorphism*). *Poly* là nhiều và *morph* là hình dạng. Đa hình là khả năng kích hoạt các thao tác khác nhau của đối tượng mà không biết gì về kiểu đối tượng. Mỗi lớp con kế thừa đặt tả thao tác của lớp cha và có khả năng sửa đổi để có được các tính chất riêng của mức trừu tượng hiện hình. Đa hình không ảnh hưởng đến nhiệm vụ phân tích, nhưng nó lại phụ thuộc vào phân tích.

Hình 5.26 là thí dụ về tính đa hình. Các con vật như *Sư tử, Hổ* và *Gấu* đều có thói quen khác nhau khi ngủ. Các lớp con cụ thể hóa thao tác *Sleep()* theo từng loại con vật.

5.6.5 - Gán đặc tính cho quan hệ

5.6.5.1 - Tính nhiều

Tính nhiều (*multiplicity*) là kết hợp biểu diễn quan hệ cấu trúc giữa các đối tượng. Tính nhiều của quan hệ cho biết bao nhiêu hiện thực của lớp có quan hệ với một hiện thực của lớp khác vào một thời điểm. Khái niệm "bao nhiêu" được gọi là tính nhiều của một "vai trò" của kết hợp. Với hệ thống đăng ký môn học tại một trường đại học, ta có các lớp *Course* và *Student.* Các lớp này có quan hệ với nhau, lớp học có sinh viên và sinh viên có các lớp học. Tính nhiều của quan hệ có thể trả lời hai câu hỏi sau: "Một sinh viên có thể học bao nhiêu môn học trong một học kỳ?" và "Bao nhiêu sinh viên có thể đăng ký một môn học?" Thí dụ một sinh viên có thể học đồng thời từ

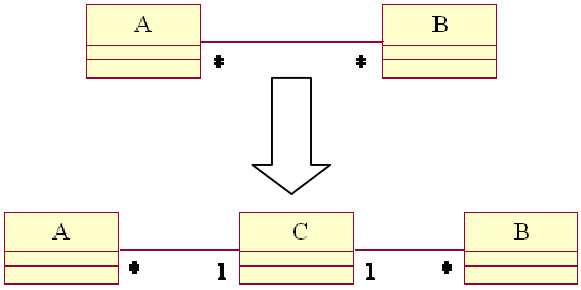
0 đế 4 môn học và một lớp có thể có từ 10 đến 20 sinh viên. Tính nhiều của qun hệ này được thể hiện trên hình 5.27.

Hình 5.27 Tính nhiều

Cần phân biệt hai khái niệm, *Cardinality* và *Tính nhiều.* Trong khi *Cardinality* là tổng số các phần tử mong đợi trong một tập, còn *tính nhiều* là đặc tả khoảng trong đó *Cardinality* được phép. Trong UML có các *tính nhiều* sau đây:

Tính nhiều	Ý nghĩa
*	Nhiều
0	Không
1	Một
0..*	Từ không đến nhiều
1..*	Từ một đến nhiều
0..1	Không hay một
1..1	Chỉ một

Trong một số ngôn ngữ lập trình, quan hệ kết hợp một chiều hay hai chiều một-nhiều dễ cài đặt hơn quan hệ nhiều-nhiều. Tuy nhiên, quan hệ kết hợp hai chiều nhiều-nhiều có thể chuyển thành quan hệ kết hợp một-nhiều như trên hình 5.28.

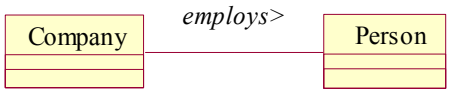


Hình 5.28 Loại bỏ quan hệ nhiều-nhiều

5.6.5.2 - Tên quan hệ

Quan hệ có thể có tên và nhiệm vụ của nó. Tên quan hệ thường là động từ hay câu ngắn mô tả tại sao có quan hệ. Thí dụ, quan hệ kết hợp giữa lớp *Person* và lớp *Company,* các câu hỏi dành cho quan hệ này là: "Tại sao lại có quan hệ này?", "*Person* là khách hàng, nhân viên hay ông chủ?". Ta có thể đặt tên quan hệ là *employs* để chỉ rõ tại sao lại có quan hệ này (hình 5.29). Tên quan hệ là tùy chọn và được đặt trên đường vẽ quan hệ, nó chỉ được sử dụng khi quan hệ giữa hai lớp không rõ ràng.

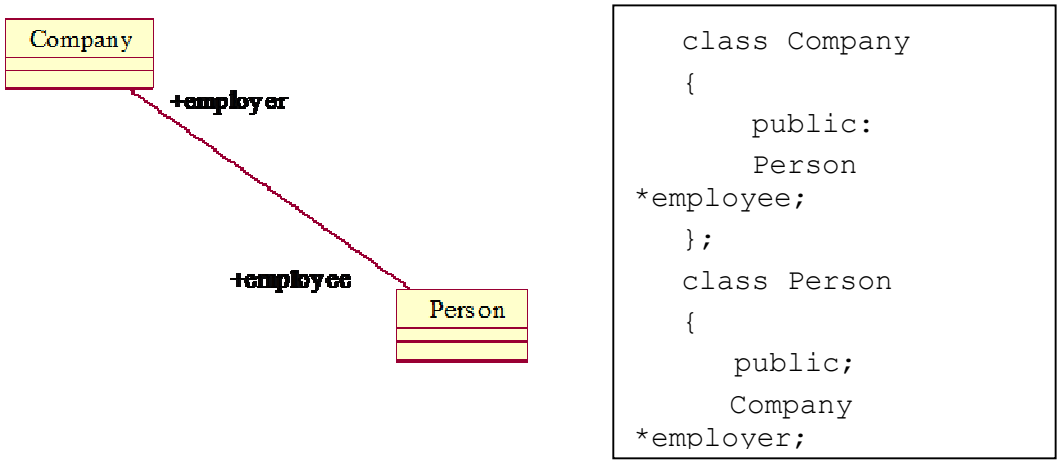
UML còn cho khả năng đặt hướng quan hệ bằng tam giác nhỏ. Để cho đơn giản, đôi khi tam giác nhỏ này được thay thế bằng ký tự <hay> có sẵn trong mọi bộ phông chữ. Thí dụ với quan hệt rên, ta phải nói là "*Company employs the person*", không thể nói ngược lại.



Hình 5.29 Tên quan hệ

5.6.5.3 - Nhiệm vụ của lớp

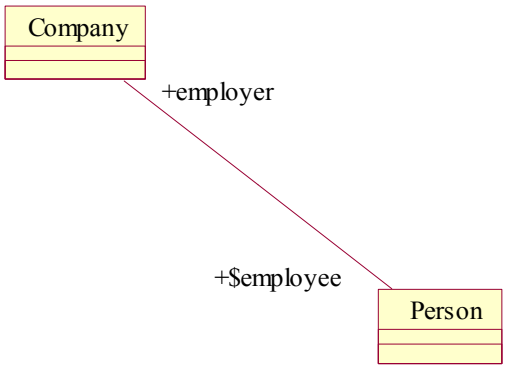
Tên nhiệm vụ có thể được sử dụng trong các quan hệ kết hợp và tổ hợp thay cho tên quan hệ. Mỗi quan hệ nhi phân có hai niệm vụ, mỗi nhiệm vụ ở đầu cuối trên quan hệ trên biểu đồ. Nhiệm vụ mô tả lớp này "nhìn" lớp khác thông qua quan hệ kết hợp như thế nào. Hãy xem xét thí dụ quan hệ giữa *Company* và *Person* trên hình 6.30, có thể nói rằng *Person* đóng vai trò *employee* khi có quan hệ với *Company*. Tên nhiệm vụ là danh từ hay câu ngắn và được đặt gần biểu tượng lớp. Thông thường việc sử dụng tên quan hệ hay sử dụng tên nhiệm vụ là độc lập nhau. Tương tự tên quan hệ, tên nhiệm vụ là tùy ý và thường được sử dụng khi quan hệ giữa hai lớp là không được thể hiện rõ ràng trên biểu đồ.



Hình 5.30 Nhiệm vụ của quan hệ

5.6.5.4 - Quan hệ tĩnh

Khi phát sinh mã trình thì thuộc tính được hình thành cho quan hệ kết hợp và tổ hợp. Thuộc tính tĩnh sẽ được chia sẽ cho mọi hiện thực lớp. Nếu đặc nhiệm vụ của quan hệ là tĩnh thì trên biểu đồ lớp sẽ có thêm dấu $ trước nó. Thí dụ trên hình 5.31 cho thấy nhiệm vụ *emplyee* là tĩnh.



Hình 5.31 Quan hệ tĩnh

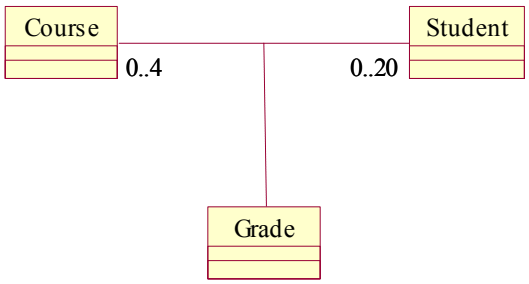
5.6.5.5 - Quan hệ friend

Giả sử ta có quan hệ *friend* giữa hai lớp *Supplier* và *Client*. Quan hệ này chỉ ra lớp *Client* có khả năng thâm nhập các thành phần không phải *public* của lớp *Supplier*. Các quan hệ kết hợp, tập hợp, phụ thuộc và tổng quát hóa đều có thể có đặc tính *friend*. Mã nguồn của lớp *Supplier* sẽ chứa logic cho phép lớp *Client* có phạm vị là *friend.*

Thí dụ, quan hệ kết hợp một chiều từ lớp *Person* đến lớp *Company* có đặt tính *friend* thì khi phát sinh mã nguồn cho *Company*, tệp .h sẽ chứa dòng lệnh *friend class Person.* Có nghĩa rằng lớp *Person* có khả năng xâm nhập các thành phần với tính chất không phải là *public* của lớp *Company.*

5.6.5.6 - Phần tử liên kết

Phần tử liên kết, còn gọi là lớp kết hợp, là nơi lưu trữ thuộc tính liên quan đến kết hợp. Thí dụ có quan hệ giữa hai lớp *Student* và *Course.* Vậy thuộc tính *Grade* sẽ được đặt ở đâu? Nếu đặt trong lớp *Student* thì phải bổ sung thuộc tính vào lớp *Student* cho từng môn học mà sinh viên đăng ký học. Nếu đặc nó trong lớp *Course* thì phải bổ sung thuộc tính cho mỗi sinh viên đăng ký môn học. Vấn đề này có thể giải quyết bằng tạo lập ra lớp mới, đó là lớp kết hợp (*association*). Thuộc tính *Grade* liên quan đến liên kết giữa hai lớp hơn là với từng lớp, do vậy đặt chúng vào lớp mới này. Ký pháp của phần tử liên kết trong UML như trên hình 5.32.



Hình 5.32 Phần tử liên kết

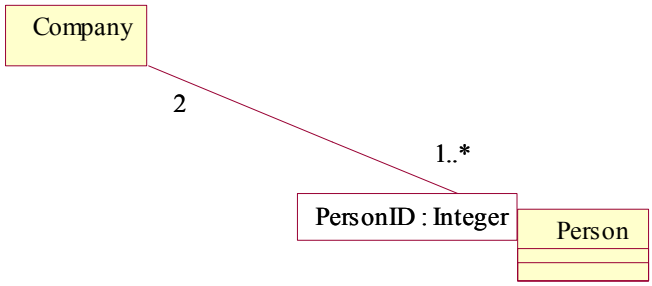
Cài đặt quan hệ này bằng C++ có thể như sau:

```
class Course; class Student;
class Grade
{
…
private:
    Course* the_course;
Student* the_student;
};

#include “Grade.h”
class Course
{
…
private:
    Grade* the-grade;
};
```

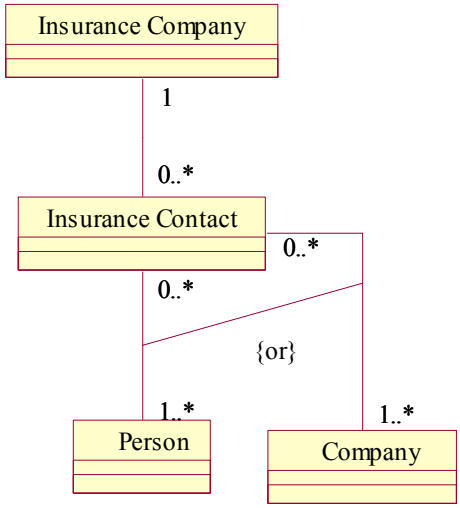
```
#include “Grade.h”
class Student
{
…
private:
    Grade* the_grade;
};
```

5.6.5.7 - ***Hạn chế phạm vi quan hệ (Qualifers)***

Qualifier được sử dụng để hạn chế phạm vi sử dụng của quan hệ kết hợp. Thí dụ ta có quan hệ kết hợp giữa lớp *Person* và lớp *Company*. Giả sử ta phải mô hình hóa vấn đề như sau: với nhân viên có số chỉ danh *PersonID* cho trước, chỉ tồn tại quan hệ giữa họ với hai công ty. Biểu đồ mô hình có thể như trên hình 5.33.



Hình 5.33 Hạn chế phạm vi quan hệ



Hình 5.34 Quan hệ kết hợp or

5.6.5.8 - ***Quan hệ kết hợp or***

Trong một số mô hình không phải mọi tổ hợp là hợp lệ. Hãy quan sát thí dụ mô hình trên hình 5.34. Một người (*Person*) có thể ký hợp đồng bảo hiểm với công ty bảo hiểm (*Insurrance Contract*) với công ty bảo hiểm, tuy nhiên người và công ty không thể có cùng hợp đồng bảo hiểm. Để giải quyết vấn đề này ta sử dụng quan hệ kết hợp *or*. Đó là quan hệ ràng buộc giữa hai

hay nhiều kết hợp. Nó chỉ ra rằng các đối tượng của lớp chỉ có thể tam gia vào một liên kết vào một thời điểm. Quan hệ kết hợp *or* được thể hiện bằng đường nối gạch giữa các kết hợp và đặc tả *{or}*.

5.7 CƠ CHẾ DUY TRÌ ĐỐI TƯỢNG

Trong pha phân tích ta đã chi tiết các thông tin cần duy trì (*persistence*) còn cơ chế duy trì được xác định trong pha thiết kế. Các đối tượng có thể được duy trì trong ROM, tệp trên đĩa cứng, tạo ra các cây *B-tree* hay sử dụng một quản trị CSDL thương mại nào đó. Khi đã quyết định sử dụng hệ quản trị CSDL thì phải quyết định sử dụng hệ quản trị CSDL quan hệ (RDBMS) hay hệ quản trị CSDL đối tượng (ODBMS). Lợi thế của sử dụng ODBMS là không cần chuyển đổi các đối tượng sử dụng sang bảng và ngược lại. Tuy nhiên cho đến nay ODBMS chưa đạt đến độ chín muồi, do vậy nó chưa chạy trơn tru như RDBMS [LIBE98]. Ngọc lại, công nghệ CSDL quan hệ đã được kiểm nghiệm, có nhiều công ty hỗ trợ phát triển và quản trị các ứng dụng CSDL quan hệ lớn.

Sự duy trì là khả năng khôi phục đối tượng về trạng thái trước đó sau khi tắt máy hay ra khỏi hệ thống. Đó là khả năng duy trì đối tượng sẵn sàng ngay cả khi chương trình đã kết thúc, và đối tượng của chương trình khác vẫn có khả năng xâm nhập đến chúng. Đặc biệt, trạng thái của đối tượng được duy trì sao cho có thể tạo lập lại nó sau này trong chương trình khác.

5.7.1.1 - Ánh xạ một-một hay sử dụng Blob

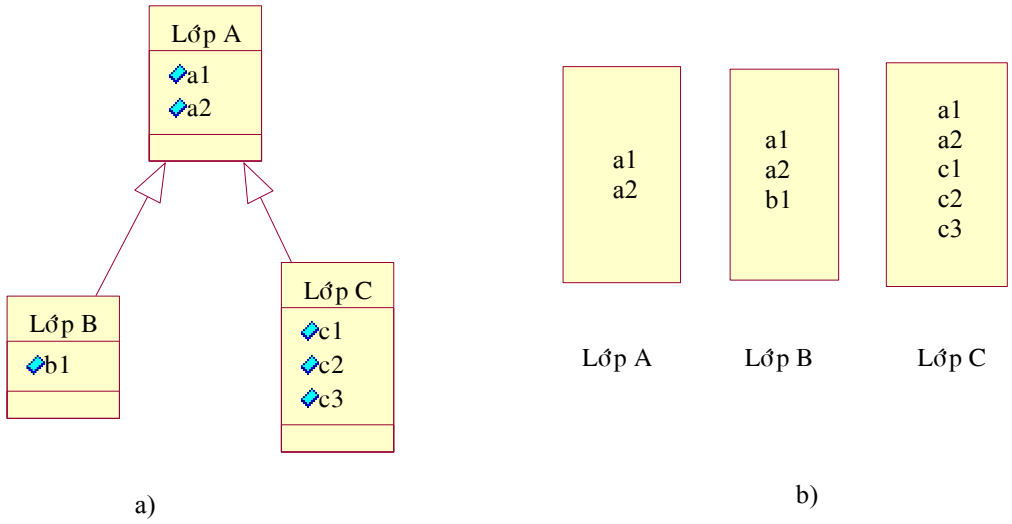
Một ánh xạ duy trì là ánh xạ mỗi thuộc tính của đối tượng vào một cột bảng CSDL. Trong mô hình ánh xạ một-một, mỗi bản ghi được hình thành từ thiết lập giá trị cột tương ứng với thuộc tính cụ thể bởi giá trị của thuộc tính đó. Với nhiều thuộc tính, ta phải chuyển đổi giá trị để phù hợp với yêu cầu CSDL. Thí dụ, cách lưu ngày tháng trong đối tượng và trong CSDL là khác nhau. Kỹ thuật này làm giảm đáng kể hiệu năng của hệ thống. Một giải pháp khác là lưu trữ toàn bộ đối tượng như dãy nhị phân của các *byte* (được gọi là *Blob*). Các giá trị khóa tìm kiếm sẽ được lưu như thuộc tính trong cột. Không thể xử lý, sắp xếp dữ liệu trong CSDL loại này, nhưng có thể truy vấn *Blob* nhờ các khóa để phục hồi lại đối tượng trong ứng dụng. Kỹ thau65t này gây ra nhiều khó khăn cho việc phát triển và bảo trì. Do vậy nhiều người phát triển hệ thống không sử dụng *Blob*.

Trong C++ ta sử dụng các con trỏ để đặc tên đối tương trong mô tả đối tượng khác. Việ này được thực hiện để biểu diễn các quan hệ kết hợp. Khi ánh xạ vào bảng CSDL, thì việc đặc tên này là do chỉ danh đối tượng (*Object Identifiers – OID*) đảm nhiệm. *OID* thường được phat sinh trong môi trường duy trì khi đối tượng được lưu trữ lần đầu vào CSDL.

OID được cài đặt như kiểu dữ liệu CSDL (số *long* hay xâu dài từ 6 đến 12 ký tự), về mặt lý thuyết thì nó phải là số duy nhất trong không gian và thời gian. Việc tạo ra *OID* duy nhất tuyệt đối là vô cùng khó khăn. *Microsodt* đã sử dụng thuật toán phức tạp để phát sinh số duy nhất dài 128 bit với tên *Chỉ danh duy nhất toàn cầu (Globally Unique Identifiers – GUID). GUID* được sử dụng làm chỉ danh duy nhất cho các lớp, giao diện hay bất kỳ đối tượng nào khác. Tuy nhiên thuật toán này còn có vấn đề trong thiết kế. Mỗi *GUID* bao gồm 60 bit để chứa số biểu diễn khoảng thời gian 10ns đếm từ 00:00:00,00 ngày 15 tháng 10 năm 1582. Đồng hồ này sẽ bị tràn vào năm 3400, do vậy sẽ xuất hiện vấn đề Y3,4K vào thiên niên kỷ thứ tư.

5.7.1.2 - Biểu diễn kế thừa

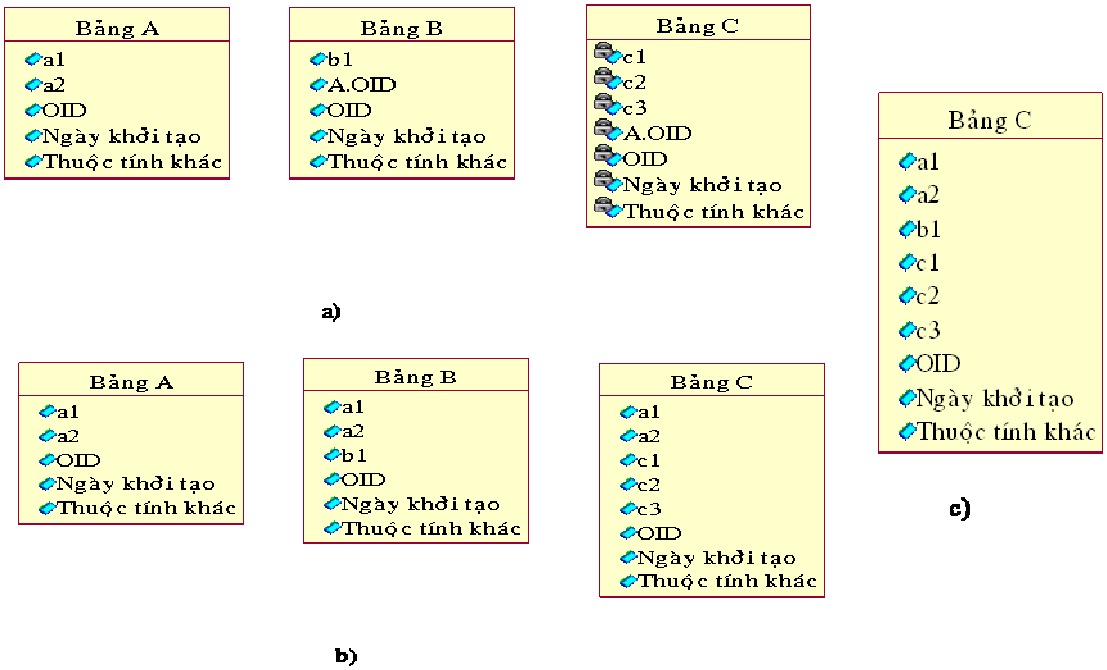
Thí dụ về quan hệ kế thừa được biểu diễn trong biểu đồ trên hình 5.35a. Biểu đồ bao gồm lớp cơ sở A và hai lớp kế thừa B và C. Hình 5.35b chỉ ra xếp đặt chúng trong bộ nhớ.



Hình 5.35 Thí dụ kế thừa lớp

Chú ý rằng thuộc tính định nghĩa trong A (A.a1 và A.a2) sẽ là một phần của đối tượng B và đối tượng C. Vậy, ta phải ánh xạ lớp cơ sở và các lớp kế thừa vào bảng CSDL như thế nào? Có ba lược đồ để lựa chọn như sau:

Lưu toàn bộ các thuộc tính của lớp A vào một bảng, các thuộc tính định nghĩa trực tiếp trong B được lưu vào bảng thứ hai, còn bảng thứ ba chứa các thuộc tính định nghĩa trực tiếp trong C. Hình 5.36a mô tả các bảng CSDL cho lớp A, B và C theo quan điểm này. Chú ý rằng *OID* của lớp cơ sở đượcsử dụng như *khóa ngoài (foreign key)* trong các bảng biểu diễn lớp suy diễn.



Hình 5.36 Biểu diễn kế thừa

Khóa: Mỗi hàng trong bảng chỉ có danh duy nhất, nó bao gồm một hay nhiều trường trong hàng. Trường sử dụng làm chỉ danh cho hàng được gọi là khóa.

Khóa ngoài: Các bảng trong CSDL quan hệ có quan hệ với nhau. Một giải pháp tạo quan hệ giữa hai bảng A và bảng B là bảng B chứa khóa của bảng A, hay nói cách khác bảng B có khóa ngoài.

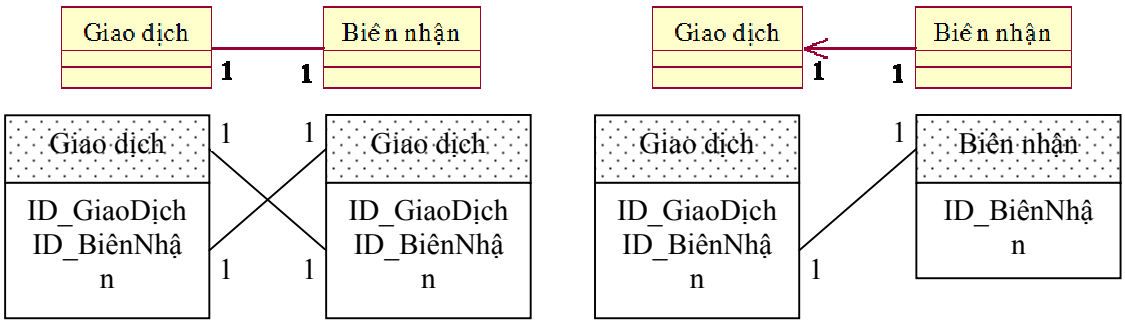
Theo cách tiếp cận này thì để tạo lập lại đối tượng sau này ta phải thực hiện thao tác kết nối (*join*). Lợi thế của nó là làm giảm đáng kể không gian lưu trự nhưng lại làm giảm tốc độ thực hiện chương trình.

Lưu trữ các thuộc tính được định nghĩa trực tiếp trong lớp B vào một bảng, các thuộc tính định nghĩa trực tiếp trong C được lưu vào bảng thứ hai, các thuộc tính định nghĩa trong lớp A được lưu vào cả hai bảng nói trên. Hình 5.36b mô tả cách tiếp cận này. Không cần thao tác kết nối ở đây để tạo lập đối tượng. Giải pháp này đòi hỏi ít không gian lưu trữ xo với bảng phẳng và chương trình chạy nhan h hơn so với giải pháp trên.

Lưu trữ toàn bộ các thuộc tính của lớp A, B, C vào cùng một bảng lớn, cách tiếp cận này được gọi là tiếp cận bảng phẳng. Thí dụ, bảng phẳng của các lớp A, B, C được mô tả tong hình 5.36c. Các thuộc tính không có ý nghĩa trong lớp kế thừa được gán *NULL*. Giải pháp này không đòi hỏi thao tác kết nối để tạo lập lại đối tượng, nó đòi hỏi nhiều không gian hơn để lưu trữ, nhưng chương trình có thể chạy nhanh hơn.

5.7.1.3 - Biểu diễn quan hệ

Việc biểu diễn quan hệ giữa hai đối tượng phụ thuộc vào tính nhiều của quan hệ. Tính nhiều của quan hệ được tìm thấy trong mô hình đối tượng và phải được cài đặt trong CSDL. Có hai cách ánh xạ quan hệ giữa các đối tượng vào CSDL quan hệ. Theo cách tiếp cận thứ nhất thì quan hệ giữa hai đối tượng được ánh xạ vào CSDL bằng cách sử dụng chỉ danh *OID* của một đối tượng như giá trị của cột trong bảng CSDL. Theo cách tiếp cận thứ hai thì sử dụng bảng tách biệt để lưu trự thông tin quan hệ. Quyết định sử dụng giải pháp nào phụ thuộc vào tính nhiều của quan hệ và quan hệ đó là một chiều hay hai chiều.

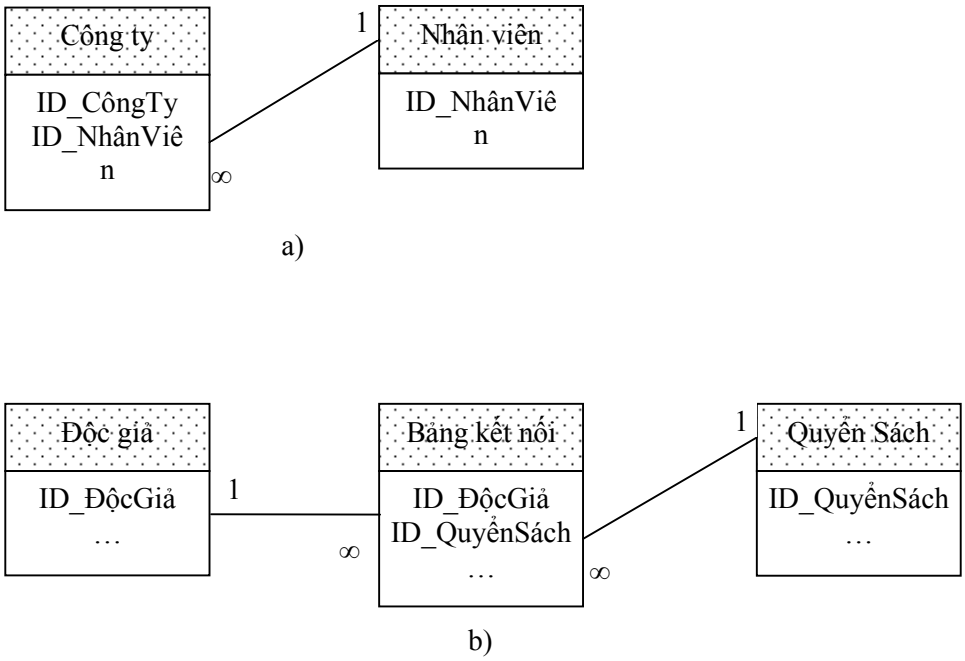


Hình 5.37 Quan hệ một-một

Quan hệ một-một: Để biểu diễn quan hệ một-một hai chiều thì phần định nghĩa bảng cho mỗi đối tượng phải chứa *OID* đối tượng khác như con trỏ đến thự thể đối tượng đó. Thí dụ, giả sử ta có đối tượng *Giao dịch* và đối tượng *Biên nhận*. Mỗi *Giao dịch* có một biên nhận và mỗi *Biên nhận* biểu diễn một *Giao dịch*. Nếu ta mô hình hóa *Biên nhận* như một bản ghi và *Giao dịch* như một bản ghi khác thì phải gắn ID của *Biên nhận* vào bản ghi *Giao dịch* và ID của *Giao dịch* vào bảng ghi *Biên nhận* (hình 5.37a).

Với quan hệ một-một một chiều, đối tượng đang bị trỏ tới sẽ có *OID* của đối tượng khác. Thí dụ đối tượng *Biên nhận* cần tìm *Giao dịch*, nhưng không làm ngược lại. Do vậy, ta có quan hệ một-một một chiều, *Biên nhận* sẽ có ID của *Giao dịch* (hình 5.37b).

Quan hệ một-nhiều. Quan hệ một-nhiều mộ chiều được quan tâm nhiều hơn. Giả sử ta có *TênSách* và ta có rất nhiều *QuyểnSách, TênSách* trỏ đến từng đối tượng của *QuyểnSách*. Trong trường hợp này, đối tượng *QuyểnSách* sẽ có *OID* của đối tượng *TênSách*. Bây giờ có truy vấn CSDL để có quyển sách mà có tên trong trường của nó (hình 5.38a).



Hình 5.38 Quan hệ một-nhiều

Trong quan hệ hai chiều nhiều-nhiều ta không thể gán *OID* của đối tượng này làm giá trị của cột trong đối tượng khác. Ta phải sử dụng bảng riêng với tên gọi là bảng kết nối (*join table*). Thí dụ, rất nhiều *ĐộcGiả* mượn một *QuyểnSách* và mỗi *ĐộcGiả* mượn nhiều *QuyểnSách*. Ta phải tạo bảng kết nối, nó kết nối *ĐộcGiả* vào *QuyểnSách* sao cho mỗi đầu vào sẽ tương ứng một *ID_ĐộcGiả* và một *QuyểnSách* (hình 5.38b).

5.8 THỰC HÀNH

5.8.1 - Sử dụng Rational Rose

5.8.1.1 - Tạo lập và hủy bỏ biểu đồ lớp

Trong *Rose*, biểu đồ lớp được lập trong khun nhìn logic (*logical view*). Số lượng biểu đồ lớp được thành lập phụ thuộc vào nhu cầu mô tả trọn vẹn bức tranh hệ thống. Khi tạo lập, mô hình mới, biểu đồ lớp *Main* được hình thành ngay dưới khun nhìn logic. Thông thường, biểu đồ lớp này chứa các gói lớp trong mô hình. Ta có thể lập biểu đồ lớp khác trong gói hay ngay dưới khung nhìn logic.

Tạo lập biểu đồ lớp mới như sau:

1. Nhấn phím phải trên khung nhìn logic trong browser
2. Chọn *New>Class diagram*

3. Nhập tên mới cho biểu đồ lớp vừa lập
4. Nhấn đúp trên biểu đồ lớp mới lập trong browser để mở chúng.

Sau khi tạo lập biểu đồ lớp, bước tiếp theo là bổ sung lớp vào mô hình. Trong quá trình xây dựng, đôi khi cần hủy bỏ lớp trong biểu đồ hay hủy bỏ chính biểu đồ lớp mà ta vừa lập. Khi hủy bỏ biểu đồ lớp, các lớp trong biểu đồ vẫn tồn tại trong browser và trong các biểu đồ khác.

Hủy bỏ phần tử biểu đồ lớp như sau:

1. Chọn phần tử trên biểu đồ
2. Nhấn phím *Delete*

Hủy bỏ phần tử mô hình khỏi mô hình như sau:

1. Chọn phần tử trên biểu đồ
2. Nhấn phím *Ctrl+D*.

Hủy bỏ biểu đồ lớp như sau:

1. Nhấn phím phải trên biểu đồ lớp trong browser
2. Chọn thực đơn *Delete*.

5.8.1.2 - Bổ sung lớp thông thường vào biểu đồ

Có nhiều cách bổ sung lớp vào biểu đồ như sử dụng phím trên thanh công cụ, browser hay thực đơn của Rose. Trình tự bổ sung lớp mới vào biểu đồ như sau:

1. Chọn phím *Class trên thanh công cụ*
2. Nhấn phím chuột bất kỳ đâu trong biểu đồ lớp, lớp mới có tên mặc định *NewClass*
3. Nhập tên mới cho lớp vừa tạo ra. Lớp mới được tự động cập nhật vào browser trong khung nhìn logic.

Ta có thể bổ sung lớp có sẵn trong mô hình vào biểu đồ lớp bằng cách di lớp từ browser vào biểu đồ lớp đang mở.

5.8.1.3 - Bổ sung lớp tham số

1. Lớp tham số là kiểu đặc biệt của lớp, nó được bổ sung vào biểu đồ theo các bước sau:
2. Chọn phím *Parameterized Class* trên thanh công cụ
3. Nhấn chuột bất kỳ đâu trong biểu đồ lớp
4. Nhập tên lớp.

5.8.1.4 - Đặt đối số cho lớp tham số

Đối số của lớp tham số được hiển thị trong chữ gạch-gạch, trên biểu đồ lớp. Đối số có thể là lớp khác, kiểu dữ liệu hay biểu thức hằng. Có thể bổ sung bao nhiêu đối số tùy ý. Trình tự bổ sung đối số như sau:

1. Mở cửa sổ đặc tả lớp
2. Chọn bảng *Detail*

3. Nhấn phím phải bất kỳ đâu trong vùng *Formal Arguments*
4. Chọn thực đơn *Insert*
5. Nhập tên đối số
6. Nhấn chuột trên cột *Type* để hiển thị danh sách kiểu đối số, chọn kiểu phù hợp.
7. Nhấn chuột trên cột *Default Value* để nhập giá trị mặc định nhưng việc này không bắt buộc.

Bãi bỏ đối số theo trình tự sau:

1. Mở cửa sổ đặc tả lớp
2. Chọn bảng *Detail*
3. Nhấn phím phải trên đối sẽ bãi bỏ
4. Chọn thực đơn *Delete*

5.8.1.5 - Bổ sung lớp hiện thực (Instantiated Class)

Bổ sung lớp hiện thực như sau;

1. Chọn phím *Instantiated Class* trên thanh công cụ
2. Nhấn bất kỳ đâu trong biểu đồ
3. Nhập tên lớp với đối trong dấu <>.

5.8.1.6 - Bổ sung lớp tiện ích (Class Utility)

Lớp tiện ích lớp là tập hợp các thao tác để lớp kah1c sử dụng. Bổi sung lớp tiện ích theo các bước như sau:

1. Chọn phím *Class Utility* trên thanh công cụ
2. Nhấn bất kỳ đâu trong biểu đồ để vẽ lớp mới
3. Nhập tên lớp.

5.8.1.7 - Bổ sung lớp tiện ích tham số

Lớp tiện ích tham số là mẫu để tạo lớp tiện ích. Bổ sung lớp tiện ích tham số như sau:

1. Chọn phím *Parametrized Class Utility* trên thanh công cụ
2. Nhấn bất kỳ đâu trong biểu đồ để vẽ lớp mới.
3. Nhập tên lớp.

5.8.1.8 - Bổ sung lớp tiện ích hiện thực

Lớp tiện ích hiện thực được bổ sung vào biểu đồ như sau:

1. Chọn phím *Instantiated Paremetrized Class Utility* trên thanh công cụ
2. Nhấn bất kù đâu trong biểu đồ để vẽ lớp mới
3. Nhập tên lớp.

5.8.1.9 - Bổ sung Metaclass

Bổ sung *metaclass* như sau:

1. Bổ sung lớp vào biểu đồ lớp hay browser như mô tả trên
2. Mở cửa sổ đặc tả lớp
3. Chọn *Metaclass* trong cử sổ *Type* của hộp thoại
4. Nhấn *OK*.

5.8.1.10 - Đặc tả lớp

Thông qua ửa sở đặc tả lớp (*Class Specification*) ta có đặc các thuộc tính cho lớp như *stereotype,* phạm vi và lưu trữ lớp (*persistance*). Mở cửa sổ đặc tả như sau:

1. Nhấn phím phải trên lớp trong biểu đồ lớp
2. Chọn thực đơn *Open Specification.*

Tên lớp. Mỗi lớp trong mô hình có tên duy nhất. Thông thường, tên lớp là danh từ đơn, tên lớp không có dấu cách (vì nhiều ngôn ngữ lập trình không trợ giúp dấu cách trong tên lớp). Tên lớp thường ngắn gọn, không phân biệt chữ hoa và chữ thường. Việc lựa chọn loại ký tự phụ thuộc vào từng hãng phần mềm.

Stereotype. Gán *stereotype* cho lớp như sau:

1. Mở cửa sổ đặc tả lớp
2. Chọn *stereotype* từ hộp danh sách hay nhập tên mới

Hiển thị tên *stereotype* trên biểu đồ theo các bước như sau:

1. Nhấn phím phải trên lớp trong biểu đồ lớp
2. Chọn *Options > Stereotype Display >Label.* Tên *stereotype* sẽ xuất hiện trong dấu <<>> ngay trên tên lớp.

Hiển thị biểu tượng *stereotype* trên biểu đồ lớp theo các bước như sau:

1. Nhấn phím phải trên lớp trong biểu đồ lớp
2. Chọn *Options > Stereotype Display > Icon.*
3. Biểu tượng *stereotype* sẽ được hiển thị trong biểu tượng lớp, trên tên lớp.

Tắt biểu tượng *stereotype* trên biểu đồ lớp theo các bước như sau:

1. Nhấn phím phải trên lớp trong biểu đồ lớp
2. Chọn *Options > Stereotype Display >None*. Lớp vẫn có *stereotype* và nhìn thấy nó trong cửa sổ đặc tả lớp, nhưng không nhìn thấy trong biểu đồ lớp.

Phạm vi lớp. Thuộc tính *Visibility* xác định lớp có đượcấy ngoài gói của nó hay không. *Rose* có ba loại phạm vi lớp, đó là:

- *Public:* lớp được nhìn thấy từ mọi lớp khác trong hệ thống
- *Protected, Private:* Lớp chỉ có thể được nhìn thấy từ lớp *friend* hay từ trong chính lớp nó.
- *Package hay Implementation:* lớp được nhìn thấy từ lớp khác trong gói.

Để đặt phạm vi lớp ta làm như sau:

1. Nhấn phím phải trên lớp trong biểu đồ lớp
2. Chọn *Open Specification* từ thực đơn
3. Chọn đặc tính *Public, Protected, Private* hay *Implemenation.*

Đặt số phiên bản (*cardinality)* cho lớp. Trường *Cardinality* là để nhập số hiện thực mong đợi của lớp. Thí dụ nếu ta muốn có nhiều hiện thực lớp *Employee* thì nhập giá trị n cho *Cardinality* của lớp này. Lớp điều khiển thường có *Cardinality* là 1. Để đặt *Cardinality* cho lớp ta làm như sau:

1. Mở cửa sổ đặc tả lớp
2. Chọn bảng *Detail*
3. Chọn *Cardinality* từ hộp danh sách hay nhập số mới.

Đặt đặc tính lưu trữ lớp. Trong Rose có thể gán đặc tính lưu trữ cho lớp, các đặc tính này có thể là:

- *Persistent.* Lớp vẫn còn tồn tại ngay cả khi ứng dụng không thực hiện. Có nghĩa rằng thông tin trong đối tượng của lớp được lưu trữ trong CSDL hay trong khuôn mẫu khác nào đó.
- *Transient.* Thông tin trong đối tượng của lớp không được lưu trữ vào bộ nhớ ngoài.

Không được gán thuộc tính *Persistent* cho lớp tiện ích, lớp tiện ích tham số và lớp tiện ích hiện thực. Các bước gán đặc tính này như sau:

1. Mở cửa sổ đặc tả lớp
2. Chọn bảng *Detail*
3. Chọn *Persistent* hay *Transient* trong vùng *Persistent.*

Đặt đặc tính lớp trừu tượng. Lớp A là trừu tượng thì không bao giờ có đối tượng kiểu A trong bộ nhớ. Tạo lập lớp trừu tượng như sau:

1. Tạo lập lớp bằng phương pháp mô tả trên
2. Mở cửa sổ đặc tả lớp
3. Chọn bảng *Detail*
4. Đánh dấu *Abstract* trên hộp thoại.

5.8.1.11 - Tạo lập và hủy bỏ gói

Trong các chương trước ta thấy gói được xây dựng từ các khung nhìn UC, chúng chứa tác nhân, UC, biểu đồ trình tự hoặc/và biểu đồ cộng tác. Gói còn được bổ sung vào khung nhìn logic, chúng các lớp có tính chất chung. Tạo lập gói mới được hiện thực như sau:

1. Chọn phím *Pakage* trên thanh công cụ
2. Nhấn bất kỳ đâu trong biểu đồ lớp để tạo gói mới
3. Nhập tên gói.

Bổ sung gói trong browser được hiện thực như sau:

1. Nhấn phím phải chuột trên khung nhìn logic trong browser. Nếu có nhu cầu tạo gói trong gói thì nhấn phím phải chuột trên gói trong browser.

2. Chọn thực đơn *New> Pakage* để bổ sung gói mới với tên mặc định *NewPakage* trong browser.
3. Nhập tên cho gói mới.

Một khi gói được tạo, ta có thể di chuyển các phần tử mô hình hóa vào gói. Hủy bỏ gói trong biểu đồ đượ thực hiện như sau:

1. Chọn gói trong biểu đồ lớp
2. Nhấn phím *Delete*
3. Chú ý rằng gói chỉ bị loại bỏ khỏi biểu đồ lớp, nhưng vẫn còn trong browser và biểu đồ lớp khác.

Hủy bỏ gói trong mô hình được thực hiện như sau;

1. Nhấn phím phải trên gói trong browser
2. Chọn thực đơn *Delete.*

5.8.1.12 - Bổ sung thuộc tính cho lớp

Bổ sung thuộc tính vào lớp theo các bước như sau:

1. Nhấn phím phải trên lớp trong biểu đồ lớp
2. Chọn *New>Attribute*
3. Nhập tên thuộc tính theo khuôn mẫu: *Data Type = Intial Value,* thí dụ:
   Address:String
   IDNumber:Integer=0
   Trật tự kiểu dữ liệu ảnh hưởng đến phát sinh mã trình, nhưng giá trị mặc định của thuộc tính là tùy ý.
4. Để gắn thêm các thuộc tính khác hãy nhấn *Enter* và nhập trực tiếp thuộc tính mới vào lớp.

5.8.1.13 - Đặt thuộc tính

Có thể gán các đặc tính sau đây cho thuộc tính lớp: kiểu dữ liệu của thuộc tính, giá trị khởi đầu, *stereotype* và phạm vị. Để mở cửa sổ thuộc tính ta làm như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification.*

Đặt kiểu dữ liệu thuộc tính. Có thể gán kiểu dữ liệu của ngôn ngữ như *string, integer, long* hay *boolean* cho kiểu thuộc tính. Cũng có thể gắn tên lớp mới tạo lập ra trong mô hình vào kiểu thuộc tính của lớp. Trình tự gắn kiểu dữ liệu thuộc tính như sau:

1. Nhấn phím phải vào thuộc tính trong browser
2. Chọn thực đơn *Open Specification.*
3. Chọn kiểu dữ liệu từ hộp danh sách *Type* hay nhập kiêu dữ liệu mới.

Đặt stereotype cho thuộc tính. Tương tự UC và tác nhân, có thể gắn *stereotype* cho thuộc tính để phân loại chúng. Việc làm này cũng không bắt buộc trong Rose vì nó chỉ làm cho mô hình dễ đọc mà thôi.

Trình tự gắn *stereotype* cho thuộc tính theo các bước như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification*. Rose sẽ mở cửa sổ *Class Attribute Specification.*
3. Chọn *stereotype* từ hộp danh sách hay nhập *stereotype* mới.

Đặt giá trị ban đầu cho thuộc tính. Trình tự gắn giá trị ban đầu cho thuộc tính như sau đây:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification*. Rose sẽ mở cửa sổ *Class Attribute Specification.*
3. Nhập giá trị mặc định cho thuộc tính vào ô *Intial Value.*

Đặt phạm vi cho thuộc tính. Việc gán phạm vi thuộc tính được thực hiện như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification*.
3. Chọn phạm vi thuộc tính trong cửa sổ *Export Control*. Mặc định phạm vi thuộc tính sẽ là *Private*.

Đặt cách thức lưu trự thuộc tính. Cách thức lưu trữ thuộc tính mô tả thuộc tính sẽ được lưu trữ trong lớp như thế nào, chúng bao gồm ba loại: *by value, by reference* và *unspecified.* Các bước đặc cách thức lưu trữ thuộc tính như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification*. Rose sẽ mở cửa sổ đặc tả thuộc tính.
3. Chọn bảng *Detail*
4. Chọn *By Value, By Reference* hay *Unspecified.* Mặc định thuộc tính được gán *Unspecified.*

Gán thuộc tính là tĩnh. Thuộc tính tĩnh được mọi hiện thức trong hệ thống chia sẽ. Gán thuộc tính là tĩnh được thực hiện như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification*. Rose sẽ mở cửa sổ đặc tả thuộc tính.
3. Chọn bảng *Detail*
4. Đánh dấu *Static* trên hộp thoại. Rose sẽ đặt đấu $ trước tên thuộc tính trong biểu đồ lớp.

Gán thuộc tính là suy diễn. Thuộc tính suy diễn là thuộc tính được tính toán từ một hay nhiều thuộc tính khác. Gán thuộc tính là suy diễn theo các bước sau đây:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification*. Rose sẽ mở cửa sổ đặc tả thuộc tính.
3. Chọn bảng *Detail*
4. Đánh dấu *Derived* trên hộp thoại. Rose sẽ đặt đấu / trước tên thuộc tính trong biểu đồ lớp.

5.8.1.14 - Bổ sung và hủy bỏ thao tác

Tương tự thuộc tính, thao tác được bổ sung vào mô hình thông qua biểu đồ lớp, browser hay thông qua đặc tả lớp. Khi bổ sung thao tác ta cũng có thể bổ sung tài liệu cho chúng. Tài liệu thường mô tả mục đích thao tác, mô tả tham số và giá trị cho lại.

Trình tự thực hiện bổ sung thao tác như sau:

1. Nhấn phím phải trên lớp trong biểu đồ lớp
2. Chọn *New>Operation*
3. Nhập tên lớp theo khuôn mẫu như sau:

   Name(arg1:arg1 data type): Operation Return Type
   Thí dụ:
   Add(X:Integer, Y:Integer): Integer
   Print(EmployeeID:Long): Boolean
   Delete():Long
4. Nhấn *Enter* để tiếp tục nhập trực tiếp các thao tác khác vào biểu đồ lớp.

Trình tự thực hiện bãi bỏ thao tác như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Delete*

5.8.1.15 - Đặc tả thao tác

Trong đặc tả thao tác ta có thể gán tham số thao tác, kiểu cho lại và phạm vi thao tác. Để thực hiện việc này trước hết ta phải mở cửa sổ đặc tả, trình tự thực hiện như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification.*

Gán lớp trả lại cho thao tác. Lớp trả lại cho thao tác là kiểu kết quả của thao tác. Thực hiện việc này theo các bước như sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification.*
3. Chọn lớp trả lại từ hộp danh sách hay nhập kiểu trả lại mới.

Gán stereotype cho thao tác. Bốn loại *stereotype* hay sử dụng cho thao tác là: *Implemetor, Manager, Access* và *Helper.* Trình tự đặt *stereotype* cho thao tác tương tự như sau đây:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn thực đơn *Open Specification.*
3. Chọn *stereotype* từ hộp danh sách hay nhập kiểu *stereotype* mới.

Gán phạm vi cho thao tác. Bốn tùy chọn cho phạm vi thao tác là: *Public, Private, Protected* và *Pakage (Implementation)*. Đặt phạm vi cho thao tác theo trình tự sau:

1. Nhấn phím phải trên thuộc tính trong browser
2. Chọn phạm vi thao tác trong cửa sổ *Export Control.* Mặc định, phạm vi của thao tác là *Public*.

5.8.1.16 - Bổ sung đối số vào thao tác

Đối số của thao tác (hay tham số) là dữ liệu vào của thao tác. Hai loại thông tin cho đối số là tên đối và kiểu cho lại. Trong biểu đồ lớp, đối và kiểu của thao tác đặt trong dấu(). Nếu gán giá trị mặc định cho đối thì ký pháp của UML như sau:

Operation Name(arg1:arg1 data type=arg1 default value):operation return type

Trình tự thực hiện bổ sung đối cho thao tác như sau:

1. Mở cửa sổ đặc tả thao tác.
2. Chọn bảng *Detail*
3. Nhấn phím phải trên hộp *argument,* chọn thực đơn *insert*
4. Nhập tên đối số
5. Nhấn chuột trên cột *Type* và nhập kiểu đối
6. Nhấn cột *Default* và nhập giá trị mặc định nếu muốn.

5.8.1.17 - Ánh xạ thao tác thành thông điệp

Trong biểu đồ tương tác, mỗi thông điệp sẽ được ánh xạ vào một thao at1c bên trong lớp. Trình tự ánh xạ các thông điệp thành thao tác theo thứ tự sau:

1. Đảm bảo rằng đối tượng nhận thông điệp đã được ánh xạ đến lớp
2. Nhấn phím phải trên thông điệp trong biểu đồ tương tác
3. Danh sách các thao tác xuất hiện
4. Chọn thao tác trong danh sách cho thông điệp.

Loại bỏ ánh xạ thao tác thông điệp như sau;

1. Nhấn đúp trên thông điệp trong biểu đồ tương tác
2. Trong cửa sổ *Name:* xóa tên thao tác và nhập tên thông điệp mới.

Tạo lập thao tác mới cho thông điệp như sau;

1. Đảm bảo rằng đối tượng nhận thông điệp đã được ánh xạ đến lớp
2. Nhấn phím phải trên thông điệp trong biểu đồ tương tác
3. Chọn <*new operation*>
4. Nhập tên và các chi tiết khác của thao tác mới
5. Nhấn phím *OK*
6. Nhấn phím phải trên thông điệp
7. Chọn thao tác mới trong danh sách.

5.8.1.18 - Tính nhiều của lớp

Để đặt tính nhiều (*multiplicity*) cho các lớp ta làm như sau:

1. Nhấn phím phải trên một đầu cuối của quan hệ
2. Chọn thực đơn *Multiplicity*
3. Chọn *multiplicity* mong muốn
4. Lặp lại các bước 1-3 cho đầu cuối còn lại của quan hệ.

Cách khác là:

1. Mở cửa sổ đặc tả quan hệ (Nhấn phím phải trên quan hệ, chọn *Specification*)
2. Chọn bảng *Role Detail* cho đầu cuối của quan hệ

3. Thay đổi *multiplicity* thông qua trường *cardinality*
4. Lặp lại từ 1-3 cho đầu kia của quan hệ.

5.8.1.19 - Đặt tên và hướng quan hệ

Đặt tên quan hệ theo các bước như sau:

1. Chọn quan hệ, mở cữa sổ đặt tả quan hệ
2. Chọn bảng *General*
3. Nhập tên quan hệ vào cửa sổ *name*

Đặt hướng tên quan hệ như sau:

1. Mở cửa sổ đặc tả của quan hệ mong muốn
2. Chọn bảng *Detail*
3. Chọn hướng tên quan hệ bằng cửa sổ hướng tên quan hệ.

5.8.1.20 - Gán stereotype cho quan hệ

Đặt *stereotype* cho quan hệ theo các bước sau đây:

1. Mở cửa sổ đặc tả cho quan hệ
2. Chọn bảng *General*
3. Nhập *stereotype* cho quan hệ

5.8.1.21 - Gán nhiệm vụ cho quan hệ

Đặc tên nhiệm vụ theo thứ tự như sau:

1. Nhấn phím phải trên đầu cuối của quan hệ kết hợp
2. Chọn thực đơn *Role Name*
3. Nhập tên nhiệm vụ

5.8.1.22 - Phạm vi của quan hệ

Trong quan hệ kết hợp, Rose sẽ tạo các thuộc tính khi phát sinh mã tình. Sử dụng cử sổ *Export Control* trong Rose để phát sinh phạm vi thuộc tính. Phạm vi có thể là *Public, Private, Protected* hay *Pakage (Inplementation*).

Trong quan hệ hai chiều có thê đặt phạm vi cho hai thuộc tính, mỗi chúng ở một đầu quan hệ. Trong quan hệ một chiều thì chỉ cần đặt mộ đầu cho quan hệ. Phạm vi được đặc bằng chọn bảng *Role A General* hay *Role B General* trong cửa sổ đặc tả cửa quan hệ.

Đặt phạm vi quan hệ như sau:

1. Nhấn phím phải trên tên nhiệm vụ
2. Chọn *Export Control* thích ứng

5.8.1.23 - Đặt quan hệ tĩnh

Để đặt quan hệ kết hợp là tĩnh ta làm như sau:

1. Nhấn phím phải trên tên nhiệm vụ
2. Chọn thực đơn *Static*.

5.8.1.24 - Đặt quan hệ friend

Đặt quan hệ là *friend* như sau:

1. Nhấn phím phải trên quan hệ
2. Chọn thực đơn *Friend*

5.8.1.25 - Đặt Containment

Đặt tính này cho biết cách lưu trữ thuộc tính trong lớp như thế nào. Đặt tính chất chứa như sau:

1. Nhấn phím phải chuột trên quan hệ
2. Chọn *Contaiment*
3. Chọn *By Reference, By Value* hay *Unspecified.*

5.8.1.26 - Đặt thành phần liên kết

Trình tự đặt phần tử liên kết cho quan hệ:

1. Mở cửa sổ đặc tính quan hệ
2. Chọn bảng *Detail*
3. Đặt phần tử liên kết nhờ trường *Link Element.*

5.8.1.27 - Đặt giới hạn phạm vi quan hệ

Trình tự đặc giới hạn phạm vi cho quan hệ như sau:

1. Nhấn phím phải trên đầu cuối của quan hệ kết hợp
2. Chọn thực đơn *Key/Qualifier*
3. Nhập tên và kiểu *qualifier*

5.8.2 - Thí dụ: Hệ thống bán hàng (tiếp theo)

Sau khi khách hàng và người phát triển phần mềm thống nhất cho rằng biểu đồ tương tác phù hợp với yêu cầu tác nghiệp, người thiết kế quyết định nhóm các lớp thành nhóm theo *stereotype*. Vậy các gói sau này sẽ được tạo lập: *Entities, Boundaries* và *Control*. Sau đó chuyển các lớp vào nhóm tương ứng. Tiếp theo các biểu đồ lớp sẽ được lập trong mỗi gói: biểu đồ lớp *Main* hiển thị các gói, biểu đồ lớp *Enter New Order* chỉ ra lớp cho UC này.

*5.8.2.1 - **Đặt cấu hình Rose***

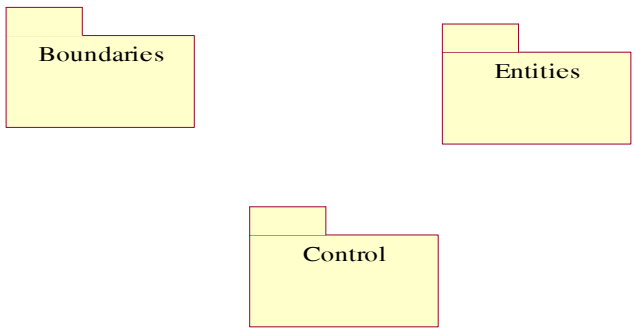
1. Chọn thực đơn *Topols>Options*
2. Chọn bảng *Diagram*
3. Đảm bảo các hộp đánh dấu sau được chọn: *Show Stereotype, Show all Attributes, Show all Operations.*
4. Đảm bảo các hộp đán dấu sau không được chọn: *Suppress Attributes, Suppress Operations.*

*5.8.2.2 - **Tạo lập gói***

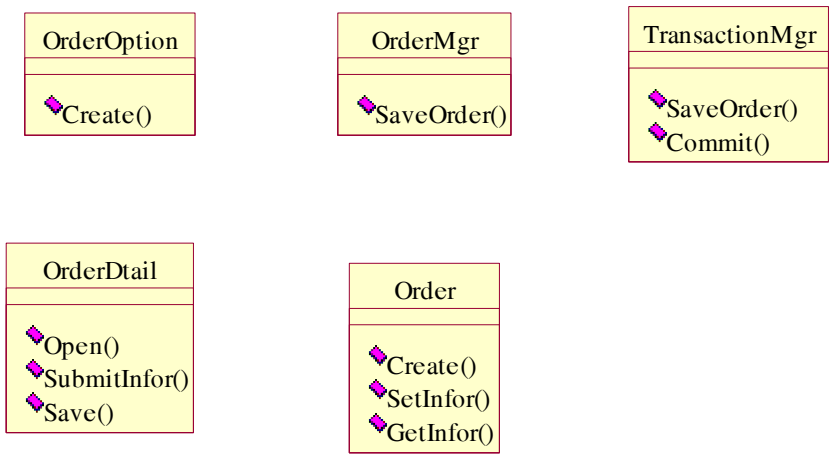
1. Nhấn phím phải trên *Logical View* trong Browse
2. Chọn thực đơn *New>Pakage*
3. Đặt tên *Entities*
4. ặp lại để tạo các gói *Boundaries* và *Control*.

*5.8.2.3 - **Tạo lập biểu đồ lớp Main***

1. Nhấn đúp trên *Main class diagram* ngay dưới *Logical View* để mở nó
2. Di gói *Entities* từ browser vào biểu đồ
3. Di các gói *Boundaries* và *Control* vào biểu đồ. Hình 5.39 là biểu đồ lớp *Main* kết quả.



Hình 5.39 Các gói



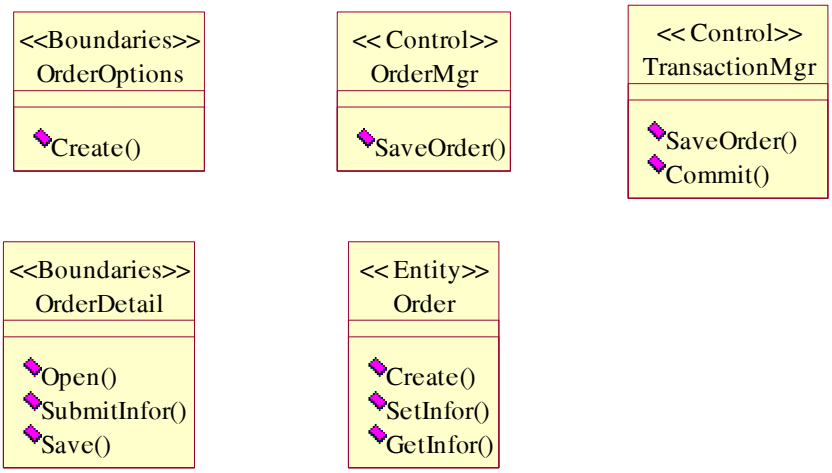
Hình 5.40 Biểu đồ lớp của Uc Entry New Order

5.8.2.4 - Tạo lập biểu đồ lớp cho mọi lớp trong UC Enter New Order

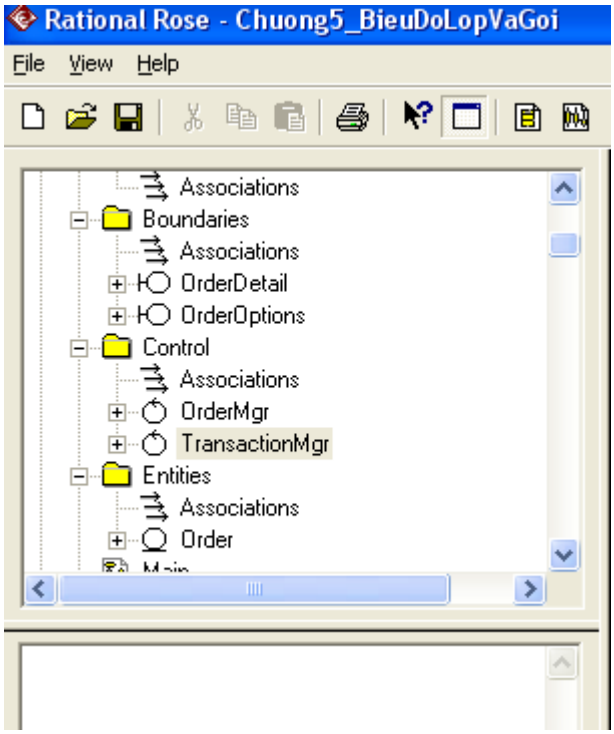
1. Nhấn phím phải trên *Logical View* trong Browse
2. Chọn thực đơn *New>Class Diagram*
3. Đặt tên mới cho biểu đồ lớp: **Add New Order**
4. Nhấn đúp trên biểu đồ lớp *Add New Order* để mở nó
5. Di các lớp *OrderOptions, OrderDetail, OrderMgr, Order* và *TransactionMgr* từ browser vào biểu đồ. Biểu đồ kết quả trên hình 5.40.

5.8.2.5 - Bổ sung Stereotype vào lớp

1. Nhấn phím phải trên lớp *OrderOptions* trong biểu đồ
2. Chọn *Open Specification*
3. Nhập **Boundaries** vào vùng *stereotype*
4. Nhấn phím *OK*
5. Nhấn phím phải trên lớp *OrderDetail* trong biểu đồ
6. Chọn *Open Specification*
7. Chọn **Boundaries** từ hộp danh sách
8. Nhấn *OK*
9. Lặp lại để gán các lớp *OrderMgr, TransactionMgr* cho *stereotype Control* và lớp *Order* cho *stereotype Entity.* Kết quả được thể hiện trên hình 5.41.



Hình 5.41 Stereotype của lớp



Hình 5.42 Các gói trong browser

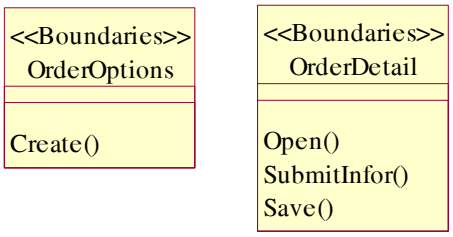
5.8.2.6 - Nhóm các lớp vào gói

1. Di lớp *OrderOptions* và *OrderDetail* trong browser vào gói *Boundaries.*
2. Di các lớp *OrderMgr, TransactionMgr* vào gói *Control*
3. Di lớp *Order* vào gói *Entities.*

5.8.2.7 - Bổ sung lớp vào các gói

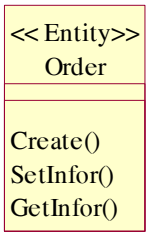
1. Nhấn phím phải vào gói *Boundaries* trong browser

2. Chọn thực đơn *New>Class Diagram*
3. Đặt tên biểu đồ mới: **Main**
4. Nhấn đúp trên *Main* để mở nó
5. Di các lớp *OrderOptions* và *OrderDetail* từ browser vào biểu đồ
6. Đóng biểu đồ. Biểu đồ kết quả trên hình 5.43.
7. Nhấn phím phải trên gói *Entities* trong browser
8. Chọn thực đơn *New>Class Diagram*
9. Đặt tên biểu đồ mới: **Main**

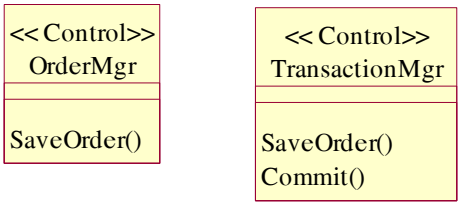


Hình 5.43 Lớp trong gói Boundaries

10. Nhấn đúp trên *Main* để mở nó
11. Di lớp *Order* từ browser vào biểu đồ
12. Đóng biểu đồ. Biểu đồ kết quả trên hình 5.44.
13. Nhấn phím phải trên gói *Control* trong browser
14. Chọn thực đơn *New>Class Diagram*
15. Đặt tên biểu đồ mới: **Main**
16. Nhấn đúp trên *Main* để mở nó
17. Di các lớp *OrderMgr* và *TransactionMgr* từ browser vào biểu đồ
18. Đóng biểu đồ. Biểu đồ kết quả trên hình 5.45.

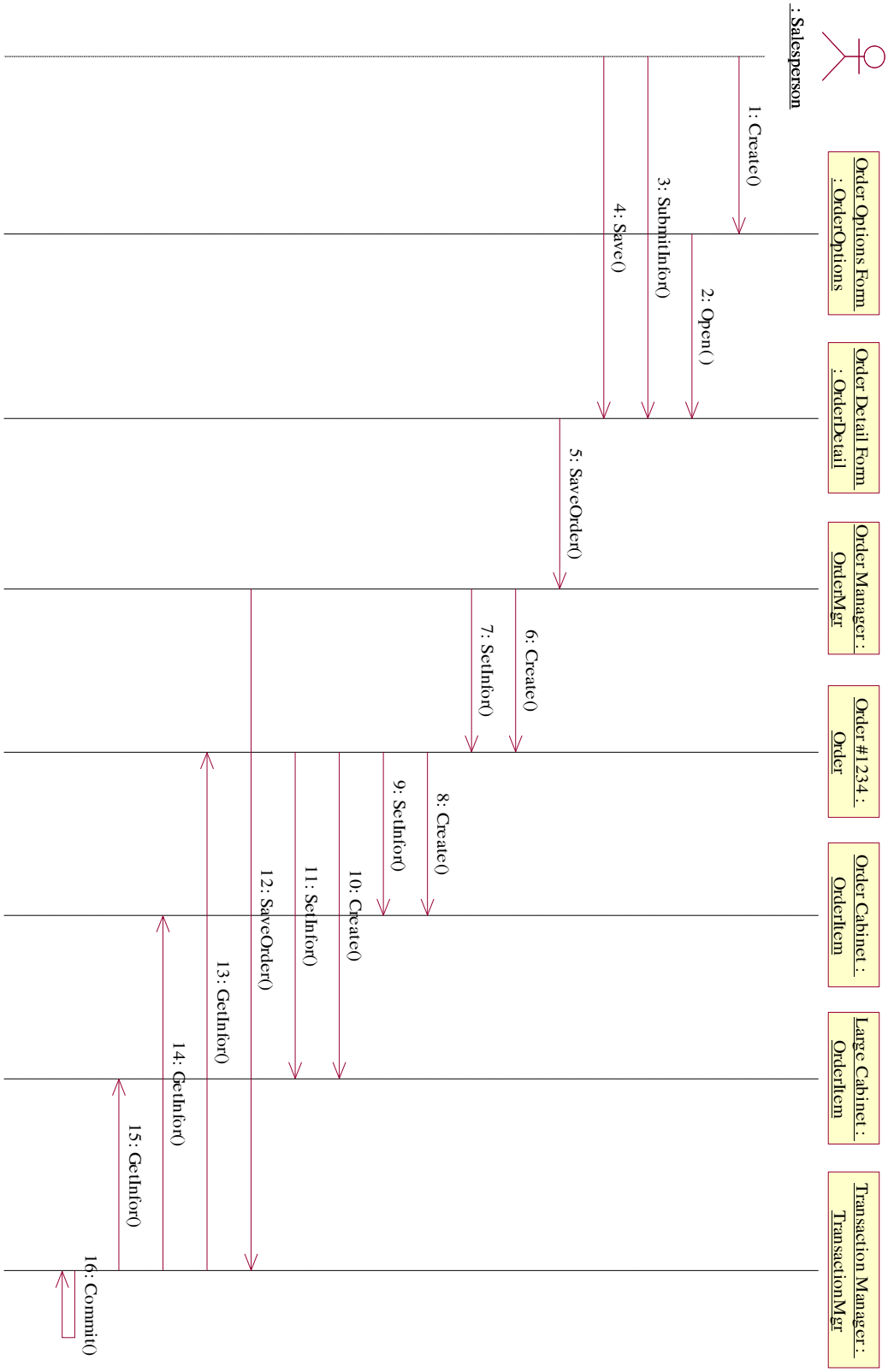


Hình 5.44 Lớp trong gói Entity



Hình 5.45 Lớp trong gói Control

Ta đã có biểu đồ lớp cho UC *Enter New Order*. Bây giờ cần bổ sung tham số, kiểu dữ liệu và kiểu giá trị cho lại vào lớp. Ta phải đi trở lại luồng sự kiện để tìm ra ac1c thuộc tính. Các thuộc tính *Order Number* và *Customer Name* sẽ được bổ sung vào lớp *Order* trong biểu đồ lớp. Vì có nhiều chi tiết trong một đơn hàng (*order items*) và mỗi đơn hàng lại có thông tin và hành vi riêng, do vậy các mục đơn hàng sẽ được mô hình hóa thành lớp, thay cho thuộc tính của *Order*. Biểu đồ trình tự sẽ được cập nhật như trên hình 5.46.



Hình 5.46 Biểu đồ trình tự

5.8.2.8 - Bổ sung thuộc tính và thao tác

Khách hàng bây giờ có nhu cầu thay đổi yêu cầu như: hệ thống có khả năng yêu cầu thay đổi đặt hàng và thời gian cung cấp hàng; bổ sung thêm nguồn hàng; thay đổi cút ít về thủ tục nhập hàng vào kho.

Tài liệu về các yêu cầu mới hay thay đổi phải được cập nhật. Nhận thấy rằng yêu cầu mới về ngày tháng ảnh hưởng đến UC đang thiết kế *Enter New Order*. Do vậy chúng được quan tâm ở đây bằng cách bổ sung hai thuộc tính mới cho lớp *Order.*

5.8.2.9 - Đặt cấu hình Rose

1. Chọn thực đơn *Tools>Options*
2. Chọn bảng *Diagram*
3. Đảm bảo các hộp đánh dấu sau được chọn: *Show Visibility, Show Stereotype, Show Operation Signatures, Show all Attributes* và *Show all Operarions*
4. Đảm bảo các hộp đánh dấu sau không được chọn: *Suppress Attributes, Suppress Operations*
5. Chọn bảng *Notation*
6. Đảm bảo hộp đánh dấu sau không được chọn: *Visibility as Icons.*

5.8.2.10 - Bổ sung lớp mới

1. Tìm biểu đồ lớp *Add New Order* trong browser
2. Nhấp đúp để mở biểu đồ
3. Chọn phím công cụ *Class* trên thanh công cụ
4. Nhấn bất kỳ đâu trong biểu đồ để vẽ lớp mới
5. Nhập tên *OrderItem* cho lớp mới
6. Gán *Entity* cho *stereotype* của lớp *OrderItem*
7. Di lớp *OrderItem* từ browser vào gói *Entities*

5.8.2.11 - Bổ sung thuộc tính

1. Nhấn phím phải trên lớp *Order*
2. Chọn *New Attribute*
3. Nhập thuộc tính mới: **OrderNumber: Integer**
4. Nhấn phím *Enter*
5. Nhập tiếp thuộc tính mới: **CustomerName: String**
6. Lặp để nhập các thuộc tính sau: **OrderDate: Date, OrderFillDate: Date**
7. Nhấn phím phải trên lớp *OrderItem*
8. Chọn *New Attribute*

9. Nhập thuộc tính mới: **ItemID: Integer**
10. Nhấn phím *Enter*
11. Nhập tiếp thuộc tính mới: **ItemDescription: String**

5.8.2.12 - Bổ sung thao tác vào lớp OrderItem

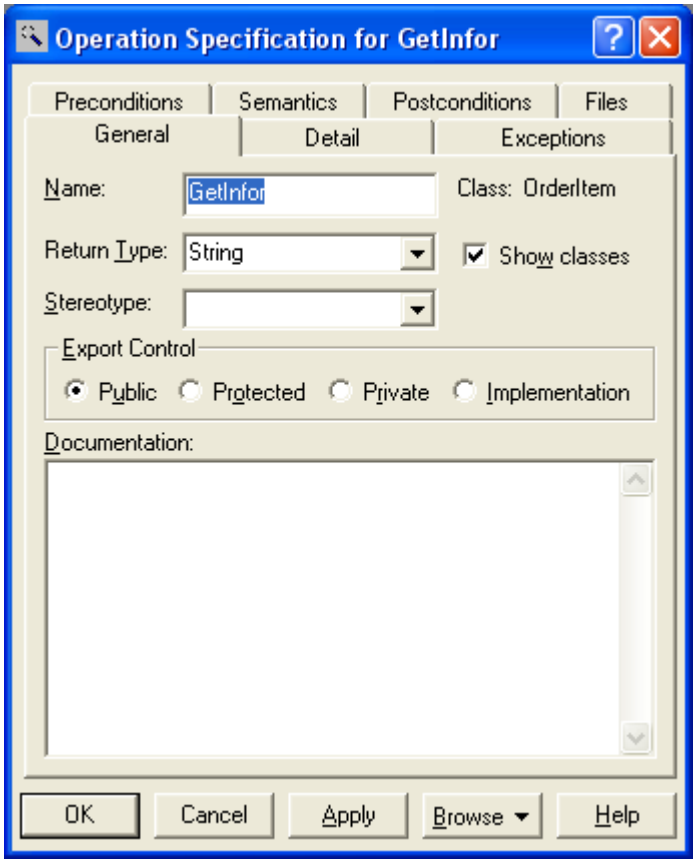
1. Nhấn phím phải trên lớp *OrderItem*
2. Chọn *New Operation*
3. Nhập thao tác mới: **Create**
4. Nhấn phím *Enter*
5. Nhập thao tác mới: **SetInfo**
6. Nhấn phím *Enter*
7. Nhập thao tác mới: **GetInfo**

5.8.2.13 - Bổ sung chi tiết thao tác trong biểu đồ lớp

1. Nhấn phím chuột trên lớp *Order* để mở nó
2. Nhấn chuột trong lớp *Order*
3. Cập nhật thao tác **Create()** thành **Create(): Boolean**
4. Cập nhật thao tác **SetInfo()** thành **Setinfo(OrderNum: Integer, Customer: String, OrderDate: Date, OrderFillDate: Date): Boolean**
5. Cập nhật thao tác **GetInfo()** thành **GetInfo():Boolean**

5.8.2.14 - Bổ sung chi tiết thao tác trong browser

1. Tìm lớp *OrderItem* trong browser
2. Nhấn trên ký tự + cạnh *OrderItem* để mở nó
3. Nhấn đúp trên *GetInfo()* để mở *Operation Specification* (hình 5.47).

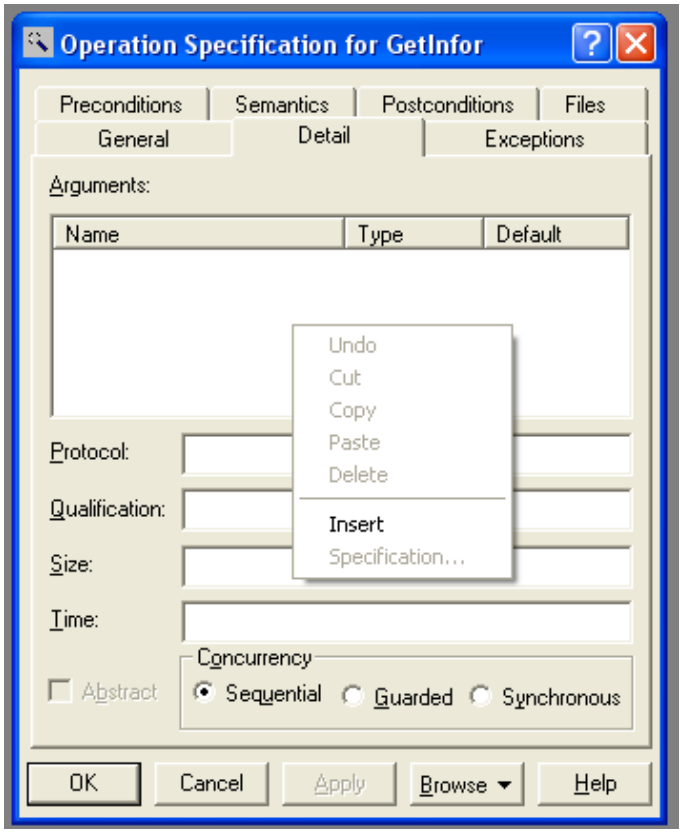


Hình 5.47 Cửa sổ thao tác đặc biệt

4. Chọn **String** trong hộp danh sách *Return type*
5. Nhấn *OK*

Thực hiện tương tự như trên để chọn Boolean cho kiểu gái trị cho lại của thao tác *SetInfo()*

6. Chọn bảng *Detail*
7. Nhấn phím phải trên khoảng trắng của vùng đối số để bổ sung tham số như hình 5.48.



Hình 5.48 Bổ sung thao tác

8. Chọn *Select. Rose tự gán đối số có tên* **argname**
9. Đổi tên từ *argname* thành **ID**
10. Nhấn trong cột *Type* để mở hộp danh sách kiểu. Chọn **Integer**
11. Nhấn cột *Default* để bổ sung giá trị mặc định. Nhập giá trị **0**
12. Nhấn *OK*
13. Nhấn đúp trên thao tác *Create()* của lớp *OrderItem* để mở cửa sổ đặc tả thao tác
14. Chọn *Boolean* cho *combobox* kiểu cho lại (*Return type*)
15. Nhấn *OK*.

5.8.2.15 - Bổ sung quan hệ

Sau khi đã bổ sung thuộc tính và các thao tác vào lớp, ta đã tiến gần đến phát sinh mã trình. Nhưng trước hết phải quan tâm đến quan hệ giữa các lớp. Hãy quan s1t việc bổ sung thực đơn trong biểu đồ trình tự để tìm ra quan hệ. Bất kỳ lớp nào trong biểu đồ trình tự đều cần quan hệ trên biểu đồ lớp. Một khi tìm ra quan hệ thì bổ sung vào biểu đồ. Thí dụ sau đây thực hiện bổ sung quan hệ vào các lớp trong UC *Enter New Order*.

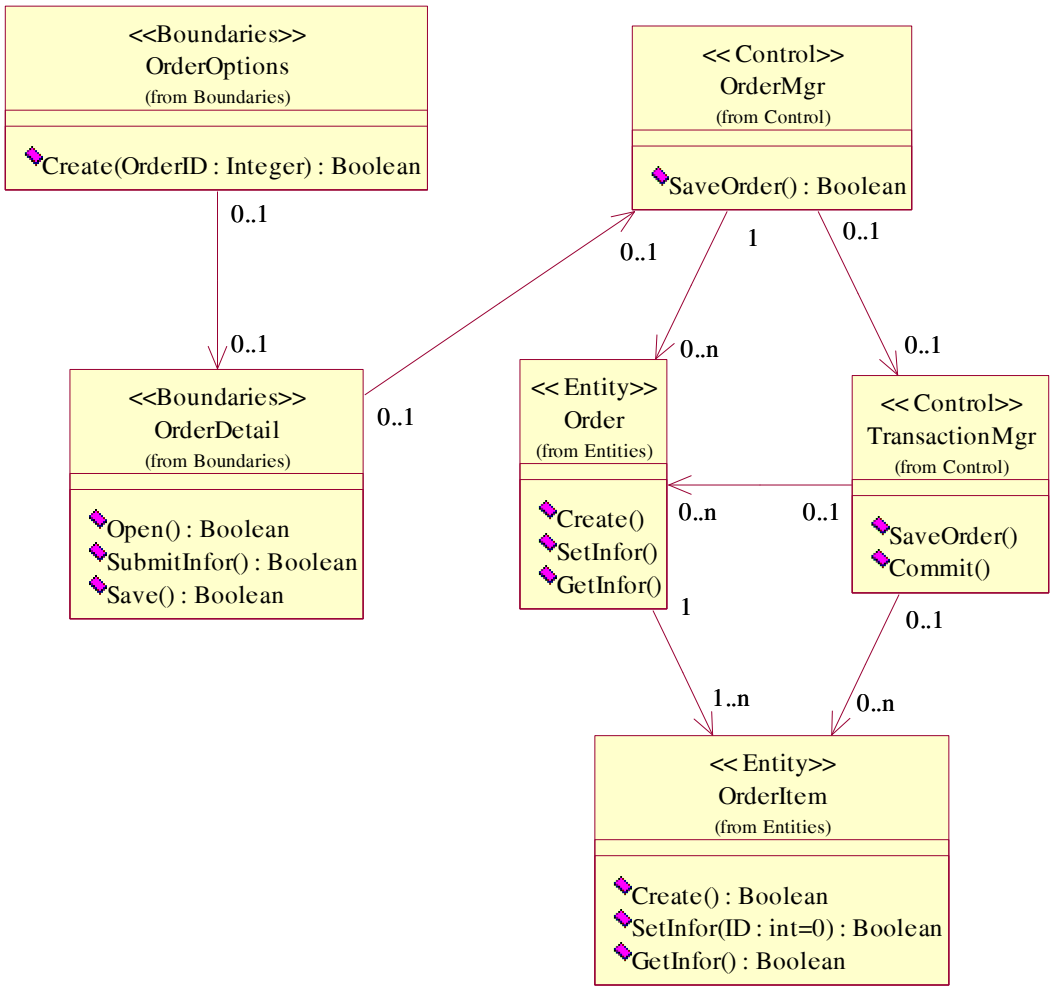
5.8.2.16 - Đặt cấu hình Rose

1. Định vị biểu đồ lớp *Add New Order* trong browser
2. Nhấp đúp để mở biểu đồ

3. Tìm phím công cụ *Unidirectional Association,* nếu không tìm thấy thì tiếp tục tiếp bước sau.
4. Nhấn phím phải trên thanh công cụ *Dagram* để chọn *Customize*
5. Bổ sung phím tên *Creates A Unidirectional Asscoiation.*

*5.8.2.17 - **Bổ sung kết hợp***

1. Chọn phím công cụ *Unidirectional Association*
2. Vẽ kết hợp từ lớp *OrderOptions* đến lớp *OrderDetail*
3. Lặp lại để vẽ các kết hộp sau:
    a. Từ *OrderDetail* đến *OrderMgr*
    b. Từ *OrderMgr* đến *Order*
    c. Từ *OrderMgr* đến *TransactionMgr*
    d. Từ *TransactionMgr* đến *Order*
    e. Từ *TransactionMgr* đến *OrderItem*
    f. Từ *Order* đến *OrderItem*
4. Nhấn phím phải trên kết hợp một chiều giữa *OrderOptions* và lớp *OrderDetail,* gần phía lớp *OrderOptions.*
5. Chọn *Multiplicity>Zero or One*
6. Nhấn phím phải trên phía kia của quan hệ một chiều.
7. Chọn *Multiplicity>Zero or One*
8. Lặp lại và bổ sung các *Multiplicity*
9. Sơ đồ kết quả như trên hình 5.49.



Hình 5.49 Quan hệ kết hợp trong Add New Order

CHƯƠNG 6

BIỂU ĐỒ CHUYỂN TRẠNG THÁI VÀ BIỂU ĐỒ HOẠT ĐỘNG

Biểu đồ chuyển trạng thái mô tả chu kỳ tồn tại của đối tượng, phân hệ và hệ thống. Biểu đồ hoạt động là mở rộng của biểu đồ trạng thái. Cả hai loại biểu đồ này điều mô tả khía cạnh động của hệ thống. Chương này trình bày các phần tử biểu đồ, biểu đồ chuyển trạng thái và biểu đồ hoạt động. Cuối chương là thực hành sử dụng UML để vẽ biểu đồ trạng thái cho một lớp trong thí dụ quản lý bán hàng.

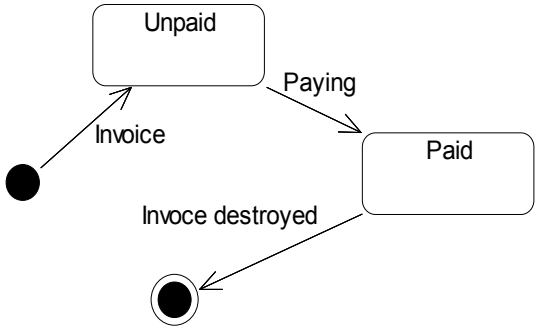
6.1 BIỂU ĐỒ CHUYỂN TRẠNG THÁI

Biểu đồ chuyển trạng thái bao gồm các thông tin về các trạng thái khác nhau của đối tượng, thể hiện các đối tượng chuyển đổi từ trạng thái này sang trạng thái khác thế nào, hành vi của mỗi đối tượng trong mỗi trạng thái ra sao. Biểu đồ trạng thái chỉ ra chu kỳ sống của đối tượng, từ khi nó được tạo ra đến khi bị phá hủy. Nó còn cho biết các sự kiện (thông điệp nhận được, kết thúc khoảng thời gian, điều kiện nào đó thành *true*) tác động lên các trạng thái như thế nào.Biểu đồ này là giải pháp tốt nhất để mô hình hóa hành vi động của lớp. Thông thường trong một dự án sẽ không phải tạo ra mọi biểu đồ trạng thái cho mọi lớp. Nhiều dự án không sử dụng loại biểu đồ này. Nếu có lớp với nhiều hành vi động (lớp ở một trong nhiều trạng thái khác nhau) thì biểu đồ này có ích lợi.

Mọi đối tượng đều có trạng thái; trạng thái là kết quả của các hoạt động do các đối tượng thực hiện trước đó. Đối tượng luôn luôn ở trong một trạng thái xác định vào một thời điểm. Thông thường trạng thái được xác định bởi các giá trị thuộc tính và các liên kết đến các đối tượng khác. Thí dụ nếu hai đối tượng của lớp *Person* và *Company* có quan hệ thì con người cụ thể của lớp *Person* không phải lúc nào cũng có việc làm mà nó sẽ ở một trong ba trạng thái là *Employed* (người lao động), *Retired* (người về hưu) hay *Unemployed* (thất nghiệp). Thí dụ về các trạng thái của đối tượng như sau:

- Hóa đơn (đối tượng) đã được thanh toán (trạng thái).
- Xê ô tô (đối tượng) đang đứng (trạng thái).
- Chị B (đối tượng) làm nhiệm vụ bán hàng (trạng thái).

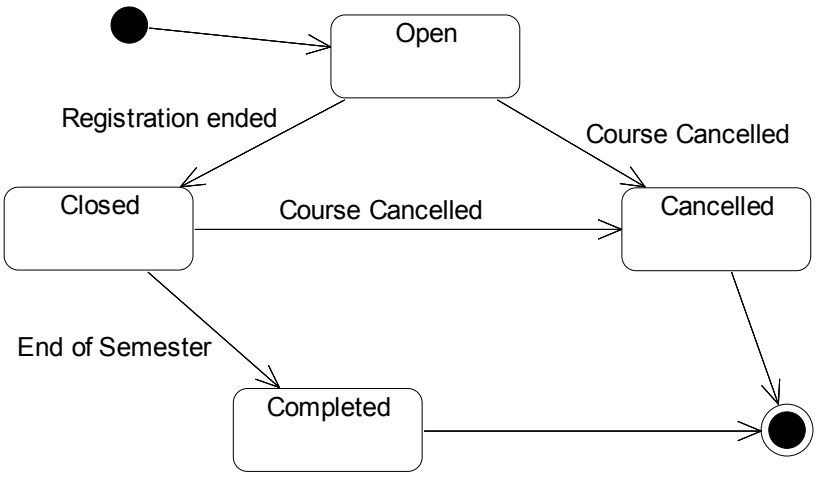
Đối tượng thay đổi trạng thái khi cái gì đó xảy ra (gọi là sự kiện); thí dụ, ai đó thanh toán hóa đơn bán hàng, bắt đầu lái xe… Biểu đồ trạng thái được sử dụng để chỉ ra đối tượng phản ứng với các sự kiện như thế nào và trạng thái bên trong của chúng thay đổi ra sao. Khi quan sát đối tượng động ta phải quan tâm đến tương tác và sự thay đổi bên trong nó. Tương tác mô tả hành vi bên ngoài đối tượng và tương tác với các đối tượng khác như thế nào (bằng gửi thông điệp). Giả sử ai đó thanh toán hóa đơn hàng thì hóa đơn thay đổi trạng thái từ chưa thanh toán (*Unpaid)* sang thanh toán (*Paid*). Khi hóa đơn mới được lập ra thì nó ở trạng thái chưa thanh toán như mô tả trên hình 6.1.



Hình 6.1 Trạng thái của đơn hàng

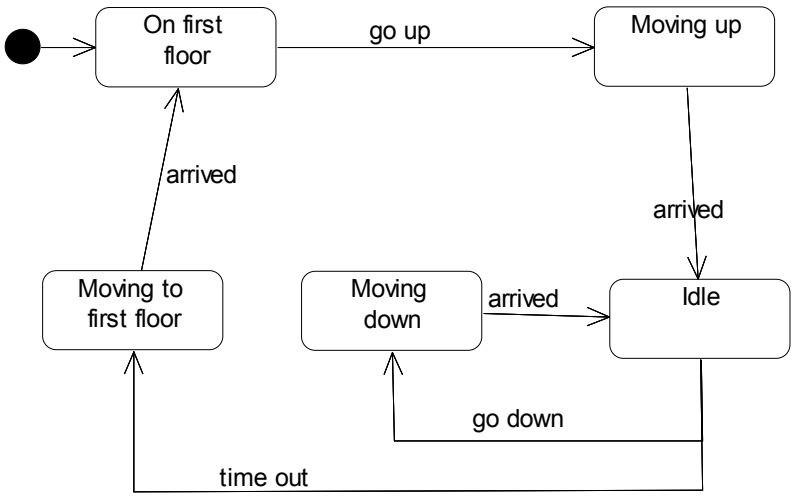
Biểu đồ trạng thái có điểm khởi đầu (hình tròn đen) và vài điểm kết thúc (hình tròn đen có đường bao); các hoạt động trong biểu đồ được đặt trong chữ nhật góc tròn. Trong chữ nhật có các dòng văn bản để chỉ ra các hành động. Giữa các trạng thái là quá độ trạng thái (thể hiện bằng mũi tên). Quá độ có thể có tên sự kiện gây ra biến đổi trạng thái. Khi sự kiện xảy ra thì có sự biến đổi từ trạng thái này sang trạng thái kia (đôi khi còn gọi là cháy). Trên hình 6.1, trả tiền (*Paying*), lập hóa đơn (*Invoice creaded*) hủy hóa đơn (*Invoice destroyed*) là các sự kiện làm chuyển trạng thái.

Hình 6.2 là thí dụ biểu đồ chuyển trạng thái của lớp Đăng ký môn học (*Course*). Trong thí dụ này, lớp *Course* có thể ở một trong các trạng thái như mở môn học (*Open*), kết thúc đăng ký môn học (*Closed*), bãi bỏ môn học (*Cancelled*) và hoàn thành môn học (*Completed*).



Hình 6.2 Biểu đồ chuyển trạng thái lớp Course

Hình 6.3 là thí dụ biểu đồ trạng thái của thang máy. Thang máy bắt đầu từ tầng một, có nghĩa rằng nó đang ở trạng thái *On first floor*, nó có thể đi lên (*Moving up*) hay đi xuống (*Moving down*). Nếu thang máy đang dừng tại tầng nào đó (*Idle*) thì sau khoảng thời gian nhất định nó đi lên tầng một (*Moving to first floor*). Biểu đồ trạng thái của thí dụ này không có điểm cuối.

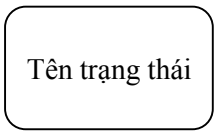


Hình 6.3 Biểu đồ trạng thái thang máy

Biểu đồ trạng thái là cần thiết bởi vì nó giúp phân tích viên, người thiết kế và người phát triển hiểu hành vi đối tượng trong hệ thống. Đặc biệt, người phát triển phải hiểu rõ hành vi đối tượng vì họ phải cài đặt hành vi trong phần mềm. Họ không chỉ cài đặt đối tượng mà còn làm đối tượng thực hiện cái gì đó.

6.1.1 - Trạng thái

Trạng thái là một trong các điều kiện có thể để đối tượng tồn tại. Trạng thái được xãc định từ hai vùng: thuộc tính và quan hệ giữa các lớp. Tương tự các thành phần khác của biểu đồ UML, ta có thể bổ sung tài liệu vào trạng thái. Tuy nhiên, tài liệu này sẽ không chuyển thành mã nguồn sau này. Ký pháp trạng thái trong UML như sau:



Khi đối tượng ở trong trạng thái nào đó, nó thực hiện nhiều hoạt động. Thí dụ, phát sinh báo cáo, thực hiện một vài tính toán hay gởi sự kiện tới đối tượng khác. Có thể gộp năm loại thông tin để mô tả trạng thái; chúng bao gồm hoạt động, hành động vào, hành động ra, sự kiện, lịch sử trạng thái.

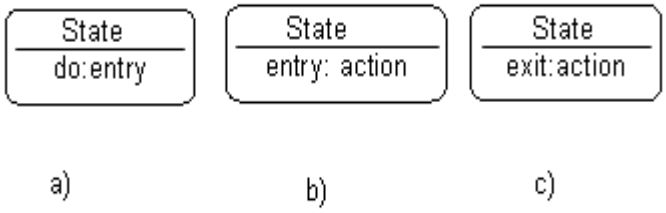
Lần đầu tiên đối tượng được tạo ra thì nó ở trong *trạng thái khởi động*. Trong biểu đồ UML thì trạng thái khởi động được vẽ bằng hình tròn đen. Chỉ có một trạng thái bắt đầu trong biểu đồ. *Trạng thái dừng* là trạng thái khi đối tượng bị phá hủy. Số lượng trạng thái dừng trong biểu đồ là tùy ý, có bao nhiêu trạng thái dừng cho biểu đồ cũng được.

6.1.1.1 - Hoạt động

Hoạt động (*activity*) là hành vi mà đối tượng thực thi khi nó ở trong trạng thái cụ thể. Thí dụ, khi tài khoản ở trong trạng thái đóng thì máy đọc thẻ không chấp nhập thẻ tín dụng của khách hàng, nếu đã mở tài khoản thì trạng thái có thể là gửi thông điệp, chờ hay tính toán. Hoạt động là hành vi có thể ngắt được, nó có thể được hoàn thành khi đối tượng đang trong trạng thái hay nó bị ngắt vì đối tượng chuyển sang trạng thái khác. Hoạt động được biểu diễn trong phần tử biểu đồ trạng thái (hình 6.4a), trước đó có từ *do* và dấu *:* (hoặc dấu */*).

6.1.1.2 - ***Hành động vào***

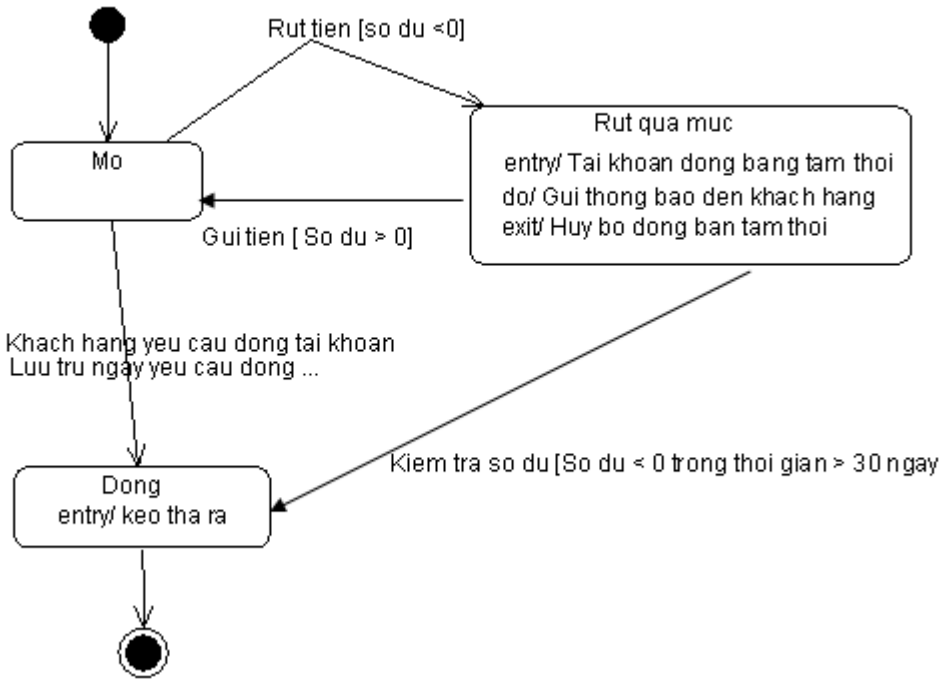
Hành động vào (*Entry action*) là hành vi xảy ra khi đối tượng đang chuyển đổi trạng thái. Thí dụ khi tài khoản đang chuyển vào trạng thái rút tiền quá mức thì "hành động vào" đóng băng tạm thời tài khoản (*Temporarily freeze account*) xảy ra (hình 6.4). Tuy nhiên hành động náy sẽ không xảy ra khi đối tượng đã vào trạng thái rút quá qui định. Nó xảy ra như một bộ phận của biến đổi vào trạng thái. Không giống như hoạt động, hành vi vào là không ngắt được. Ký pháp của hành động vào trong UML như trên hình 6.4b.



Hình 6. 4 Thông tin trong trạng thái

6.1.1.3 - ***Hành động ra***

Hành động ra (*exit action*) tương tự như hành động vào. Tuy nhiên, hành động ra là bộ phận của chuyển đổi ra khỏi trạng thái. Thí dụ, khi tài khoản rời bỏ trạng thái rút quá quy định thì hành động xảy ra hủy trạng thái đóng băng tạm thời (*Remove Temporary Freeze*) xảy ra như một phần của biến đổi (hình 6.5). Hành động ra cũng không bị ngắt. Ký pháp của hành động ra trong UML như trên hình 6.4c.



Hình 6.5 Biểu đồ biến đổi trạng thái của lớp Tài khoản

Tóm lại, cú pháp hình thức cho các hoạt động trong một trạng thái được thể hiện như sau:

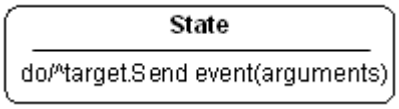
event_name argument_list '/' action_expression

trong đó, *event_name* có thể là sự kiện bất kỳ nào, kể cả các sự kiện chuẩn như *entry, exit* hay *do; action_expression* cho biết hành động nào sẽ được thực hiện, có thể thay ký tự *:* cho ký tự */*. Các sự kiện có thể có đối.

Hình 6.5 là thí dụ biểu đồ biến đổi trạng thái của lớp *Tài khoản* trong hệ thống rút tiền tự động ATM. Hành vi trong hoạt đông (*do*), hành động vào (*entry*), hành động ra (*exit*) có thể gửi sự kiện đến một vài đối tượng khác. Khi nhận được sự kiện thì hành động nào đó có thể xảy ra. Thí dụ đối tượng lớp *Tài khoản* có thể gửi sự kiện đến đối tượng đọc thẻ tín dụng. Trong trường hợp này phải đặt dấu ^ vào trước hành động, hành động vào, hành động ra. Trên biểu đồ UML có thể là:

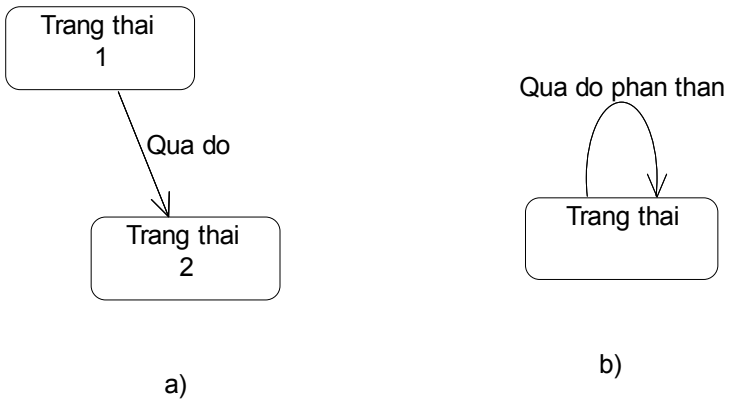
Do:^Target.Event(Arguments)

trong đó, *Target* là đối tượng nhận sự kiện, *Event* là thông điệp gửi và *Arguments* là tham số thông điệp.



6.1.2 - Quá độ

Quá độ (*transition*) là chuyển động từ trạng thái này sang trạng thái khác. Ký pháp quá độ trên biểu đồ UML như trên hình 6.6a. Quá độ có thể phản thân, khi xảy ra hiện tượng là đối tượng chuyển tiếp trở lại trạng thái củ của nó. Ký pháp quá độ phản thân trong UML như hình 6.6b.

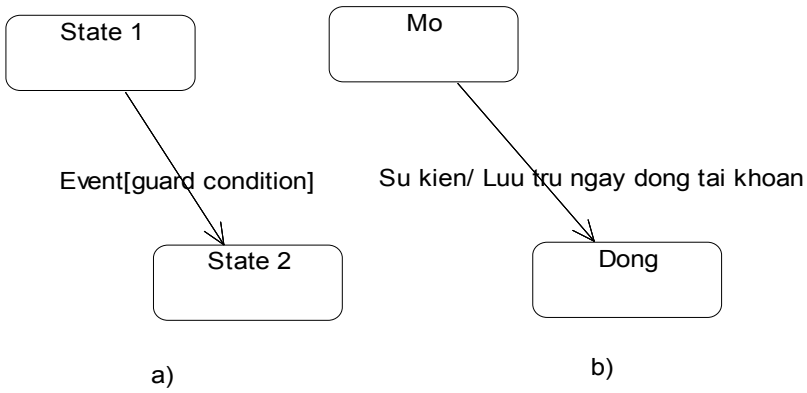


Hình 6.6 Chuyển tiếp trạng thái

UML cho khả năng gộp nhiều đặc tả vào quá độ. Chúng có thể là sự kiện, đối, điều kiện bảo vệ, hành động và gửi sự kiện.

Sự kiện. sự kiện là cái gì đó làm nguyên nhân của chuyển tiếp từ trạng thái này sang trạng thái khác. Thí dụ sự kiện *Khách hàng yêu cầu đóng tài khoản* sẽ gây ra tài khoản chuyển từ *trạng thái mở* sang *trạng thái đóng* (hình 6.5). Sự kiện *Gửi tiền vào* sẽ chuyển *Tài khoản* từ trạng thái *Rút quá* sang trạng thái *Mở* có đối số là *Tổng số*, chứa số tiền gửi vào. Hầu hết các chuyển tiếp đều có sự kiện. Với chuyển tiếp tự động (không có sự kiện) thì đối tượng tự động chuyển tiếp từ trạng thái này sang trạng thái khác.

Điều kiện canh (guard). Điều kiện canh điều khiển chuyển tiếp để có thể hay không có thể xảy ra (hình 6.7a). Thí dụ, sự kiện gửi tiền sẽ chuyển tài khoản từ trạng thái *Rút quá* sang trạng thái *Mở,* nhưng chỉ khi *Số dư tài khoản* lớn hơn 0.



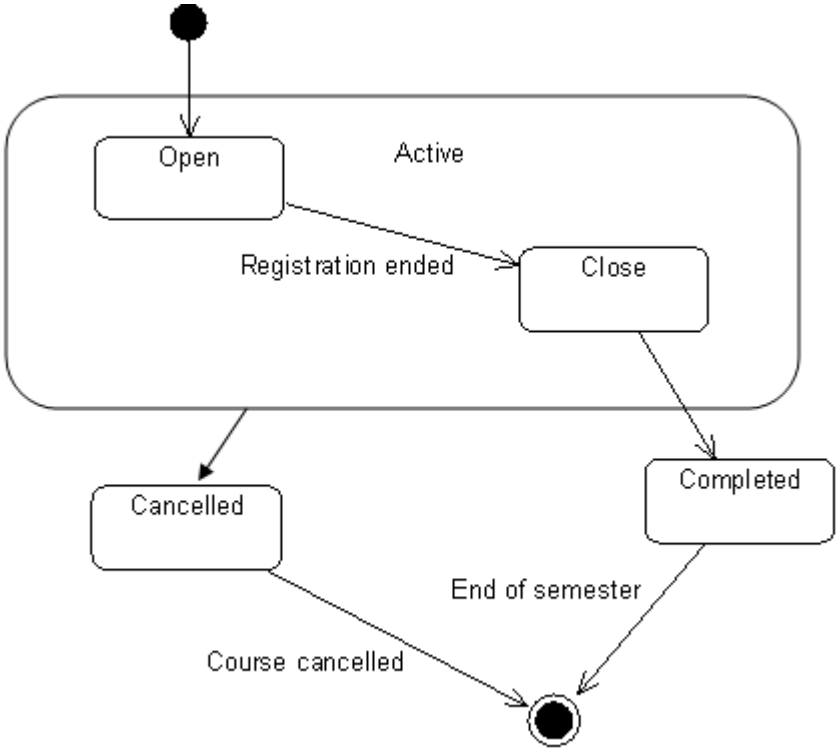
Hình 6.7 Điều kiện canh

Hành động (action). Hành động là hành vi không ngắt được, xảy ra như một phần của chuyển tiếp. Hành động vào và hành động ra được thể hiện trong trạng thái. Phần lớn hành động được vẽ theo mũi tên chuyển tiếp. Thí dụ, khi chuyển từ trạng thái mở sang trạng thái đóng, hành động *Lưu thời điểm yêu cầu đóng tài khoản* xảy ra. Đó là hành vi không ngắt được, xảy ra khi tài khoản chuyển tiếp từ trạng thái mở sang trạng thái đóng. Hành động biểu thị trên mũi tên chuyển tiếp được đặt sau tên sự kiện và dấu / (hình 6.7b).

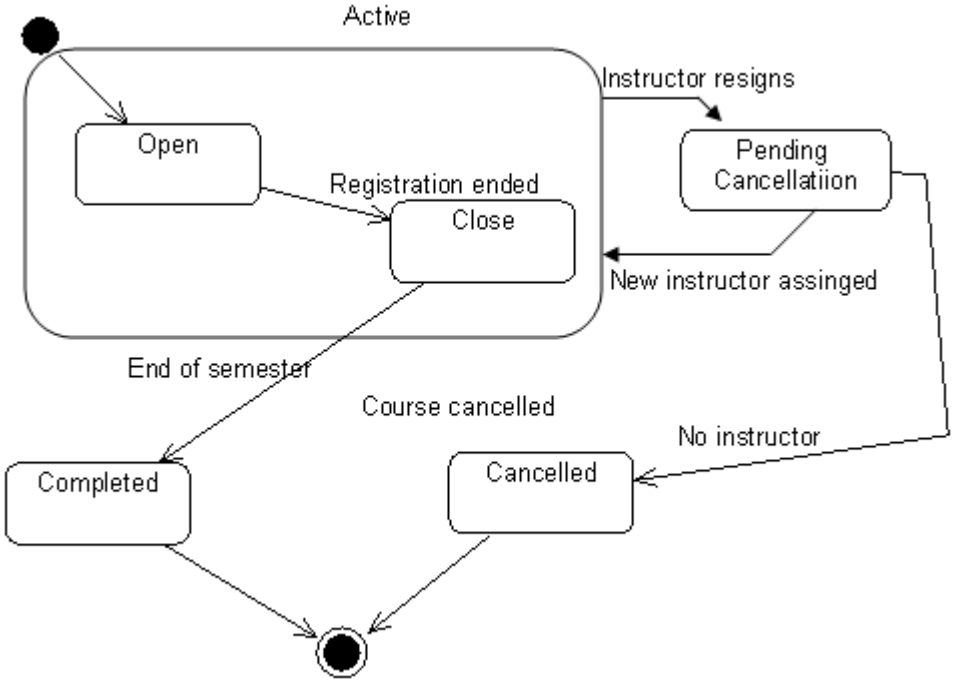
6.1.3 - Trạng thái ẩn

Để giảm thiểu số lượng trạng thái trong biểu đồ, ta có thể ẩn một số trạng thái trong trạng thái khác. Các trạng thái ẩn (*nested*) được xem như tiểu trạng thái (*substates*), trong khi trạng thái lớn hơn gọi là siêu trạng thái (*superstates*).

Nếu hai hay nhiều trạng thái có cùng chuyển tiếp, thì có thể nhóm chúng vào cùng siêu trạng thái. Sau đó thay vì quản lý hai chuyển tiếp như nhau, ta quản lý chuyển tiếp của siêu trạng thái. Biểu đồ trạng thái trên hình 6.8 là biểu đồ trạng thái ẩn xây dựng từ biểu đồ trên hình 6.2.



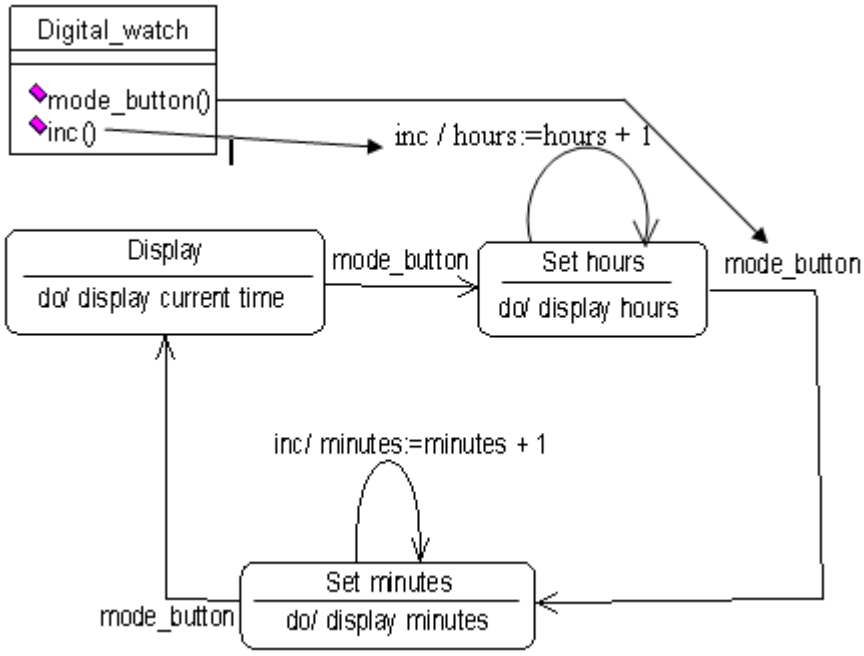
Hình 6.8 Trạng thái ẩn



Hình 6.9 Lịch sử siêu trạng thái

Trong biểu đồ loại này, hệ thống cần phải nhớ trạng thái bên trong nào là cuối cùng. Giả sử ta có ba trạng thái trong siêu trạng thái, khi rời khỏi siêu trạng thái ta muốn hệ thống nhớ lại nơi vừa ra khỏi từ siêu trạng thái để trở lại sau này. Để giải quyết vấn đề này ta cần làm hai việc sau: Thứ nhất là bổ sung trạng thái khởi đầu trong siêu trạng thái để chỉ ra điểm khởi đầu mặc định trong siêu trạng thái; thứ hai là sử dụng *chỉ báo lịch sử trạng thái* để nhớ nơi đối tượng vừa đi qua. Nếu chỉ báo lịch sử được đặt thì đối tượng có thể rời khỏi siêu trạng thái, khi trở lại nó sẽ đến đúng

nơi nó rời bỏ trước đó. Chỉ báo lịch sử được ký hiệu bằng chữ "**H**" trong vòng tròn tại góc biểu đồ (hình 6.9).



Hình 6.10 Lớp và biểu đồ trạng thái tương ứng

6.1.4 - Lớp và biểu đồ trạng thái

Quan hệ giữa lớp và biểu đồ trạng thái tương ứng được mô tả thông qua thí dụ trên hình 6.10. Trên hình này có lớp *Digital_watch* và biểu đồ trạng thái tương ứng của nó. Hình này cho thấy các sự kiện trong biểu đồ trạng thái liên quan đến các thao tác trong lớp như thế nào. Trong thí dụ này đồng hồ có ba trạng thái sau: trạng thái hiển thị thông thường và hai trạng thái tăng giờ tăng phút.

6.2 BIỂU ĐỒ HOẠT ĐỘNG

Chắc chắn rằng khái niệm biểu đồ tiến trình (*flowchart*) quen thuộc với người lập trình viên. Biểu đồ này chỉ ra trình tự các bước, tiến trình, các điểm quyết định và các nhánh. Các lập trình viên mới thường sử dụng khái niệm này để khái quát vấn đề và đề xuất giải pháp. Biểu đồ hoạt động của UML tương tự như biểu đồ tiến trình. Nó chỉ ra các bước (các hoạt động), các điểm quyết định và các nhánh. Biểu đồ hoạt động là biểu đồ mới trong UML vì nó không phải kết quả của *Booch, Jacobson* hay *Rumbagh.* Nó được hình thành trên biểu đồ sự kiện của *Odell* và được cập nhật vào phiên bản cuối cùng của UML.

Biểu đồ hoạt động được sử dụng để mô hình hóa khía cạnh động của hệ thống, mô hình hóa các bước trình tự hay tương tranh trong quá trình tính toán. Biểu đồ hoạt động còn được sử dụng để mô hình hóa luồng đối tượng đi từ trạng thái này sạng trạng thái khác tại từng vị trí trong luồng điều khiển. Trong khi biểu đồ tương tác tập trung vào luồng điều khiển từ đối tượng đến đối tượng thì biểu đồ hoạt động tập trung vào luồng điều khiển từ hoạt động đến hoạt động. Biểu đồ hoạt động bao gồm trạng thái hoạt động và trạng thái hành động, quá độ và đối tượng. Tương tự như các biểu đồ khác, biểu đồ loại này cũng có ghi chú và ràng buộc.

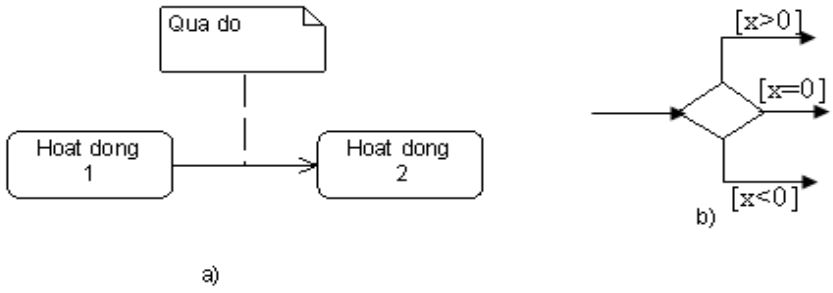
6.2.1 - Trạng thái hành động và trạng thái hoạt động

Trong luồng điều khiển bằng mô hình hóa bằng biểu đồ hoạt động ta có thể thực hiện vài biểu thức để cho lại giá trị hay thiết lập giá trị thuộc tính, ta có thể gọi thao tác của đối tượng, gửi tín hiệu đến đối tượng hay sự kiện lập và hủy bỏ đối tượng. Các tính toán cơ bản này được gọi là trạng hành động (*action state*) vì nó là trạng thái của hệ thống. Trạng thái hành động là không chia nhỏ được, công việc của trạng thái hành động là không ngắt được và nó chiếm khoảng thời gian ngắn để thực hiện.

Ngược lại, trạng thái hoạt động (*activity state*) là tách được, nó có thể bị ngắt và cần khoảng thời gian đáng kể để hoàn thành công việc. Có thể xem trạng thái hành động là trường hợp đặc biệt của trạng thái hoạt động. Ngược lại, trạng thái hoạt động được xem như tổ hợp mà luồng điều khiển của nó được lập từ các trạng thái hoạt động khác và từ các trạng thái hành động. Trong trạng thái hoạt động ta có thể tìm ra các biểu đồ hoạt động khác. Ký pháp đồ họa của trạng thái hoạt động giống như ký pháp trạng thái hành động, điểm khác nhau là trạng thái hoạt động còn có các bộ phận khác như hành động vào, hành động ra hay đặc tả *submachine*. Ký pháp đồ họa của trạng thái hoạt động trên hình 6.11. Trạng thái khởi đầu và trạng thái kết thúc được biểu diễn tương tự như biểu đồ trạng thái.

6.2.2 - Quá độ

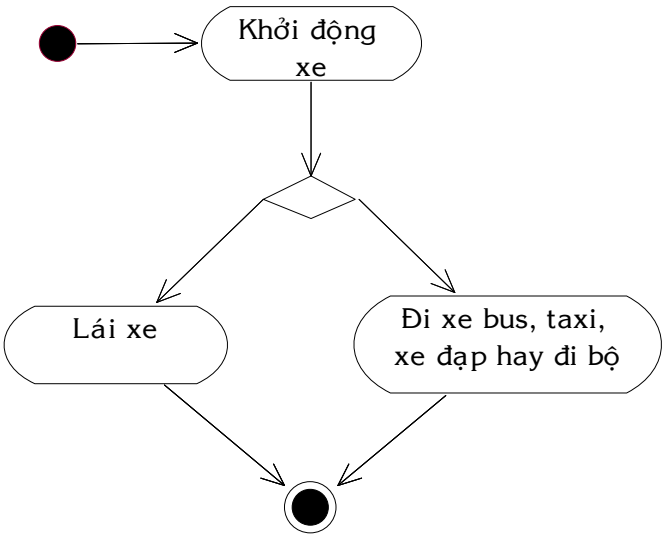
Khi hành động hay hoạt động của trạng thái hoàn thành, luồng điều khiển chuyển sang trạng thái hành động hay hoạt động khác. Luồng này được mô tả bởi quá độ (*transition*), nó cho thấy đường đi từ trạng thái hành động hay hoạt động đến trạng thái hành động hay hoạt động khac. Việc chuyển tiếp từ trạng thái này sang trạng thái khác được thực hiện tức thì vì hệ thống phải luôn ở trong trạng thái biết trước. Trong UML, quá độ được biểu diễn bằng mũi tên đơn (hình 6.11a).



Hình 6.11 Quá độ và rẽ nhánh

6.2.3 - Rẽ nhánh

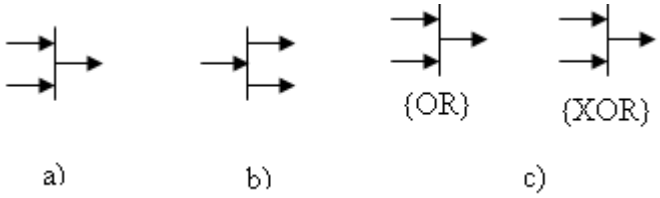
Thông thường thì quá độ là trình tự. Nhưng cần phải có kiểu đường đi khác để mô hình hóa luồng điều khiển. Trong biểu đồ tiến trình, rẽ nhánh xác định đường đi phụ trên cơ sở biểu thức *Bool*. Trên biểu đồ UML, rẽ nhánh được biểu diễn bởi viên kim cương trắng. Rẽ nhánh có thể có một quá độ vào và hai hay nhiều quá độ ra. Quá độ ra thường được gán điều kiện viết trong ngoặc vuông như trên hình 6.11b hay được gắn biểu thức *Bool*, nó được đánh giá tại đầu vào nhánh. Thí dụ, ông A hàng ngày đi làm việc bằng xe máy. Ông A lên xe, cắm chìa khóa và khởi động xe. Hai trường hợp này hình thành hai hoạt động: ông A lái xe và ông A đi xe bus, taxi, xe đạp hay đi bộ. Kịch bản này thể hiện trên hình 6.12.



Hình 6.12 Biểu đồ hoạt động hai nhánh

6.2.4 - Đường dẫn tương tranh

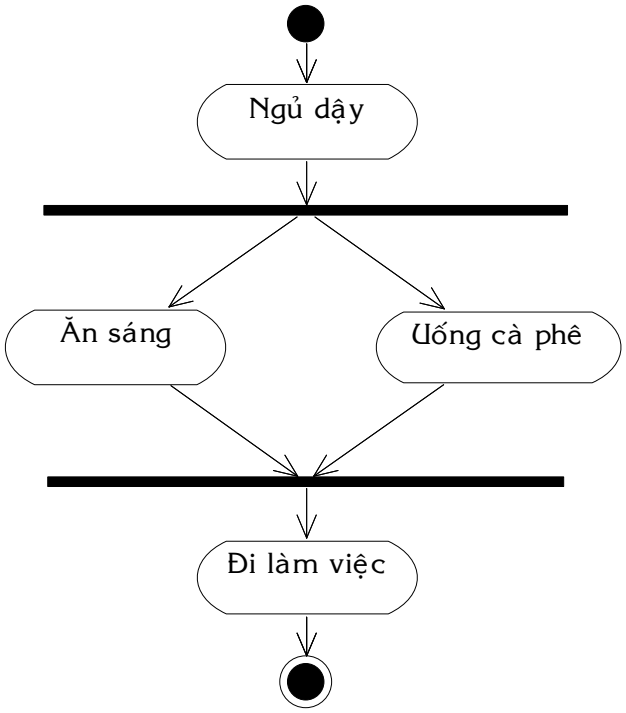
Đường dẫn hay gặp nhất trong biểu đồ hoạt động là có quá độ trong trình tự rẽ nhánh. Tuy nhiên trong khi mô hình hóa luồng công việc của nhiều tiến trình ta gặp phải các luồng tương tranh. Trong UML, thanh đồng bộ được sử dụng để kết hợp (hình 6.13a) và chia nhánh (hình 6.13b) các luồng điều khiển song song.



Hình 6.13 Đồng bộ và rẽ nhánh

Thanh đồng bộ được vẽ bằng đường đậm. Đồng bộ có nghĩa rằng mọi quá độ đi vào phải có mặt đầy đủ trước khi quá độ "cháy" (hình 6.13a). Quá độ này còn được gọi là đồng bộ *AND*. Tuy nhiên, để mô tả quá độ đồng bộ *OR* (chỉ một quá độ vào là đủ cho "cháy") hay vẽ quá độ đồng bộ *XOR*, *B. Oesterich* đã đề nghị ký pháp đồ họa như trên hình vẽ 6.13c [OEST001].

Thí dụ trên hình 6.14 mô tả chia luồng điều khiển đơn thành hai luồng điều khiển tương tranh. Thanh đồng bộ chia nhánh có một quá độ đến và hai quá độ đi ra. Hai hoạt động *Ăn sáng* và *Uống cà phê* song song tiếp tục. Thanh kết hợp có hai quá độ đi vào và một quá độ ra. Thanh này thực hiện đồng bộ luồng tương tranh, có nghĩa rằng các luồng này phải chờ đến khi mọi luồng đều đến thanh kết hợp (*Ăn sáng* và *Uống cà phê* xong) để sau đó chỉ có một luồng điều khiển tiếp tục (*Đi làm việc*). Như vậy, thanh đồng bộ cho khả năng "mở" hay "đóng" các nhánh song song trong luồng thực hiện của thao tác hay UC.



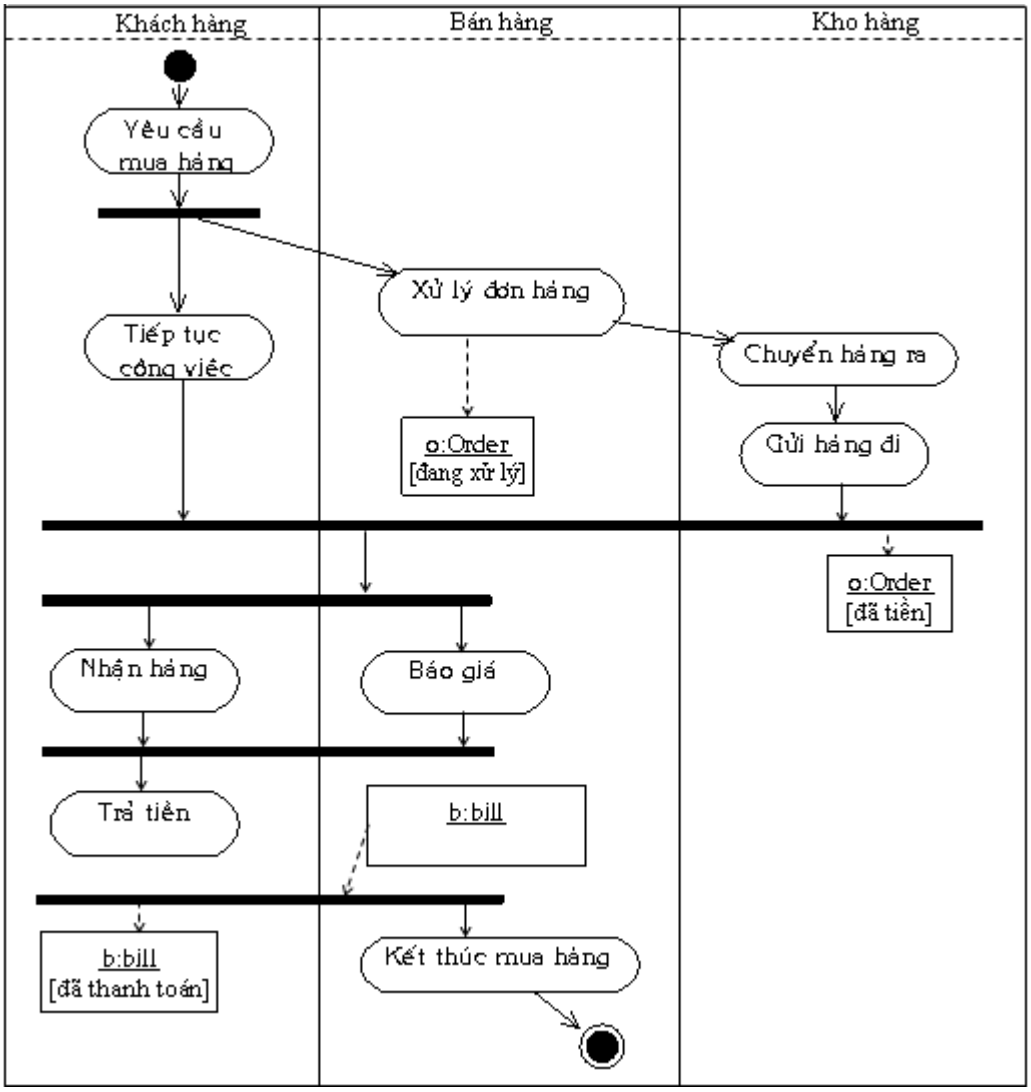
Hình 6.14 Các hoạt động tương tranh

6.2.5 - Đường bơi

Đường bơi (*swimlanes*) được sử dụng để mô hình hóa luồng công việc của các tiến trình thương mại, để phân hoạch các trạng thái hoạt động trên biểu đồ hoạt động vào nhóm. Trong UML, mỗi nhóm này được gọi là đường bơi vì các nhóm được phân tách bởi đường thẳng đứng như trên hình 6.15. Mỗi đường bơi có tên và nhiệm vụ duy nhất và được biểu thị trên đỉnh biểu đồ.Trong biểu đồ hoạt động. mỗi hoạt động thuộc về một đường bơi nhưng quá độ có thể được vẽ trải qua các đường bơi. Mỗi đường bơi được cài đặt bởi một hay nhiều lớp.

6.2.6 - Luồng đối tượng

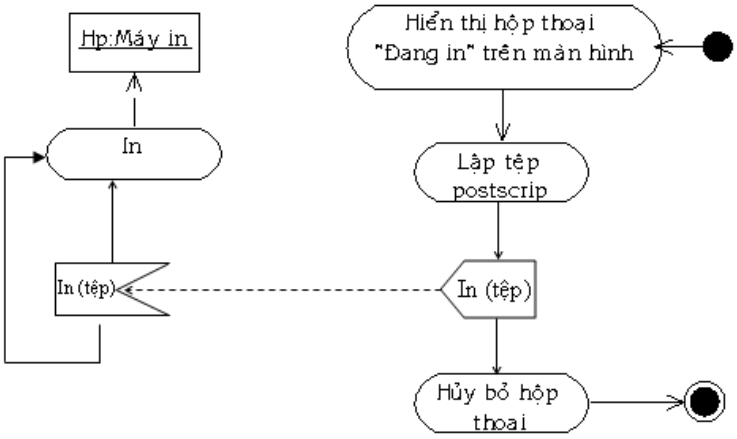
Các đối tượng có thể được kích hoạt trong luồng điều khiển trong biểu đồ hoạt động. Khảo sát tiếp tục thí dụ trên hình 6.13. Giả xử hoạt động *Xử lý đơn hàng* tạo ra đối tượng của lớp *Order*, hoạt động *Gửi hàng đi* làm thay đổi trạng thái đối tượng *Order* thành đã được xử lý. Biểu đồ hoạt động hình 6.15 còn được bổ sung các đối tượng. Mũi tên nét đứt trên biểu đồ là luồng đối tượng.



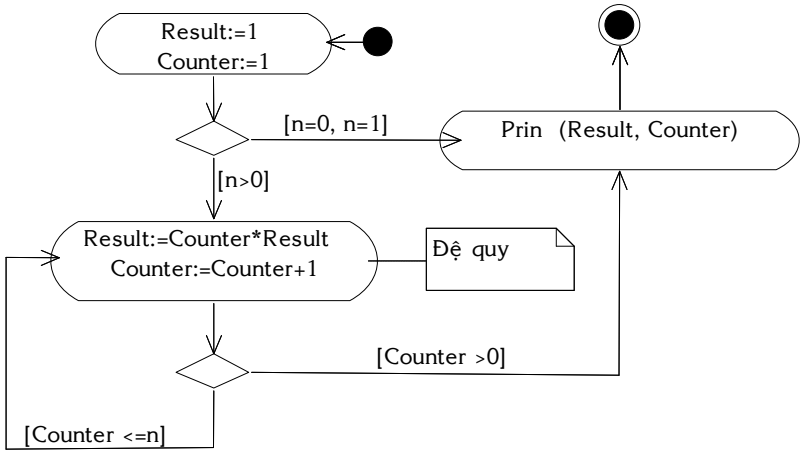
Hình 6.15 Mô hình hóa luồng công việc

6.2.7 - Gửi và nhận tín hiệu

Có thể gửi và nhận tín hiệu trong biểu đồ hoạt động. Có hai loại biểu tượng cho gửi và nhận tín hiệu. Hình ngũ giác lồi là biểu tượng của gửi và ngũ giác lõm là biểu tượng của nơi nhận. Trên biểu đồ chúng được gắn vào đường quá độ của đối tượng gửi và đối tượng nhận thông điệp. Hình 6.16 là biểu đồ hoạt động mô tả tiến trình in tệp trong *Windows*. Giữa hai hành động *Tạo lập tệp postcript* và *Hủy bỏ hộp thoại* là gửi tín hiệu *Yêu cầu in*. Tín hiệu này còn chứa tệp và được đối tượng *Máy in* in.



Hình 6.16 Tín hiệu trong biểu đồ hoạt động



Hình 6.17 Mô hình hóa thao tác

Tóm lại, khi mô hình hóa khía cạnh động của hệ thống thì thông thường biểu đồ hoạt động được sử dụng theo hai cách sau:

- Mô hình hóa luồng công việc.
- Mô hình hóa thao tác. Thí dụ trên hình 6.17 vừa mô tả trên đây là mô hình hóa thao tác tính giai thừa của số n.

6.3 THỰC HÀNH

6.3.1 - Sử dụng Rational Rose

6.3.1.1 - Tạo lập biểu đồ trạng thái

Trong Rose có thể tạo một biểu đồ trạng thái cho mỗi lớp. Mọi trạng thái và chuyển tiếp trạng thái của lớp đều ở trong biểu đồ này. Biểu đồ chuyển trạng thái nằm dưới lớp trong *browser*. Quá trình tạo lập biểu đồ chuyển trạng thái như sau:

1. Nhấn phím phải chuột trên lớp trong *browser*
2. Chọn thực đơn Open State Diagram

6.3.1.2 - Bổ sung trạng thái vào biểu đồ

Bổ sung trạng thái vào biểu đồ như sau:

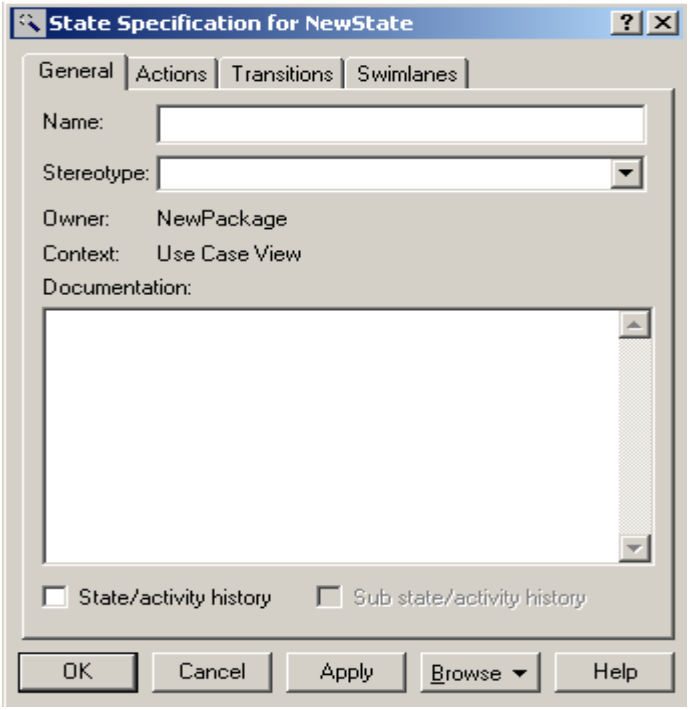
1. Chọn phím *State* từ thanh công cụ
2. Nhấn phím chuột trên biểu đồ biến đổi trạng thái nơi muốn vẽ trạng thái.

6.3.1.3 - Bổ sung chi tiết cho trạng thái

Sử dụng bảng *Detail* trong cửa sổ hoặc đặc tả trạng thái như trên hình 6.18 để gộp các thông tin vào trạng thái, bao gồm hành động, hành động vào, hành động ra, sự kiện và lịch sử trạng thái.

Hoạt động (*activity*). Bổ sung hoạt động theo các bước sau đây:

1. Mở cửa sổ đặc tả cho trạng thái mong muốn
2. Chọn bảng *Detail*
3. Nhấn phím phải trên hộp *Actions*
4. Chọn thực đơn *Insert*
5. Nhấp đúp trên hành động mới
6. Nhập hành động trong trường *Action*
7. Chọn Entry Until Exit trong hộp *When*



Hình 6.18 Cửa sổ đặc tả trạng thái

Các bước bổ sung hành động vào (*entry action*) và hành động ra (*exit action*) được vào trạng thái được thực hiện tương tự như các bước như các bước mô tả trên đây.

Gửi sự kiện. Gửi sự kiện được thực hiên theo các bước sau đây:

1. Mở cửa sổ đặc tả cho trạng thái mong muốn
2. Chọn bảng *Detail*

3. Nhấn phím phải trên hộp *Actions*
4. Chọn thực đơn *Insert*
5. Nhấn đúp trên hành động mới
6. Chọn kiểu *Send Event*
7. Nhập sự kiện, đối số và đích trong các trường tương ứng.

6.3.1.4 - Bổ sung chuyển trạng thái

Vẽ phần tử chuyển trạng thái vào biểu đồ như sau:

1. Chọn phím *Transition* trên thanh công cụ
2. Nhấn trên trạng thái nơi bắt đầu vẽ
3. Di chuyển đường chuyển trạng thái đến phần tử trạng thái đích.

Vẽ phần tử chuyển trạng thái phản thân như sau:

- Chọn phím *Transition to Self* trên thanh công cụ
- Nhấn trên trạng thái nơi vẽ chuyển trạng thái phản thân.

6.3.1.5 - Bổ sung chi tiết cho chuyển trạng thái

Đặc tả chuyển trạng thái trong UML bao gồm sự kiện, đối số, điều kiện gác, hành động và gửi sự kiện. Gán sự kiện cho chuyển trạng thái theo các bước như sau:

Nhấn đúp vào chuyển tiếp để mở cửa sổ đặc tả

Chọn bảng General

Nhập sự kiện trong trường Event

Gán sự kiện cho chuyển trạng thái theo các bước như sau:

1. Nhấn đúp vào chuyển tiếp để mở cửa sổ đặc tả
2. Chọn bảng General
3. Nhập đối số trong trường Argument.

Bổ sung điều kiện gác cho biểu đồ như sau:

4. Nhấn đúp vào chuyển tiếp để mở cửa sổ đặc tả
5. Chọn bảng *Detail*
6. Nhập điều kiện trong trường *Condition.*

Bổ sung hành động cho biểu đồ như sau:

7. Nhấn đúp vào chuyển tiếp để mở cửa sổ đặc tả
8. Chọn bảng *Detail*
9. Nhập hành động trong trường *Condition.*

Bổ sung gửi sự kiện cho biểu đồ như sau:

10. Nhấn đúp vào chuyển tiếp để mở cửa sổ đặc tả
11. Chọn bảng *Detail*

12. Nhập sự kiện trong trường *Send Argument*

13. Nhập đích vào trường *Send Target*

Để ẩn (*nest*) trạng thái ta làm như sau:

14. Chọn phím S*tate* từ thanh công cụ

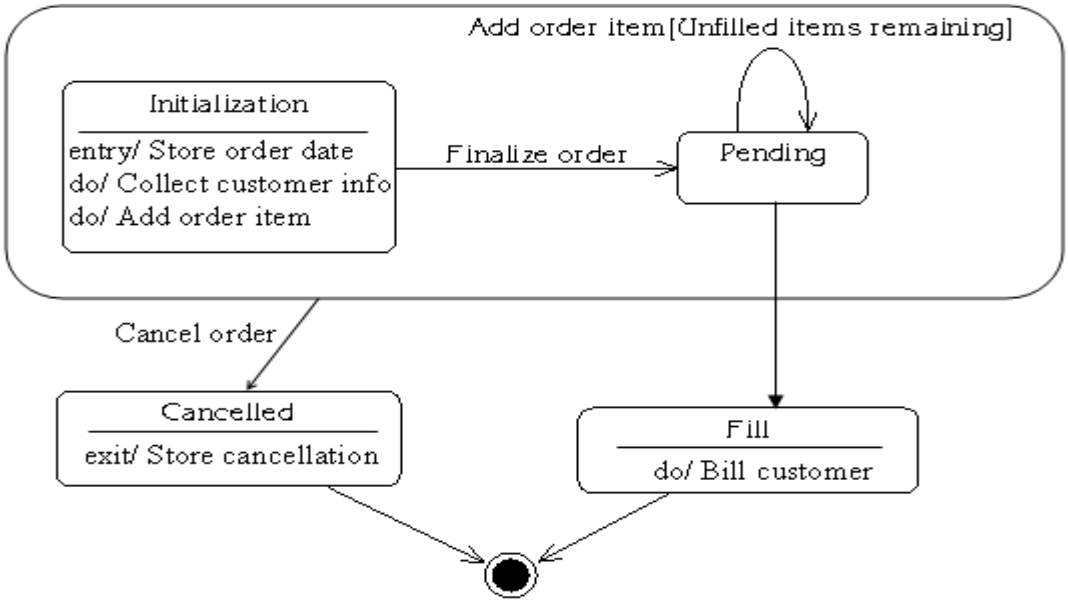
Nhấn phím chuột vào trạng thái sẽ ẩn trong trạng thái mới.

Để sử dụng lịch sử trạng thái ta làm như sau:

15. Mở cửa sổ đặc tả cho trạng thái mong muốn
16. Chọn bảng Detail
17. Đánh dấu hộp States History
18. Nếu ta có trạng thái trong trạng thái ta có thể áp dụng đặc trưng lịch sử cho mọi trạng thái ẩn trong siêu trạng thái. Để thực hiện việc này ta đánh dấu hộp Substates History.

6.3.2 - Thí dụ: Hệ thống bán hàng (tiếp theo)

Trong bài thực hành này chúng ta sẽ lập biểu đồ biến đổi trạng thái cho lớp *Order* của ứng dụng quản lý hóa đơn bán hàng. Hóa đơn bán hàng có thể có nhiều trạng thái: hóa đơn chưa giải quyết (*pending order*), hóa đơn đã giải quyết (*filled order*), hóa đơn hủy bỏ (*cancelled order*). Biểu đồ trạng thái của chúng được vẽ trên hình 6.19



Hình 6.19 Biểu đồ trạng thái lớp Order

6.3.2.1 - Lập biểu đồ

19. Tìm lớp Order trong browser
20. Nhấn phím phải của chuột trên lớp, chọn *Open State Diagram*

6.3.2.2 - Bổ sung trạng thái Start, Stop

21. Chọn công cụ trạng thái Start

22. Đặt chúng trong biểu đồ
23. Làm tương tự cho trạng thái Stop

6.3.2.3 - Bổ sung siêu trạng thái

24. Chọn phím *State* từ thanh công cụ

Đặt trạng thái vào biểu đồ

6.3.2.4 - Bổ sung các trạng thái khác

25. Chọn trạng thái từ thanh công cụ
26. Đặt tả trạng thái vào biểu đồ
27. Đặt tên trạng thái *Cancelled*
28. Chọn trạng thái từ thanh công cụ
29. Đặt trạng thái vào biểu đồ
30. Đặt tên trạng thái *Filled*
31. Chọn trạng thái từ thanh công cụ
32. Đặt trạng thái vào trong siêu trạng thái
33. Đặt tên trạng thái *Initializatiion*
34. Chọn trạng thái từ thanh công cụ
35. Đặt trạng thái vào trong siêu trạng thái
36. Đặt tên trạng thái *Pending*

6.3.2.5 - Bổ sung chi tiết trạng thái

37. Nhấn đúp phím chuột trên trạng thái *Initialization*
38. Chọn *Detail Tab*
39. Nhấn phím phải trên hộp *Actions*
40. Chọn Insert trong thực đơn pop-up
41. Nhấn đúp trên *new action*
42. Đặt tên hành động là *Store Order Date*
43. Đảm bảo rằng *On Entry* được chọn trong hộp soạn thảo *When*
44. Lặp lại các bước 3-7 để bổ sung các hành *động Collect Customer Info, Entry until Exit* và *Add Order Items*, *Entry until Exit*.
45. Nhấn *OK* hai lần để đóng *specification*.
46. Nhấn dúp trên trạng thái *Cancelled*
47. Lặp lại các bước 2-7 để bổ sung hành động: *Store Cancellation Date*, *On Exit*.
48. Nhấn *OK* hai lần để đóng *specification*
49. Nhấn đúp trên trạng thái *Filled*
50. Lặp lại các bước 2-7 để bổ sung hành động: *Bill Customer*, *Entry until Exit*.

51. Nhấn *OK* hai lần để đóng *specification*.

*6.3.2.6 - **Bổ sung quá độ***

52. Chọn phím *Transition* từ thanh công cụ
53. Nhấn trạng thái *Start*
54. Di đường *transition* đến trạng thái *Initialization*
55. Lặp lại bước 1-3 để bổ sung *transitions*:
    Initialization đến *Pending*
    Pending đến *Filled*
    Siêu trạng thái (*uperstate*) đến *Cancelled*
    Cancelled đến trạng thái *End*
    Filled đến trạng thái *End*
56. Chọn phím *Transition to Self* từ thanh công cụ
57. Nhấn trên trạng thái *Pending*

*6.3.2.7 - **Bổ sung chi tiết quá độ***

58. Nhấn đúp trên đường biến đổi trạng thái từ *Initialization* đến *Pending* để mở *specification*
59. Nhập *Finalize Oder* vào ô *Event*
60. Nhấn *OK* để đóng *specification*
61. Lặp lại các bươc 1-3 để bổ sung sự kiện *Cancel Order* vào sườn giữa siêu trạng thái và trạng thái *Cancelled*
62. Nhấn đúp trên sườn từ *Pending* đến *Filled* để mở *specification*
63. Nhập *Add Order Item* vào ô *Event*.
64. Chọn *Detail tab*
65. Nhập *No unfilled items remaining* vào ô *Condition*
66. Nhấn *OK* để đóng *Specification*
67. Nhấp đúp trên sườn phản thân của trạng thái *Pending*
68. Nhập *Add Order Item* vào ô *Event*
69. Chọn *Detail tab*
70. Nhập *Unfilled items remaining* vào ô *Condition*
71. Nhấn *OK* để đóng *Specification*.

CHƯƠNG 7
BIỂU ĐỒ KIẾN TRÚC VẬT LÝ VÀ PHÁT SINH MÃ TRÌNH

Kiến trúc hệ thống là kế hoạch chi tiết của các bộ phận hình thành hệ thống, bao gồm cấu trúc, giao diện và cơ chế được sử dụng để giao tiếp. Xác định kiến trúc tốt cho khả năng dễ dàng tìm kiếm vị trí của các chức năng, khái niệm cụ thể hay dễ dàng xác định vị trí để bổ sung các chức năng và khái niệm mới để phù hợp với hệ thống. UML định nghĩa kiến trúc như sau: "*Kiến trúc* là cấu trúc tổ chức của hệ thống. Kiến trúc có thể tách thành các bộ phận (thành nhiều mức khác nhau) tương tác thông qua giao diện, các quan hệ nối các bộ phận và ràng buộc để lắp ráp các bộ phận". Còn kiến trúc phần mềm được *Buschmann* (1996) định nghĩa như sau: "*Kiến trúc phần mềm* là mô tả các hệ con và các thành phần của hệ thống phần mềm và các quan hệ giữa chúng. Các hệ con và các thành phần được xác định từ các góc nhìn khác nhau để chỉ ra các thuộc tính chức năng và phi chức năng của hệ thống phần mềm. Kiến trúc phần mềm là vật phẩm, là kết quả của hoạt động thiết kế phần mềm".

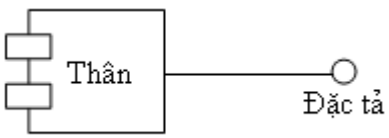
Kiến trúc hệ thống được phân chia thành hai loại: logíc và vật lý. *Kiến trúc logíc* chỉ ra các lớp và đối tượng, các quan hệ và cộng tác để hình thành khía cạnh chức năng của hệ thống. Kiến trúc logíc được mô tả bằng các biểu đồ UC, lớp, trạng thái, trình tự, hợp tác và biểu đồ hoạt động như đã trình bày trong các chương trước đây. Kiến trúc chung là cấu trúc ba tầng, trong đó hệ thống được chỉ ra tầng giao diện, tầng đối tượng tác nghiệp và tầng CSDL. Biểu đồ gói của UML sẽ chỉ ra cấu trúc logíc của chúng bao gồm kiến trúc bên trong của mỗi tầng. *Kiến trúc vật lý* đề cập đến mô tả chi tiết hệ thống về phương diện phần cứng và phần mềm của hệ thống. Nó phản ánh cấu trúc phần cứng, bao gồm các nút khác nhau và chúng nối với nhau như thế nào. Đồng thời nó mô tả cấu trúc vật lý và các phụ thuộc của các mođun mã trình để cài đặt các khái niệm định nghĩa trong kiến trúc logíc; và sự phân bố các phần mềm khi chạy theo tiến trình, chương trình và các thành phần khác.

Kiến trúc vật lý liên quan đến cài đặt, do vậy nó được mô hình hóa trong các biểu đồ cài đặt. Biểu đồ cài đặt trong UML bao gồm biểu đồ thành phần (*compoment diagram*) và biểu đồ triển khai (*deployment diagram*). Biểu đồ thành phần chứa các thành phần phần mềm, bao gồm các đơn vị mã trình và cấu trúc các tệp (mã nguồn và nhị phân). Biểu đồ triển khai chỉ ra kiến trúc hệ thống khi chạy, bao gồm các thiết bị vật lý và các phần mềm đặt trên nó.

7.1 BIỂU ĐỒ THÀNH PHẦN

7.1.1 - Thành phần là gì?

Thành phần là *mô đun vật lý mã trình*, thành phần phần mềm có thể là thư viện mã nguồn và các tệp chạy được. Mặc định mỗi lớp trong mô hình logíc sẽ có phần đặc tả phần thân. Đặc tả chứa ghép nối lớp, thân chứa cài đặt của cùng lớp đó. UML có các biểu tượng đồ họa để gán cho các kiểu thành phần khác nhau (hình 7.1).



Hình 7.1 Biểu đồ thành phần của UML

Dưới đây là mô tả vắn tắt một vài kiểu thành phần của UML

Thành phần mã nguồn. Thành phần mã nguồn có ý nghĩa vào thời điểm dịch chương trình. Thông thường đó là tệp mã nguồn cài đặt một hay nhiều lớp. Thí dụ C++, mỗi tệp *.cpp* và tệp *.h* là thành phần. Trước khi phát sinh mã nguồn, ta phải ánh xạ từng tệp vào thành phần tương ứng; trong C++, mỗi lớp được ánh xạ vào hai tệp (*.cpp* và *.h*).

Thành phần nhị phân. Thành phần nhị phân thường là mã trình có được sau khi dịch thành phần mã nguồn. Nó có thể là tệp mã đối tượng (*obj*), tệp thư viện tĩnh (*lib*) hay tệp thư viện động (*Dynamic Linking library-dll*). Thành phần nhị phân có ý nghĩa vào thời điểm liên kết, hoặc thời điểm chạy chương trình (thí dụ thư viện động).

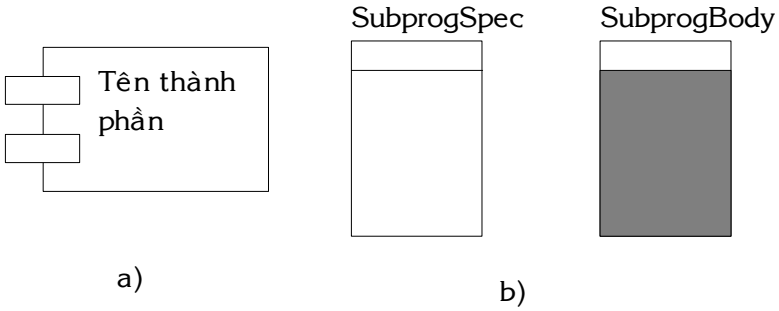
Thành phần khả thi. Thành phần thực hiện được là tệp chương trình thực hiện được (các tệp .EXE), là kết quả của liên kết các thành phần nhị phân (liên kết tĩnh hay liên kết động). Thành phần thực hiện được biểu diễn đơn vị thực hiện được chạy trên bộ xử lý (máy tính).

Một khi thành phần được tạo lập, ta có thể gộp chúng vào biểu đồ thành phần và vẽ các quan hệ giữa chúng. Giữa các thành phần chỉ có một loại quan hệ phụ thuộc. Kết nối quan hệ giữa các thành phần cho biết một thành phần cần một thành phần khác để có được định nghĩa đầy đủ. Thí dụ, kết nối phụ thuộc cho biết thành phần này phải được dịch trước thành phần kia. Thí dụ khác là với thành phần khả thi thì kết nối phụ thuộc được sử dụng để nhận ra thư viện động nào cần phải có để thành phần khả thi chạy được.

7.1.2 - Biểu tượng thành phần trong Rational Rose

Trong Rose có một số biểu tượng để thực hiện thành phần, đó là:

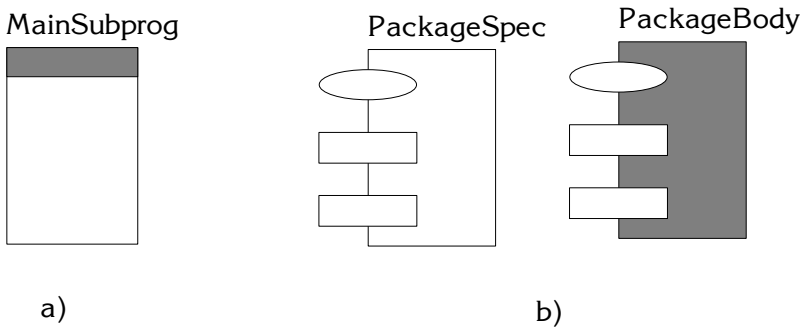
Thành phần. Biểu tượng thành phần được sử dụng để biểu diễn mô đun phần mềm có giao diện (hình 7.2a). Trong đặc tả thành phần có xác định kiểu *stereotype (ActiveX, Applet, DLL, exe…).*



Hình 7.2 Biểu tượng thành phần và chương trình con

Đặc tả và thân chương trình con. Đó là loại biểu tượng được sử dụng cho đặc tả (*SubprogSpec*) và cài đặt của chương tring con (*SubprogBody*). Hình 7.2b thể hiện các biểu tượng cho các thành phần này. Chương trình con không chứa định nghĩa lớp.

Chương trình chính. Là tệp chứa điểm vào chương trình (hình 7.3b). Thí dụ trongn chương trình C/C++ đó là tệp chứa hàm *main()* hay trong chương trình Visual C++ thì đó là tệp chứa hàm *WinMain()*.

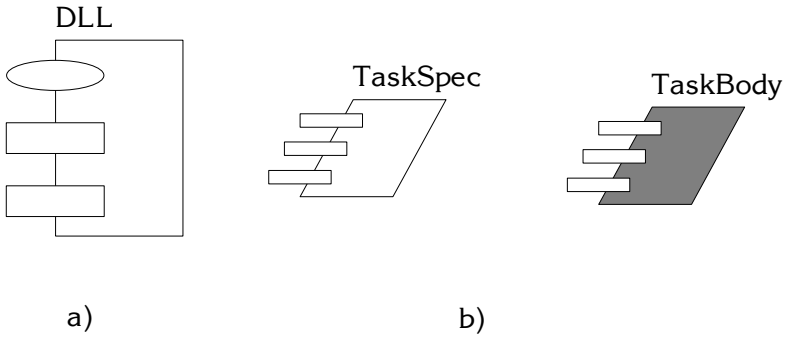


Hình 7.3 Biểu tượng chương trình chính và gói

Đặc tả và thân gói. Gói là cài đặt của lớp. Đặc tả gói (*PackageSpec*) là tệp *header,* chứa thông tin về nguyên hàm của lớp. Trong C++, đặc tả gói là tệp với phần mở rộng tên H. Thân gói (*PackageBody*) chứa mã lệnh của các thao tác của lớp. Trong C++, thân gói là tệp .*CPP* (hình 7.3b).

Các biểu tượng thành phần áp dụng cho các thành phần run-time (các tệp *exe, dll và task*) như sau:

Tệp thư viện động (*DLL*). Biểu tượng cho tệp *DLL* như trên hình 7.4a.

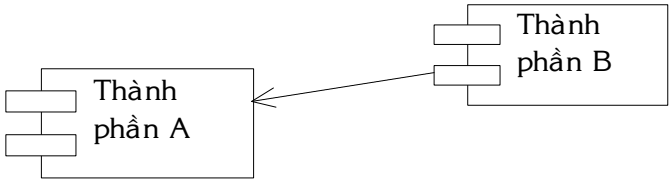


Hình 7.4 Biểu tượng thư viện động và nhiệm vụ

Đặc tả nhiệm vụ và thân. Các biểu tượng này biểu tượng các gói co luồng điều khiển độc lập. Thí dụ, tệp khả thực (*exe*) được biểu diễn như đặc tả nhiệm vụ (*TaskSpec*) với mở rộng *exe* (*TaskBody*) như trên hình 7.4b.

7.1.3 - Phụ thuộc thành phần

Chỉ có một kiểu quan hệ phụ thuộc giữa các thành phần trong biểu đồ loại này. Phụ thuộc thành phần cho rằng một thành phần phụ thuộc vào thành phần khác. Phụ thuộc thành phần được vẽ như hình 7.5.



Hình 7.5 Phụ thuộc thành phần

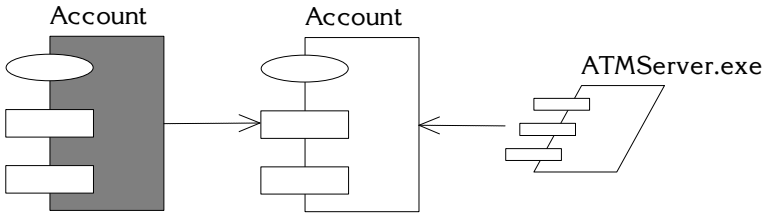
Trong ví dụ trên hình 7.5 ta thấy thành phần A phụ thuộc thành phần B. Nói cách khác là một vài lớp trong A phụ thuộc một vài lớp trong B. Do vậy, không thể biên dịch A nếu chưa biên dịch

B. Tránh phụ thuộc vòng trong biểu đồ loại này. Thí dụ, nếu A phụ thuộc B và B phụ thuộc A thì sẽ không biên dịch được chúng. Do vậy phải loại bỏ phụ thuộc vòng trước khi phát sinh mã trình. Phụ thuộc cho biết cái nào dễ sử dụng lại và cái nào không. Thí dụ trê cho thấy A phụ thuộc B thì không thể sử dụng lại A nếu không sử dụng B, còn B thì dễ được sử dụng lại vì nó không phụ thuộc vào bất cứ thành phần nào

7.1.4 - Biểu đồ thành phần

Biểu đồ thành phần là biểu đồ UML hiển thị các thành phần của hệ thống và phụ thuộc giữa chúng. Thí dụ, hình 7.6 là biểu đồ thành phần của hệ thống rút tiền tự động ATM. Trên biểu đồ thành phần ta có thể nhìn thấy mã nguồn và thành phần run-time trong hệ thống. Thí dụ này cho thấy hệ thống có thành phần mã nguồn là các tệp *account.h* và *account.cpp*, tệp khả thi là *ATMServer.exe*. Với biểu đồ này, người phát triển thực hiện dịch, triển khai hệ thống sẽ biết thư viện mã trình nào tồn tại và tệp khả thi (*.exe*) nào sẽ được sẽ được tạo ra khi mã trình được dịch. Các phụ thuộc thành phần (mũi tên nét đứt) cho biết thành phần sẽ được dịch theo thứ tự nào.

Sau khi tạo biểu đồ thành phần thì ta bổ sung phần tử đồ họa thành phần vào chúng. Trong C++, các thành phần hay được dùng nhất là đặc tả gói (*Package Specification*), thân gói (*Package Body*) và thành phần thực hiện (*Executuon*). Như đã trình bày trên đây, biểu tượng *Package Specification* hoặc *Component* được sử dụng cho tệp *.DLL*. Biểu tượng *Package Body* dành cho tệp *CPP*. Cũng có thể gắn tài liệu vào thành phần để mô tả mục đích của thành phần hay mô tả các lớp trong thành phần.



Hình 7.6 Biểu đồ thành phần

Tương tự như lớp, có thể tổ chức các thành phần thành gói. Thông thường tạo gói khung nhìn thành phần cho mỗi gói khung nhìn logíc. Thí dụ, trong hệ thống quản lý bán hàng ta có các gói khung nhìn logic *Orders* chứa các lớp *Order, OrderItem và OrderForm*. Do vậy, gói khung nhìn thành phần tương ứng cần phải chứa các thành phần dành cho các lớp *Order, OrderItem và OrderForm.*

Tương tự như các phần tử mô hình khác trong UML, ta có thể bổ sung các đặc tả chi tiết vào mỗi thành phần, chúng bao gồm:

Stereotype. *Stereotype* điều khiển biểu tượng nào sẽ được sử dụng để biểu diễn thành phần. Chúng có thể là một trong các lựa chọn sau: *<none>*, đặc tả chương trình con, thân chương trình con, chương trình chính, đặc tả gói, thân gói, khả thực, *DLL*, đặc tả nhiệm vụ và thân nhiệm vụ. Ngoài ra trong Rose còn có *stereotype* cho *ActiveX, Applet*, ứng dụng… Ta cũng có thể tạo *stereotype* mới, phù hợp với loại ngôn ngữ lập trình và ứng dụng cụ thể.

Ngôn ngữ. Trong Rose có thể gán ngôn ngữ cho từng thành phần. Do vậy có khả năng phát sinh ra các ngôn ngữ khác nhau cho từng thành phần mô hình, thí dụ phát sinh một thành phần mô hình ra C++, một thành phần mô hình khác ra *Java*...

Khai báo. Các khai báo (*declaration*) phụ được gộp vào mã trình cho mỗi thành phần. Các khai báo bao gồm các lệnh phụ thuộc ngôn ngữ để khai báo biến, lớp … Lệnh *#include* của C++ cũng được xem như khai báo.

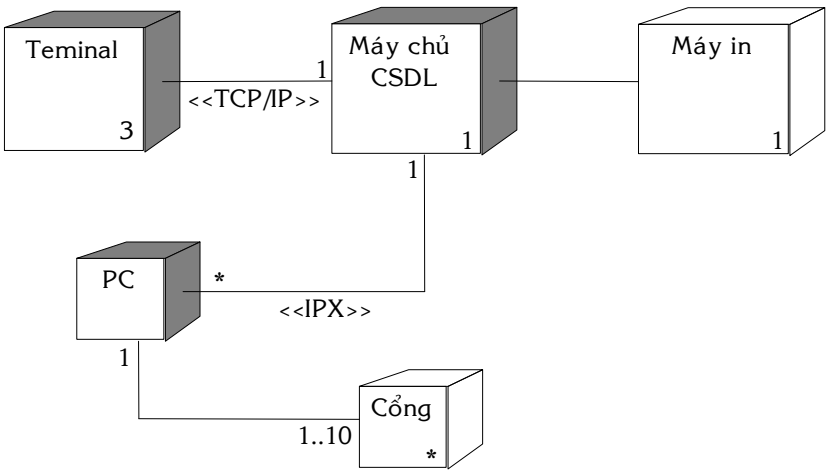
Lớp. Trước khi phát sinh mã trình cho lớp thì phải ánh xạ nó vào thành phần. Điều này cho Rose biết mã trình của lớp sẽ được ghi vào tệp nào. Có thể ánh xạ một hay nhiều lớp vào mỗi thành phần.

7.2 BIỂU ĐỒ TRIỂN KHAI

Biểu đồ triển khai mô tả kiến trúc hệ thống phần cứng (nút) khác nhau như bộ xử lý, các thiết bị và các thành phần phần mềm thực hiện nên kiến trúc đó. Nó là mô tả vật lý của tôpô hệ thống, mô tả cấu trúc của các đơn vị phần cứng và phần mềm chạy trên nó. Biểu đồ triển khai chỉ ra toàn bộ các nút trên mạng, kết nối giữa chúng và các tiến trình chạy trên chúng.

Nút là đối tượng vật lý (các thiết bị) có tài nguyên tính toán. Chúng có thể là máy tính, máy in, đọc thẻ từ, thiết bị truyền tin… Các nút được kết nối thông qua kết hợp giao tiếp. Các nút trao đổi thông điệp hay đối tượng theo đường dẫn kết nối. Kiểu giao tiếp được thể hiện bằng *stereotype,* chỉ ra thủ tục giao tiếp hay mạng được sử dụng.

Biểu đồ triển khai có thể hiển thị các lớp nút hay hiện thực nút. Tương tự như mọi biểu đồ khác, phân biệt giữa lớp và đối tượng cài đặt lớp bằng tên gạch chân.

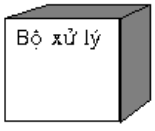


Hình 7.7 Biểu đồ triển khai của hệ thống đóng/mở cửa

Hình 7.7 là thí dụ về biểu đồ triển khai hệ thống điều khiển đóng/mở cửa trong một tòa nhà. Biểu đồ cho thấy hệ thống bao gồm máy chủ nối với các máy tính PC điều khiển đóng/mở cửa ra vào. Biểu đồ chưa xác định tổng số máy tính PC. Nhưng biểu đồ cho thấy một máy tính PC có thể điều khiển nhiều nhất 10 cửa ra vào. Ba máy tính đầu cuối (*terminal*) được sử dụng để xâm nhập hệ thống. Máy in được nối trực tiếp vào máy tính chủ. Biểu đồ còn thể hiện các giao tiếp giữa các nút khác nhau. Máy chủ và các máy PC kết nối với nhau thông qua giao thức IPX, kết nối máy tính đầu cuối với máy chủ thông qua giao thức TCP/IP. Kết nối giữa các nút còn lại trên biểu đồ chưa xác định.

7.2.1 - Phần tử mô hình của biểu đồ

Bộ xử lý (*processor*). Bộ xử lý là máy xử lý. Các máy chủ (*server*), trạm làm việc…thuộc loại này. Ký pháp trong Rose của bộ xử lý như sau:



Chi tiết cho bộ xử lý. Trong đặc tả bộ xử lý có thể bổ sung các thông tin: *stereotype,* đặc tính và *scheduling.*

Stereotype: Tương tự như thành phần mô hình, sử dụng nó để phân nhóm bộ xử lý.

Đặc tính: Là mô tả vật lý của bộ xử lý, thí dụ tốc độ và dung lượng nhớ.

Scheduling: Mô tả loại lập biểu thời gian (*scheduling*) xử lý, bao gồm: *Preemptive* (tiến trình có mức ưu tiên cao hơn có thể chiếm quyền ưu tiên của các tiến trình có mức ưu tiên thấp hơn), *Non Preemptive* (các tiến trình không có mức ưu tiên, tiến trình đang thực hiện chỉ dừng khi nó tự kết thúc), *Cyclic* (chỉ ra chu kỳ điều khiển giữa các tiến trình), *Executive* (một vài thuật toán tính toán điều khiển *scheduling*), *Manual* (tiến trình được người sử dụng điều khiển).

Thiết bị. Thiết bị là máy móc hay bộ phận phần cứng không phải là bộ xử lý trung tâm. Thiết bị bao gồm màn hình, máy in…Cả thiết bị và bộ xử lý được xem như *nút* trong mạng. Ký pháp của thiết bị trong UML như sau:

Chi tiết thiết bị.Tương tự bộ xử lý, thiết bị cũng có nhiều thông tin chi tiết như *stereotype* và đặc tính. Sự khác nhau giữa thiết bị và bộ xư lý phụ thuộc mạnh vào quan điểm. Máy tính đầu cuối nối với máy chủ được người sử dụng xem như thiết bị, nhưng đôi khi người khác lại coi nó là bộ xử lý.

Kết nối. Kết nối là liên kết vật lý giữa hai bộ xử lý, hai thiết bị hay giữa thiết bị và bộ xử lý. Thông thường, kết nối biểu diễn kết nối mạng vật lý giữa các nút trong mạng. Chúng cũng có thể liên kết *Internet* giữa các nút. Có thể gán *stereotype* cho kết nối, đồng thời nó cũng được gán đặc tính nói về chi tiết của kết nối vật lý.

7.2.2 - Tiến trình

Tiến trình là luồng thực hiện đơn chạy trong bộ xử lý. Một tệp khả thi được xem là tiến trình. Tiến trình có thể được hiển thị trong biểu đồ triển khai, nếu được hiển thị thì nó được liệt kê ngay dưới bộ xử lý nơi nó chạy. Các tiến trình được gán mức ưu tiên. Nếu bộ xử lý mà trên đó các tiến trình chạy sử dụng *scheduling* theo mức ưu tiên. Thì mức ưu tiên của tiến trình sẽ được xác định khi nó chạy.

7.3 THỰC HÀNH

7.3.1 - Sử dụng Rational Rose

7.3.1.1 - Tạo lập và hủy bỏ biểu đồ thành phần

Biểu đồ thành phần được lập ra trong khung nhìn thành phần theo các bước như sau:

1. Trong *Browser*, nhấn chuột phải trên gói chứa *Component diagram*
2. Chọn thực đơn New->Component Diagram
3. Nhập tên cho biểu đồ thành phần mới

Hủy bỏ biểu đồ thành phần theo các bước sau:

1. Trong Browser, nhấn chuột phải trên Component diagram

2. Chọn thực đơn Delete

7.3.1.2 - Bổ sung thành phần

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

1. Chọn phím *Component* từ thanh công cụ
2. Nhấn trên biểu đồ nơi định vẽ thành phần
3. Nhập tên cho thành phần mới

Hủy bỏ thành phần khỏi biểu đồ như sau:

1. Nhấn phím phải trên thành phần trong *Browser*
2. Nhấn các phím *Ctrl+D*

7.3.1.3 - Bổ sung chi tiết thành phần

Gán *stereotype* như sau:

1. Mở cửa sổ đặc tả trong thành phần mong muốn
2. Chọn bảng *General*
3. Nhập stereotype vào trường *Stereotype*.

Gán ngôn ngữ như sau:

1. Mở cửa sổ đặc tả thành phần mong muốn
2. Chọn bảng *General*
3. Nhập ngôn ngữ trong cửa sổ *Language*.

Để ánh xạ lớp cho thành phần:

1. Mở cửa sổ đặc tả thành phần mong muốn
2. Chọn bảng *Realizes*
3. Nhấn phím phải trên lớp để ánh xạ
4. Chọn thực đơn *Assign.*

7.3.1.4 - Gắn phụ thuộc thành phần

Bổ sung phụ thuộc thành phần như sau:

1. Chọn biểu tượng *Dependency* trên thanh công cụ
2. Di đường phụ thuộc từ thành phần *Client* đến thành phần *Supplier*.

7.3.1.5 - Tạo lập biểu đồ triển khai

Để mở biểu đồ triển khai ta làm như sau:

1. Nhấn đúp phím chuột trên khung nhìn triển khai trong *brower*
2. *Rose* sẽ mở *Deployment diagram* cho mô hình.

7.3.1.6 - Bổ sung và bãi bỏ bộ xử lý (proccessor)

Bổ sung bộ xử lý vào biểu đồ sau:

1. Chọn phím *Processor* từ thanh công cụ
2. Nhấn trên biểu đồ triển khai để đặt bộ xử lý
3. Nhập tên cho bộ xử lý.

Hủy bỏ bộ xử lý khỏi biểu đồ như sau:

1. Chọn bộ xử lý trong biểu đồ
2. Nhấn phím *Delete*

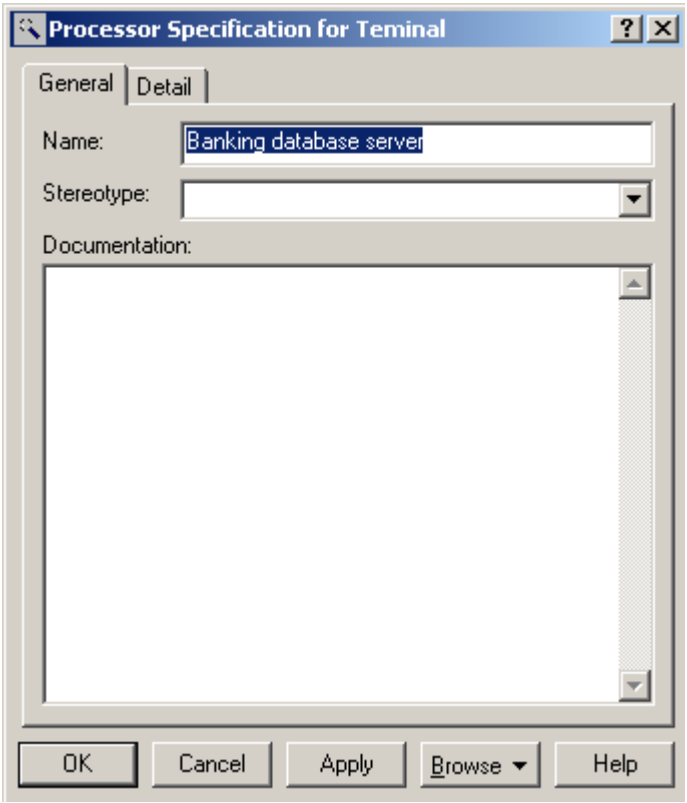
Hủy bỏ bộ xử lý khỏi mô hình như sau:

1. Chọn bộ xử lý trong *Deloyment diagram*
2. Nhấn phím *Ctrl+D*.

7.3.1.7 - Bổ sung chi tiết cho bộ xử lý

Để gán *stereotype* cho bộ xử lý ta làm như sau:

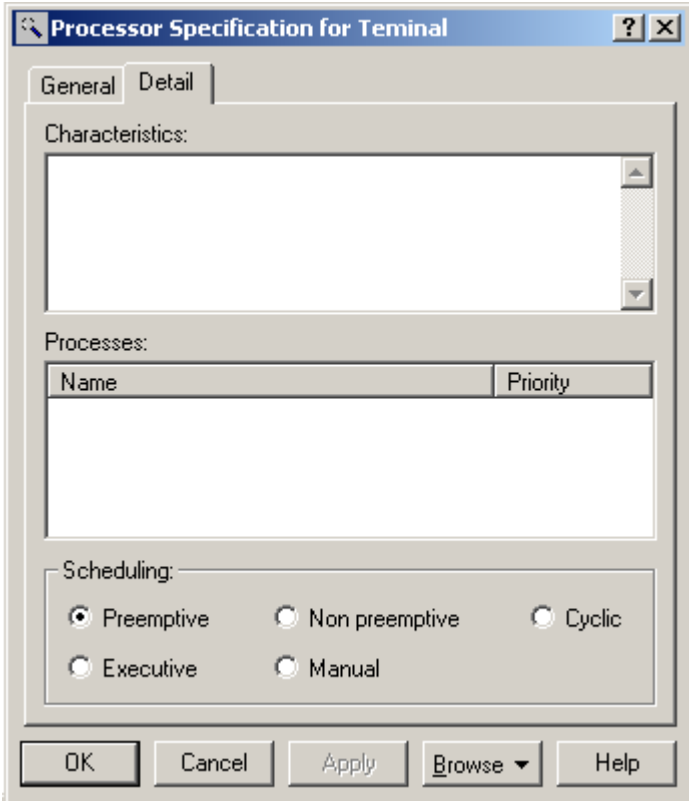
1. Mở cửa sổ đặc tả bộ xử lý mong muốn
2. Chọn bảng *General* (hình 7.8)



Hình 7.8 Cửa sổ đặc tả bộ xử lý

3. Nhập *stereotype* vào cửa sổ *Stereotype*

Bổ sung đặc tính và scheduling cho bộ xử lý được thực hiện tương tự như trên nhưng lần này chọn bảng *Detail* (hình 7.9)



Hình 7.9 Bổ sung đặc tính

7.3.1.8 - Bổ sung thiết bị

Bổ sung thiết bị vào biểu đồ triển khai như sau:

1. Chọn phím *Device* từ thanh công cụ
2. Nhấn trên biểu đồ triển khai để đặt thiết bị
3. Nhập tên cho thiết bị mới

7.3.1.9 - Bổ sung chi tiết thiết bị

Thực hiện bổ sung chi tiết thiết bị tương tự bổ sung chi tiết cho bộ xử lý trên đây.

7.3.1.10 - Bổ sung kết nối

Bổ sung kết nối vào biểu đồ triển khai như sau:

Chọn phím *Connection* từ thanh công cụ

Nhấn trên nút trong biểu đồ triển khai để nối

Di đường kết nối đến nút khác.

7.3.1.11 - Bổ sung chi tiết kết nối

Có thể gán *stereotype* cho kết nối như sau:

1. Mở cửa sổ đặc tả cho kết nối mong muốn

2. Chọn bảng *General*
3. Nhập stereotype trong hộp *Stereotype*

7.3.1.12 - Bổ sung và hủy bỏ tiến trình

Trình tự bổ sung tiến trình như sau:

1. Nhấn phím phải chuột trên bộ xử lý mong muốn trong *Browser*
2. Chọn thực đơn *New->Process*
3. Nhập tên cho tiến trình mới

Trình tự hủy bỏ tiến trình mới như sau:

1. Nhấn phím phải chuột trên tiến trình mong muốn trong Browser
2. Chọn thực đơn Delete.

7.3.2 - Phát sinh mã trình bằng *Rose*

Có sáu bước cơ bản để phát sinh mã trình, bao gồm:

1. Kiểm tra mô hình
2. Tạo lập thành phần
3. Thực hiện ánh xạ lớp vào thành phần
4. Đặt thuộc tính phát sinh mã trình
5. Chọn lớp, thành phần hày gói
6. Phát sinh mã trình.

Không phải ngôn ngữ nào cũng cần đầy đủ các bước trên. Thí dụ C++ không cần bước hai, hay không ngôn ngữ nào cũng bắt buộc hiện bước một. Tuy vậy, khuyến cáo là nên thực hiện đầy đủ năm bước đầu trước khi phát sinh mã trình.

7.3.2.1 - Bước 1: Kiểm tra mô hình

Rose có chức năng kiểm tra mô hình độc lập ngôn ngữ để bảo đảm mô hình là phù hợp trước khi phát sinh mã trình. Chúng ta nên thực hiện chức năng này trước khi phát sinh mã trình. Nó có thể tìm ra tính không thống nhất và lỗi trong mô hình, chúng ảnh hưởng đến sai xót trong mã trình.

Trình tự thực hiện như sau:

1. Chọn *Tools-> Check Model*
2. Lỗi mô hình sẽ được hiển thị trong cửa sổ *log.*

Các lỗi hay xảy ra là thông điệp trong biểu đồ tương tác không được ánh xạ vào thao tác hay đối tượng trong biểu đồ tương tác không được ánh xạ vào lớp. Chú ý các đối tượng trong biểu đồ trình tự và biểu đồ cộng tác, mỗi hợp chữ nhật phải có tên đối tượng, dấu **:**, tên lớp.

Vi phạm (*violation*) xâm nhập. Thực đơn *Check Model* sẽ tìm ra hầu hết tính không chắc chắn và các vấn đề xảy ra trong mô hình. Thực đơn *Access Violation* sẽ tìm ra vi phạm khi có quan hệ giữa hai lớp trong hai gói khác nhau, nhưng không có quan hệ giữa hai gói. Thí dụ, nếu lớp *Order* trong gói *Entities* có quan hệ với lớp *OrderManager* trong gói *Control* thì phải có quan hệ giữa hai gói *Entities* và *Control*.

Tìm kiếm vi phạm xâm nhập như sau:

1. Chọn Report-> Show Access Violations
2. Rose sẽ hiển thị vi phạm xâm nhập trong cửa sổ.

Kiểm tra ngôn ngữ. Để thực hiện kiểm tra ngôn ngữ hãy chọn thực đơn *Tools>Check Model*. Kiểm tra này cho biết lỗi xảy ra như nhiều lớp *public* lại đặt vào một đơn vị dịch.

7.3.2.2 - Bước 2: Tạo lập thành phần

Bước thứ hai trong tiến trình phát sinh mã lệnh là tạo lập các thành phần để chứa các lớp. Có nhiều loại thành phần như các tệp mã nguồn, tệp *exe*, tệp thư viện, *ActiveX*, *applets*…Trước khi phát sinh mã trình phải ánh xạ các lớp vào thành phần mã nguồn tương ứng. Một khi các thành phần đã được tạo lập, ta có gắn các phụ thuộc giữa chúng trong biểu đồ thành phần. Các phụ thuộc giữa các thành phần là phụ thuộc biên dịch trong hệ thống. Nếu phát sinh mã trình *C++, Java, VisualBasic* thì bước này phải được thực hiện.

Để tạo lập thành phần ta làm như sau:

1. Mở *Comonent diagram*
2. Sử dụng biểu tượng *Component* trong thanh công cụ để bổ sung thành phần mới vào biểu đồ.

7.3.2.3 - Bước 3: Ánh xạ lớp vào thành phần

Mỗi thành phần mã nguồn biểu diễn tệp mã nguồn cho một vài lớp. Trong C++, mỗi lớp được ánh xạ đến hai thành phần mã nguồn: tệp *header* và tệp chứa thân lớp. Bước thứ ba này thực hiện ánh xạ mỗi lớp vào các thành phần thích ứng. Với C++ thì bước này là tùy chọn. Rose có thể phát sinh mã trình cho nó mà không đi qua bước này.

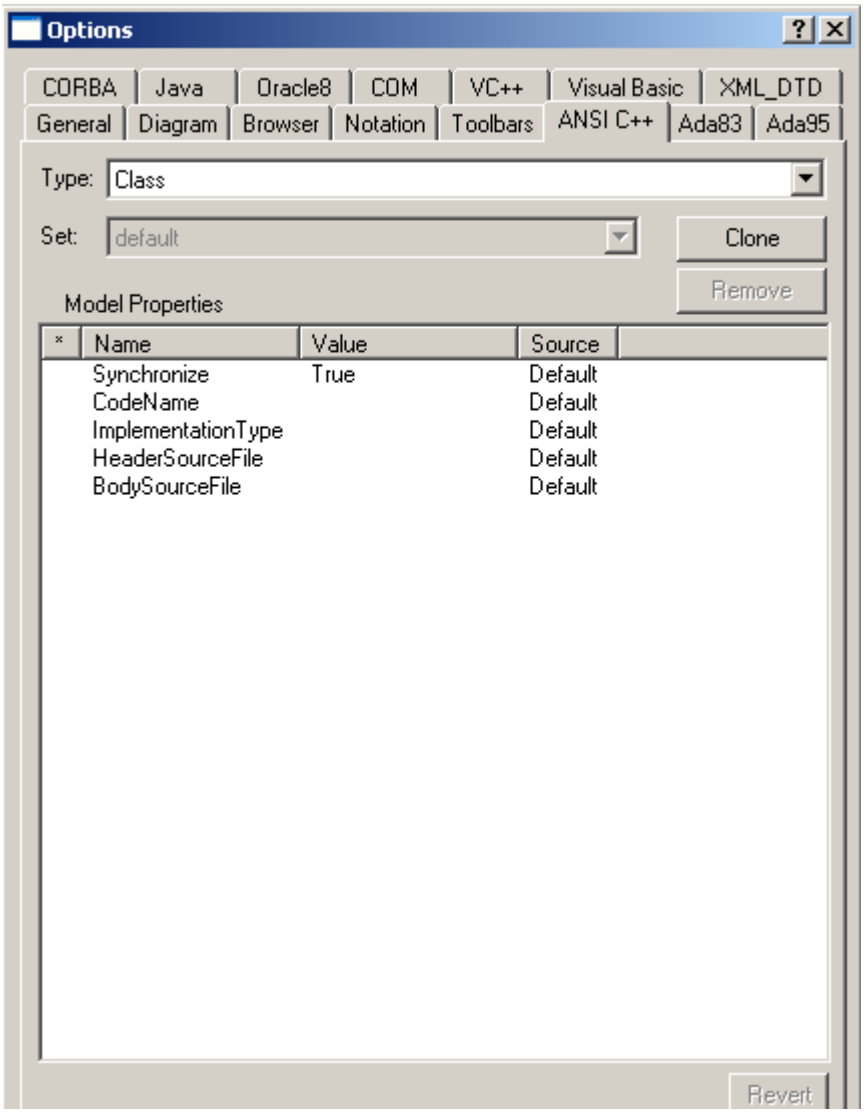
Để ánh xạ lớp vào thành phần ta làm như sau:

1. Nhấn phím phải trên thành phần trong biểu đồ thành phần hay *browser*.
2. Chọn open Specification
3. Chọn bảng *Realizes*
4. Nhấn phím phải trên lớp (những lớp) thích úng và chọn *Assign*
5. *Browser* sẽ chỉ ra tên thành phần trong dấu <<>> sau tên lớp trong *Logical View*.

7.3.2.4 - Bước 4: Đặt các thuộc tính cho phát sinh mã trình.

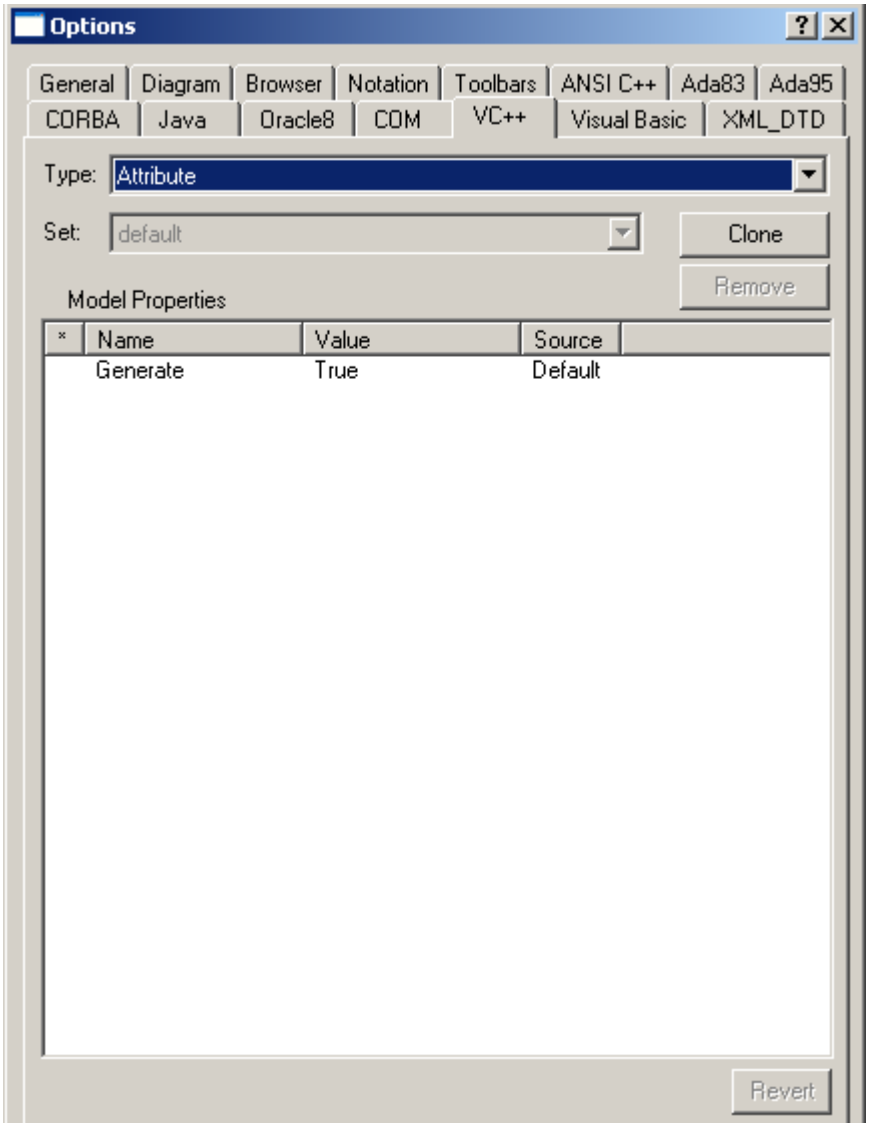
Có nhiều thuộc tính cho phát sinh mã nguồn có thể gán cho lớp, thuộc tính và các thành phần khác nhau của mô hình. Các thuộc tính này điều khiển mã trình sẽ được phát sinh như thế nào. *Rose* cung cấp bộ thuộc tính mặc định. Thí dụ, một đặc tính cho phát sinh mã trình của thuộc tính C++ là *GenerateOperation*, nó điều khiển các *Get( )* sẽ được phát sinh cho thuộc tính này hay không. Trước khi phát sinh mã trình nên xem xét đặc tính phát sinh mã trình và thay đổi chúng nếu cần.

Để quan sát đăc tính phát sinh mã trình hãy chọn *Tools>Options*, sau đó chọn bảng ngôn ngữ thích ứng (hình 7.10).



Hình 7.10 Đặc thuộc tính

Từ cửa sổ type ta có thể chọn thành phần mô hình như *Class, Attribute, Operation*…Mỗi ngôn ngữ có thành phần mô hình khác nhau. Hình 7.11 là chọn thành phần Attribute để thay đổi đặc tính.

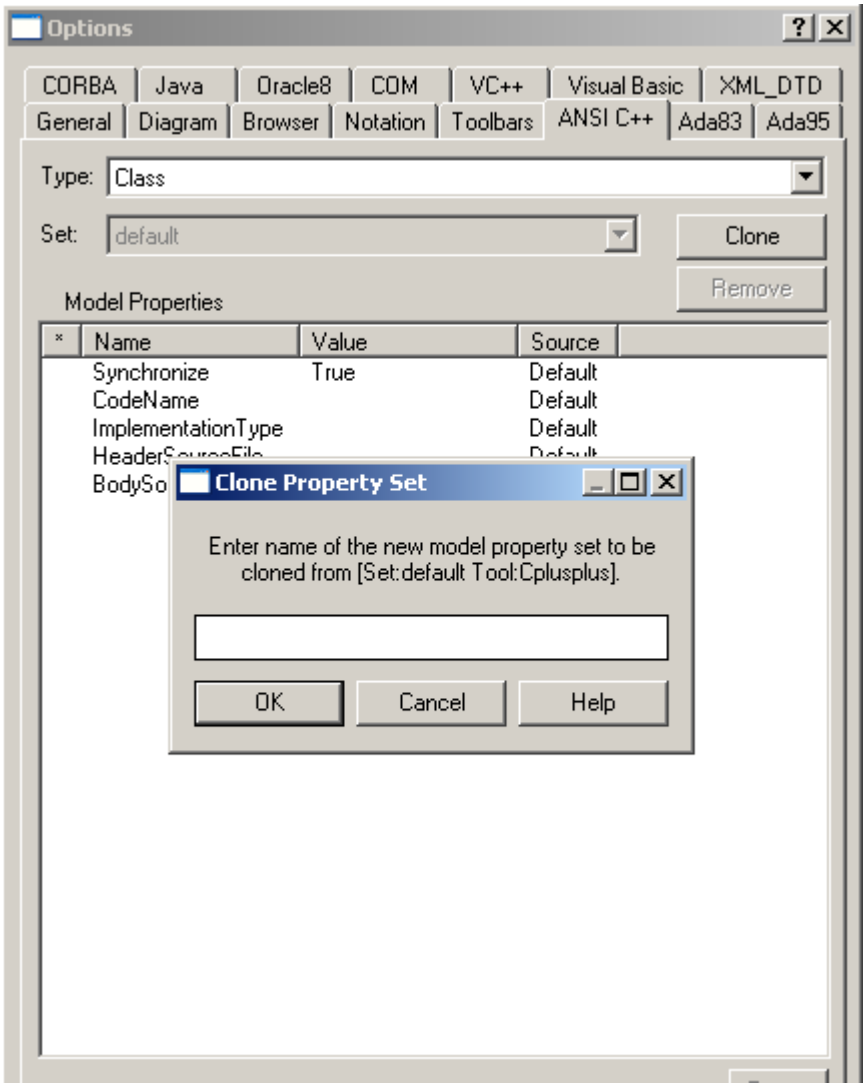


Hình 7.11 Đặt thuộc tính (tiếp)

Mọi thay đổi tại đây ảnh hưởng đến toàn bộ mô hình. Thí dụ nếu chọn đặc tính lớp là *GenerateDefaultConstructor* thì nó sẽ tác động đến mọi đối tượng trong mô hình. Nếu chọn đặc tính phát sinh mã trình cho một thành phần như lớp, thuộc tính… hãy mở cửa sổ *Specification* cho nó. Chọn bảng ngôn ngữ (C++) và thay đổi đặc tính tại đây. Mọi thay đổi trong cửa sổ này chỉ tác động đến một lớp mà thôi.

Tập đặc tính tạm thời. Thay vì thay đổi trực tiếp tập đặc tính mặc định, ta có thể tạo lập tập đặc tính tạm thời để sử dụng. Để lập tập đặc tính tạm thời ta chọn thực đơn *Tools>Model Properties>Edit>C++ tab>>Edit Set*, sau đó nhấn phím *Clone* trong cửa sổ *Clone the Property Set*. Rose sẽ yêu cầu nhập tên cho tập mới này.

Mở điều khiển *Set* trong *Clone the Property Set* để thay đổi thuộc tính nào cần thay đổi (hình 7.12). Tập đặc tính này sẽ không ảnh hưởng đến tập đặc tính mặc định.



Hình 7.12 Đặc tính tạm thời

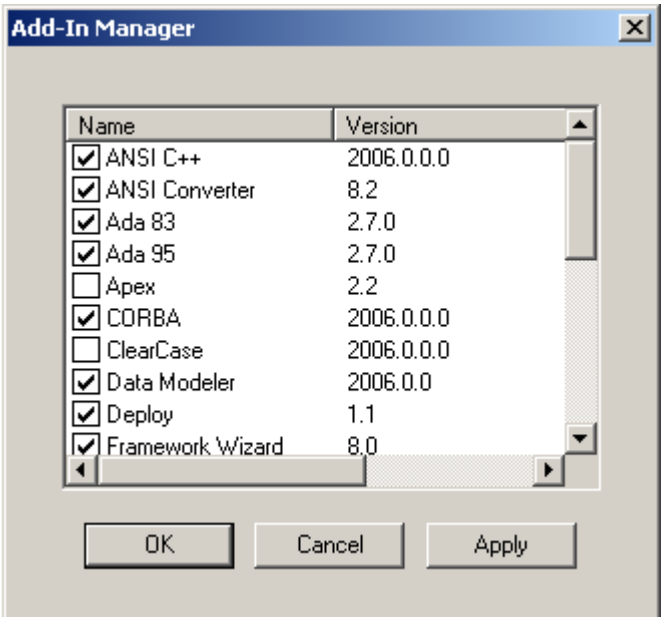
Bỏ tập đặc tính tạm thời. Khi không còn cần đến tập đặc tính tạm thời, thì có thể bãi bỏ nó khỏi mô hình theo trình tự sau: chọn thực đơn *Tools>Options*; chọn bảng ngôn ngữ thích hợp; chọn tập đặc tính nhái trong hộp danh sách *Set*; nhấn *Remove* để bãi bỏ chúng.

7.3.2.5 - Bước 5: Chọn lớp, thành phần hay gói

Khi phát sinh mã trình ta có thể phát sinh cho lớp, thành phần hay gói vào cùng thời điểm. Mã trình có thể phát sinh từ biểu đồ hay *browser*. Nếu phát sinh mã từ gói, thì có thể chọn gói *Logical view* trên biểu đồ lớp hay chọn gói *Component view* trên biểu đồ thành phần. Cũng có thể phát sinh mã trình đồng thời cho nhiều lớp, thành phần hay gói.

7.3.2.6 - Bước 6: Phát sinh mã trình

Nếu cài đặt *Rose Enterprise* thì có thể lựa chọn nhiều ngôn ngữ để phát sinh mã. Để hiển thị hay ẩn các lựa chọn, hãy chọn *Add-Ins>Add-In Manager* trong thực đơn *Tools*. Hộp thoại lựa chọn sẽ xuất hiện như hình 7.13.



Hình 7.13 Quản lý mã đích

Khi chọn lớp hay thành phần trên biểu đồ, hãy chọn ngôn ngữ thích ứng trong thực đơn phát sinh mã trình. Nếu có lỗi xảy ra trong quá trình phát sinh mã trình, chúng được hiển thị trên cửa sổ log.

7.3.2.7 - Cái gì đã được phát sinh?

Khi phát sinh mã trình, *Rose* tập hợp các thông tin từ *Logical view* và *Component view* của mô hình. Thực tế là không có công cụ mô hình hóa nào phát sinh mã trình đầy đủ. Rose cũng chỉ phát sinh khung chương trình như sau:

Class. Mọi lớp trong biểu đồ sẽ được phát sinh mã trình.

Attributes. Mã trình sẽ có thuộc tính của lớp, bao gồm miền tác động, loại dữ liệu và giá trị mặc định.

Operation signatures. Các thao tác được khai báo trong mã trình kèm theo tham số, kiểu dữ liệu tham số và kiểu giá trị cho lại các thao tác.

Relationships. Một vài quan hệ trong mô hình sẽ phát sinh thuộc tính khi phát sinh mã.

Components. Mỗi thành phần sẽ được cài đặt trong tệp mã nguồn thích hộp. Khi đã phát sinh tệp, hai bước còn lại là: 1) người phát triển thu nhận tệp và lập trình các thao tác của lớp; 2) thiết kế giao diện đồ họa.

Rational Rose của Rational Software Corporation không hướng vào thiết kế giao diện đồ họa. Hãy sử dụng ngôn ngữ lập trình để thiết kế màn hình và báo cáo. Khi phát sinh mã trình, *Rose* sử dụng cấu trúc gói trong *Component view* để tạo ra các danh mục thích ứng. Mặc định, danh mục gốc cho mã trình phát sinh là nơi ứng dụng *Rose*. Có thể thay đổi danh mục thông qua đặc tính phát sinh mã nguồn. Nếu không có cấu trúc thành phần, *Rose* sẽ sử dụng cấu trúc gói trong *Logical view* để tạo lập cấu trúc danh mục.

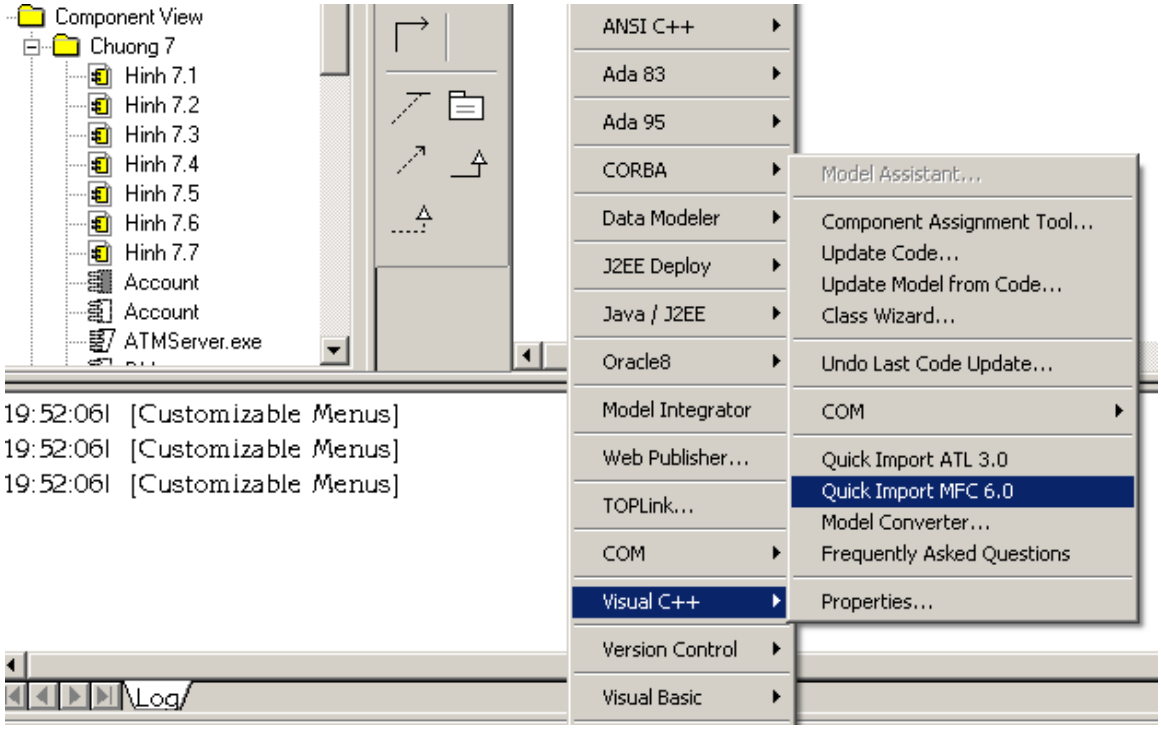
Khi gặp khó khăn chuyển đổi từ mô hình sang mã trình, hãy kiểm tra kỹ lưỡng phiên bản Rose cài đặt trên máy tính có phù hợp yêu cầu hay không. *Rose* có một số phiên bản chính như sau:

- Rose Modeler. Cho phép tạo lập mô hình nhưng không cho khả năng phát sinh mã trình hay chuyển đổi ngược từ mã trình thành mô hình.
- Rose Professional. Cho phép phát sinh mã trình cho một loại ngôn ngữ.
- Rose Enterprise. Cho phép phát sinh mã trình trong nhiều ngôn ngữ khác nhau như C++, Java, Visual Basic, lược Oracle8…

7.3.3 - Rational Rose và Visual C++

Rational Rose là công cụ hiển thị giúp phân tích và thiết kế hệ thống phần mềm. Visual C++ là môi trường lập trình tích hợp (*Intergrated Development Environmet– IDE*) mạnh cho phép người phát triển xây dựng ứng dụng bằng C++. Hai công cụ này làm việc chặt chẽ với nhau để xây dựng ứng dụng phần mềm có chất lượng tốt. Thông thường, trong môi trường phát triển phần mềm, các yêu cầu được thu thập bằng UC và các tài liệu khác, chúng được mô hình hóa bằng mô hình UC, sau đó xây dựng các mô hình phân tích và thiết kế. Mô hình thiết kế là trừu tượng hệ thống, được ánh xạ trực tiếp thành mã trình. Sau khi có mã trình là các công việc dịch, thực hiện và gỡ lỗi. Trong khi Rose là công cụ quản lý các mô hình thì *Visual C++* là công cụ quản lý mã trình và thực hiện nó.

Quá trình chuyển tiếp từ mô hình sang mã trình và qui trình ngược lại từ mã trình sang mô hình được gọi là *RTE (Round Trip Engineering*). Khả năng RTE là chìa khóa thắng lợi của dự án phần mềm. Khi người cài đặt phát hiện lỗi thiết kế và sửa nó trong mã trình thì sự thay đổi này sẽ được tự động phản ánh ngược lại mô hình. Hơn nữa, việc thay đổi mô hình sẽ được phản ánh trong mã trình đang có. Khả năng này đảm bảo tính đồng bộ giữa mã trình và mô hình.



Hình 7.14 Nhập thư viện MFC vào mô hình

Để hỗ trợ ứng dụng *MFC (Microsoft Foundation Class*), Rose có khả năng chuyển đổi ngược toàn bộ thư viện *MFC*. Hãy sử dụng thực đơn *Tools/Visual C++/ Quick Import MFC* 6.0 (hình 7.14) để nhập chúng vào mô hình. Gói mới với tên MFC được bổ sung vào *Logical view*. Người sử dụng có thể sử dụng các lớp *MFC* này để thiết kế chi tiết hệ thống. Mỗi thành phần mô hình được kết hợp vào dự án (*project*) khác nhau của *Visual C++,* hơn nữa *Visual C++ Project* phải được tạo lập trước khi gán cho thành phần mô hình.

Để sử dụng lợi thế giữa *Rose* và *Visual C++* thì trên máy tính phải có sẵn hai phần mềm sau:

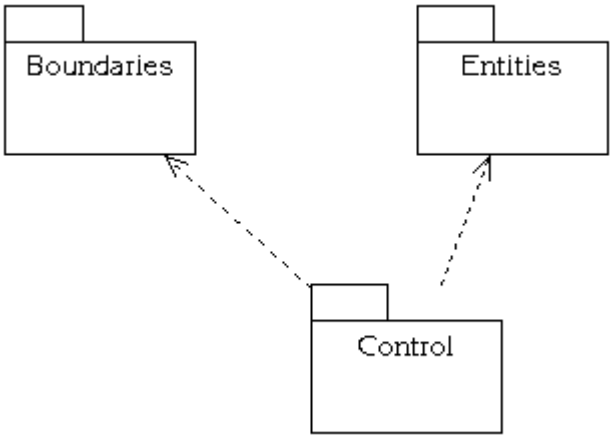
1. Visual C++ 6.0
2. Phiên bản *Rational Rose Enterprise* hay *Rational Rose Professional C++ Edition*.

Nếu *Visual C++* chưa sẵn sàng trong *Rational Rose* thì hãy thực hiện các bước sau:

1. Khởi động Rational Rose
2. Chọn thực đơn *Add-Ins> Add-In Manager*
3. Chọn *VC++ add-in*
4. Nhấn phím *OK*.

Đặt ngôn ngữ mặc định cho *Rational Rose*:

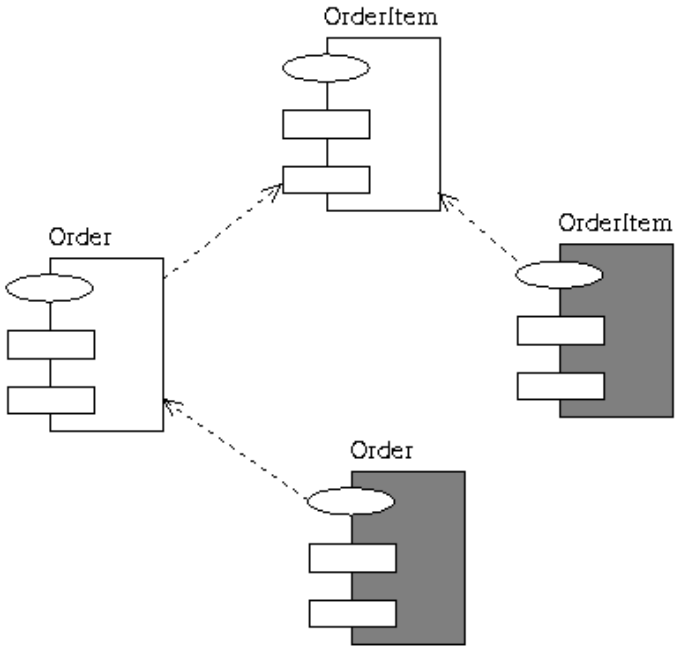
1. Chọn thực đơn *Tools > Options*
2. Chọn bảng *Notation*
3. Chọn hộp danh sách *Default Language*
4. Chọn Visual C++
5. Nhấn phím *OK*.



Hình 7.15 Biểu đồ thành phần của hệ thống đơn hàng

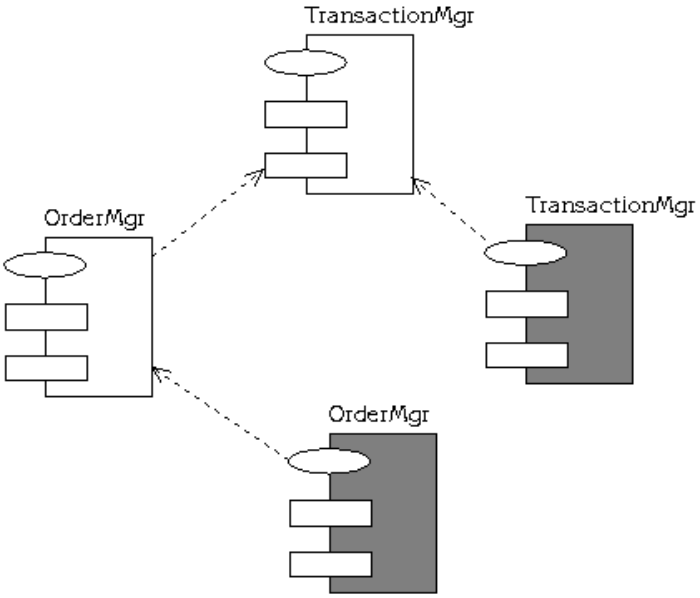
7.3.4 - Thí dụ: Hệ thống bán hàng (tiếp theo)

Phần này thực hành tạo lập biểu đồ thành phần cho hệ thống quản lý bán hàng. Trong chương trước ta đã tìm ra các lớp cần cho *UC New Order*. Sau khi hoàn chỉnh phân tích và thiết kế, ta có thể xây dựng biểu đồ thành phần. Giả sử ta sẽ sử dụng ngôn ngữ lập trình C++ để cài đặt hệ thống. Biểu đồ thành phần chính tập trung vào các gói của thành phần được chỉ ra trên hình 7.15.



Hình 7.16 Biểu đồ thành phần gói Entities

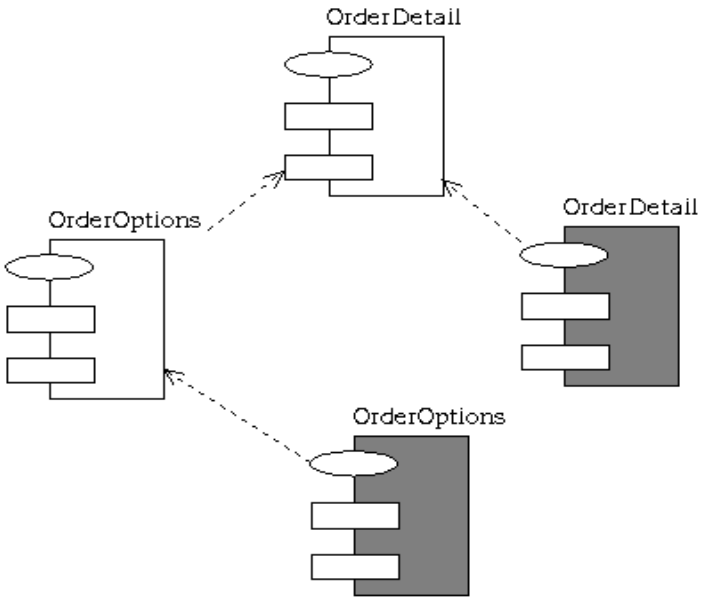
Hình 7.16 bao gồm các thành phần trong gói *Entities*. Các thành phần này chứa các lớp trong gói *Entities* trong khung nhìn logic.



Hình 7.17 Biểu đồ thành phần gói Control

Hình 7.17 bao gồm các thành phần trong gói *Contro*l. Các thành phần này chứa các lớp trong gói *Contro*l trong khung nhìn logic.

Hình 7.18 bao gồm các thành phần trong gói *Boundaries*. Các thành phần này chứa các lớp trong gói Boundaries trong khung nhìn *l*ogic.



Hình 7.18 Biểu đồ thành phần gói Boundaries

Hình 7.19 chỉ ra toàn bộ các thành phần trong hệ thống. Biểu đồ này cho toàn bộ phụ thuộc giữa các thành phần trong hệ thống.

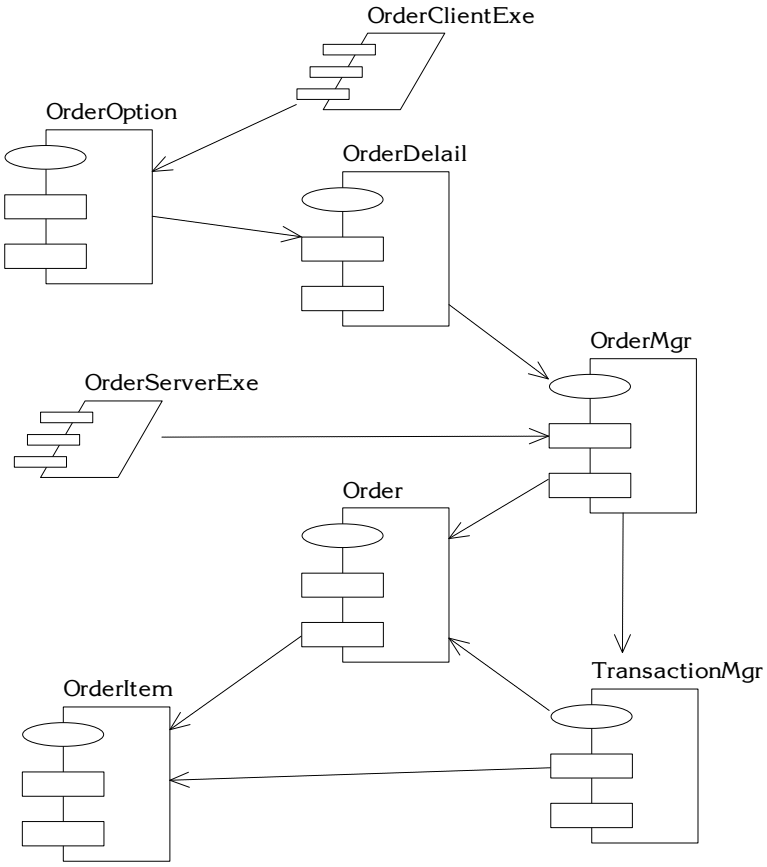
7.3.4.1 - Lập các gói thành phần

1. Nhấn phím phải trên *Component view* trong *brower*
2. Chọn *New > Package*
3. Đặt tên gói là *Entities*
4. Lặp bước 1-3 cho các gói *Boundaries* và *Control*

7.3.4.2 - Bổ sung gói vào biểu đồ thành phần Main

Mở biểu đồ thành phần *Main*

Di các gói *Entities, Boundaries* và *Control* từ brower vào *Main Component diagram*.



Hình 7.19 Biểu đồ thành phần của hệ thống đơn hàng

7.3.4.3 - Vẽ các phụ thuộc gói

1. Chọn phím *Dependency* từ thanh công cụ
2. Nhấn trên gói Boundaries trong Main Component diagram
3. Di phụ thuộc đến gói *Control*
4. Thực hiện tương tự để vẽ các phụ thuộc cho các gói còn lại.

7.3.4.4 - Bổ sung thành phần cho các gói và vẽ các phụ thuộc

1. Nhấn đúp vào gói *Entities* trong biểu đồ thành phần *Main*.
2. Chọn phím *Package specification* (đặc tả gói) trên thanh công cụ
3. Đặt đặc tả gói vào biểu đồ
4. Nhập tên *OrderItem* cho đặc tả gói
5. Lặp các bước 2-4 để bổ sung đặc tả gói *Order*
6. Chọn phím *Package Body* (thân gói) từ thanh công cụ
7. Đặt thân gói vào biểu đồ
8. Nhập tên *OrderItem* cho thân gói

9. Lặp các bước 6-8 để bổ sung thân gói *Order*
10. Chọn phím *Dependency* từ thanh công cụ
11. Nhấn phím chuột trên thân gói *OrderItem*
12. Di đường phụ thuộc tới đặc tả gói *OrderItem*
13. Lặp để vẽ các phụ thuộc còn lại
14. Làm tương tự cho cách thành phần và phụ thuộc sau đây:

Với gói *Boundaries*

- Đặc tả gói *OrderOptions*
- Thân gói *OrderOptions*
- Đặc tả gói *OrderDetail*
- Thân gói *OrderDetail*

Các phụ thuộc trong gói *Boundaries*

- Thân gói *OrderOptions* đến đặc tả gói *OrderOptions*
- Thân gói *OrderDetail* đến đặc tả gói *OrderDetail*
- Đặc tả gói *OrderOptions* đến đặc tả gói *OrderDetail*

Với gói *Control*

- Đặc tả gói *OrderMgr*
- Thân gói *OrderMgr*
- Đặc tả gói *TransactionMgr*
- Thân gói TransactionMgr

Các phụ thuộc trong gói *Control*

- Thân gói *OrderMgr* đến đặc tả gói *OrderMgr*
- Thân gói *TransactionMgr* đến đặc tả gói *TransactionMgr*
- Đặc tả gói *OrderMgr* đến đặc tả gói *TransactionMgr*

7.3.4.5 - Tạo biểu đồ thành phần System

1. Nhấn phím phải trên *Component View* trong *browser*
2. Chọn thực đơn *New>Component Diagram*
3. Đặt tên biểu đồ mới là *System*
4. Nhấn đúp trên biểu đồ thành phần *System*

7.3.4.6 - Đặt các thành phần vào biểu đồ thành phần System

1. Nhấn đặc tả gói *Order* trong gói *Entities*
2. Di đặc tả gói *Order* vào biểu đồ
3. Lặp để đặt đặc tả gói *OrderItem* vào biểu đồ
4. Làm tương tự để đặt các thành phần sau đây vào biểu đồ.

Trong gói thành phần *Boundaries*:

- Đặc tả gói *OrderOptions*
- Đặc tả gói *OrderDetail*

Trong gói thành phần *Control:*

- Đặc tả gói *OrderMgr*
- Đặc tả gói *TransactionMgr*
- Đặc tả nhiệm vụ *OrderClientExe*
- Đặc tả nhiệm vụ *OrderServerExe*

5. Chọn *Task Specification* trên phím công cụ
6. Đặt đặc tả nhiệm vụ vào biểu đồ và đặt tên là *OrderClientExe*
7. Lặp cho đặc tả nhiệm vụ *OrderServerExe*

7.3.4.7 - Bổ sung các phụ thuộc khác vào biểu đồ thành phần System

1. Chọn phím *Depedency* từ thanh công cụ
2. Nhấn trên đặc tả gói *OrderDetail*, di đến đặc tả gói *OrderMgr*.
3. Lặp để bổ sung các phụ thuộc sau:
   - Đặc tả gói OrderMgr đến đặc tả gói Order
   - Đặc tả gói TransactionMgr đến đặc tả gói OrderItem
   - Đặc tả gói TransactionMgr đến đặc tả gói OrderOptions
   - Đặc tả nhiệm vụ OrderClientExe đến đặc tả gói OrderOptions
   - Đặc tả nhiệm vụ OrderServerExe đến đặc tả gói OrderMgr

Ánh xạ các lớp vào thành phần:

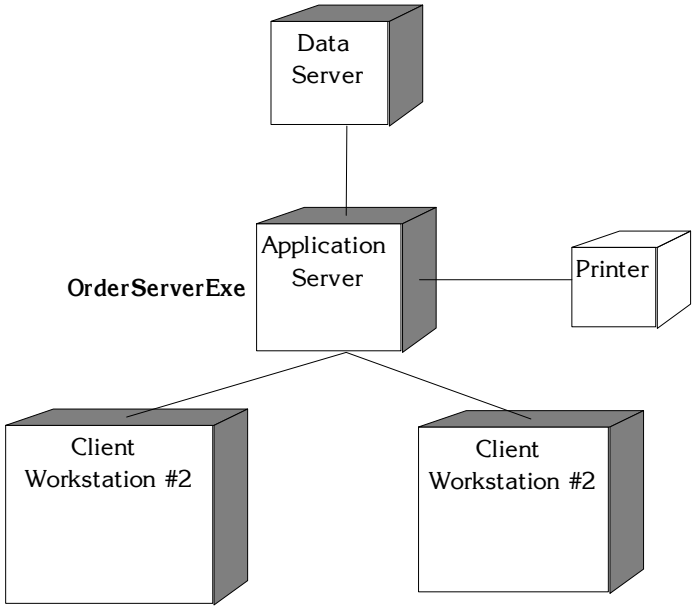
1. Định vị lớp *Order* trong gói *Entities* trong khung nhìn logic của *browser*.
2. Di nó đến đặc tả gói thành phần *Order* trong *Component view*, có nghĩa là ánh xạ lớp Order vào đặc tả gói thành phần *Order*.
3. Di lớp *Order* vào thân gói thành phần *Order* trong *Component view* (có nghĩa là ánh xạ lớp *Order* vào thân thành phần *Order*).
4. Lặp lại để ánh xạ các lớp sau đây vào thành phần:
   - Lớp *OrderItem* vào đặc tả gói *OrderItem*
   - Lớp *OrderItem* vào thân gói *OrderItem*
   - Lớp *OrderOptions* vào đặc tả gói *OrderOptions*
   - Lớp *OrderOptions* vào thân gói *OrderOptions*
   - Lớp *OrderDetail* vào đặc tả gói *OrderDetail*
   - Lớp *OrderDetail* vào thân gói *OrderDetail*
   - Lớp *OrderMgr* vào đặc tả gói *OrderMgr*
   - Lớp *OrderMgr* vào thân gói *OrderMgr*

- Lớp *TransactionMgr* vào đặc tả gói *TransactionMgr*
- Lớp *TransactionMgr* vào thân gói *TransactionMgr*

7.3.4.8 - Lập biểu đồ triển khai

Phần này tạo lập biểu đồ triển khai cho hệ thống xử lý đơn hàng. Biểu đồ hoàn thiện được thể hiện trên hình 7.20. Bổ sung nút vào biểu đồ triển khai theo các bước sau đây:

Nhấn đúp trên *Deployment view* trong *browser* để mở biểu đồ triển khai



Hình 7.20 Biểu đồ triển khai của hệ thống Xử lý đơn hàng

1. Chọn phím *Processor* từ thanh công cụ,
2. Nhấn trên biểu đồ để vẽ bộ xử lý
3. Nhập tên *Database Server* cho bộ xử lý
4. Lặp bước 2-4 để bổ sung các biểu đồ sau:
   - *Application Server*
   - *Client Workstation #1*
   - *Client Workstation #2*
5. Chọn phím *Device* từ thanh công cụ
6. Nhấn phím chuột trên biểu đồ để vẽ thiết bị
7. Nhập tên thiết bị là *Printer*

7.3.4.9 - Bổ sung kết nối

1. Chọn phím *Connection* từ thanh công cụ
2. Nhấn trên bộ xử lý *Database Server*
3. Di đường kết nối với bộ xử lý *Application Server*

4. Lặp để vẽ các kết nối sau:
   - *Application Server* đến bộ xử lý *Client Workstation #1*
   - *Application Server* đến bộ xử lý *Client Workstation #2*
   - *Application Server* đến thiết bị *Printer*

*7.3.4.10 - **Bổ sung tiến trình***

1. Nhấn phím phải trên bộ xử lý *Application Server* trong *Browser*
2. Chọn thực đơn *New>Process*
3. Nhập tên tiến trình *OrderServerExe*
4. Lặp các bước 1-3 để bổ sung các tiến trình sau:
   - *OrderClientExe* cho bộ xử lý *Client Workstation #1*
   - *OrderClientExe* cho bộ xử lý *Client Workstation #2*

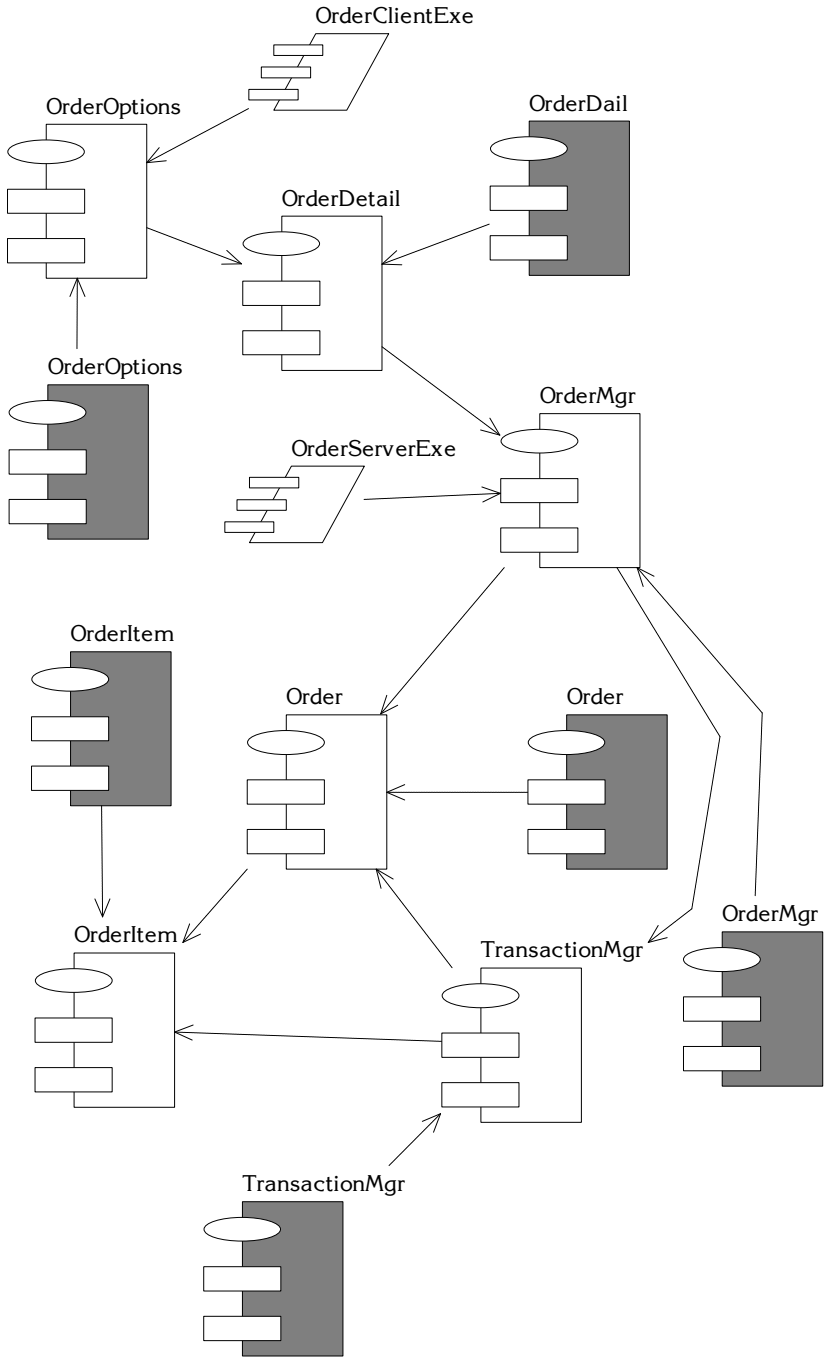
*7.3.4.11 - **Hiển thị tiến trình***

1. Nhấn phím phải trên tiến trình *Application Server*
2. Chọn *Show Processes* từ thực đơn
3. Lặp các bước 1-2 để hiển thị tiến trình của các bộ xử lý sau:
   - Bộ xử lý *Client Workstation #1*
   - Bộ xử lý *Client Workstation #2*

*7.3.4.12 - **Phát sinh mã trình Visual C++***

Bổ sung thân gói vào biểu đồ thành phần hệ thống trong *Component view*:

1. Mở *System Component Diagram*
2. Chọn thân gói Order trong *Entities* từ *browser*
3. Di thân gói *Order* tới *System Component Diagram*
4. Lặp lại các bước 2-3 cho các thành phần sau:
   a. Thân gói *OrderItem* trong *Entities*
   b. Thân gói *OrderOptions* trong *Boundaries*



Hình 7.21 Biểu đồ thành phần của hệ thống nhập đơn hàng

c. Thân gói *OrderDetail* trong *Boundaries*

d. Thân gói *Transaction* trong *Control*

e. Thân gói *OrderMgr* trong *Control*

Biểu đồ thành phần hệ thống nhập đơn hàng như trên hình 7.21.

Đặt ngôn ngữ phát sinh là *Visual C++*

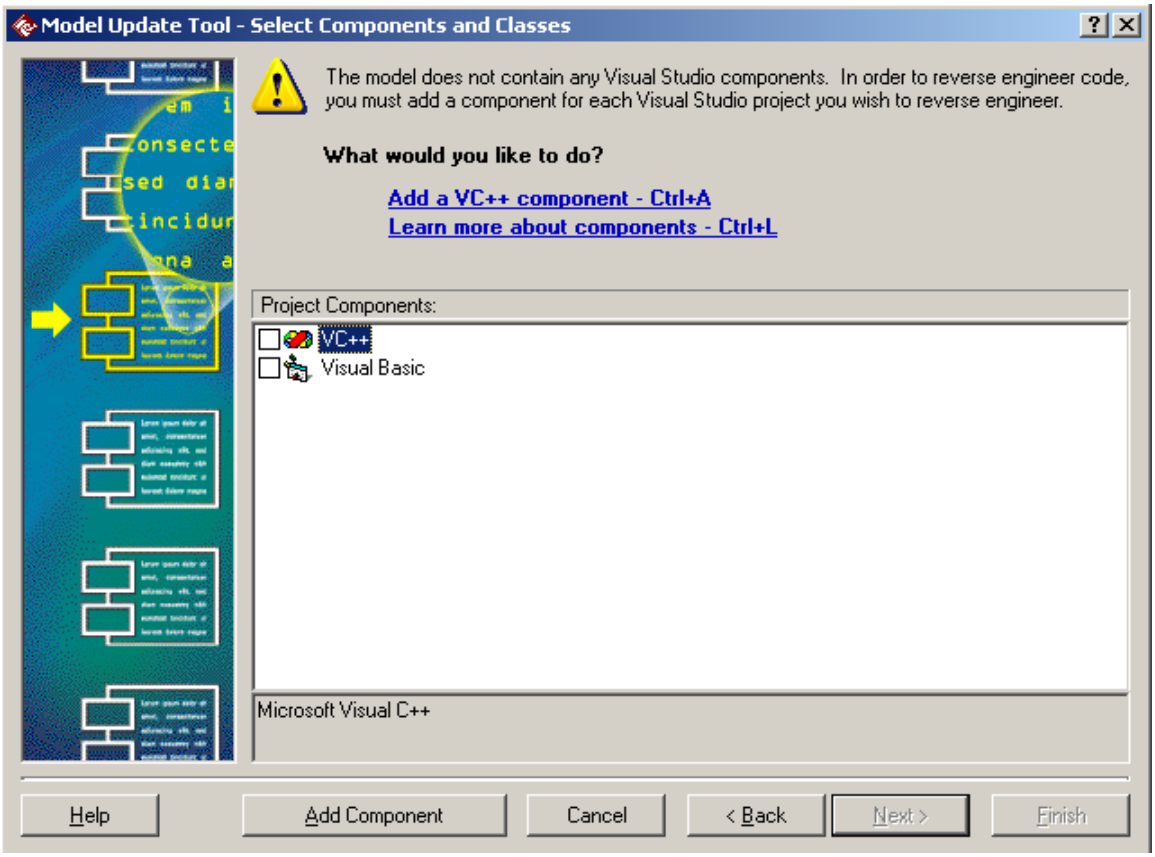
1. Mở *Component Specification* của thành phần *Order* trong gói thành phần *Entities*

2. Chọn ngôn ngữ *Visual C++*
3. Lặp lại bước 1-2 cho các thành phần sau:
    a. *Entities*: Thân gói *Order*
    b. *Entities*: Đặc tả gói *OrderItem*
    c. *Entities*: Thân gói *OrderItem*
    d. *Boundaries*: Đặc tả gói *OrderOptions*
    e. *Boundaries*: Thân gói *OrderOptions*
    f. *Boundaries*: Đặc tả gói *OrderDetail*
    g. *Boundaries*: Thân gói *OrderDetail*
    h. *Control*: Đặc tả gói *TransactionMgr*
    i. *Control*: Thân gói *TransactionMgr*
    j. *Control*: Đặc tả gói *OrderMgr*
    k. *Control*: Thân gói *OrderMgr*
    l. Đặc tả nhiệm vụ *OrderClientExe*
    m. Đặc tả nhiệm vụ *OrderServerExe.*

Phát sinh mã trình Visual *C++* được thực hiện như sau:

1. Mở biểu đồ *System Component*
2. Chọn toàn bộ đối tượng trong biểu đồ *System Component*
3. Chọn *Tools > Visual C++ > Update Code*.

Trên màn hình sẽ xuất hiện hộp thoại cho phép chọn thành phần hay lớp sẽ chuyển sang *Visual C++* (hình 7.22). Các tệp mã trình mới sẽ được gán vào tệp *Project* nào đó của *Visual C++*. Nếu không sử dụng tệp *Project* nào có sẵn thì phải tạo tệp mới.



Hình 7.22 Chọn thành phần hay lớp để chuyển sang Visual C++

Hình 7.23 Tạo lập Project trong Visual C++ 6.0

1. Nhấn phím chuột trên thành phần *Order* trong cửa sổ *Component in model*
2. Chọn *Yes* để gán thành phần vào *VC++ Project*

3. Nhấn phím…của cửa sổ *Workspace File (Optional)* để xuất hiện hộp thoại *Select a VC++ Project File*
4. Nhấn đúp trên biểu tượng *New Project* trong cửa sổ *New*. Cửa sổ *New* của *Microsoft Visual C++* xuất hiện. Lựa chọn các thuộc tính cho *Visual C++* cho phù hợp. Một thí dụ chọn lựa như trên hình 7.23
5. Nhấn phím *OK* > *Finish* để trở về Rational Rose
6. Nhấn phím *OK* > *OK* để trở về hộp thoại *code Update Tool* (hình 7.22)
7. Nhấn phím lần nữa trên thành phần *Order* trong cửa sổ *Components in Model*
8. Thực hiện tương tự cho các thành phần còn lại trong cửa sổ *Components in Model*. Nhưng lần này chọn thuộc tính bổ sung vào *Project* hiện hành.
9. Nhấn phím *OK* để *Rational Rose* tự động phát sinh mã trình *Visual C++* cho các thành phần đã đánh dấu
10. Mở *Microsoft Visual C++* để bổ sung các mã trình cho lớp và xây dựng các lớp giao diện người sử dụng.

Dưới đây là hai tệp thí dụ được *Rational Rose* phát sinh từ mô hình UML.

Tệp Order.cpp

```
//Copyright (C ) 1991- 1999 Rational Software Corporation-
#include “stdafx.h”
#include “Order.h”
//##Modelld=3A77E3CD0280
Boolean Order::Create()
{
    //TODO:Add your specialized code here
    //NOTE:Requires a correct return value to compile.
}
//##Modelld=3A77E3E603316
Boolean Order::SetInfo(Integer OrderNum, String Customer, Date OrderDate, Date FillDate)
{
    //TODO:Add your specialized code here.
    //NOTE:Requires a correct return value to compile.
}
//##Modelld=3A77E4E0230
String Order::GetInfo()
{
    //TODO:Add your specialized code here.
    //NOTE:Requires a corrct return value to compile.
}
```

Tệp Order.h

```
//Copyright (C) 1991-1999 Rational Software Corporation
#if defined (_MSC_VER) &&(_MSC_VER >=1000)
```

```
#pragma once
#endif
#ifndef_INC_ORDER_3A77E11400C8_INCLUDED
#define_INC_ORDER_3A77E11400C8_INCLUDED
//##Modelld=3A77E11400C8
class Order
{
pubbic:
    //##Modelld=3A7F695F019A
    OrderItem* theOrderItem;
    //##ModelId=3A77E3CD0280
    Boolean Create();
    //##ModelId=3A77E3E60316
    Boolean SetInfo(Integer OrderNum, String Customer, Date OrderDate, Date  FillDate);
//##ModelId=3A77E4E0230
String GetInfo();
private:
    //##ModelId=3A7E13F9038E
    Integer OrderNumber;
    //ModelId=3A7E14260122
    String CustomerName:
    //##ModelId=3A7E14470208
    Date OrderDate;
    //##ModelId=3A7E145303D4
    Date OrderFillDate;
};
#endif /*_INC_ORDER_3A77E11400C8_INCLUDED*/
```

CHƯƠNG 8
VÍ DỤ ÁP DỤNG

Trong các thư viện truyền thống, thủ thư phải làm việc vật vả với khối lượng lớn giấy tờ, và độc giả rất vất vả để tìm ra quyển sách mà họ quan tâm. Do công nghệ phần mềm và đặc biệt *Internet* phát triển nhanh, khái niệm mới, *thư viện điện tử*, đã xuất hiện. Loại thư viện này đã cho nhiều lợi ích như khả năng xâm nhập từ mọi nơi trên thế giới, khả năng tìm kiếm sách, tạp chí, các tiệp đa phương tiện một cách nhanh chóng… Chương này có sử dụng một số thí dụ trong tài liệu [KALA01] để trình bày một trong những cách ứng dụng hiệu quả UML vào phân tích, thiết kế hệ thống thư viện điện tử. Hệ thống thư viện điện tử được mô tả trong chương này được xem như bài học thực nghiệm (*Case Study*) chứ chưa phải là một ứng dụng hoàn chỉnh.

8.1 KHẢO SÁT TIẾN TRÌNH TÁC NGHIỆP

Các công việc của một thư viện điện tử được làm rõ dần dần thông qua đối thoại giữa phân tích viên và khách hàng, người đặt hang xây dựng hệ thống. Hãy xem xét kỹ đoạn hội thoại giữa họ được mô tả vắn tắt dưới đây.

Phân tích viên: Độc giả ci đến đâu trong thư viện và để làm những việc gì?

Khách hàng: Độc giả đi đến hang lang của thư viện để tìm thông tin về tài liệu muốn mượn. Sau khi đã tìm thấy sách (tạp chí) để mượn, độc giả đến giá sách (tạp chí) để lấy sách (tạp chí)

Phân tích viên: Độc giả tìm kiếm những thông tin gì?

Khách hàng: Thông tin chính về sách hay tài liệu và về tác giả của tài liệu được sắp theo thứ tự abc. Các thông tin này được ghi trong phích và đặt trong các hộp khác nhau hoặc nó được truy cập từ máy tính đặt tại hành lang.

Phân tích viên: CSDL của các thông tin này để ở đâu

Khách hàng: CSDL được lưu trong máy của mạng LAN, còn các máy đầu cuối được đặt trong thư viện

Phân tích viên: Có thể tìm tài liệu theo loại dữ liệu nào?

Khách hàng: Dữ liệu để tìm sách và tạp chí tương tự nhau, bao gồm số xuất bản (ISBN), tên, năm xuất bản, nhà xuất bản. Nhưng sách còn có loại dữ liệu khác như tác giả, thể loại và tạp chí có tập và lĩnh vực.

Phân tích viên: Có những loại sách và tạp chí nào trong thư viện?

Khách hàng: Ngoài sách và tạp chí dưới dạng giấy trong còn có dưới dạng điện tử. Các tài liệu điện tử được lưu trong CSDL của máy chủ

Phân tích viên: Ai quản trị khối lượng lớn tài liệu và dữ liệu điện tử này?

Khách hàng Nhân viên thư viện.

Phân tích viên: Đến nay đã có thể thấy rõ ràng rằng mỗi độc giả đều có quyền tìm kiếm tài liệu trên máy tính để mượn. Có thể đọc ngay sách hay tạp chí lưu trữ dưới dạng số. Có thể mượn sách hay tạp chí trong. Bây giờ thảo luận về tiến trình mượn sách.

Khách hàng: Sau khi tìm ra tài liệu mượn thì bất kỳ ai có thẻ thư viện đều có thể mượn tài liệu. Có thể mượn tài liệu in tại bàn cho mượn của thư viện. Nếu còn quyển sách hay tập tạp chí trong thư viện thì thủ thư đưa cho độc giả.

Phân tích viên: Vậy, có hai trường hợp xảy ra, đó là còn và hết sách (tạp chí) trong thư viện?

Khách hàng: Nếu hết tài liệu thì độc giả có thể đặt mượn trước; khi nào có tài liệu thì thư viện sẽ thông báo cho độc giả. Độc giả phải đến thư viện để mượn. Chú ý rằng tài liệu này sẽ chỉ dành cho độc giả đã đăng ký trước trong vòng hai tuần lễ kể từ ngày thông báo.

Phân tích viên: Nếu còn tài liệu trong thư viện thì sao?

Khách hàng: Trong trường hợp này thủ thư sẽ kiểm tra dữ liệu về đọc giả xem độc giả có thẻ thư viện hay không. Nếu đã có thẻ. Nếu đã có thẻ, thủ thư ghi lại ISBN của tài liệu mượn và thông tin về độc giả mượn vào sổ mượn, đồng thời ghi vào CSDL mượn. Mỗi độc giả chỉ được mượn một số sách (tạp chí) nhất định, trong khoản thời gian nhất định.

Phân tích viên: Thủ thư có thể mượn sách hay tạp chí không?

Khách hàng: Có, mỗi thủ thư cũng có một bản ghi mượn.

Phân tích viên: Ai thực hiện việc ghiu thông tin mượn tài liệu vào CSDL?

Khách hàng: Ngoài máy trạm trong mạng để tìm kiếm tài liệu, thủ thư còn có máy trạm để kiểm tra thông tin về độc giả,tài liệu có còn khôngvà xem danh sách đôc giả phải trả tài liệu hết hạn mượn.

Phân tích viên: Với tài liệu điện tử thì sao?

Khách hàng: Nó luôn luôn có sẵn trong thư viện.

Phân tích viên: Trường hợp độc giả không tìm thấy sáchtrong danh sách tìm kiếm thì thủ thư có khuyên nhủ gì không?

Khách hàng: Thủ thư có thể khuyến cáo độc giả , nhưng độc giả có đống ý hay không.

Phân tích viên: Trên đây ta đã trao đổi các hoạt động của nhân viên thư viện liên quan đến việc mượn, đặt trước, khuyến cáo và tìm kiếm.Bây giờ trao đổi về giao thiệp giữa thủ thư với độc giả,sách và tạp chí.

Khách hàng: Nhiệm vụ của thủ thư tập trung vào mượn và trả sách,tạp chí. Họ gửi thông báo đến ai phải trả sách hết hạn mượn, đến ai đặt trước để mượn và nay đã có sách trong thư viện.

Phân tích viên:Thư viện có thuê mướn nhân viên để quản trị khối dữ liệu lớn hay không?

Khách hàng:Thư viện có nhân viên nhập liệu làm việc trên các máy trạm. Họ nhập liệu và bảo trì,còn chuyên gia CSDL thực hiện quản trị CSDL.

Phân tích viên:Nếu ai đó muốn trở thành độc giả của thư viện thì cần làm gì?

Khách hàng:Họ phải đến phòng tiếp đón để làm thủ tục.Thủ thư ghi bản ghi mới để ghi họ và tên ,đại chỉ nơi ở và chứng minh thư, số điện thoại và e-mail và ngày tháng năm sinh.Sau khi đăngvà in bản ký,thủ thư in thẻ thư viện và đưa cho độc giả và in bản ghi trên giấy để lưu trong hồ sơ thư viện.

Phân tích viên:Nhập thông tin về sách và tạp chí như thế nào?

Khách hàng:Sách và tạp chí có rất nhiều thuộc tính chung,bao gồm ISBN,tên, năm xuất bản và nhà xuất bản.Mỗi ấn phẩm còn có số lượng bản trong thư viện đôi khi chúng còn kèm theo dữ liệu đa phương tiện.

Phân tích viên: Thư viện có hỗ trợ dữ liệu đa phương tiện cho các ấn phẩm điện tử và ấn phẩm giấy?

Khách hàng: Không quan tâm đến hình thức ấn phẩm.Thư viện hỗ trợ dữ liệu đa phương tiện cho cả ấn phẩm điện tử và ấn phẩm giấy .

Phân tích viên: Khi tìm kiếm thông tin về tài liệu ,tôi có thể xem ngay dữ liệu đa phương tiện?Dữ liệu đa phương tiện là những dữ liệu loại gì?

Khách hàng:Nếu kết quả tìm kiếm trên máy tính là dữ liệu đa phương tiện thì độc giả có thể xem ngay nó.Dữ liệu đa phương tiện bao gồm văn bản ,âm thanh, ảnh và hình ảnh động.

Phân tích viên:Nhập thông tin về ta`c giả như thế nào?

Khách hàng:Thông tin về tác giả được nhập riêng.Chúng bao gồm họ và tên,ngày tháng năm sinh và nơi sinh.Khi nhập thông tin về sách,nếu tên tác giả đã có trong danh sách tác giả thì nhân viên nhập liệu chỉ việc chọn dữ liệu từ danh sách,nếu không thì phải tạo thông tin tác giả.Ngoài ra còn phải nhập loại sách.

Phân tích viên:Còn tạp chí thì sao?

Khách hàng: Số tạp chí và lĩnh vực của nó.

Phân tích viên: Như đã nói trênlà tồn tại hai loai lưu trữ thông tin(điện tửvà giấy),vậy,biểu diễn dữ liệu đa phương tiện trên bì giấy bằng cách nào?

Khách hàng:Dữ liệu đa phương tiện chỉ lưu trữ dưới khuôn dạng số.Thẻ giấy được in từ bản ghi và chỉ chứa có một vài thông tin cần thiết cho thẻ.

Phân tích viên: Ta đã khảo sát thấy công việc của thủ thư bao gồm nhập tên sách,tạp chí và thông tin tác giả , độc giả .Nhưng còn số lượng ấn phẩm thì sao?

Khách hàng:Số lượng ấn phẩm được nhập đồng thời khi nhập thông tin về sách .Số liệu cần quan tâmlà số lượng mới và số lượng đang có trong thư viện.

Phân tích viên: Có thể cho rằng công việc phân tích về công việc của thủ thư đã được thực hiện đầy đủ?

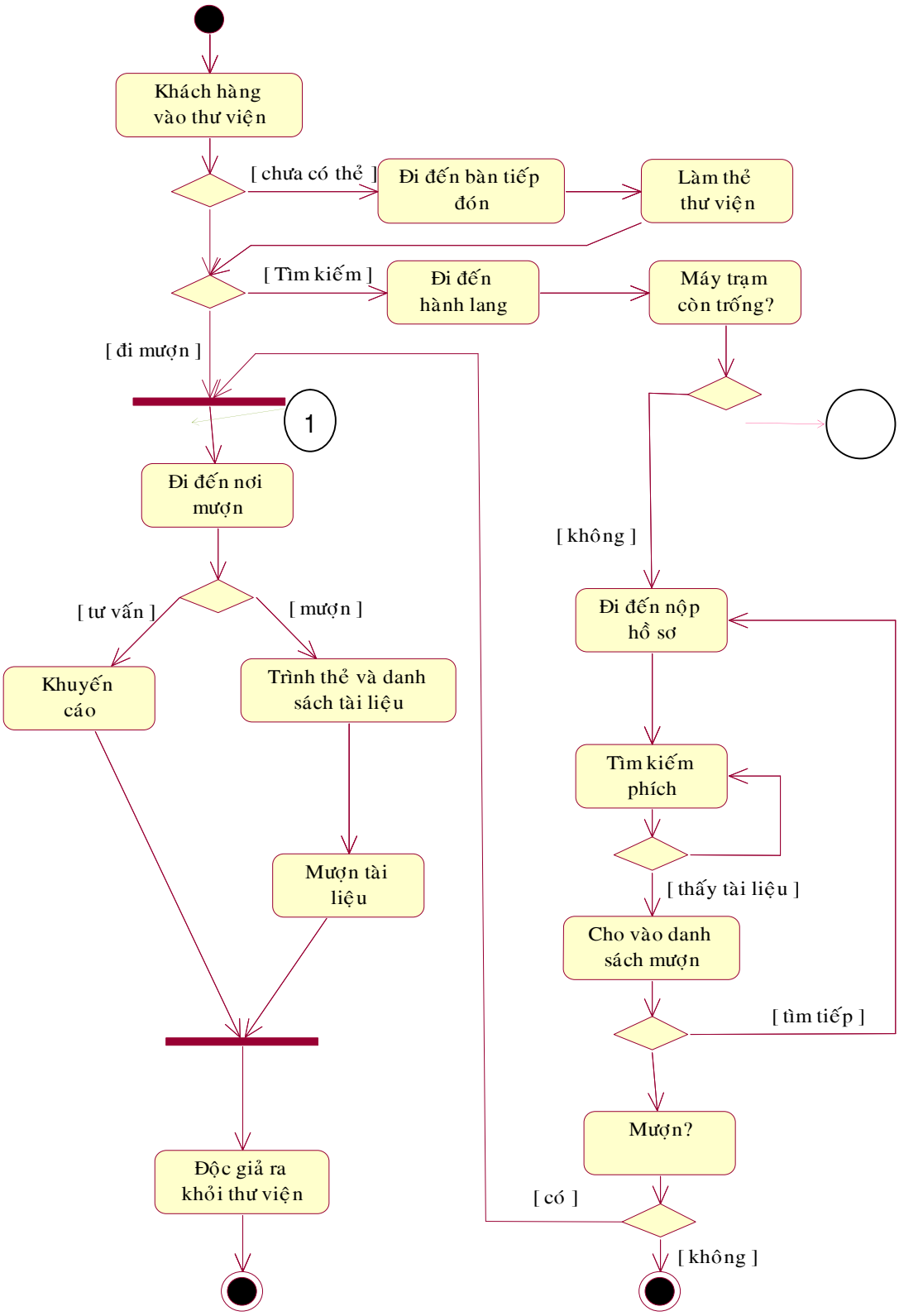
Khách hàng:Đúng vậy!

Từ mô tả trên đây về cuộc trao đổi giữa khách hàng(người yêu cầu xây dựng hệ thống thư viện)và phân tích viên hệ thống,ta có thể vẽ một số biểu đồ hoạt động của hệ thống thư viện điện tử.Sau đây là một vài biểu đồ biểu diễn các khía cạnh khác nhau của hệ thống:

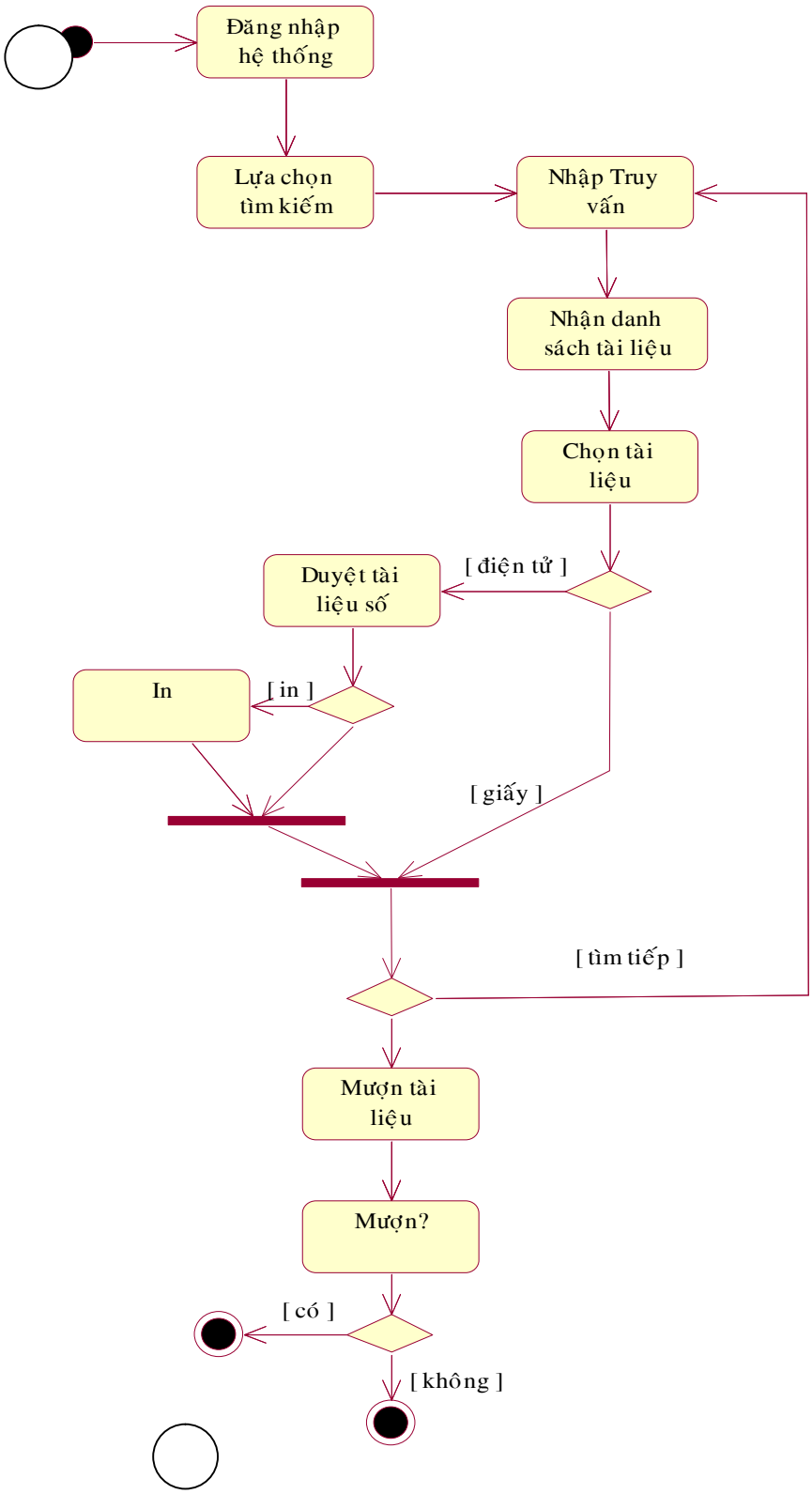
Hình 8.1 và hình 8.2 là biểu đồ nhoạt động của cả hệ thống.

Hình 8.3 là biểu đồ hoạt động của thủ thư khi cho mượn sách hay tạp chí.

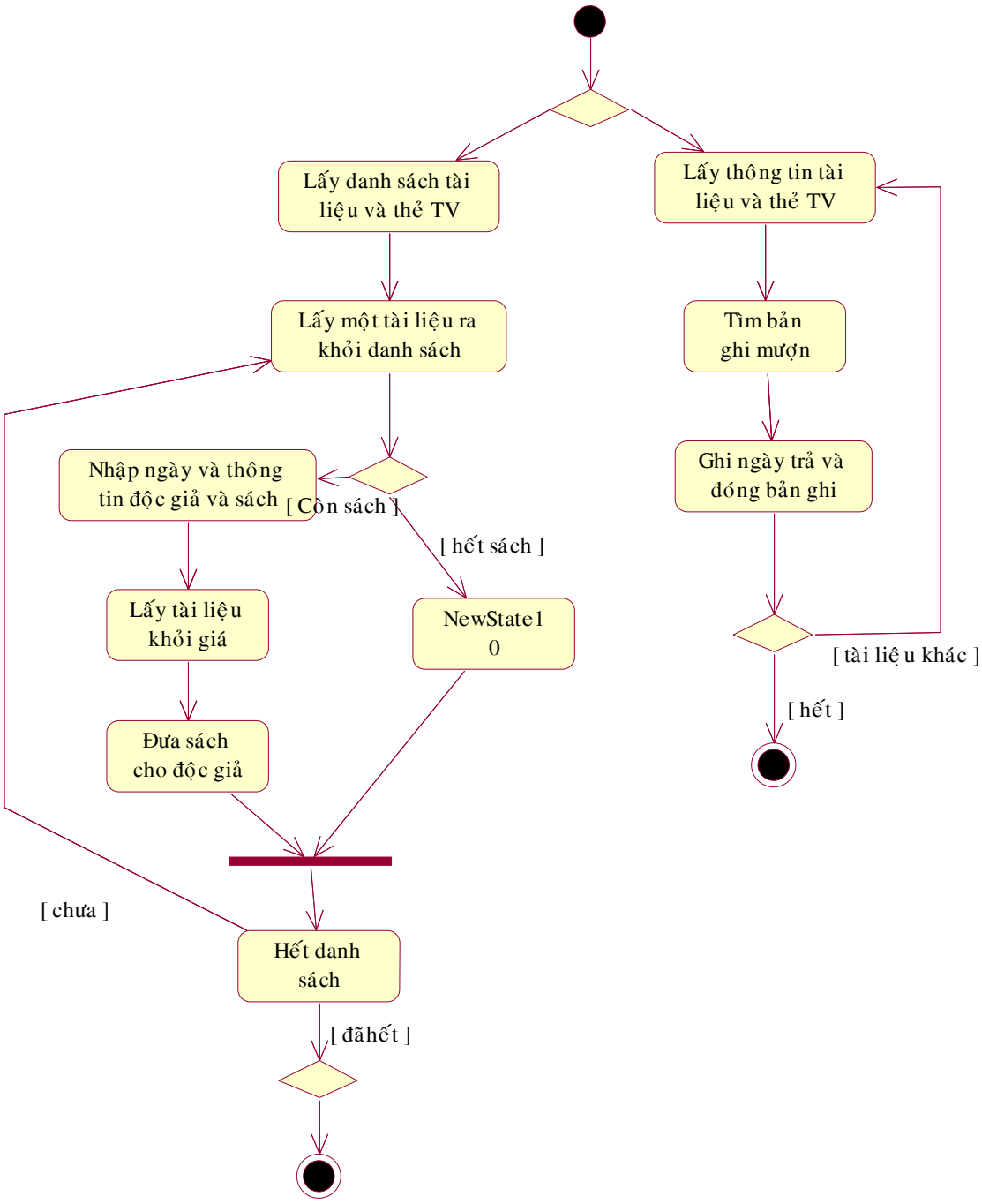
Hình 8.4 biểu đồ hoạt động của nhân viên nhập liệu.



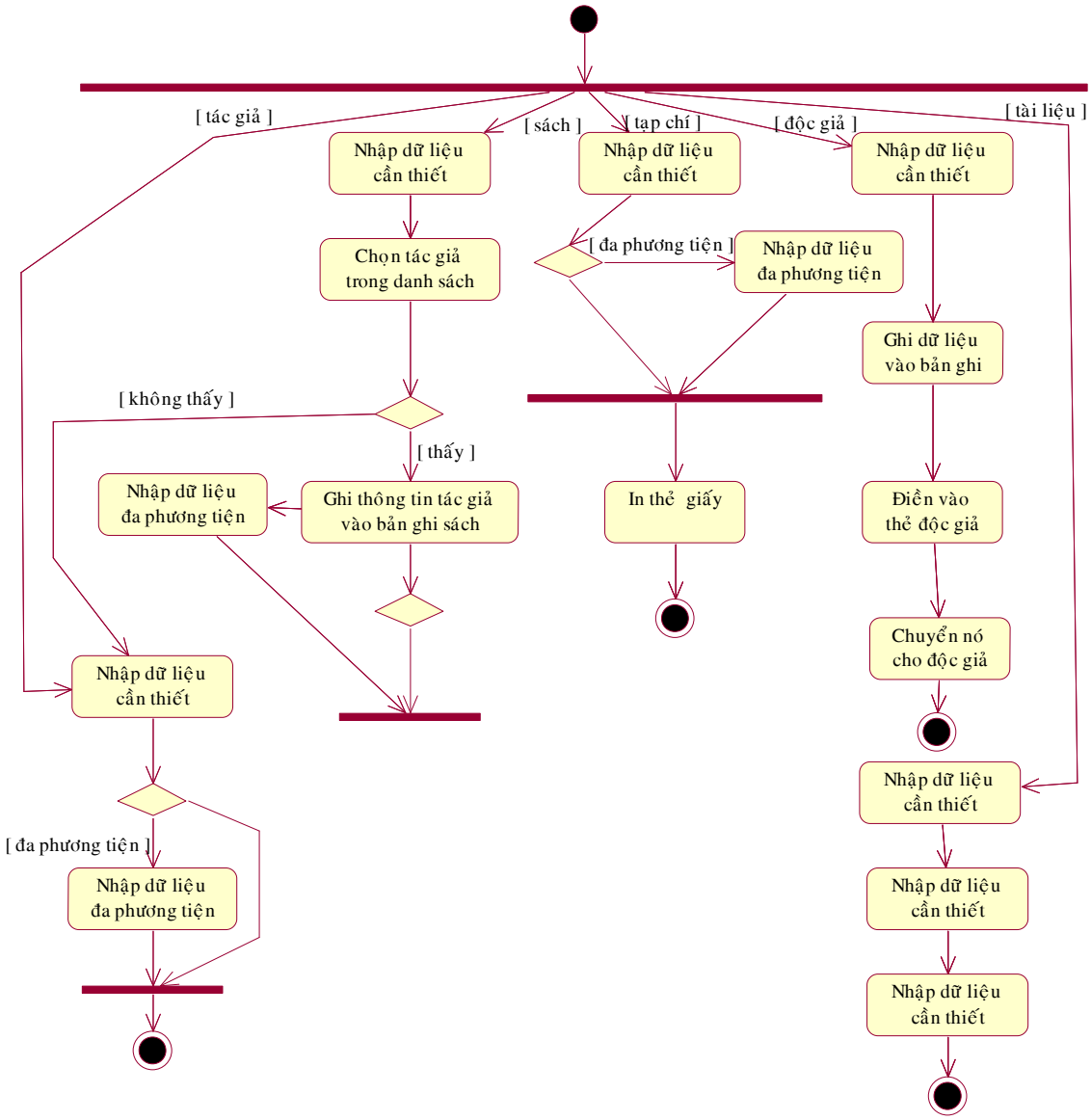
Hình 8.1 Biểu đồ hoạt động



Hình 8.2 Biểu đồ hoạt động (tiếp theo)



Hình 8.3 Biểu đồ hoạt động của thủ thư



Hình 8.4 Biểu đồ hoạt động của nhân viên nhập liệu

8.2 PHÂN TÍCH LĨNH VỰC

Bước tiếp theo là tìm kiếm các đối tượng hệ thống tương ứng với đối tượng thế giới thực cho mô hình .Trước mắt ta tìm các đối tượng từ phỏng vấn trong lĩnh vực vấn đề mô tả trên bằng cách tìm ra các danh từ , động từ và các nhóm động từ.Một số danh từ trở thành lớp, một số khác trở thành thuộc tính của mô hình, còn động từ và nhóm động từ trở thành thao tác hay nhãn của các kết hợp.Sau đây là danh sách danh từ ,phần in nghiêng có thể là thuộc tính và sẽ bị tách khỏi danh sách.

Hành lang ,bản ghi dữ liệu ,cặp lưu trữ,mạng LAN,máy chủ,máy trạm,sách in,tạp chí in,sách điện tử,tạp chí điện tử ,quyển sách,ngày tháng,giới hạn ngày tháng,sách ,tạp chí,tài liệu,tác giả ,phích,thư viện,CSDL,ISBN,tiêu đề năm xuất bản,loại,nhà xuất bản,số tạp chí,lĩnh vực,đặt trước,nhân,mượn,nơi mượn,thủ thư,người sử dụng,độc giả,bản ghi cho mượn,báo

cáo,tổng số quyển sách,nhân viên nhập liệu,chuyên gia CSDL,nơi tiếp đón,họ và tên,địa chỉ ,nơi ở ,số đăng ký cá nhân,điện thoại,e-mail,ngày sinh,thẻ bạn đọc,tài liệu thư viện,số liệu đa phương tiện,thông tin,văn bản,âm thanh ,hình ảnh,hình ảnh động.

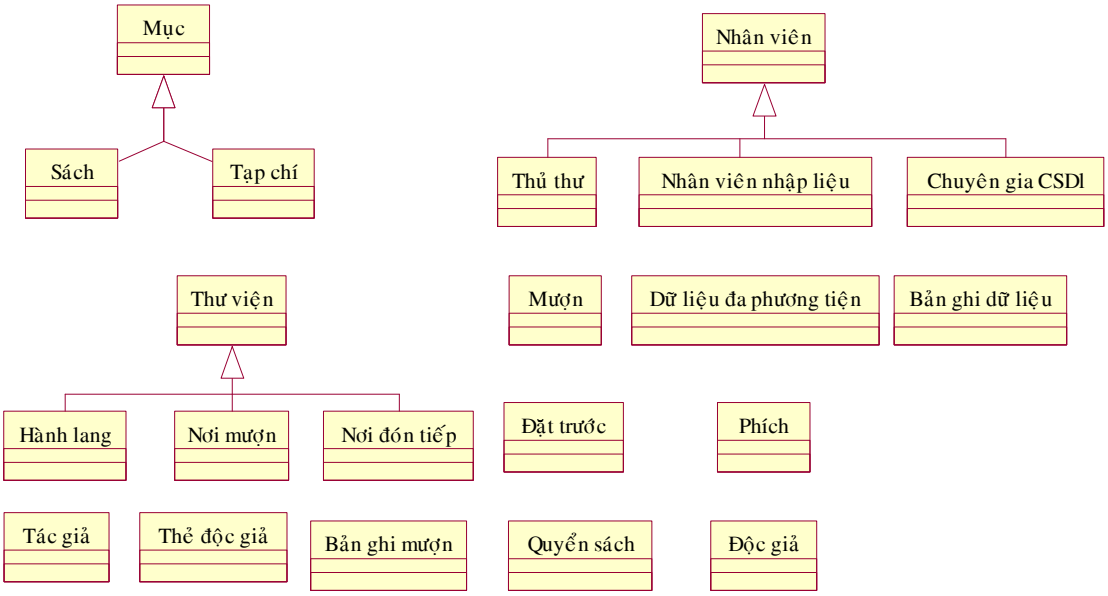
Danh sách động từ tìm ra từ phỏng vấn khách hàng như sau:

Đi,tìm kiếm,mượn,sắp xếp,in,lấy,có,chuyển đổi,ghi,quản lý ,xâm nhập,đọc ,duyệt,lấy quyển sách,đặt trước ,thông báo,đưa cho,kiểm tra dữ liệu,khuyến cáo,gửi,mua,thao tác,quản trị,nhập.

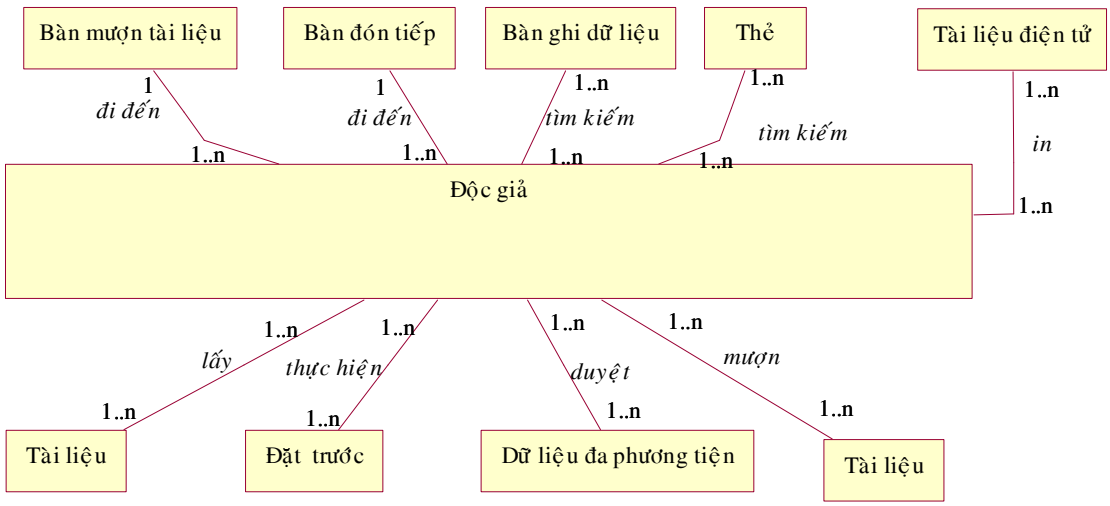
Có thể nhóm một số danh từ để thử hình thành những lớp trừu tượng sau này.Thí dụ ta có các nhóm sau:

- Nhóm con người:Độc giả,tác giả,nhân viên,thủ thư,chuyên gia CSDL,nhân viên nhập liệu.
- Nhóm các mục thư viện:sách,tạp chí,quyển sách,đặt trước,mượn tài liệu.
- Nhóm các vùng trong thư viện:hành lang,nơi mươn,nơi đón tiếp.
- Nhóm khác:thẻ thư viện,bản ghi dữ liệu,số liệu đa phương tiện,bản ghi mượn.

Kết quả là đã tìm ra các lớp trừu tượng như trên hình 8.5 .Chúng sẽ được chi tiết hóavà bổ sung trong các bước tiếp theo.



Hình 8.5 Các lớp



Hình 8.6 Lớp "Độc giả"

Từ các động từ hay cụm động từ trong phỏng vấn ta có thể hình thành các kết hợp giữa một vài lớp và đặt tên cho chúng.Thí dụ,khi khảo sát lớp "Độc giả"ta có thể hình thành các câu thể hiện kết hợp như dưới đây:

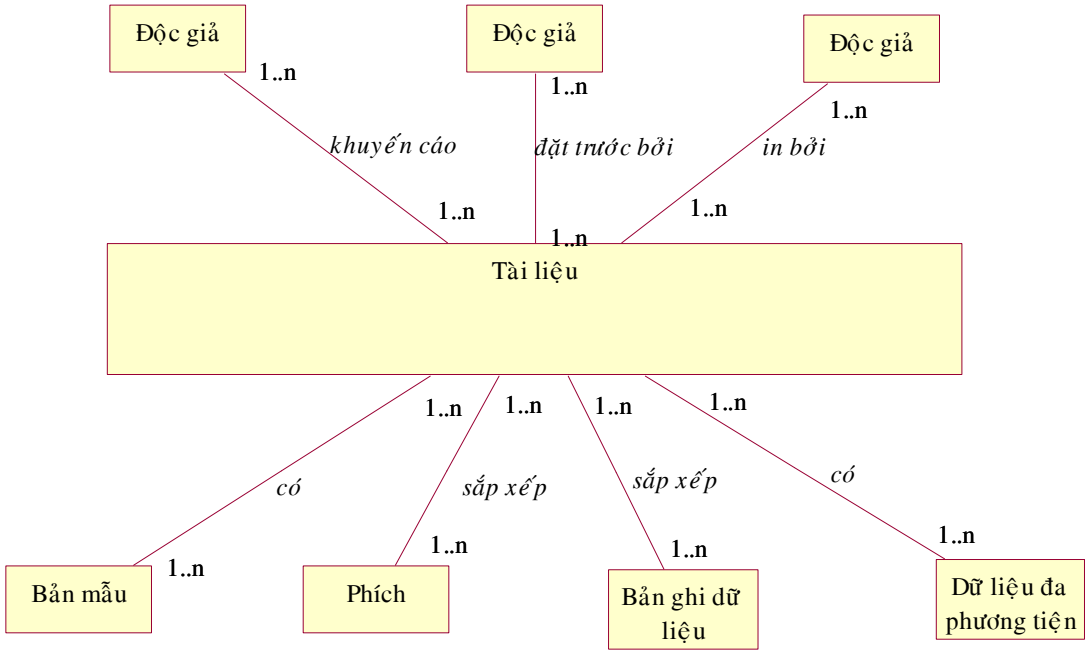
- Độc giả tìm phích hay bản ghi dữ liệu
- Độc giả đi đến bàn mượn tài liệu
- Độc gỉa mượn tài liệu.
- Độc giả đặt trước để mượn.
- Độc giả lấy tài liệu(sách,tạp chí)
- Độc giả duyệt dữ liệu đa phương tiện.
- Độc giả đi đến bàn đón tiếp.
- Độc giả in ấn phẩm điện tử.

Sau khi bổ sung tính nhiều(multiplicity) cho các kết hợp ta có "Độc giả"như trên hình 8.6.

Khi khảo sát lớp "Tài liệu" (sách hoặc tạp chí) ta có thể hình thành các câu thể hiện kết hợp như dưới đây:

- Tài liệu(sách hay tạp chí)có thể được khuyến cáo cho tác giả.
- Tài liệu được độc giả đặt trước.
- Tài liệu điện tử được độc giả in.
- Tài liệu được sắp xếp theo thứ tự trong phích và bản ghi dữ liệu.
- Tài liệu có bản mẫu.
- Tài liệu có dữ liệu đa phương tiện.

Sau khi bổ sung tính nhiều cho các kết hợp ta có lớp "Tài liệu"như trên hình 8.7.

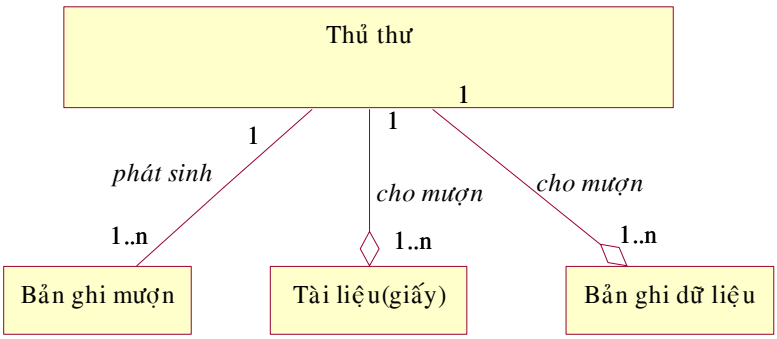


Hình 8.7 Lớp "Tài liệu"

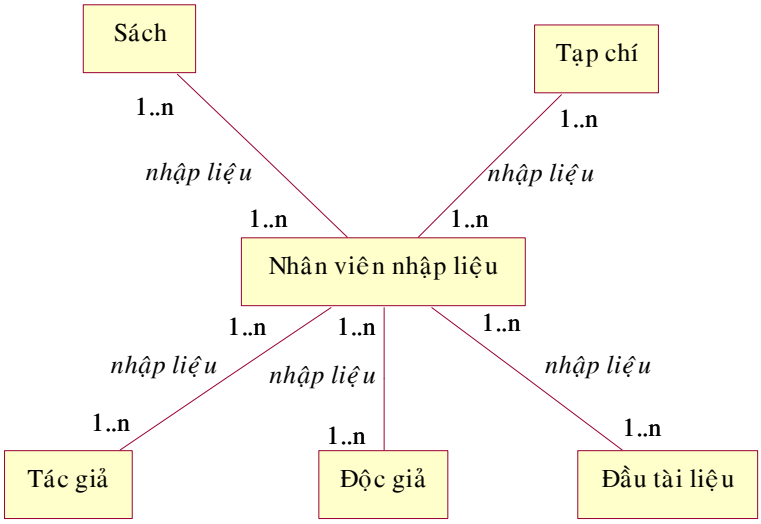
Tương tự,ta có thể xây dựng kết hợp và tính nhiều cho lớp "Nhân viên".Cần chú ý rằng nhân viên thư viện cũng là độc giả, do vậy nó cũng có các kết hợp như lớp "Độc giả", nhưng nhân viên có thể thực hiện các công việc khác mà độc giả không làm.

- Thủ thư lập bản ghi mượn tài liệu.
- Thủ thư cho độc giả mượn tài liệu.
- Nhân viên nhập liệu nhập thông tin về sách.
- Nhân viên nhập liệu nhập thông tin về tài liệu.
- Nhân viên nhập liệu nhập thông tin về tác giả
- Nhân viên nhập liệu nhập thông tin về độc giả
- Nhân viên nhập liệu nhập thông tin về bản mẫu.

Sau khi bổ sung tính nhiều cho các kết hợp ta có lớp "Thủ thư"như trên hình8.8 và lớp "Nhân viên nhập liệu" như trên hình 8.9.



Hình 8.8 Lớp "Thủ thư"



Hình 8.9 Lớp "Nhân viên nhập liệu"

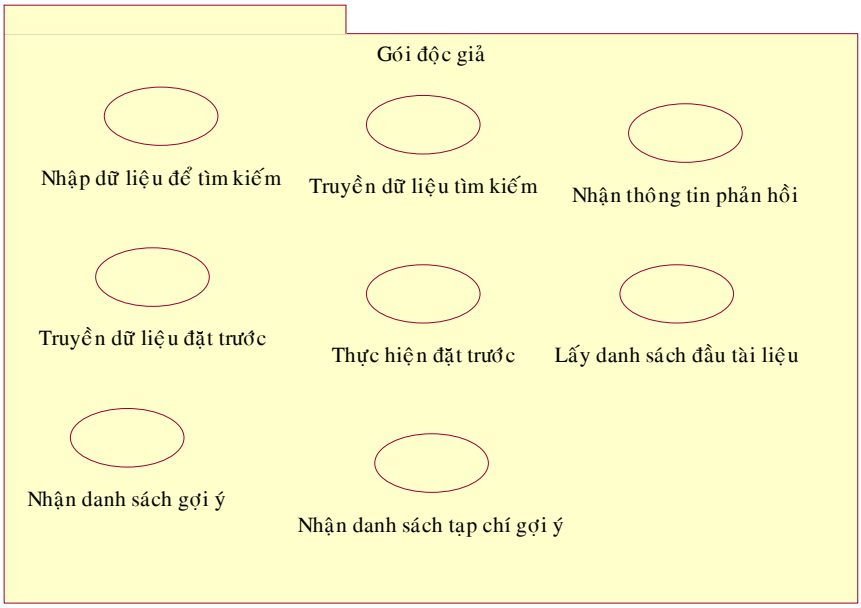
8.3 PHÂN TÍCH HỆ THỐNG.

8.3.1 - Xây dựng biểu đồ trường hợp sử dụng (Use Case-UC)

Danh sách tác nhân.Từ phân tích yêu cầu phần mềm ta nhận ra các tác nhân như sau:Độc giả,Thủ thư,Nhân viên nhập liệu,chuyên gia CSDL.

Danh sách các trường hợp sử dụng.Từ phân tích yêu cầu phần mềm,các chức năng hệ thống được hình thành.Đó là các trường hợp sử dụng của hệ thống,chúng được chia thành nhiều gói để dễ quan sát như gói Độc giả,gói Thủ thư ,gói Sách,gói Bản mẫu đầu tài liệu,gói Mượn sách,gói Tác giả và gói Đặt trước.Tương tự,ta có thể nhận ra các UC cho gói khác như gói Nhân viên nhập liệu,Đa phương tiện và gói Tạp chí.

Gói Độc giả.Gói này bao gồm các UC sau đây(hình8.10).



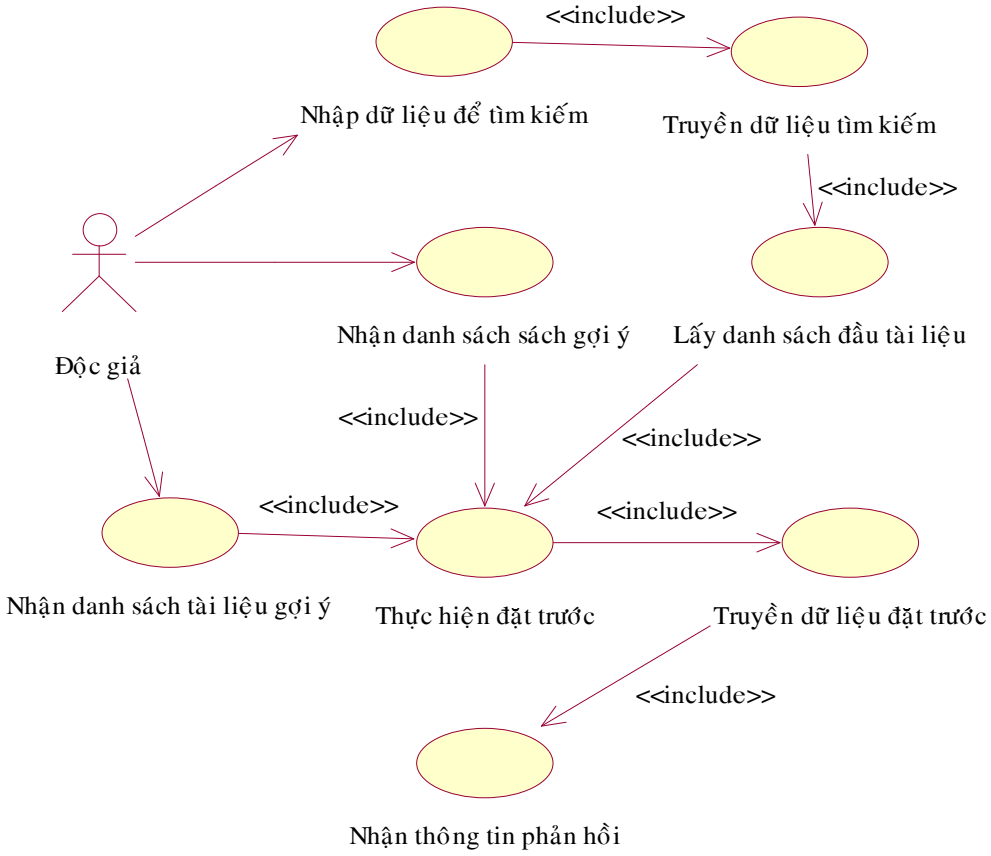
Hình 8.10 Gói UC "Độc giả"

- Trường hợp sử dụng "*Nhập dữ liệu để tìm kiếm*"
  - Mô tả UC:Độc giả nhập dữ liệu tìm kiếm vào cửa sổ (form)tìm kiếm.
  - Tác nhân kích hoạt:Độc giả
  - Tiền điều kiện:Độc giả có dữ liệu tìm kiếm;Hậu điều kiện:Độc giả nhập dữ liệu tìm kiếm vào thư điện tử.
  - Các bước trong UC này:
    - Từ mạng LAN của thư viện hay từ nơi khác độc giả kích hoạt giao diện người sử dụng để nhập liệu tìm kiếm.
    - Cửa sổ mẫu tìm kiếm xuất hiện trên màn hình.
    - Độc giả nhập dữ liệu vào mẫu.
- UC "Truyền dữ liệu để tìm kiếm".
  - Mô tả UC:Lấy dữ liệu từ cửa sổ rồi gửi đến mô tơ tìm kiếm CSDL thư viện.
  - Tiền điều kiện:Dữ liệu mẫu tìm kiếm được nhập vào cửa sổ.Hậu điều kiện:Dữ liệu đến mô tơ tìm kiếm CSDL thư viện.
  - Các bước trong UC này:
    - Nhấn phím "Đệ trình" trên cửa sổ giao diện.
    - Kích hoạt cơ chế truyền dữ liệu qua mạng.
    - Dữ liệu đi đến mô tơ tìm kiếm CSDL thư viện.
    - Trên màn hình xuất hiện thông báodữ liệu đã gửi và chờ để đọc lại dữ liệu.

UC này liên quan đếnUC "Nhập liệu để tìm kiếm", chúng có quan hệ phụ thuộc <<*include*>>(hình 8.11).

- UC "Lấy danh sách đầu tài liệu tìm thấy".
  - Mô tả UC:Lấy danh sách sách hay tạp chí sau khi nhập dữ liệu vào cửa sổ và gửi đến CSDL thư viện.
  - Tác nhân kích hoạt :Độc giả.
  - Tiền điều kiện:Độc giả sử dụng cửa sổ tìm kiếm và liên lạc mạng LAN tốt,phím "Đệ trình" bị nhấn,Hậu điều kiện:Danh sách đầu tài liệu trên màn hình.
  - Các bước trong UC này:
    - Chờ trong khi tực hiện tìm kiếm.
    - CSDL thư viện truyền danh sách đầu tài liệu đến giao diện người dùng thông qua mạng.
    - Độc giả nhận danh sách đầu tài liệu phù hợp với mẫu dữ liệu đã nhập.

UC này liên quan đến UC "Truyền dữ liệu để tìm kiếm",chúng có quan hệ phụ thuộc <<*include*>>.



Hình 8.11 Quan hệ giữa các UC trong gói "Độc giả"

- UC "*Đặt trước*"
  - Mô tả UC:Độc giả muốn đặt trước đầu tài liệu sau khi tìm kiếm hay được tư vấn.
  - Tiền điều kiện:Có danh sách đầu sách hay tạp chí.Nhấn phím "Đặt trước" ;Hậu điều kiện:Bản ghi giữ chỗ được truyền đến thư viện điện tử.
  - Các bước trong UC này:
    - Độc giả nhận được danh sách đầu tài liệu.
    - Độc giả nhấn phíM "Đặt trước"tài liệu mà họ muốn.
    - Độc giả điền mẫu cho đặt trước.
- UC " *Truyền dữ liệu đặt trước*":

UC này tương tự như UC "Truyền dữ liệu để tìm kiếm", nó liên quan đến các UC "Thực hiện đặt trước" và "Nhận phản hồi".Kết nối là <<*include*>>.

- UC "*Nhận thông tin phản hồi*":
  - Mô tả UC:Sau khi nhấn phím "Đệ trình"trong cửa sổ đặt trước,phản hồi từ thư viện điện tử sẽ trở lại và thông báo cho độc giả trạng thái của đặt trước.

Trạng thái đặt trước có thể là còn đầu tài liệu mong muốn hay đã cho mượn hết.

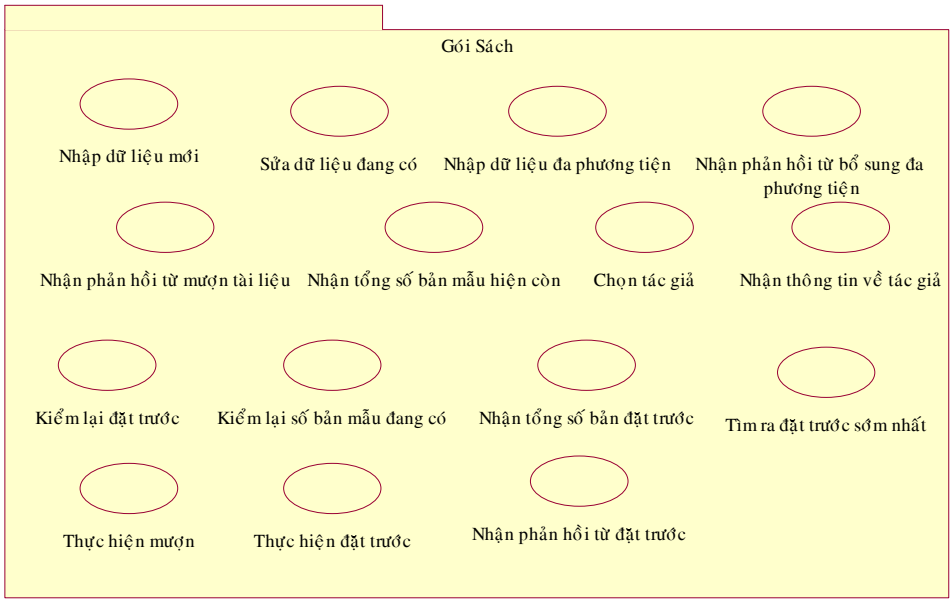
  - Tiền điều kiện:Phím "Đệ trình"bị nhấn,truyền thông trên mạng tốt ;Hậu điều kiện:Nhận trạng thái đặt trước.
  - Các bước trong UC này:
    - Chờ cho đến khi nhận được thông tin phản hồi.

- Truyền thông tin cần thiết qua mạng đến giao diện độc giả.
- Độc giả nhận trạng thái đặt trước.

- UC "Lấy danh sách đầu mục gợi ý":
  - Mô tả UC: Thủ thư kích hoạt chức năng gợi ý của hệ thống và nhận lại các đầu tài liệu khuyến cáo.
  - Tác nhân kích hoạt: Độc giả .
  - Tiền điều kiện: Phím "Gợi ý"bị nhấn; Hậu điều kiện:Nhận danh sách đầu sách.
  - Các bước trong UC này:
    - Kích hoạt chức năng gợi ý của hệ thống.
    - Thư viện điện tử kích hoạt thuật toán gợi ý để phát sinh danh sách đầu tài liệu.
    - Hiển thị danh sách đầu mục tài liệu gợi ý trên màn hình độc giả.

Gói Sách.Gói này bao gồm các UC sau đây (hình 8.12).

- Trường hợp sử dụng "*Nhập dữ liệu mới*":
  - Mô tả UC: Nhập dữ liệu tìm kiếm vào cửa sổ (form)sách.
  - Tác nhân kích hoạt: Nhân viên nhập liệu.
  - Tiền điều kiện: Cửa sổ nhập liệu đã được kích hoạt ,có sẵn dữ liệu cho sách mới .Hậu điều kiện :Dữ liệu nhập vào CSDL thư viện điện tử.
  - Các bước trong UC này:
    - Nhân viên nhập liệu kích hoạt giao diện nhập thông tin sách qua LAN hay Internet.
    - Mẫu nhập liệu xuất hiện trên màn hình.
    - Nhân viên nhập thông tin cần thiết.

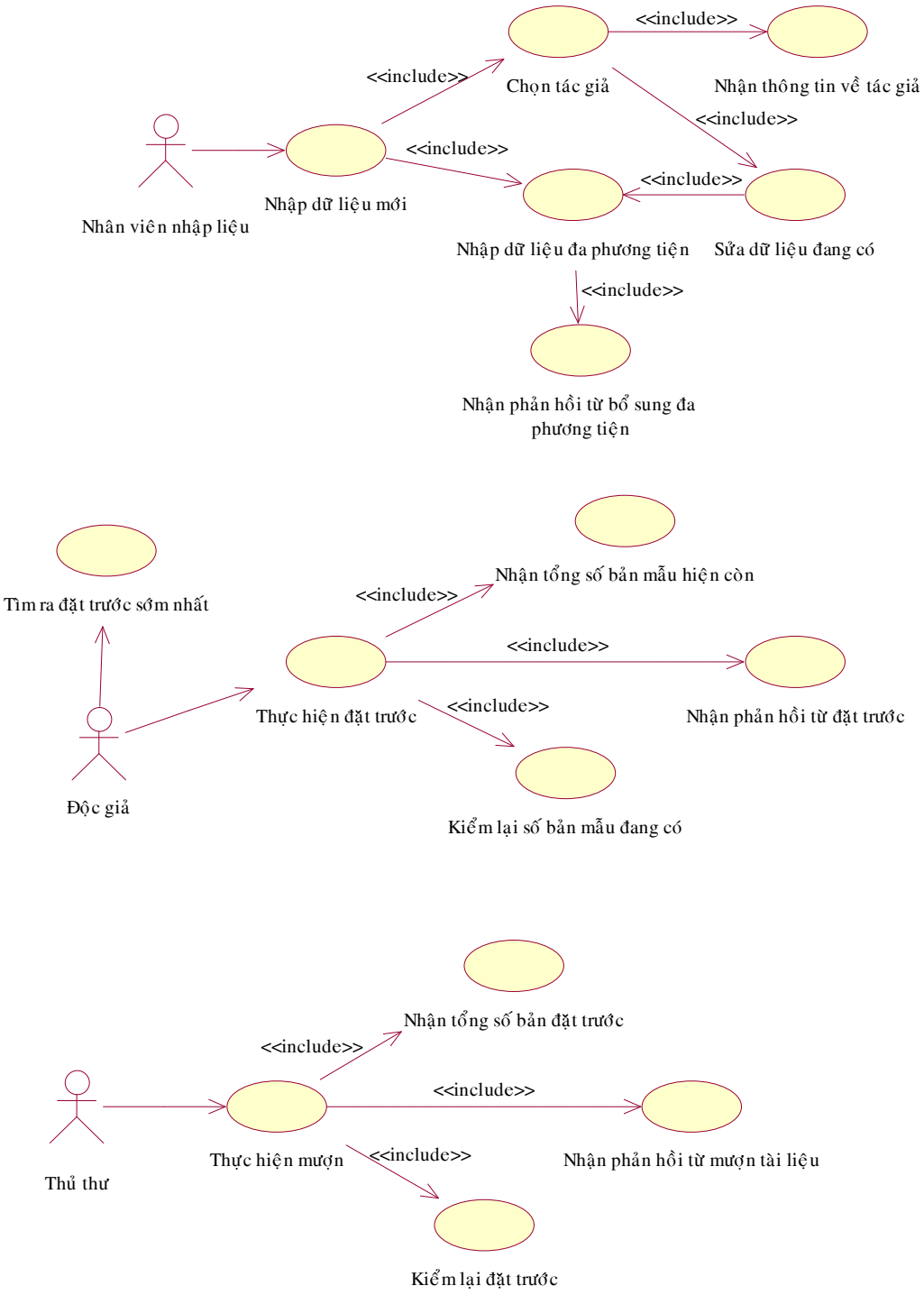


Hình 8.12 Gói UC "Sách"

- Trường hợp sử dụng "*Chọn tác giả*".
  - Mô tả UC: Chọn tác giả sách từ danh sách tác giả.
  - Tác nhân kích hoạt : Nhân viên nhập liệu.
  - Tiền điều kiện: Bản ghi tác giả có trong CSDL.Hậu điều kiện:Dữ liệu về tác giả sách.
  - Các bước trong UC này:

- Nhấn phím “Đệ trình”trong cửa sổ tìm kiếm.
- Kích hoạt cơ chế truyền dữ liệu trên mạng.
- Mẫu dữ liệu tìm kiếm đến mô tơ tìm kiếm trong CSDL.
- Trên màn hình độc giả xuất hiện thông báo đã truyền và chờ đến khi nhận được dữ liệu nào đó.

UC này liên quan đến UC “Nhập thông tin sách mới”,do vậy phải bổ sung quan hệ phụ thuộc<<include>>giữa chúng trong biểu đồ UC(hình8.13).



Hình 8.13 Quan hệ giữa các UC trong gói "Sách"

- Trường hợp sử dụng "*Khai thác dữ liệu tác giả*".
  - Mô tả UC: Khai thác toàn bộ thông tin cần thiết về tác giả đã lựa chọn từ UC "Chọn tác giả".
  - Tiền điều kiện: Tác giả đã được chọn từ danh sách tác giả.Hậu điều kiện: Thông tin về tác giả sách.

- Các bước trong UC này:
  - Sau khi tìm kiếm, CSDL chọn bản ghi tác giả và tách ra các thông tin cần thiết về tác giả.
  - Gửi thông tin đến cửa sổ nhập dữ liệu tác giả .
  - Hiển thị thông tin trên màn hình người sử dụng.

UC này chứaUC "*Chọn tác giả*".

- Trường hợp sử dụng: "*Nhập dữ liệu đa phương tiện*".
  - Mô tả UC: Nếu sách có dữ liệu đa phương tiện thì nó được nhập vào bản ghi sách.
  - Tiền điều kiện: Sách có dữ liệu đa phương tiện và tệp đa phương tiện đang được lưu ở đâu đó trong máy chủ, phím "Đa phương tiện" được nhấn.Hậu điều kiện:Dữ liệu đa phương tiện được nhập vào.
  - Các bước trong UC này:
    - Nếu sách có dữ liệu đa phương tiện, phím "Đa phương tiện"được nhấn và cửa sổ nhập liệu hiển thị.
    - Người sử dụng nhập liệu đa phương tiện, vị trí trên máy chủ nơi lưu dữ liệu lưu trữ.
    - Nhấn phím "Đệ trình" và bản ghi đa phương tiện được gửi đến CSDL.

UC này được gộp vào UC "Sửa dữ liệu của sách đã có" và UC "Nhập dữ liệu cho sách mới .

- Trường hợp sử dụng "*Nhận phản hồi từ bổ sung đa phương tiện*".
  - Mô tả UC: Sau khi làm đầy cửa sổ nhập ,CSDL gửi thông báo kết quả thực hiện tiến trình bổ sung dữ liệu đa phương tiện .
  - Tiền điều kiện: Cửa sổ nhập dữ liệu đa phương tiện đã được điền, có tệp đa phương tiện và mạng thông.Hậu điều kiện:Nhận được trạng thái giao dịch.
  - Các bước trong UC này:
    - Dữ liệu đa phương tiện đến CSDL, tiến trình tạo bản ghi mới bắt đầu.
    - Kết quả của tiến trình có thể tốt hay có thể hỏng
    - Loại kết thúc tiến trình tạo lập được trả lại cửa sổ nhập liệu sách.

UC này được gộp vào UC "Nhập dữ liệu đa phương tiện"

- Trường hợp sử dụng "*Nhận dữ liệu cho sách đang tồn tại*".
  - Mô tả UC:Nếu nhập nhầm thông tin quyển sách nào đó,có thể sửa lại chúng.
  - Tiền điều kiện:Bản ghi sách đã có,phím "Sửa"được nhấn,nhập sai dữ liệu sách.Hậu điều kiện:Thay đổi bản ghi sách.
  - Các bước trong UC này:
    - Người sử dụng nhấn phím "Sửa", cửa sổ chứa thôn g tin sách xuất hiện trên màn hình.
    - Chọn và sửa thông tin sai trong bản ghi
    - Nếu thay đổi thông tin tác giả hay thông tin đa phương tiện , tiến trình được nhập như dữ liệu mới được thực hiện.

- Trường hợp sử dụng "*Đặt trước*".
  - Mô tả UC:Độc giả có thể đặt trước sách sẽ mượn.
  - Tiền điều kiện:Sách phải có trong danh sách tìm kiếm từ tiến trình tìm kiếm hay tiến trình quản lý, độc giả muốn đặt trước.Hậu điều kiện:Dữ liệu của sách cần thiết và độc giả được truyền đến CSDL thư viện điện tử.
  - Các bước trong UC này:
    - Người sử dụng nhấn phím "Đặt trước" và nhập các thông tin cần thiết để đặt trước
    - CSDL nhận yêu cầu đặt trước.
    - Thực hiện kiểm tra dữ liệu và bản ghi CSDL.

- Hệ thống kiểm tra đầu sách còn lại, thựchiện đặt trước.
- Chờ đến khi nhận được thông tin phản hồi.

- Trường hợp sử dụng "*Nhận phản hồi từ đặt trước*".
  - Mô tả UC:Sau khi độc giả điền dữ liệu vào cửa sổ và nhấn phím "Đặt trước",hệ thống nhận yêu cầu và thực hiện tiến trình đặt trước rồi nhận thông báo trạng thái.
  - Tiền điều kiện: "Đặt trước"bị nhấn.Hậu điều kiện:Trạng thái đặt trước.
  - Các bước trong UC này:
    - Lấy trạng thái thực hiện đặt trước.
    - Gửi trạng thái đến người dùng qua mạng.
    - Thông điệp trạng thái đặt trước hiển thị trên màn hình.

  UC này và UC "Thực hiện đặt trước"có quan hệ phụ thuộc<<*include*>>.

- Trường hợp sử dụng "*Kiểm tra đầu sách còn trong thư viện*".
  - Mô tả UC:Đếm đầu sách còn chưa cho mượn, kết quả đượ gửi đến tiến trình đặt trước.
  - Tiền điều kiện:Có yêu cầu đặt trước.Hậu điều kiện :Tổng số đầu sách còn trong thư viện.
  - Các bước trong UC này:
    - Nhận tổng số đầu sách yêu cầu đặt trước.
    - Đếm số sách còn trong thư viện.
    - Gửi giá trị này cho hệ thống.

  UC này và UC "Thực hiện đặt trước"có quan hệ phụ thuộc<<*include*>>.

- Trường hợp sử dụng "Cho lại đầu sách còn trong thư viện".
  - Mô tả UC :Cho lại đầu sách còn chưa cho mượn.
  - Tiền điều kiện:Có yêu cầu một số đầu sách.Hậu điều kiện:Truyền tổng số đầu sách còn trong thư viện.
  - Các bước trong UC này:
    - Nhận tổng số đầu sách yêu cầu đặt trước truyền đến.
    - Kiểm tra kết quả để xác định trạng thái đặt trước.
    - Truyền trạng thái đặt trước tới hệ thống.

  UC này và UC "Kiểm tra đầu sách còn trong thư viện" có quan hệ phụ thuộc<<*include*>>.

- Trường hợp sử dụng "*Thực hiện mượn*".
  - Mô tả UC :Độc giả muốn mượn sách.
  - Tiền điều kiện:Độc giả phải có mặt tại thư viện,sách in, còn đầu sách trong thư viện,độc giả yêu cầu mượn.Hậu điều kiện:Yêu cầu mượn được truyền đến thư viện .
  - Các bước trong UC này:
    - Thủ thư kích hoạt giao diện để nhận sách trả hay cho mượn.
    - Thủ thư nhập dữ liệu của tài liệu và tổng số vào cửa sổ giao diện cho mượn.
    - Sau khi nhấn phím "Cho mượn",tiến trình cho mượn được thực hiện.

- Trường hợp sử dụng "*Nhận phản hồi từ tiến trình cho mượn*"
  - Mô tả UC:Sau khi nhấn phím "Cho mượn"trong cửa sổ giao diện, bản ghi đặt trước mới được ghi vào CSDL,kết qủa của phiên giao dịch này được trả lại giao diện cho mượn.
  - Tiền điều kiện:Truyền tin qua LAN,có phím "Cho mượn"trên giao diện, dữ liệu đã được nhập vào cửa sổ cho mượn.Hậu điều kiện:Trạng thái của phiên giao dịch.
  - Các bước trong UC này:
    - Chờ cho đến khi tiến trình cho mượn kết thúc ,nhận lại kết quả trạng thái cho mượn.
    - Truyền trạng thái qua mạng đến màn hình thủ thư.
    - Nếu còn đầu sách thì độc giả lấy nó.

UC này và UC "Thực hiện mượn"có quan hệ phụ thuộc<<*include*>>.

- Trường hợp sử dụng "*Nhận tổng số bản đặt trước*".

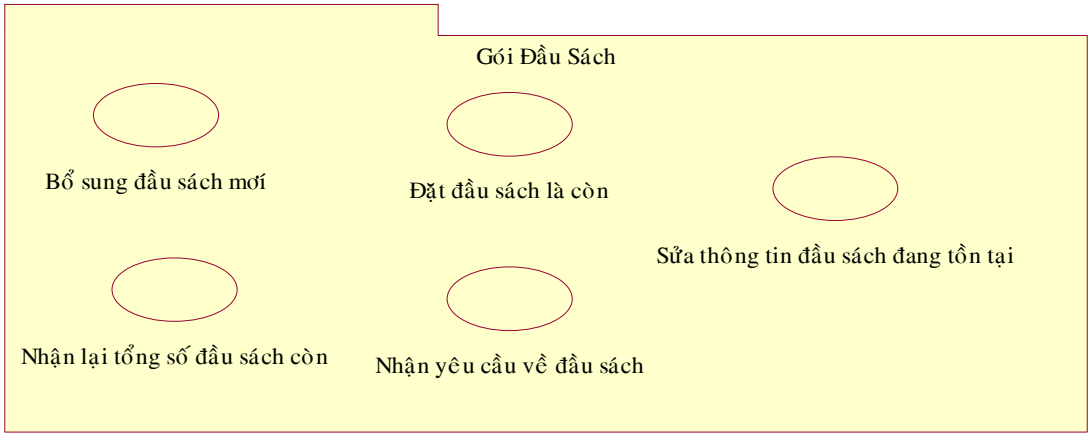
UC này xảy ra khi bản ghi mượn mới được tạo lập.Nếu sách được đặt trước mà không còn đầu sách thì tiến trình mượn thất bại.

- Trường hợp sử dụng "*Kiểm tra đặt trước*".

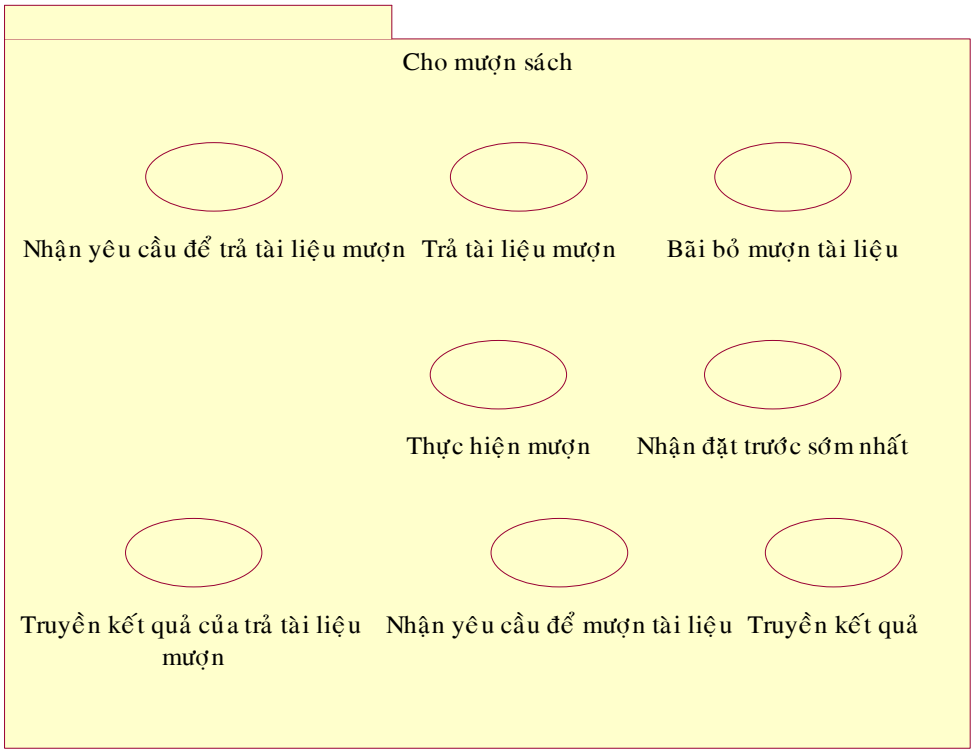
Nếu độc giả đã đặt trước sách để mượn, thì nếu đầu sách còn thì độc giả nậhn sách,nếu không thì tiến trình cho mượn thất bại.

- Trường hợp sử dụng "*Tìm lần đặt trước đầu tiên*".

UC này được sử dụng trong tiến trình cập nhật bản ghi đặt trước .Nếu độc giả yêu cầu đặt trước và không còn đầu sách,thì khi có đầu sách UC này tìm ra đặt trước sớm nhất cho đầu sách này,đặt lại trạng thái đầu sách từ "sẵn có" sang "đã đặt trước".Đặt lại đầu sách là sẵn sàng khi ai đó trả nó cho thư viện.



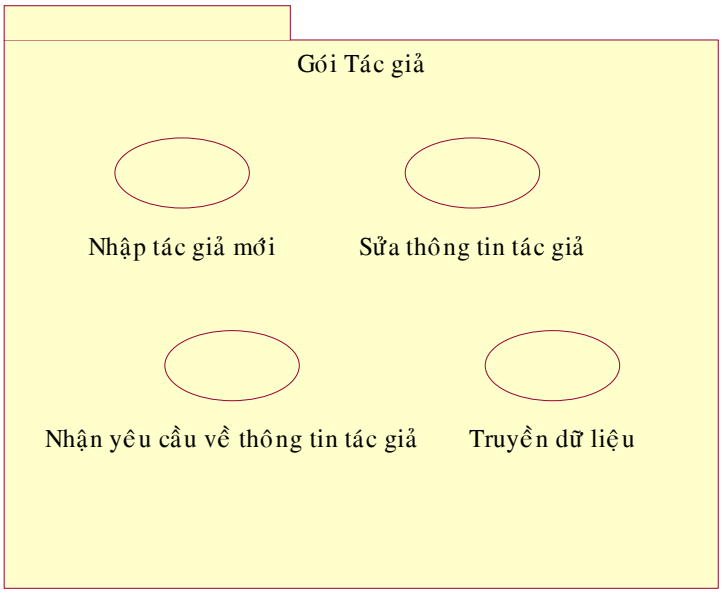
Hình 8.14 Gói UC "Đầu sách"



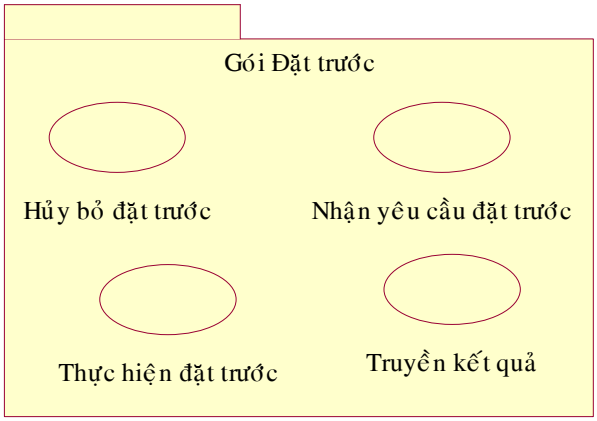
Hình 8.15 Gói UC "Cho mượn sách"

Giữa UC "Thực hiện mượn" và UC "Kiểm tra đặt trước",giữa UC "Thực hiện mượn" và "Nhận số bản đặt trước"có quan hệ <<*include*>>.

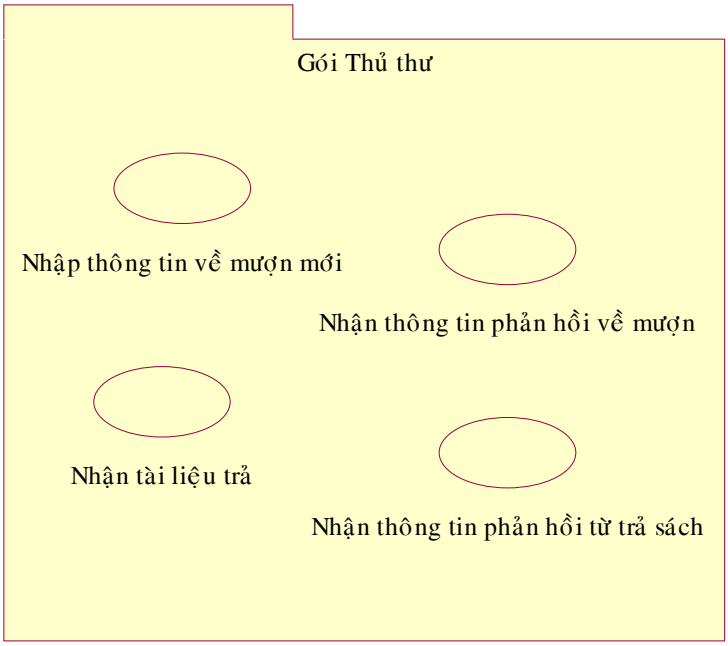
Việc phân tích trường hợp sử dụng được thực hiện tương tự cho các gói Đầu sách(hình 8.14),gói Cho mượn sách (hình 8.15),gói Tác giả (hình 8.16),gói Đặt trước (hình 8.17) và gói Thủ thư (hình 8.18).



Hình 8.16 Gói UC "Tác giả"



Hình 8.17 Gói UC "Đặt trước"



Hình 8.18 Gói UC "Thủ thư"

Tương tự ta có thể nhận ra các UC cho gói khác như gói Nhân viên nhập liệu,Đa phương tiện và gói Tạp chí.

8.4 BIỂU ĐỒ TƯƠNG TÁC

Một trong các cách tìm ra tương tác là xem xét các cấu phần hệ thống trong từng gói UC.Nhiệm vụ ở đây là chỉ ra các cấu phần hệ thống tương tác với nhau như thế nào để hoàn thiện từng UC .Hãy khảo sát gói "Độc giả" và gói "Sách" làm thí dụ.

8.4.1 - Tiến trình đặt trước sách để mượn

Giả sử rằng độc giả đặt trước sách có trong danh sách sách gợi ý.Để thực hiện thao tác này ta cần có sự tham gia của một số UC trong gói "Độc giả",bao gồm Lấy danh sách sách gợi ý,Thực

hiện đặt trước,Truyền dữ liệu đặt trước và nhận thông tin phản hồi (hình 8.11).Sau đây là mô tả các bước của từng UC.

Nhận danh sách sách gợi ý.

1. Kích hoạt phần giao diện về gợi ý sách mượn.
2. Sau khi nhấn phím "Gợi ý"trên giao diện,tín hiệu được gửi tới hệ thống để kích hoạt thuật toán gợi ý.
3. Hệ thống kích hoạt thuật toán gợi ý để phát sinh danh sách sách gợi ý.
4. Hệ thống gửi danh sách sách đến giao diện độc giả.
5. Hiển thị danh sách tài liệu trên màn hình độc giả.

Thực hiện đặt trước.

1. Độc giả nhận danh sách sách,tạp chí(từ tìm kiếm hay từ gợi ý).
2. Độc giả nhấn phím "Đặt trước"cho sách,tạp chí mong muốn.
3. Cửa sổ để xác nhận đặt trước xuất hiện.
4. Độc giả điền thông tin vào cửa sổ.

Truyền thông tin đặt trước.

1. Nhấn phím trên "Đệ trình"trên cửa sổ đặt trước.
2. Hệ thống truyền yêu cầu đặt trước đến CSDL Đặt trước.
3. Yêu cầu đến CSDL Đặt trước
4. Trên màn hình độc giả xuất hiện các thông báo "Đã gửi yêu cầu"và "Chờ xác nhận đã nhận yêu cầu".

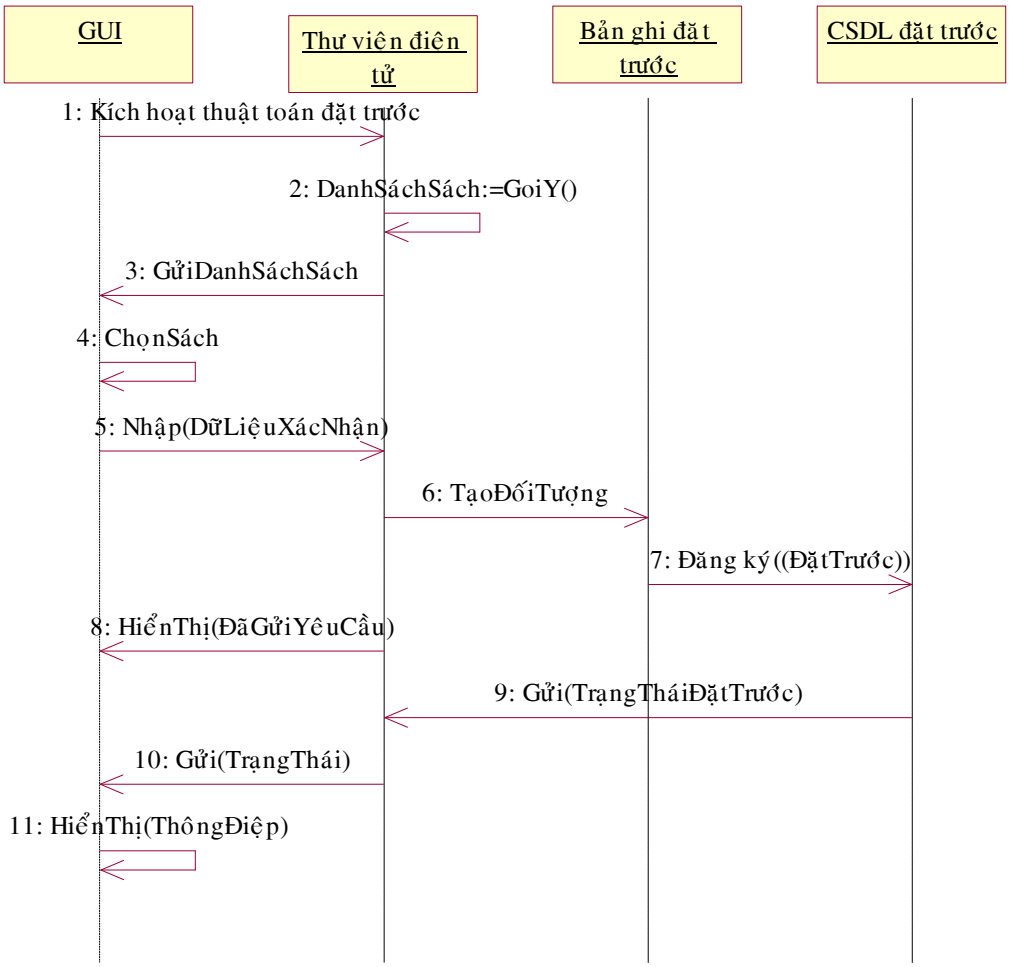
Trước khi đi đến UC "Nhận thông tin phản hồi"ta thấy các UC "Đặt trước mới" và UC "Truyền thông tin phản hồi"của gói "Đặt trước" sẽ tham gia vào tiến trình tương tác này.

Đặt trước mới.

1. Yêu cầu đến CSDL đặt trước,kích hoạt tiến trình tạo đặt trước để mượn
2. Kết quả của tiến trình là trạng thái đặt trước ,nó được gửi đến hệ thống.
3. Gửi trạng thái đăt trước đến hệ thống thư viện

Truyền thông tin phản hồi.

1. Hệ thống thư viện gửi trạng thái đặt trước đến giao diện độc giả .
2. Trạng thái đặt trước đến phía độc giả .



Hình 8.19 Biểu đồ trình tự của tiến trình đặt trước

Nhận thông tin phản hồi .

1. Giao diện độc giả nhận trạng thái do hệ thống gửi đến.
2. Giải mã trạng thái.
3. Hiển thị thông báo lên màn hình phù hợp với trạng thái đặt trước .

Biểu đồ trình tự của tiến trình đặt trước tài liệu gợi ý để mượn như trên hình 8.19.

8.4.2 - Tiến trình mượn sách , tạp chí.

Giả sử rằng độc giả muốn mượn sách,ông ta đi đến thư viện,sách muốn mượn có trong thư viện và là sách in (không phải sách điện tử).Các UC trong gói "Sách" cần thiết cho thao tác này(hình 8.13),chúng bao gồm "Thực hiện mượn","Truyền dữ liệu mượn", "Nhận tổng số bản đặt trước", "Kiểm tra lại đặt trước" và "Nhận phản hồi từ mượn tài liệu".

Thực hiện mượn.

1. Kích hoạt tiến trình mượn/trả từ giao diện thủ thư
2. Cửa sổ cho mượn xuất hiện trên màn hình thủ thư

3. Thủ thư nhập dữ liệu cần thiết
4. Dữ liệu mượn được truyền đến hệ thống thư viện

Truyền dữ liệu mượn tài liệu

1. Thủ thư nhấn phím "Đệ trình"trên cửa sổ cho mượn,dữ liệu mượn được truyền đến hệ thống thư viện .
2. Hệ thống nhận dữ liệu ,yêu cầu mới về mượn tài liệu được gửi đến CSDL cho mượn.
3. Yêu cầu được gửi đến CSDL cho mượn,chờ thông tin xác nhận.

Khi dữ liệu được truyền đến CSDL cho mượn,cơ chế bổ sung bản ghi mới vào CSDL được kích hoạt .Thao tác này sẽ kết hợp với các UC trong gói "Cho mượn"(hình 8.15),bao gồm "Nhận yêu cầu mượn sách", "Thực hiện mượn","Truyền kết quả mượn".

Nhận yêu cầu mượn sách .

1. CSDL cho mượn nhận yêu cầu thực hiện tạo bản ghi mới.
2. Kích hoạt tiến trình tạo và bổ sung bản ghi mới vào CSDL.

Thực hiện mượn .

1. CSDL kích hoạt cơ chế tạo bản ghi mới.
2. Nhận kết quả tạo bản ghi.
3. Gửi kết quả đến hệ thống thư viện.

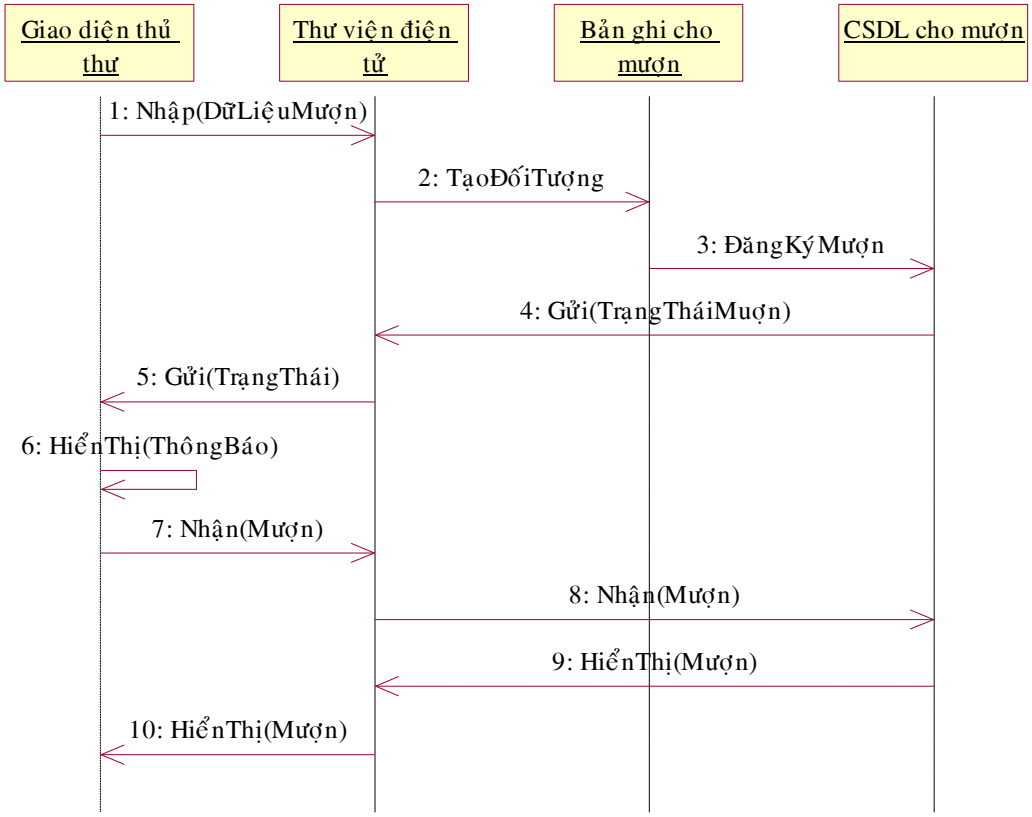
Truyền kết quả mượn .

1. Kết quả tiến trình mượn được gửi đến hệ thống .
2. Truyền kết quả đến giao diện thủ thư .

Tiến trình được tiếp tục lại với UC trong gói "Sách" .

Nhận phản hồi từ mượn sách .

1. Giao diện thủ thư đã nhận kết quả từ tiến trình mượn.
2. Giải mã kết quả , hiển thị thông điệp trên màn hình.
3. Gửi yêu cầu nhận trạng thái trong bản ghi mượn của độc giả đến CSDL cho mượn.
4. Danh sách bản ghi được lấy ra từ CSDL cho mượn.
5. Hiển thị danh sách trên màn hình.

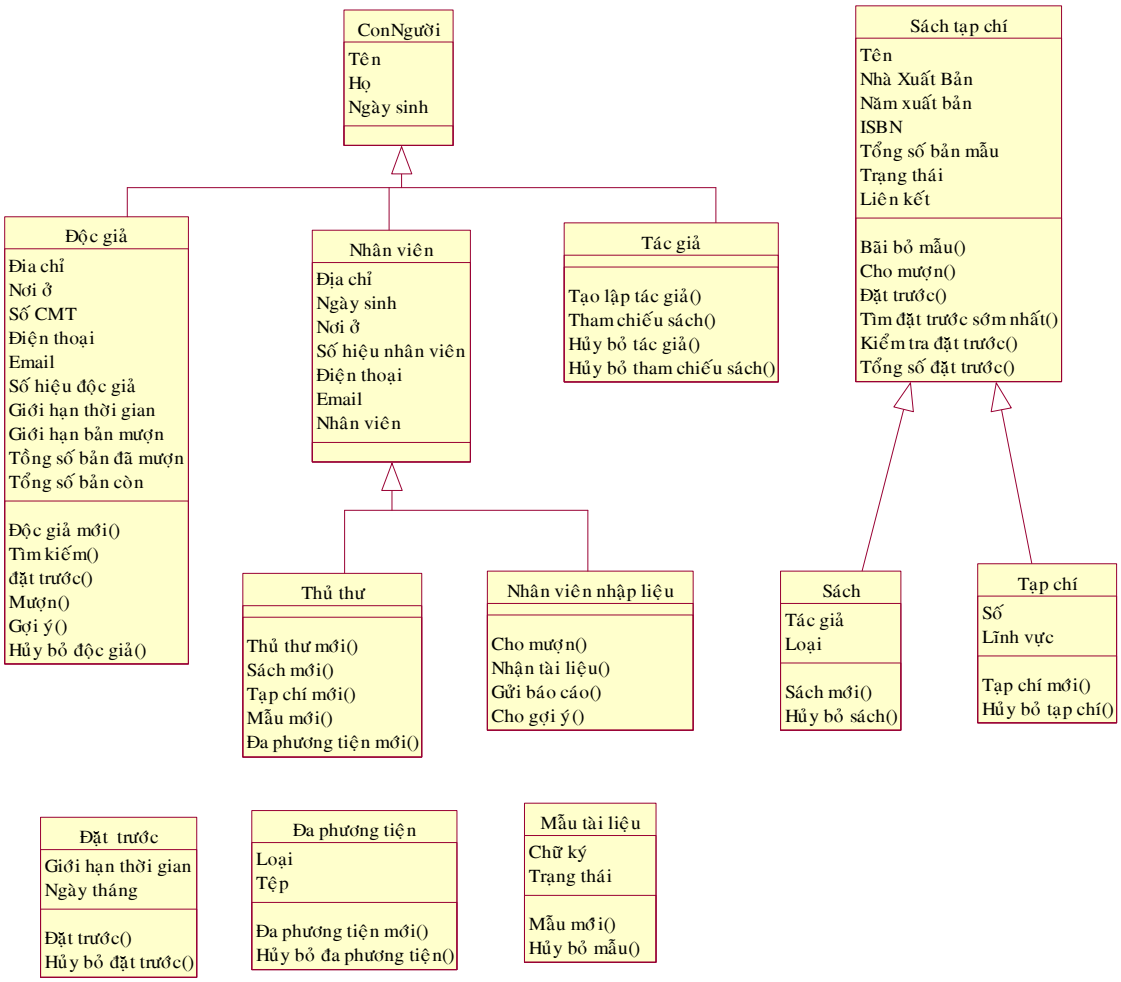


Hình 8.20 Biểu đồ trình tự của tiến trình mượn sách

Biểu đồ trình tự của các bước trong các UC này được vẽ trên hình 8.20.

8.5 BIỂU ĐỒ LỚP.

Phần 2 của chương này đã tìm ra được một số lp của hệ thống như lớp "Độc giả",lớp "Tài liệu",lớp "Thủ thư"…Tiếp theo ta phải bổ sung các thuộc tính và các thao tác cho chúng để hình thành biểu đồ lớp.



Hình 8.21 Một vài lớp của hệ thống thư viện

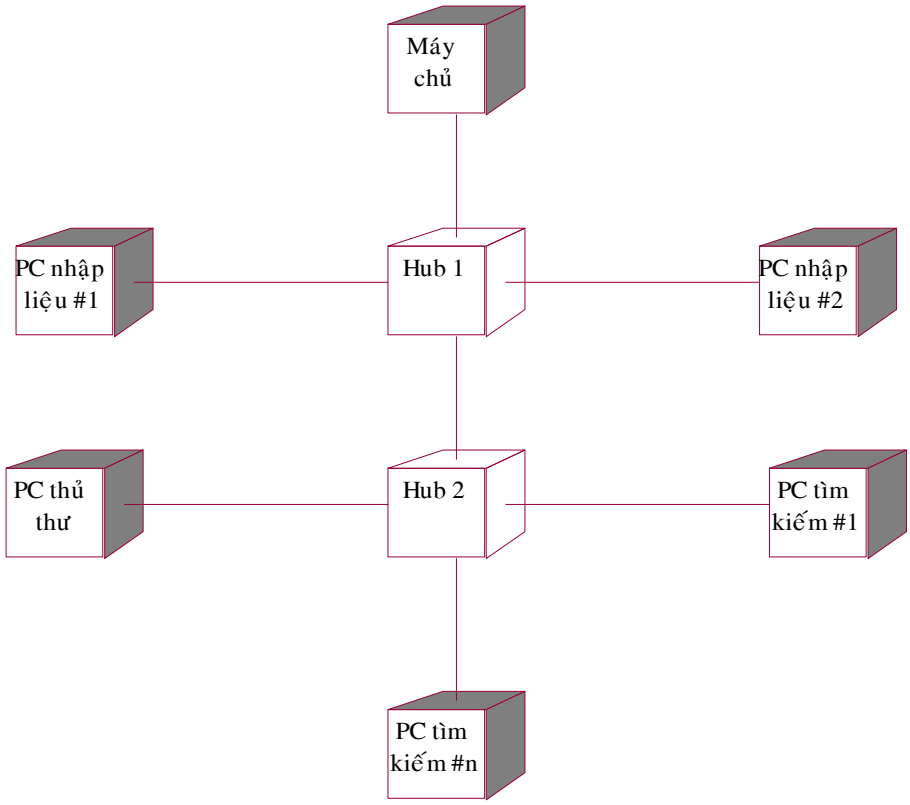
Bổ sung thuộc tính và thao tác.

Từ danh sách danh từ và động từ của phân tích lĩnh vực ta có các lớp "Độc giả",lớp "Tác giả",lớp "Nhân viên",lớp "Sách",lớp "Tạp chí",lớp "Bản mẫu"…như trên hình 8.21.Các lớp "Thủ thư", "Nhân viên nhập liệu", "Độc giả" và "Tác giả" kế thừa từ lớp trừu tượng "Con người".Các lớp "Sách'và "Tạp chí" kế thừa từ lớp trừu tượng "Sách tạp chí",lớp này có thuộc tính "Trạng Thái"để lưu loại tài liệu(bản in hay bản điện tử).Lớp "Mẫu tài liệu" có thuộc tính "Trạng thái" để cho biết bản mẫu hiện hành đã cho mượn hay còn trong thư viện.Thuộc tính "Loại" của lớp "Đa phương tiện" để lưu loại dữ liệu như văn bản , hình ảnh,âm thanh,còn thuộc tính "Tệp"của nó lưu địa chỉ vật lý của tệp dữ liệu.Trong lớp "Độc giả"có thuộc tính suy diễn "Tổng số bản còn" để lưu số đầu tài liệu còn trong thư viện,đó là hiệu của hai thuộc tính "Giới hạn bản mượn" và "Tổng số bản đã mượn".Thuộc tính "Số hiệu độc giả"của lớp "Độc giả"lưu số thẻ thư viện.

8.6 BIỂU ĐỒ TRIỂN KHAI

Kỹ sư hệ thống bắt đầu suy nghĩ về kiến trúc hệ thống, xem xét tôpô và cài đặt mạng như thế nào.Sau đó họ chỉ ra mođun phần mềm nào sẽ được đặt tại nút nào trong mạng.Gỉa sử rằng đội

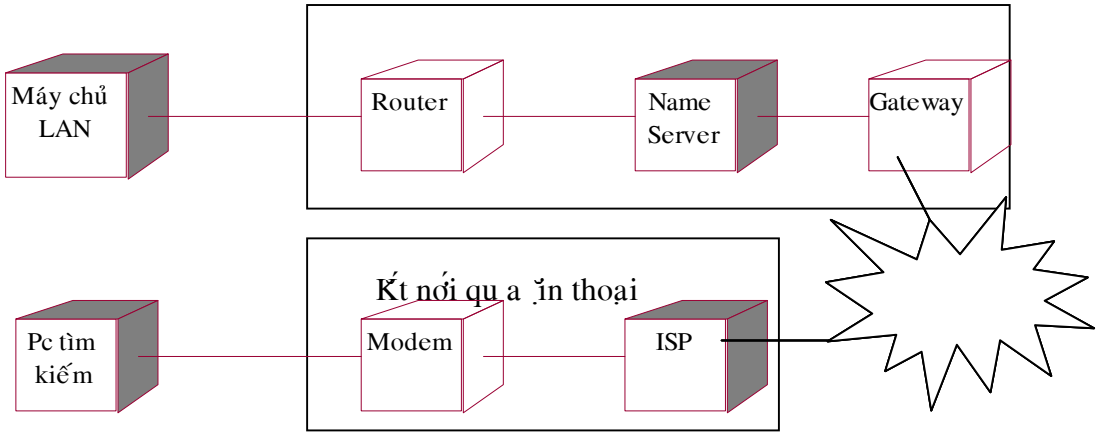
ngũ phát triển quyết định xây dựng mạng LAN cho hệ thống, giao diện Internet được cài đặt trên máy chủ LAN.Biểu đồ triển khai của LAN có thể như hình vẽ 8.22.



Hình 8.22 Biểu đồ thành phần của LAN thư viện

Máy chủ và các PC nhập liệu được đặt trong cùng một phòng làm việc với một thiết bị Hub.Một Hub khác được đặt ngoài hành lang để nối đến các PC tìm kiếm.Máy tính của thủ thư được đặt tại nơi cho mượn tài liệu.

Nếu hệ thống làm việc trên môi trường Internet thì cần bổ sung thiết bị làm giao diện Internet. Ta có thể giữ nguyên cấu trúc mạng LAN,sau đó bổ sung phương tiện kết nối mạng vào phía máy chủ ,bao gồm Name Server (là CSDL đánh giá kết nối),Router(thiết bị hỗ trợ kết nối các mạng) và Gateway(thiết bị truyền thông tin từ giao thức này sang giao thức khác).Chính Gateway cho phép các thành viên Internet xâm nhập hệ thống thư viện từ nhà hay công sở .Phía độc giả chỉ cần có PC kết nối Internet,trình duyệt ,tên và mật khẩu để xâm nhập trang WEB của thư viện điện tử .Biểu đồ triển khai của hệ thống thư viện trên cơ sở Internet như trên hình 8.23.

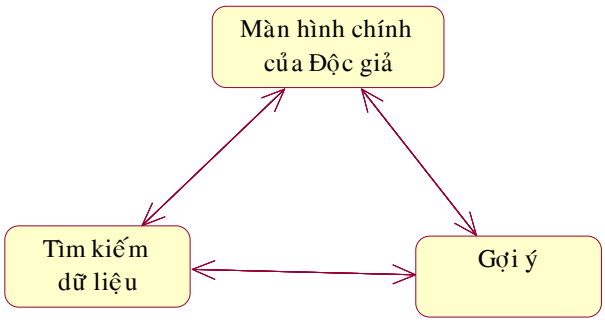


Hình 8.23 Biểu đồ triển khai hệ thống thư viện trên cơ sở Web

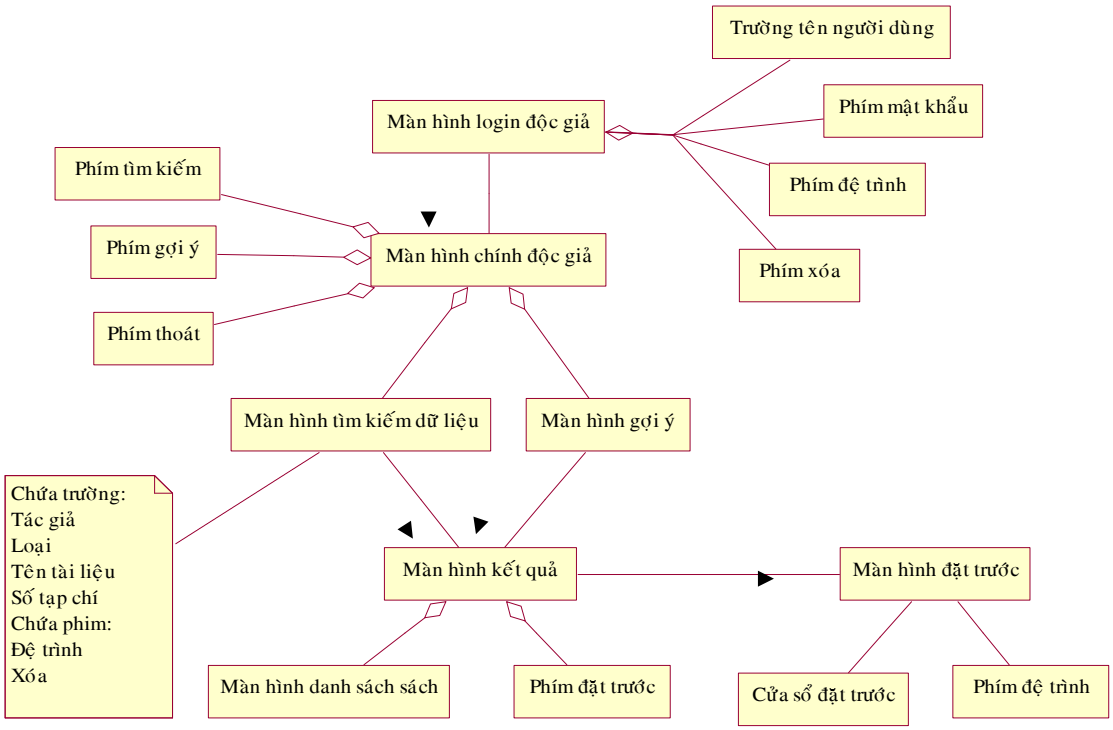
8.7 THIẾT KẾ GIAO DIỆN

Thiết kế giao diện giữa hệ thống và người sử dụng là xác định cách sử dụng hệ thống , nó liên quan đến nghệ thuật và khoa học .Thiết kế giao diện có một số nguyên tắc như trình bày sau đây:

- Hiểu cai người sử dụng làm,người thiết kế thường phải thực hiện phân tích nhiệm vụ để hiểu bản chất công việc của người sử dụng .Phân tích nhiệm vụ tương tự phân tích UC.
- Làm cho người dùng có cảm giác luôn điều khiển được tương tác,họ có thể hủy bỏ tương tác bất kỳ khi nào.
- Cung cấp nhiều cách để thực hiện một hoạt động liên quan giao diện(thí dụng đóng cửa sổ và tệp)
- Nên bắt đầu hiển thị thông tin từ góc trên,trái của màn hình.Thông tin phải dễ đọc và dễ hiểu.
- Hạn chế số màu sử dụng vì quá nhiều màu sẽ làm người sử dụng mất tập trung vào nhiệm vụ.Hạn chế sử dụng chữ Italic vì khó đọc trên màn hình.
- Cố gắng sử dụng các hộp thoại cùng kích thước.Các xâu ký tự được chỉnh hàng từ phía trái.

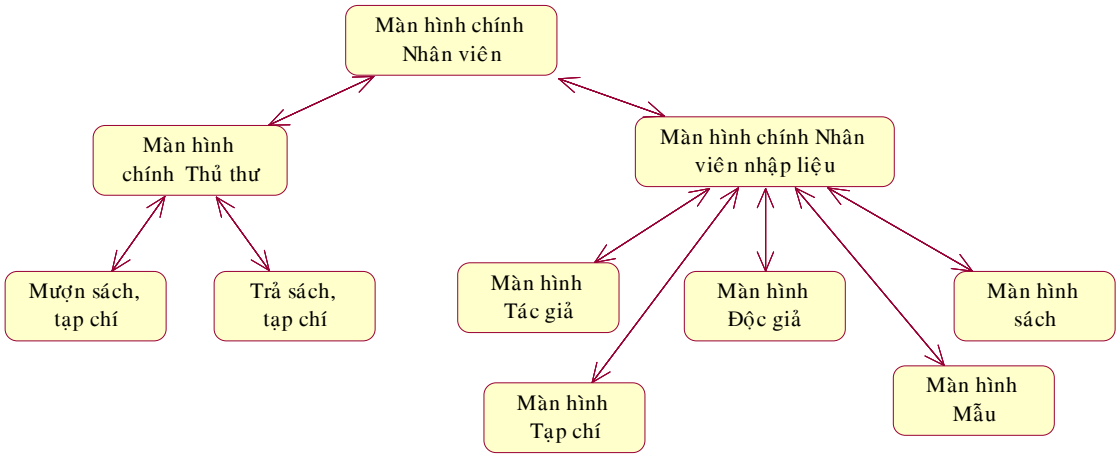


Hình 8.24 Giao diện độc giả

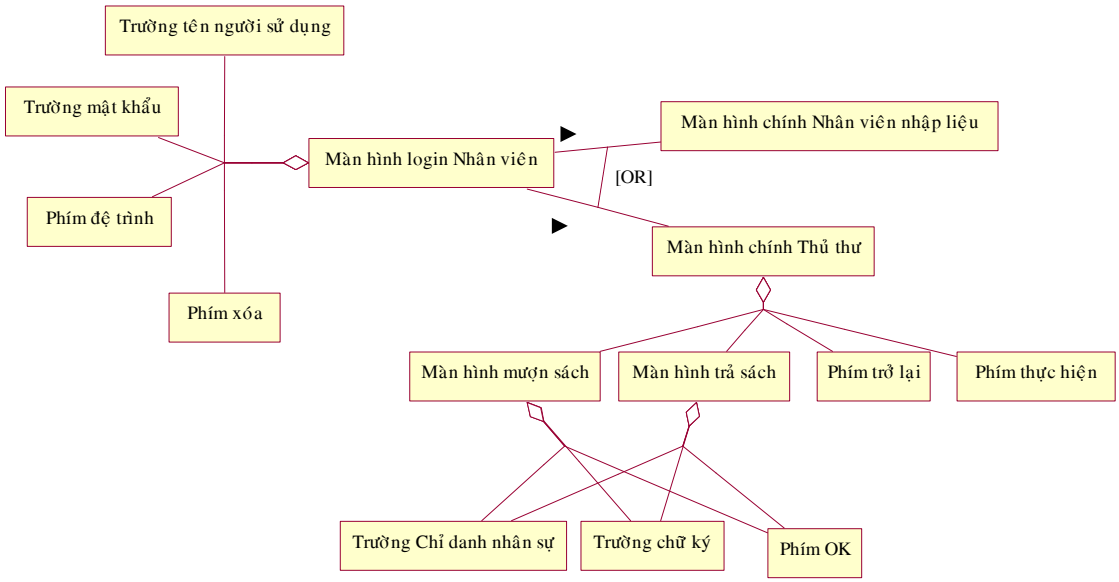


Hình 8.25 Phân cấp màn hình Độc giả

Thông thường makét giao diện được thiết kế trên giấy để mọi người tham gia dự án trao đổi thống nhất.Tiếp theo là xây dựng các màn hình giao diện mẫu.Từ các biểu đồ UML phân tích trong các phần trên,ta có thể vẽ ra được giao diện người sử dụng.Thí dụ,hình 8.24 là biểu đồ trạng thái của giao diện "Độc giả".Hình 8.25 là biểu đồ tổng hợp phân cấp màn hình "Độc giả".Hình 8.26 là biểu đồ trạng thái mức cao của giao diện "Nhân viên".Hình 8.27 là biểu đồ tổng hợp của phân cấp màn hình "Thủ thư".



Hình 8.26 Biểu đồ trạng thái của giao diện Nhân viên



Hình 8.27 Phân cấp màn hình Nhân viên

PHỤC LỤC

CHƯƠNG 9

MÃ TRÌNH PHÁT SINH TRONG ROSE

Phụ lục này trình bày một số mẫu mã trình do Rational Rose phát sinh tự động.Các thí dụ trong phụ lục này không mô tả toàn bộ khả năng phát sinh mã trình của Rose mà chỉ mô tả một số tương đương giữa UML và các ngôn ngữ lập trình như C++, Java và SQL.

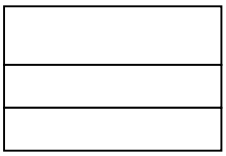
9.1 PHÁT SINH MÃ TRÌNH C++

9.1.1 - Các lớp

9.1.1.1 - Lớp rỗng

Rose phát sinh cấu tử ,hủy tử ,cấu tử sao chép,toán tử gán và hai toán tử so sánh.Chú ý rằng để cho dễ đọc mã trình thì các thao tác trong thí dụ này sẽ không được viết lặp lại trong các thí dụ về sau.

Mô hình

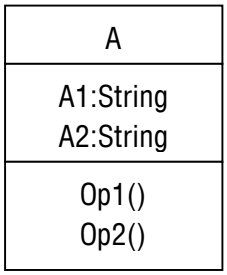


Mã trình

```
#ifndef A_h
#define A_h 1
class A
{
public:
    //## Constructor (generated)
    A ();
    A (const A& right);
    //## Destructor (generated)
    ~ A () ;//## Assingnment Operation (generated)
    Const A& operator =(const A& right)

    //## Equality Operations (generated)
    int operator == (const A& right)const;
    int operator != (const A$ right)const;
} ;
#endif
```

9.1.1.2 - Lớp có thuộc tính và thoa tác

Mô hình Mã trình




```
class A
{
public:
    …
    //## Other operations (specified)
    void Op 1()
    void Op 2()
    const String get_A 1() const;
    void set_A (const String value);

    const String get_A2() const;
    void set_A2(const String value);
private:
    String A1;
    String A2;
    };

inline const String A:: get_A1() const
{
    Return A1;
}
inline void A:: set_A1(const String value)
{
    A1=value;
}

inline const String A::get_A2() const
{
    Return A2;
}
inline void A::set_A2(const String value)
{
    A2=value;
}
```

9.1.1.3 - Lớp tham số (template)

Mô hình **Mã trình**

```
template <argtype Att>
class D
{
public:
    D();
    D(const D<Att>& right);
    ~D ();

    const D<Att>& operator=(const D<Att>& right);
    int operator == (const D<Att>& right) const;
    int operator != (const D<Att>& right) const;
…
};
```

9.1.1.4 - Lớp tiện ích (utility)

Mọi thao tác của lớp tiện ích đều có tiền tố là từ khóa static.

Mô hình **Mã trình**

```
class F
{
pubilc:
    static void Op 1();
    static void Op 2();
};
```

9.1.2 - Quan hệ kết hợp

9.1.2.1 - Kết hợp 1-1

Nếu kết hợp theo tham chiếu thì mã trình được phát sinh cho kết hợp bao gồm con trỏ trong vùng private của lớp tham gia kết hợp.

Mô hình

Mã trình

```
Class A
{
…
const B* get_Rb() const;
void set_Rb (B* const value);
private:
```

```
B* Rb;
};
inline const B* A:: get_Rb () const
{
return Rb;
}
inline void A:: set_Rb (B*const value)
{
Rb = value;
}

class B
{
…
const A* get_Ra () const;
void set_Ra(A* const value);
private:
A* Ra;
};
inline const A* B :: get_Ra() const
{
return Ra;
}
inline void A:: set_Ra (A* const value)
{
Ra=value;
}
```

9.1.2.2 - Kết hợp N-1

Mã trình được phát sinh cho kết hợp bao gồm con trỏ trong vùng private của lớp tham gia kết hợp.Tính nhiều (*multiplicity*) 0… * được cài đặt bằng tập có kích thước không giới hạn.

Mô hình

Mã trình

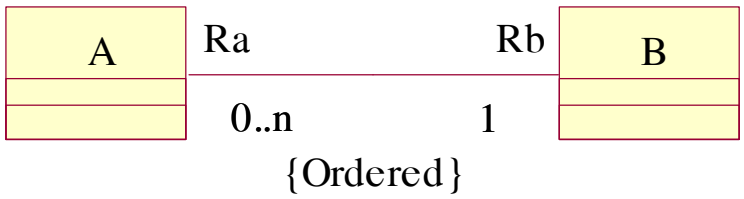
```
class B
{
…
const UnboundedSetByReference<A> get_Ra() const;
void set_Ra (const UnboundedSetByReference<A>value);
Private:
```

```
unboundedSetByReference<A>Ra;
};

inline const UnboundedSetByReference<A> B:: get_Ra () const
{
return Ra;
}
inline void A:: set_Ra(const UnboundedSetByReference<A>value)
{
Ra=value;
}
```

9.1.2.3 - Kết hợp N-1 có ràng buộc

Mã trình được phát sinh cho kết hợp bao gồm con trỏ torng vùng private của lớp tham gia kết hợp.Do có ràng buộc {Ordered} cho nên tính nhiều (multiplicity) 0…* được cài đặt bằng danh sách có kích thước không giới hạn thay cho tập như thí dụ trên đây.

Mô hình

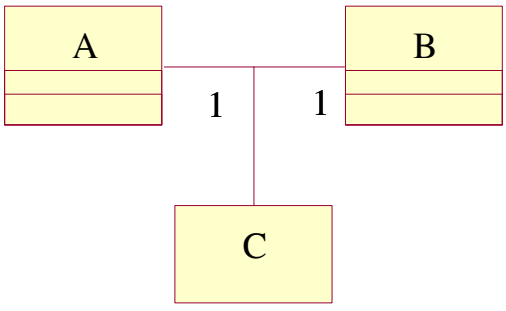


Mã trình

```
class B
{
…
const UnboundedListByReference<A> get_Ra() const;
void set_Ra(const UnboundedListByReferenc<A>value);
private:
unboundedListByReference<A>Ra;
};
```

9.1.2.4 - Lớp kết hợp 1-1

Mô hình



Mã trình

```
class A; classB
class C
{
…
const B* get_the_B () const;
void set_the_B (B* const value);
private:
A* the_A;
B* the_B;
};

#include “C.h”
class A
{
…
const C* get_the_C() const;
void set_the_C(C* const value);
private:
C* the_C;
};

#include “C.h”
class B
{
…
const C* get_the_C() const;
void set_the_C(C* const value);
private:
C* the_C;
};
```

9.1.2.5 - Lớp kết hợp N-N

Mô hình

Mã trình

```
#include “C.h”
class B
{
…
const UnboundedSetByReference<A> get_the_C() const;
void set_the_C(const UnboundedSetByReference<A> value);
private:
```

unboundedSetByReference<A>the_C;
};

9.1.3 - Quan hệ phụ thuộc tập hợp

9.1.3.1 - Phụ thuộc tập hợp (aggregation)1-1

Mô hình

Mã trình

```
#include “B.h”
class A
{
…
const B* get_the_B () const;
void set_the_B (B* const value);
private:
B* the_B;
};
#include “A.h”
class B
{
…
const A* get_Ra () const;
void set_Ra (A* const value);
private:
A* Ra;
};
```

9.1.3.2 - Phụ thuộc tập hợp với khả năng dẫn đường hạn chế

Mô hình

Mã trình

```
class A
{
…
private:
```

```
};

#include “A.h”
class B
{
…
const A* get_Ra () const;
void set_Ra (A* const value);
private:
A*Ra;
};
```

9.1.3.3 - Quan hệ gộp (composition) 1-1

Mô hình

Mã trình

```
#include “B.h”
class A
{
…
const B* get_the_B() const;
void set_the_B (B* Const value);
private:
B* THE_b;
};
#include “A.h”
class B
{
…
const A get_Ra () const;
void set_Ra (const Avalue);
private:
A Ra;
};
```

9.1.3.4 - Phụ thuộc tập hợp 1-N

Mô hình

Mã trình

```
#include “A.h”
class B
{
…
const UnboundedSetByReference<A>get_Ra () const;
void set_Ra (const UnboundedSetByReference<A>value);
private:
unboundedSetByReference <A>Ra;
};
```

9.1.4 - Quan hệ kế thừa

9.1.4.1 - Kết thừa đơn

Mô hình

Mã trình

```
#include “A.h”
class B:public A
{
    …
};
```

9.1.4.2 - Kế thừa bội

Mô hình

Mã trình

```
#include “A1.h”
#include “A2.h”
class B:public A2,public A1
{
    …
};
```

9.2 PHÁT SINH MÃ TRÌNH JAVA

Các đoạn mã trình mẫu dưới đây được Rose phát sinh từ mô hình UML.Phần này không mô tả toàn bộ khả năng của Rose mà chỉ mô tả tương ứng giữa một vài biểu đồ UMLvới ngôn ngữ Java.

9.2.1 - Các lớp

9.2.1.1 - Lớp rỗng

Rose phát sinh cấu tử,hủy tử.Chú ý rằng để cho dễ đọc mã trình thì các thao tác này sẽ không được viết lặp lại trong các thí dụ về sau.

Mô hình **Mã trình**

```
public final class A {
    public A () {
        super ();
        …
    }
    protected void finalize () throws Throwable {
        super.finalize ();
        …
    }
    …
}
```

9.2.1.2 - Lớp có thuộc tính và thao tác

Mô hình **Mã trình**

```
public final class A {
    private String m_A1;
    private String m_A2;
    public void Op 1() {
        …
    }
    public void Op 2 () {
        …
    }
    …
}
```

9.2.1.3 - Lớp trừu tượng

Mô hình **Mã trình**

Insert Figure: 08710D01 public abstract class A {

```
    …
}
```

9.2.1.4 - Giao diện

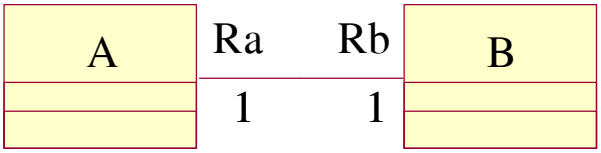
Mô hình	Mã trình

Insert Figure: 08710D01	public interface I_A {
	…
	}

9.2.2 - Quan hệ kết hợp

9.2.2.1 - Kết hợp 1-1

Mô hình

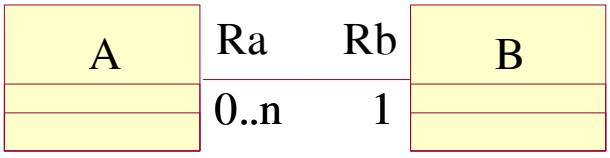


Mã trình

```
public class A {
public B m_Rb;
…
};

public class B {
public A m_Ra;
…
};
```

9.2.2.2 - Kết hợp 1-N

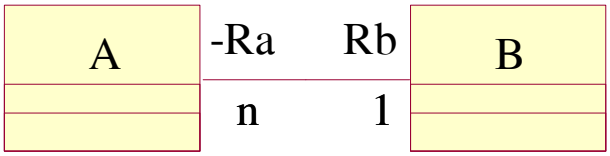
Mô hình



Mã trình

```
public class B {
public A m_Ra;
…
}
```

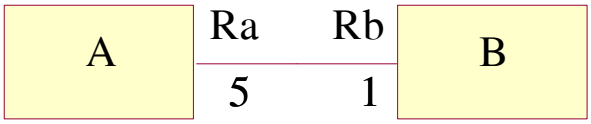
Mô hình



Mã trình

```
public class B {
public Vector m_Ra = new Vector ();
…
}
```

Mô hình



Khi tính nhiều có hạn chế thì kết hợp được cài đặt bằng mảng .

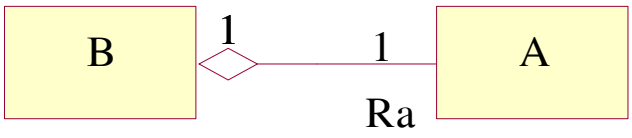
Mã trình

```
public class B {
private A[] m_Ra = new A[5];
…
}
```

9.2.3 - Quan hệ phụ thuộc tập hợp

9.2.3.1 - Phụ thuộc tập hợp (aggregation) 1-1

Mô hình

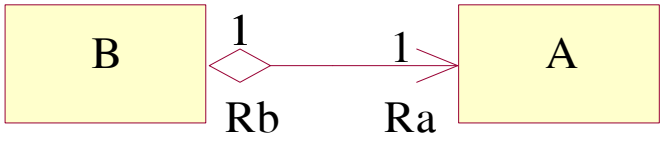


Mã trình

```
public class A {
public B m_B;
…
}
public class B {
public A m_Ra;
…
}
```

Phụ thuộc tập hợp với khả năng dẫn đường hạn chế

Mô hình



Mã trình

```
public class A {
…
}
public class B {
public A m_Ra;
…
}
```

9.2.4 - Quan hệ kế thừa

9.2.4.1 - Kế thừa đơn

Mô hình **Mã trình**

```
public class B extends A {
…
}
```

9.2.4.2 - Kế thừa giữa các giao diện

Mô hình **Mã trình**

```
public interface I_C extends I_A {
    …
}
public interface I_C extends I_A, I_B {
    …
}
```

9.2.4.3 - Cài đặt giao diện bằng lớp trừu tượng

Mô hình **Mã trình**

```
public abstract class A implements I_A {
        …
}
```

9.2.4.4 - Cài đặt giao diện bằng lớp

Mô hình	Mã trình

```
public class A implements I_A {
        …
}
```

9.2.4.5 - Cài đặt một vài giao diện bằng lớp

Mô hình	Mã trình

```
public class A implements I_A,I_B {
    …
}
```

9.3 PHÁT SINH MÃ TRÌNH VISUAL BASIC

Các đoạn mã trình mẫu dưới đây được Rose phát sinh từ mô hình UML. Phần này không mô tả toàn bộ khả năng của Rose mà chỉ mô tả tương ứng giữa một vài biểu đồ UML với ngôn ngữ Visual Basic.

9.3.1 - Các lớp

9.3.1.1 - Lớp rỗng

Mô hình	Mã trình

```
Option Base 0

Private  Sub Class_Initialize ()
End Sub

Private Sub Class_Terminate ()
End Sub
```

9.3.1.2 - Lớp có thuộc tính và thao tác

Mô hình **Mã trình**

```
Option Base 0
Public A1 As String
Public A2 As String
Private  Sub Class_Initialize ()
End Sub

Private Sub Class_Terminate ()
End Sub
Public Sub Op1()
    On Error GoTo Op1Err
    …
End Sub
Op1Err:
    Call RaiseError (MyUnhandleError, "A:Op1 Method")
End Sub
Public Property Get Op2() As Boolean
    On Error GoTo Op1Err
    …
    Exit Property
    Op2Err:
        Call RaiseError (MyUnhandleError, "A:Op2 Method")
End Property
```

9.3.1.3 - Lớp có thuộc tính và thao tác

Mô hình **Mã trình**

```
Option Base 0

Private  Sub Class_Initialize ()
End Sub

Private Sub Class_Terminate ()
End Sub

Public Sub Op2()
End Sub

Public Property Get Op2() As Boolean
End Property
```

9.3.2 - Quan hệ kết hợp

9.3.2.1 - Kết hợp 1-1

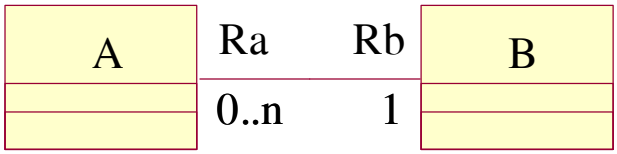
Mô hình

Mã trình

```
Option Base 0
Public Rb As B

Private Sub Class_Terminate ()
End Sub

Private Sub Class_Initialize ()
End Sub

Option Base 0
Public Ra As A

Private Sub Class_Terminate ()
End Sub

Private Sub Class_Initialize ()
End Sub
```

9.3.2.2 - Kết hợp 1-N

Mô hình



Mã trình

```
Option Base 0
Public Rb As B

Private Sub Class_Initialize ()
End Sub

Private  Sub Class_Terminate ()
End Sub

Option Base 0
Public Ra As Collection

Private Sub Class_Initialize ()
End Sub

Private Sub Class_Terminate ()
End Sub
```

9.3.3 - Quan hệ kế thừa đơn

Mô hình **Mã trình**

```
Option Base 0
Implements A
Local superclass object (generated)
Private mAObject As New A

Private Sub Class_Initialize ()
End Sub

Private Sub Class_Terminate ()
End Sub

 Public Sub Op 2()
End Sub

Private Sub A_Op1 ()
```

```
mAObject.Op1
End Sub
```

9.4 PHÁT SINH MÃ TRÌNH SQL.

Các đoạn mã trình mẫu dưới đây được Rose phát sinh từ mô hìnhUML. Phần này không mô tả toàn bộ khả năng của Rose mà chỉ mô tả tướng ứng giữa một vài biểu đồ UML với ngôn ngữ ANSI SQL.

9.4.1 - Các lớp

9.4.1.1 - Lớp rỗng

Mô hình **Mã trình**

```
CREATE TABLE T_A(
        A_Id NUMBER (5),
        PRIMARY KEY (A_Id)
)
```

9.4.1.2 - Lớp có thuộc tính và thao tác

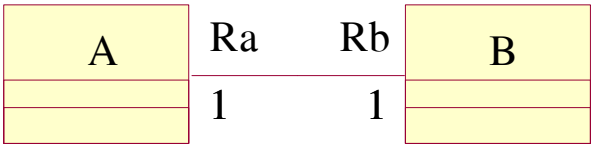
Mô hình **Mã trình**

```
CREATE TABLE T_A (
    A_ IdNUMBER (5)
    A1 VARCHAR (),
    A2 VARCHAR (),
    PRIMARY KEY (A_Id)
)
```

9.4.2 - Quan hệ kết hợp

9.4.2.1 - Kết hợp 1-1

Mô hình

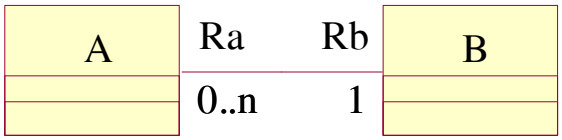


Mã trình

```
CREATE TABLE T_B (
B_Id NUMBER (5),
PRIMARY KEY (B_Id)
)

CREATE TABLE T_A (
A_Id NUMBER (5)
B_Id NUMBER (5) REFERENCE T_B (B_Id),
PRIMARY KEY (A_Id)
)
```

9.4.2.2 - Kết hợp N-1

Mô hình

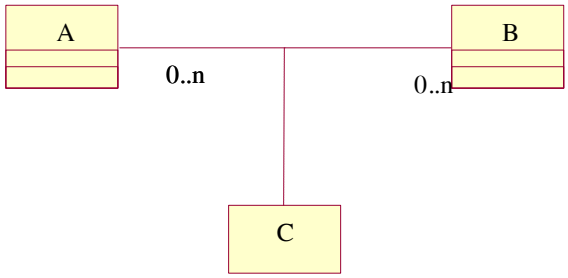


Mã trình

```
CREATE TABLE T_B(
B_Id NUMBER (5),
PRIMARY KEY (B_Id)
)

CREATE TABLE T-A(
B_Id NUMBER (5) REFERENCES T_B (B_Id),
A_Id NUMBER (5),
PRIMARY KEY (A_Id)
)
```

9.4.2.3 - Lớp kết hợp N-N

Mô hình



Mã trình

```
CREATE TABLE T_A(
A_Id NUMBER (5),
PRIMARY KEY (A_Id)
)
CREATE TABLE T_B(
B_Id NUMBER (5),
PRIMARY KEY (B_Id)
)
CREATE TABLE T_C (
A_Id NUMBER (5) REFERENCES T_A(A_Id)ON DELETE CASCADE,
B_Id NUMBER (5) REFERENCES T-B (B_Id)ON DELETE CASCADE,
PRIMARY KEY  (A_Id,B_Id)
)
```

9.4.3 - Quan hệ kế thừa

Trong các thí dụ sau đây, mỗi lớp được cài đặt trong một bảng.

9.4.3.1 - Kế thừa đơn

Mô hình

Mã trình

```
CREATE TABLE T_A (
    A_Id NUMBER (5),
    PRIMARY KEY (A_Id)
)

CREATE TABLE T_B (
    A_Id NUMBER(5) REFERENCES T_A (A_Id),
    PRIMARY KEY (A_Id)
)
```

9.4.3.2 - <u>Kế thừa bội</u>

Mô hình

Mã trình

```
CREATE TABLE T_A1 (
    A1_Id NUMBER (5),
    PRIMARY KEY (A1_Id)
)

CREATE TABLE T_A2 (
    A2_Id NUMBER (5),
    PRIMARY KEY (A2_Id)
)

CREATE TABLE T_B (
A1_Id NUMBER (5) REFERENCES T_A1 (A1_Id),
A2_Id NUMBER (5) REFERENCES T_A2 (A2_Id),
PRIMARY KEY (A1_Id, A2_Id)
)
```