MỤC LỤC

CHƯƠNG I. ĐẠI CƯƠNG VỀ PHÂN TÍCH THIẾT KẾ HỆ THỐNG

1. Các hệ thống thông tin

Ngày nay, hệ thống thông tin đã được ứng dụng trong mọi lĩnh vựa khác nhau của đời sống xã hội. Tuỳ theo quan điểm mà có thể phân loại các hệ thống thông tin theo các tiêu chí khác nhau. Xét về mặt ứng dụng, hệ thống thông tin có thể được phân chia thành một số dạng như sau:

- **Hệ thống thông tin quản lý:** Bao gồm các hệ thống thông tin hỗ trợ các hoạt động nghiệp vụ và quản lý của các doanh nghiệp, các tổ chức. Ví dụ các hệ thống quản lý nhân sự, hệ thống kế toán, hệ thống tính cước và chăm sóc khách hàng, hệ thống quản lý thư viện, hệ thống đào tạo trực tuyến ...
- **Các hệ thống Website:** là các hệ thống có nhiệm vụ cung cấp thông tin cho người dùng trên môi trường mạng Internet. Các hệ thống Website có đặc điểm là thông tin cung cấp cho người dùng có tính đa dạng (có thể là tin tức hoặc các dạng file đa phương tiện) và được cập nhật thường xuyên.
- **Hệ thống thương mại điện tử:** Là các hệ thống website đặc biệt phục vụ việc trao đổi mua bán hàng hoá, dich vụ trên môi trường Internet. Hệ thống thương mại điện tử bao gồm cả các nền tảng hỗ trợ các giao thức mua bán, các hình thức thanh toán, chuyển giao hàng hoá.
- **Hệ thống điều khiển:** là các hệ thống phần mềm gắn với các thiết bị phần cứng hoặc các hệ thống khác nhằm mục đích điều khiển và giám sát hoạt động của thiết bị hay hệ thống đó.

Mỗi loại hệ thống thông tin có những đặc trưng riêng và cũng đặt ra những yêu cầu riêng cho việc phát triển hệ thống. Ví dụ, các hệ thống điều khiển đòi hỏi những yêu cầu về môi trường phát triển, hệ điều hành và ngôn ngữ lập trình riêng; các hệ website thực thi các chức năng trên mội trường mạng phân tán đòi hỏi cách phát triển riêng...Do vậy, không có một phương pháp luận chung cho tất cả các dạng hệ thống thông tin. Phạm vi của tài liệu này nhằm giới thiệu một số khái niệm cơ bản của UML cho phát phiển các hệ thống và để dễ dàng minh hoạ chúng ta sẽ xem xét vấn đề phát triển dạng hệ thống thông tin phổ biến nhất là hệ **thống thông tin quản lý**.

2. Phân tích thiết kế là gì, tại sao phải phân tích thiết kế hệ thống ?

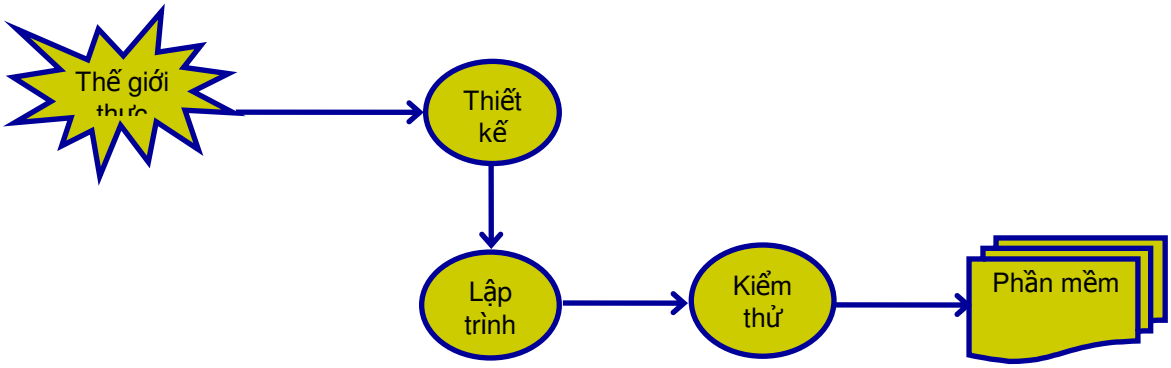
Hệ thống thông tin tin học là một ứng dụng đầy đủ và toàn diện nhất các thành tựu của công nghệ thông tin vào tổ chức, quản lý. Ngày nay, không một tổ chức hay đơn vị nào là không có nhu cầu xây dựng hệ thống thông tin. Không những nhu cầu xây dựng hệ thống thông tin tăng lên, mà quy mô và mức độ phức tạp của chúng cũng không ngừng tăng lên. Do đặc thù của hệ thống thông tin là sản phẩm "không nhìn thấy", nên phân tích thiết kế trở thành một yêu cầu bắt buộc để có được một hệ thống tốt.

Trong thực tế, để giải quyết một vấn đề hay một bài toán chúng ta cần xác định rõ yêu cầu của bài toán, xác định phương pháp và các bước để giải bài toán đó sao cho hiệu quả nhất, nhanh nhất và kết quả chính xác nhất.

Phân tích thiết kế hệ thống thông tin là quá trình tìm hiểu và mô phỏng lại hiện tượng, quy trình nghiệp vụ trong thế giới thực từ đó xây dựng hệ thống để giải quyết bài toán đặt ra trên máy tính (Hình 1.1).

Chất lượng phân tích thiết kế là nhân tố quyết định chất lượng phần mềm, không phân tích hoặc phân tích không tốt sẽ dẫn đến phần mềm chất lượng thấp:

- Không quản lý được những thay đổi về yêu cầu
- Khó kiểm thử
- Khó bảo trì
- Không có tính tiến hóa
- Không tái sử dụng được



Hình 1.1 Mô phỏng quá trình phát triển HTTT

Theo điều tra của IBM, thì những sai sót trong phân tích và thiết kế làm cho chi phí bảo trì trung bình của các hệ thống thông tin chiếm tới gần 60% tổng chi phí. Một lỗi bỏ sót trong giai đoạn phân tích đến khi lập trình và cài đặt mới phát hiện ra thì chi phí sửa chữa tăng 40 lần, và nếu để đến giai đoạn bảo trì mới phát hiện ra thì chi phí sửa chữa tăng 90 lần. Thêm vào đó, nếu thiếu các tài liệu phân tích thiết kế có thể dẫn đến hệ thống không thể bảo trì.

Thiết kế tốt sẽ đem lại một sản phẩm phần mềm chất lượng tốt:

- Dễ dàng thay đổi theo các yêu cầu của người sử dụng
- Dễ dàng kiểm thử
- Dễ bảo trì
- Có tính tiến hóa cao
- Có khả năng tái sử dụng

Do tầm quan trọng và nhu cầu thực tế, phân tích các hệ thống thông tin đã trở thành một nghề nghiệp có tính chuyên môn cao. Một kỹ sư công nghệ thông tin bất kỳ không thể không biết đọc các bản vẽ phân tích và thiết kế hệ thống thông tin. Một kỹ sư CNTT sau một năm có thể trở thành lập trình viên giỏi, thì họ cần phải mất nhiều năm mới trở thành một nhà phân tích và thiết kế viên và sau nhiều năm nữa mới trở thành một nhà phân tích thiết kế viên giỏi.

3. Các cách tiếp cận phân tích thiết kế hệ thống

Trong những năm 70 - 80, phương pháp hướng cấu trúc được coi là phương pháp chuẩn để phát triển phần mềm. Tuy nhiên, phương pháp này tỏ ra không phù hợp trong phát triển các hệ phần mềm lớn và đặc biệt là kém hiệu quả trong sử dụng lại - một yêu cầu quan trọng trong công nghiệp phần mềm. Thập niên 90 chứng kiến sự nở rộ trong nghiên cứu và xây dựng phương pháp luận phát triển phần mềm hướng đối tượng và nhanh chóng trở thành phổ biến trong công nghiệp phần mềm ngày nay. Để hiểu rõ phần nào sự khác biệt này phần này dành so sánh một số khác biệt giữa hai phương pháp này.

3.1 Giới thiệu về phương pháp phát triển hướng cấu trúc

Đặc trưng của phương pháp hướng cấu trúc là phân chia chương trình chính thành nhiều chương trình con, mỗi chương trình con nhằm đến thực hiện một công việc xác định. Trong phương pháp hướng cấu trúc, phần mềm được thiết kế dựa trên một trong hai hướng: hướng dữ liệu và hướng hành động.

- Cách tiếp cận hướng dữ liệu xây dựng phần mềm dựa trên việc phân rã phần mềm theo các chức năng cần đáp ứng và dữ liệu cho các chức năng đó. Cách tiếp cận hướng dữ liệu sẽ giúp cho những người phát triển hệ thống dễ dàng xây dựng ngân hàng dữ liệu.
- Cách tiếp cận hướng hành động lại tập trung phân tích hệ phần mềm dựa trên các hoạt động thực thi các chức năng của phần mềm đó.

Cách thức thực hiện của phương pháp hướng cấu trúc là **phương pháp thiết kế từ trên xuống (top-down)**. Phương pháp này tiến hành phân rã bài toán thành các bài toán nhỏ hơn, rồi tiếp tục phân rã các bài toán con cho đến khi nhận được các bài toán có thể cài đặt được ngay sử dụng các hàm của ngôn ngữ lập trình hướng cấu trúc.

Phương pháp hướng cấu trúc có ưu điểm là tư duy phân tích thiết kế rõ ràng, chương trình sáng sủa dễ hiểu. Tuy nhiên, phương pháp này có một số nhược điểm sau:

- Không hỗ trợ việc sử dụng lại. Các chương trình hướng cấu trúc phụ thuộc chặt chẽ vào cấu trúc dữ liệu và bài toán cụ thể, do đó không thể dùng lại một modul nào đó trong phần mềm này cho phần mềm mới với các yêu cầu về dữ liệu khác.

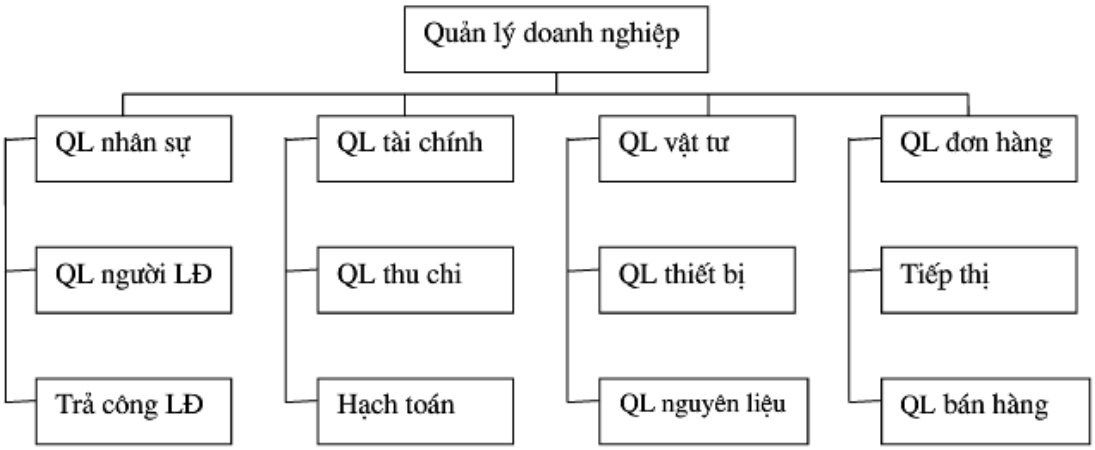
- Không phù hợp cho phát triển các phần mềm lớn. Nếu hệ thống thông tin lớn, việc phân ra thành các bài toán con cũng như phân các bài toán con thành các modul và quản lý mối quan hệ giữa các modul đó sẽ là không phải là dễ dàng và dễ gây ra các lỗi trong phân tích và thiết kế hệ thống, cũng như khó kiểm thử và bảo trì.

Mô hình dữ liệu trong phương pháp này trả lời cho câu hỏi Hệ thống sử dụng dữ liệu gì? Mô tả các dữ liệu chính sẽ có trong hệ thống và mối quan hệ ràng buộc giữa chúng sử dụng sơ đồ quan hệ thực thể, các bảng thuộc tính, các ràng buộc dữ liệu; mô hình thực thể (Entity Relationship Diagram - ERD) phản ánh hệ thống đầy đủ hơn, bổ sung cho BFD (Bussiness Function Diagram-mô hình phân rã chức năng); kho dữ liệu (Data store): Ký hiệu bằng 2 đường kẻ song song hoặc hình chữ nhật tròn góc, biểu diễn thông tin cần lưu trữ (sẽ là file hoặc CSDL)

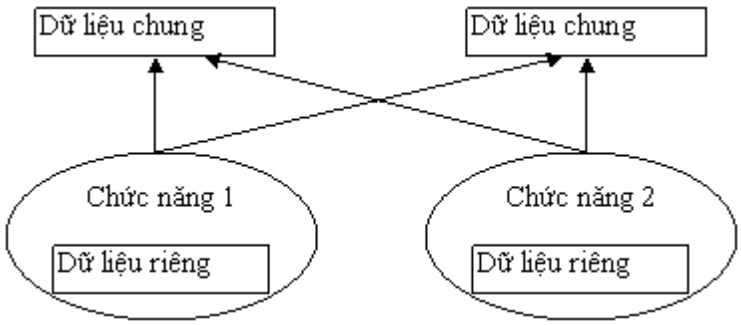
Kho dữ liệu

Tác nhân ngoài: Người/nhóm người/tổ chức bên ngoài hệ thống tiếp xúc với hệ thống. Đó là nguồn cung cấp thông tin, là nguồn sống còn của hệ thống. Tác nhân trong: Chức năng/quá trình trong hệ thống. Các mô hình có quan hệ mật thiết, xây dựng theo thứ tự: BFD, ERD, DFD(Data Flow Diagram) khi định hướng lập trình, thể hiện rõ quan hệ. Nếu cần làm rõ các yêu cầu người dùng, DFD được xây dựng trước thể hiện quy trình nghiệp vụ, sau đó xây dựng BFD và ERD

Ví dụ: Sơ đồ phân rã chức năng cho hệ thống Quản lý doanh nghiệp



Hình 1.2 Sơ đồ phân rã chức năng



Hình 1.3 Mối quan hệ giữa các chức năng trong hệ thống

3.2 Giới thiệu về phương pháp phát triển hệ thống hướng đối tượng

Khác với phương pháp hướng cấu trúc chỉ tập trung hoặc vào dữ liệu hoặc vào hành động, phương pháp hướng đối tượng tập trung vào cả hai khía cạnh của hệ thống là dữ liệu và hành động.

Cách tiếp cận hướng đối tượng là một lối tư duy theo cách ánh xạ các thành phần trong bài toán vào các đối tượng ngoài đời thực. Với cách tiếp cận này, một hệ thống được chia tương ứng thành các thành phần nhỏ gọi là các đối tượng, mỗi đối tượng bao gồm đầy đủ cả dữ liệu và hành động liên quan đến đối tượng đó. Các đối tượng trong một hệ thống tương đối độc lập với nhau và phần mềm sẽ được xây dựng bằng cách kết hợp các đối tượng đó lại với nhau thông qua các mối quan hệ và tương tác giữa chúng. Các nguyên tắc cơ bản của phương pháp hướng đối tượng bao gồm :

- Trừu tượng hóa (abstraction): trong phương pháp hướng đối tượng, các thực thể phần mềm được mô hình hóa dưới dạng các đối tượng. Các đối tượng này được trừu tượng hóa ở mức cao hơn dựa trên thuộc tính và phương thức mô tả đối tượng để tạo thành các lớp. Các lớp

cũng sẽ được trừu tượng hóa ở mức cao hơn nữa để tạo thành một sơ đồ các lớp được kế thừa lẫn nhau. Trong phương pháp hướng đối tượng có thể tồn tại những lớp không có đối tượng tương ứng, gọi là lớp trừu tượng. Như vậy, nguyên tắc cơ bản để xây dựng các khái niệm trong hướng đối tượng là sự trừu tượng hóa theo các mức độ khác nhau.

- Tính đóng gói (encapsulation) và ẩn dấu thông tin: các đối tượng có thể có những phương thức hoặc thuộc tính riêng (kiểu private) mà các đối tượng khác không thể sử dụng được. Dựa trên nguyên tắc ẩn giấu thông tin này, cài đặt của các đối tượng sẽ hoàn toàn độc lập với các đối tượng khác, các lớp độc lập với nhau và cao hơn nữa là cài đặt của hệ thống hoàn toàn độc lập với người sử dụng cũng như các hệ thống khác sử dụng kết quả của nó.
- Tính modul hóa (modularity): các bài toán sẽ được phân chia thành những vấn đề nhỏ hơn, đơn giản và quản lý được.
- Tính phân cấp (hierarchy): cấu trúc chung của một hệ thống hướng đối tượng là dạng phân cấp theo các mức độ trừu tượng từ cao đến thấp.

Ưu điểm nổi bật của phương pháp hướng đối tượng là đã giải quyết được các vấn đề nảy sinh với phương pháp hướng cấu trúc:

- Hỗ trợ sử dụng lại mã nguồn : Chương trình lập trình theo phương pháp hướng đối tượng thường được chia thành các gói là các nhóm của các lớp đối tượng khác nhau. Các gói này hoạt động tương đối độc lập và hoàn toàn có thể sử dụng lại trong các hệ thống thông tin tương tự.
- Phù hợp với các hệ thống lớn: Phương pháp hướng đối tượng không chia bài toán thành các bài toán nhỏ mà tập trung vào việc xác định các đối tượng, dữ liệu và hành động gắn với đối tượng và mối quan hệ giữa các đối tượng. Các đối tượng hoạt động độc lập và chỉ thực hiện hành động khi nhận được yêu cầu từ các đối tượng khác. Vì vậy, phương pháp này hỗ trợ phân tích, thiết kế và quản lý một hệ thống lớn, có thể mô tả các hoạt động nghiệp vụ phức tạp bởi quá trình phân tích thiết kế không phụ thuộc vào số biến dữ liệu hay số lượng thao tác cần thực hiện mà chỉ quan tâm đến các đối tượng tồn tại trong hệ thống đó.

Ví dụ: Cũng vẫn với bài toán Quản lý doanh nghiệp. Với phương pháp tiếp cận này việc phân tích bài toán xoay quanh việc xác định các lớp, đối tượng của bài toán:

Nhân sự	Vật tư	Hàng hóa	Đơn hàng

…

4. Quy trình/Chu trình phát triển hệ thống thông tin

Việc phát triển các hệ thống thông tin không chỉ đơn giản là lập trình mà luôn được xem như một tiến trình hoàn chỉnh.

Tiến trình phần mềm là phương cách sản xuất ra phần mềm với các thành phần chủ yếu bao gồm: mô hình vòng đời phát triển phần mềm, các công cụ hỗ trợ cho phát triển phần mềm và những người trong nhóm phát triển phần mềm.

Như vậy, tiến trình phát triển phần mềm nói chung là sự kết hợp cả hai khía cạnh kỹ thuật (vòng đời phát triển, phương pháp phát triển, các công cụ và ngôn ngữ sử dụng, …) và khía cạnh quản lý (quản lý dự án phần mềm).

4.1 Phát triển hệ thống thông tin – một bài toán phức tạp

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

- Những người phát triển rất khó hiểu cho đúng những gì người dùng cần.
- Yêu cầu của người dùng thường thay đổi theo thời gian phát triển.
- Yêu cầu thường được miêu tả bằng văn bản, dài dòng, nhiều khi mâu thuẫn nhau.
- Đội quân phát triển phần mềm vốn là những người "ngoài cuộc", rất khó nhận thức thấu đáo các mối quan hệ tiềm ẩn và phức tạp cần được thể hiện chính xác trong các ứng dụng lớn.
- Khả năng nắm bắt các dữ liệu phức tạp của con người (tại cùng một thời điểm) là có hạn.
- Khó định lượng chính xác hiệu xuất của thành phẩm và thỏa mãn chính xác sự mong chờ từ phía người dùng.
- Chọn phần cứng và phần mềm thích hợp cho giải pháp là một trong những thách thức lớn đối với Designer.

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

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

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

- Hiểu không đúng những gì người dùng cần.
- Không thể thích ứng cho phù hợp với những thay đổi về yêu cầu đối với hệ thống.
- Các module không khớp nhau.
- Phần mềm khó bảo trì và nâng cấp, mở rộng.
- Phát hiện trễ các lỗ hổng của dự án.
- Chất lượng phần mềm kém.
- Hiệu năng phần mềm thấp.

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

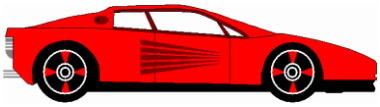
4.2 Chu trình phát triển phần mềm (Software Development Life Cycle)

Vì sao phát triển phần mềm cần phải có quy trình ? Phát triển phần mềm là một bài toán khó, nên có lẽ trước hết ta cần điểm qua một số công việc căn bản của quá trình này. Thường người ta hay tập hợp chúng theo tiến trình thời gian một cách tương đối, xoay quanh chu trình của một phần mềm, dẫn tới kết quả khái niệm Quy trình phát triển phần mềm (Software development life cycle - SDLC) như sau: Chu trình phát triển phần mềm là một chuỗi các hoạt động của nhà phân tích (Analyst), nhà thiết kế (Designer), người phát triển (Developer) và người dùng (User) để phát triển và thực hiện hệ thống thông tin. Những hoạt động này được thực hiện trong nhiều giai đoạn khác nhau, trong đó:

- ***Nhà phân tích (Analyst):*** là người nghiên cứu yêu cầu của khách hàng/người dùng để định nghĩa một phạm vi bài toán, nhận dạng nhu cầu của tổ chức, xác định xem nhân lực, phương pháp và công nghệ máy tính có thể làm sao để cải thiện một cách tốt nhất công tác của tổ chức này.
- ***Nhà thiết kế (Designer):*** thiết kế hệ thống theo hướng cấu trúc của database, screens, forms và reports – quyết định các yêu cầu về phần cứng và phần mềm cho hệ thống cần được phát triển.
- ***Chuyên gia lĩnh vực (Domain experts):*** là những người hiểu thực chất vấn đề cùng tất cả những sự phức tạp của hệ thống cần tin học hóa. Họ không nhất thiết phải là nhà lập trình, nhưng họ có thể giúp nhà lập trình hiểu yêu cầu đặt ra đối với hệ thống cần phát triển. Quá trình phát triển phần mềm có thể có rất nhiều thuận lợi nếu đội ngũ làm phần mềm có được sự trợ giúp của họ.
- ***Lập trình viên (Programmer):*** là những người dựa trên các phân tích và thiết kế để viết chương trình (coding) cho hệ thống bằng ngôn ngữ lập trình đã được thống nhất.
- ***Người dùng (User):*** là đối tượng phục vụ của hệ thống cần được phát triển.

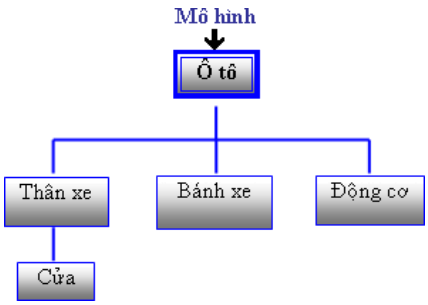
Để cho rõ hơn, xin lấy ví dụ về một vấn đề đơn giản sau: Người bình thường chúng ta khi nhìn một chiếc xe ô tô thường sẽ có một bức tranh từ bên ngoài như sau:

Vấn đề



Hình 1.4 Nhìn vấn đề ô tô của người bình thường

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



Hình 1.5 Nhìn vấn đề ô tô của chuyên gia phân tích

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

4.3 Các giai đoạn của quy trình phát triển phần mềm

Quy trình của một phần mềm có thể được chia thành các giai đoạn như sau:

- Nghiên cứu sơ bộ (Preliminary Investigation hay còn gọi là Feasibility Study)
- Phân tích yêu cầu (Analysis)
- Phân tích và thiết kế hệ thống (Analysis and Design of the System)
- Xây dựng phần mềm (Software Construction)
- Thử nghiệm hệ thống (System Testing)
- Thực hiện triển khai (System Implementation)
- Bảo trì và nâng cấp (Maintain and upgrade)

a) Nghiên cứu sơ bộ

Trước khi bắt tay vào một dự án, bạn phải có một ý tưởng cho nó. Ý tưởng này đi song song với việc nắm bắt các yêu cầu và xuất hiện trong giai đoạn khởi đầu. Nó hoàn tất một phát biểu: "Hệ thống mà chúng ta mong muốn sẽ làm được những việc như sau ....". Trong suốt giai đoạn này, chúng ta tạo nên một bức tranh về ý tưởng đó, rất nhiều giả thuyết sẽ được công nhận hay loại bỏ.

Các hoạt động trong thời gian này thường bao gồm: *thu thập các ý tưởng*, *nhận biết rủi ro*, *nhận biết các giao diện bên ngoài*, *nhận biết các các chức năng chính* mà hệ thống cần cung cấp, và có thể *tạo một vài nguyên mẫu* dùng để "minh chứng các khái niệm của hệ thống". Ý tưởng có thể đến từ nhiều nguồn khác nhau: khách hàng, chuyên gia lĩnh vực, các nhà phát triển khác, chuyên gia về kỹ nghệ, các bản nghiên cứu tính khả thi cũng như việc xem xét các hệ thống khác đang tồn tại. Một khía cạnh cần nhắc tới là code viết trong thời kỳ này thường sẽ bị "bỏ đi", bởi chúng được viết nhằm mục đích thẩm tra hay trợ giúp các giả thuyết khác nhau, chứ chưa phải thứ code được viết theo kết quả phân tích và thiết kế thấu đáo.

Trong giai đoạn nghiên cứu sơ bộ, nhóm phát triển hệ thống cần xem xét các yêu cầu của doanh nghiệp (cần dùng hệ thống), những nguồn tài nguyên có thể sử dụng, công nghệ cũng như cộng đồng người dùng cùng các ý tưởng của họ đối với hệ thống mới. Có thể thực hiện thảo luận,

nghiên cứu, xem xét khía cạnh thương mại, phân tích khả năng lời-lỗ, phân tích các trường hợp sử dụng và tạo các nguyên mẫu để xây dựng nên một khái niệm cho hệ thống đích cùng với các mục đích, quyền ưu tiên và phạm vi của nó.

Thường trong giai đoạn này người ta cũng tiến hành tạo một phiên bản thô của lịch trình và kế hoạch sử dụng tài nguyên. Một giai đoạn nghiên cứu sơ bộ thích đáng sẽ lập nên tập hợp các yêu cầu (dù ở mức độ khái quát cao) đối với một hệ thống khả thi và được mong muốn, kể cả về phương diện kỹ thuật lẫn xã hội. Một giai đoạn nghiên cứu sơ bộ không được thực hiện thoả đáng sẽ dẫn tới các hệ thống không được mong muốn, đắt tiền, bất khả thi và được định nghĩa lầm lạc – những hệ thống thường chẳng được hoàn tất hay sử dụng.

Kết quả của giai đoạn nghiên cứu sơ bộ là ***Báo cáo kết quả nghiên cứu tính khả thi***. Khi hệ thống tương lai được chấp nhận dựa trên bản báo cáo này cũng là lúc giai đoạn Phân tích bắt đầu.

b) Phân tích yêu cầu

Sau khi đã xem xét về tính khả thi của hệ thống cũng như tạo lập bức tranh sơ bộ của dự án, chúng ta bước sang giai đoạn thường được coi là quan trọng nhất trong các công việc lập trình: hiểu hệ thống cần xây dựng. Người thực hiện công việc này là *nhà phân tích*.

Quá trình phân tích nhìn chung là hệ quả của việc trả lời câu hỏi "Hệ thống cần phải làm gì ?". Quá trình phân tích bao gồm việc nghiên cứu chi tiết hệ thống doanh nghiệp hiện thời, tìm cho ra nguyên lý hoạt động của nó và những vị trí có thể nâng cao, cải thiện. Bên cạnh đó là việc nghiên cứu xem xét các chức năng mà hệ thống cần cung cấp và các mối quan hệ của chúng, bên trong cũng như với phía ngoài hệ thống. Trong toàn bộ giai đoạn này, nhà phân tích và người dùng cần cộng tác mật thiết với nhau để xác định các yêu cầu đối với hệ thống, tức là các tính năng mới cần phải được đưa vào hệ thống.

Những mục tiêu cụ thể của giai đoạn phân tích là:

- Xác định hệ thống cần phải làm gì.
- Nghiên cứu thấu đáo tất cả các chức năng cần cung cấp và những yếu tố liên quan.
- Xây dựng một mô hình nêu bật bản chất vấn đề từ một hướng nhìn có thực (trong đời sống).
- Trao định nghĩa vấn đề cho chuyên gia lĩnh vực để nhận sự đánh giá, góp ý.
- Kết quả của giai đoạn phân tích là bản ***Đặc tả yêu cầu*** (Requirements Specifications).

c) Phân tích và thiết kế hệ thống

Sau khi có được các yêu cầu từ phía khách hàng, xác định được các chức năng của hệ thống cần xây dựng, chúng ta bắt đầu phân tích hệ thống.

Ở giai đoạn này, chúng ta sẽ mô hình hóa hệ thống dưới dạng các biểu đồ. Mục đích của giai đoạn này là giúp người phát triển có cái nhìn sâu hơn về toàn cảnh hệ thống, về các hoạt động sẽ diễn ra trong hệ thống. Từ đó xây dựng dữ liệu của hệ thống.

Sau giai đoạn phân tích, khi các yêu cầu cụ thể đối với hệ thống đã được xác định, giai đoạn tiếp theo là thiết kế cho các yêu cầu mới. Công tác thiết kế xoay quanh câu hỏi chính: Hệ thống làm cách nào để thỏa mãn các yêu cầu đã được nêu trong *Đặc tả yêu cầu* ?

Một số công việc thường được thực hiện trong giai đoạn thiết kế:

- Nhận biết form nhập liệu tùy theo các thành phần dữ liệu cần nhập.
- Nhận biết reports và những output mà hệ thống mới phải sản sinh.
- Thiết kế forms (vẽ trên giấy hay máy tínhm sử dụng công cụ thiết kế).
- Nhận biết các thành phần dữ liệu và bảng để tạo database.
- Ước tính các thủ tục giải thích quá trình xử lý từ input đến output.

Kết quả của giai đoạn thiết kế *là* ***Đặc tả thiết kế*** (Designer Specifications). Đặc tả thiết kế chi tiết sẽ được chuyển sang cho các lập trình viên để thực hiện giai đoạn xây dựng phần mềm.

d) Xây dựng phần mềm

Đây là giai đoạn viết lệnh (code) thực sự, tạo hệ thống. Từng người viết code thực hiện những yêu cầu đã được nhà thiết kế định sẵn. Cũng chính người viết code chịu trách nhiệm viết tài liệu liên quan đến chương trình, giải thích thủ tục (procedure) mà anh ta tạo nên được viết như thế nào và lý do cho việc này.

Đảm bảo chương trình được viết lên thỏa mãn mọi yêu cầu trong bản Đặc tả thiết kế, người viết code cũng đồng thời phải tiến hành thử nghiệm phần chương trình của mình. Phần thử nghiệm trong giai đoạn này có thể được chia thành hai bước chính:

- Thử nghiệm đơn vị (Unit test)

Người viết code chạy thử các phần chương trình của mình với dữ liệu giả (test/dummy data). Việc này được thực hiện theo một kế hoạch thử, cũng do chính người viết code soạn ra. Mục đích chính trong giai đoạn thử này là xem chương trình có cho ra những kết quả mong đợi.

- Thử nghiệm đơn vị độc lập

Công việc này do một thành viên khác trong nhóm đảm trách. Cần chon người không có liên quan trực tiếp đến việc viết code của đơn vị chương trình cần thử nghiệm để đảm bảo tính "độc lập". Công việc thử đợt này cũng được thực hiện dựa trên kế hoạch thử do người viết code soạn lên.

e) Thử nghiệm hệ thống

Sau khi các thủ tục được thử nghiệm riêng, cần phải thử nghiệm lại toàn bộ hệ thống. Mọi thủ tục được tích hợp và chạy thử, kiểm tra xem mọi chi tiết trong Đặc tả yêu cầu và những mong đợi của người dùng có được thỏa mãn. Dữ liệu thử cần được chọn lọc đặc biệt, kết quả cần được chọn lọc đặc biệt, kết quả cần được phân tích để phát hiện mọi lệch lạc so với mong chờ.

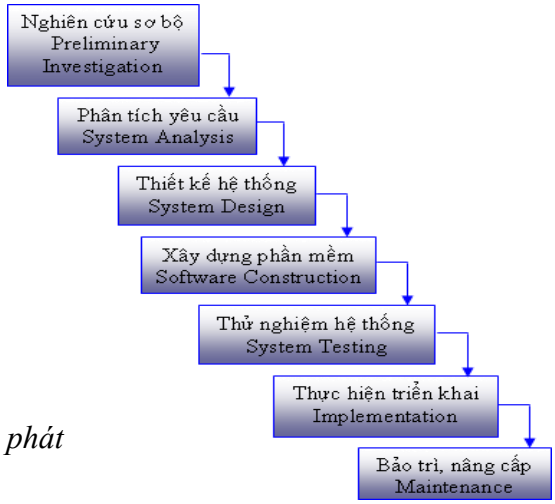
f) Thực hiện, triển khai

Trong giai đoạn này, hệ thống vừa phát triển sẽ được triển khai cho phía người dùng. Trước khi để người dùng thực sự bắt tay vào sử dụng hệ thống, nhóm các nhà phát triển cần tạo các file dữ

liệu cần thiết cũng như huấn luyện cho người dùng, để đảm bảo hệ thống được sử dụng hữu hiệu nhất.

g) Bảo trì, nâng cấp

Tùy theo các biến đổi trong môi trường sử dụng, hệ thống có thể trở nên lỗi thời hay cần phải được sửa đổi nâng cấp để sử dụng có hiệu quả. Hoạt động bảo trì hệ thống có thể rất khác biệt tùy theo mức độ sửa đổi và nâng cấp cần thiết.



Hình 1.6 Sơ đồ tổng quát các giai đoạn của Quy trình phát triển phần mềm

TỔNG KẾT CHƯƠNG 1

Chương này đã trình bày các nội dung mở đầu cho phân tích thiết kế hệ thống hướng đối tượng. Các nội dung cơ bản cần nhớ gồm:

- Có nhiều loại hệ thống thông tin khác nhau như : hệ thống thông tin quản lý, các Website, các hệ thống thương mại, các hệ thống điều khiển ... Mỗi loại hệ thống thông tin sẽ tương ứng với một phương pháp phát triển riêng.
- Việc phát triển các hệ thống thông tin nói chung được xem như một vòng đời với các pha : Xác định yêu cầu, đặc tả, thiết kế, cài đặt tích hợp, bảo trì và loại bỏ. Có hai mô hình vòng đời đơn giản và hay dùng nhất là mô hình thác nước và mô hình làm bản mẫu nhanh.
- Phương pháp phát triển phần mềm hướng đối tượng tỏ ra có nhiều ưu điểm hơn so với phương pháp hướng cấu trúc. Các pha đặc trưng trong vòng đời phát triển phần mềm hướng đối tượng là phân tích hướng đối tượng, thiết kế hướng đối tượng và lập trình hướng đối tượng.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG I

Câu 1. Tại sao khi xây dựng một HTTT cần phải có phân tích và thiết kế hệ thống?

Câu 2. Nêu các giai đoạn trong một chu trình phát triển một hệ thống thông tin? Giai đoạn nào là quan trọng? Có thể thiếu một trong các giai đoạn đó được không?

Câu 3. Kể tên một số ví dụ cho các loại hệ thống thông tin: hệ thống thông tin quản lý, hệ thống website thương mại điện tử, hệ thống điều khiển ...

Câu 5. So sánh hai phương pháp phân tích thiết kế hướng cấu trúc và hướng chức năng? Ưu và nhược điểm?

CHƯƠNG II. TỔNG QUAN VỀ PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG

1. Nhắc lại các khái niệm cơ bản về hướng đối tượng

1.1 Đối tượng (Object)

Đối tượng (Object) là chìa khoá để hiểu về lập trình hương đối tượng. Thử nhìn xung quanh bạn sẽ phát hiện nhiều ví dụ trong thực tế về đối tượng: Cái máy tính, cái đèn bàn, cái bàn của bạn, cái TV, cái xe đạp...

Các đối tượng trong thế giới thực có chung hai nét đặc trưng: tất cả đều có các *state* (trạng thái) và *behavior* (cách hành xử). Các đối tượng trong thực tế hết sức phức tạp: Cái đèn bàn của bạn có thể có 2 state (bật hoặc tắt) và 2 behavior (bật lên hoặc tắt đi), nhưng cái radio của bạn thì lại có thể có các state (bật, tắt, âm lượng hiện tại, kênh hiện tại...) và các behavior (bật lên, tắt đi, tăng/giảm âm thanh, dò kênh,...). Bạn cũng sẽ nhận ra rằng, vài đối tượng cũng sẽ chứa các đối tượng khác

Nhận ra các state và behavior của các đối tượng trong thực tế là cách tuyệt vời để bắt đầu suy nghĩ về các thuật ngữ trong lập trình hướng đối tượng.

Với mỗi đối tượng bạn nhìn thấy, hãy tự hỏi mình hai câu hỏi: Các state mà đối tượng có thể có là gì? và Các behavior mà đối tượng có thể làm là gì? Tất cả các quan sát trong thế giới thực đều được chuyển vào bên trong của thế giới lập trình hướng đối tượng.

Đối tượng trong phần mềm là khái niệm tương tự như đối tượng trong thế giới thực. Nó cũng bao gồm : các state và behavior có liên quan. Một object chứa các state của nó trong các field (gọi là biến - variable trong một số ngôn ngữ lập trình) và phơi bày các behavior thông qua các method (function ở trong một số ngôn ngữ lập trình khác). Các method tác động lên các state bên trong của đối tượng và hoạt động như một kĩ thuật gốc trong việc trao đổi từ object tới object. Việc che giấu các state bên trong và yêu cầu tất cả các tương tác phải được làm thông qua các method của object được gọi là đóng gói dữ liệu (data encapsulation) - một yếu tố cơ bản của lập trình hướng đối tượng

1.2 Lớp (Class) và thể hiện (instance)

Trong thế giới thực, bạn thường xuyên gặp các đối tượng khác nhau của cùng một kiểu. Ví dụ: có hàng nghìn chiếc xe đạp khác nhau đang tồn tại, mà tất cả chúng đều có cùng cấu tạo và tương tự nhau ở hình dáng. Mỗi cái xe đạp đều được xây dựng từ một tập hợp các thiết kế và đều chứa các thành phần giống nhau. Chúng ta gọi chung xe đạp là một ***lớp*** (class) các xe đạp, mỗi xe đạp là một ***thể hiện*** (instance) cụ thể của lớp xe đạp đó.

Trong lập trình hướng đối tượng cũng tương tự như vậy, lớp có thể được hiểu là một khuôn mẫu để tạo ra các đối tượng, trong một lớp người ta thường dùng các biến để mô tả các ***thuộc tính*** và các hàm để mô tả các ***phương thức*** (hàm) của đối tượng. Hay nói cách khác, lớp là một kiểu dữ liệu bao gồm thuộc tính và các phương thức được định nghĩa trước.

1.3 Sự trao đổi và thông điệp (message)

Các đối tượng trao đổi các thông điệp với nhau thông qua các phương thức (hàm).

1.4 Sự phân cấp (hierarchy)

1.4.1 Tính trừu tượng (abstraction), lớp trừu tượng (abstract class)

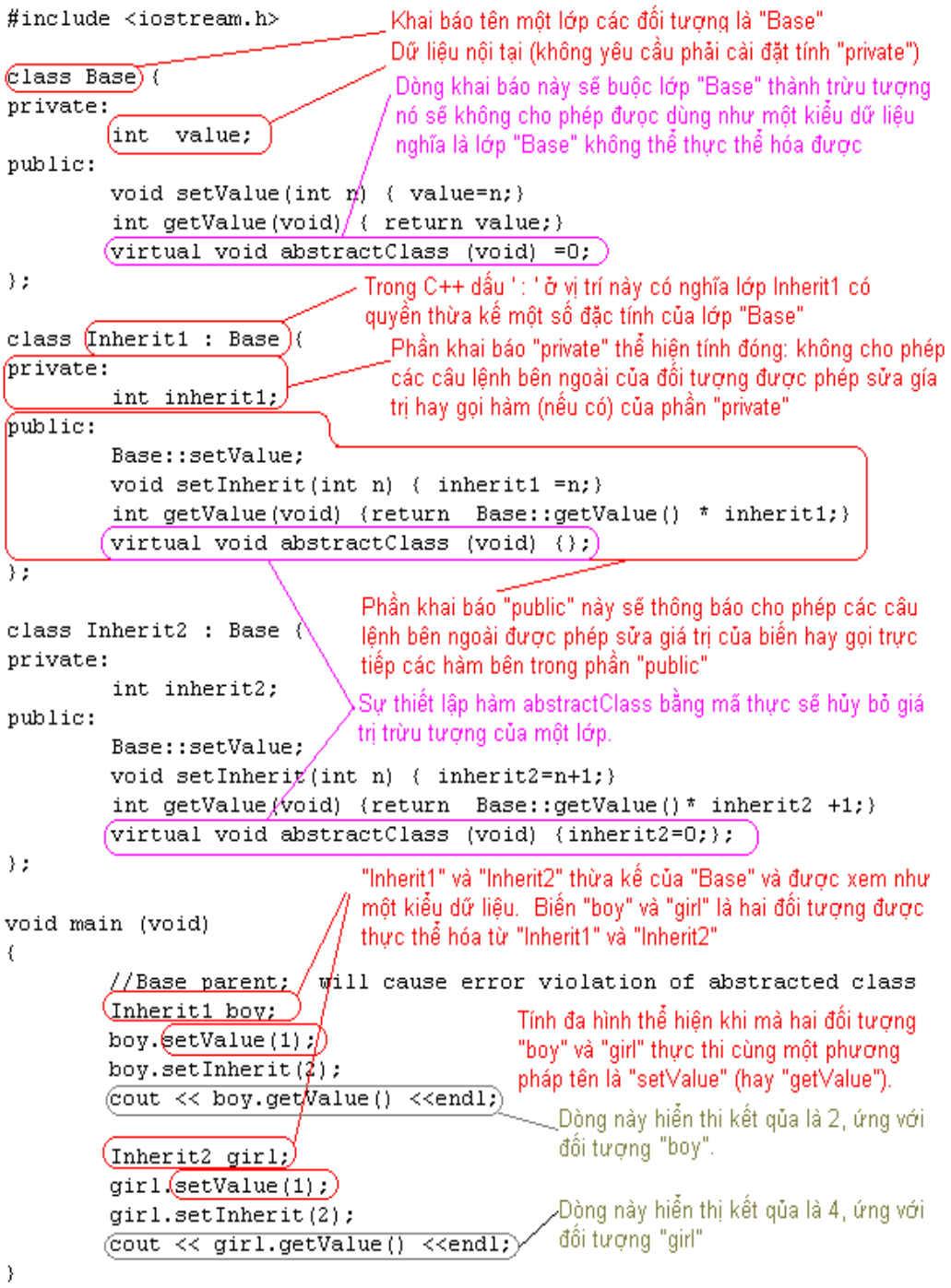
Tính trừu tượng (abstraction): Đây là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Mỗi đối tượng phục vụ như là một "động tử" có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên lạc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng tiến hành được các thao tác. Tính chất này thường được gọi là sự trừu tượng của dữ liệu.

Tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không có các biện pháp thi hành. Tính trừu tượng này thường được xác định trong khái niệm gọi là lớp trừu tượng hay hay lớp cơ sở trừu tượng.

Như vậy, ***Lớp trừu tượng:*** Là một lớp mà nó không thể thực thể hóa thành một đối tượng thực dụng được. Lớp này được thiết kế nhằm tạo ra một lớp có các đặc tính tổng quát nhưng bản thân lớp đó chưa có ý nghĩa (hay không đủ ý nghĩa) để có thể tiến hành viết mã cho việc thực thể hóa. (xem thí dụ)

Thí dụ: Lớp "hinh_thang" được định nghĩa không có dữ liệu nội tại và chỉ có các phương thức (hàm nội tại) "tinh_chu_vi", "tinh_dien_tich". Nhưng vì lớp hinh_thang này chưa xác định được đầy đủ các đặc tính của nó (cụ thể các biến nội tại là tọa độ các đỉnh nếu là đa giác, là đường bán kính và toạ độ tâm nếu là hình tròn, ...) nên nó chỉ có thể được viết thành một lớp trừu tượng. Sau đó, người lập trình có thể tạo ra các lớp con chẳng hạn như là lớp "tam_giac", lớp "hinh_tron", lớp "tu_giac",.... Và trong các lớp con này người viết mã sẽ cung cấp các dữ liệu nội tại (như là biến nội tại r làm bán kính và hằng số nội tại Pi cho lớp "hinh_tron" và sau đó viết mã cụ thể cho các phương thức "tinh_chu_vi" và "tinh_dien_tich").

Ví dụ về lớp cơ sở trừu tượng và lớp con kế thừa nó.



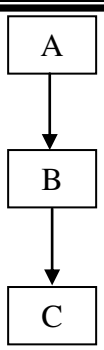
1.4.2 Sự kế thừa (inheritance)

Có hai khái niệm quan trọng làm nên thế mạnh của của lập trình hướng đối tượng đó là tính kế thừa (inheritance) và tính tương ứng bội – tính đa hình (polymorphism). Tính kế thừa cho phép các lớp được xây dựng trên các lớp sẵn có.

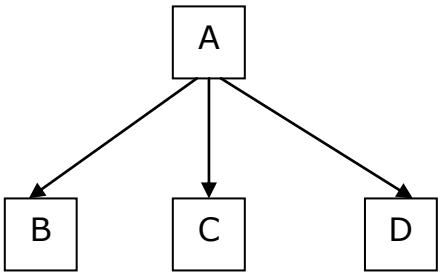
Lớp cơ sở và lớp dẫn xuất: Một lớp được xây dựng bằng cách kế thừa từ lớp khác được gọi là lớp dẫn xuất. Lớp dùng để xây dựng lớp dẫn xuất gọi là lớp cơ sở.

Lớp nào cùng có thể được xây dựng làm lớp cơ sở cho các lớp khác kế thừa nó. Hơn thế nữa, một lớp có thể làm lớp cơ sở cho nhiều lớp dẫn xuất khác nhau. Đến lượt mình lớp dẫn xuất lại dùng làm cơ sở cho các lớp dẫn xuất khác. Ngoài ra một lớp còn có thể dẫn xuất từ nhiều lớp cơ sở.

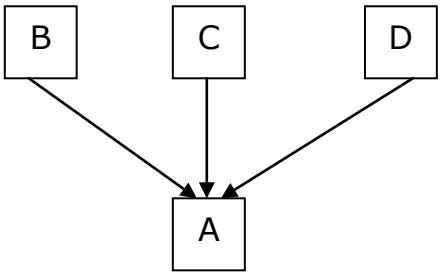
Sơ đồ 1: Lớp B dẫn xuất từ lớp A, lớp C dẫn xuất từ lớp B



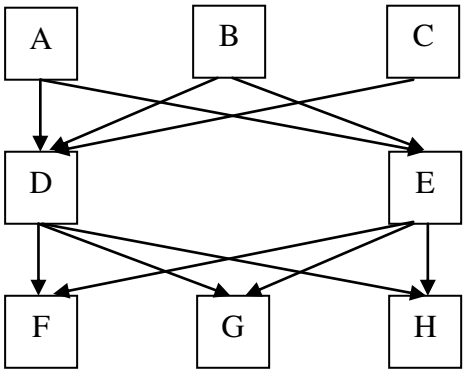
Sơ đồ 2: Lớp A là lớp cơ sở cho các lớp B, C, D



Sơ đồ 3: Lớp B, C, D là lớp cơ sở cho lớp A



Sơ đồ 4: Lược đồ dẫn xuất tổng quát



Tính kế thừa: một lớp dẫn xuất ngoài các thành phần của riêng nó, nó còn được kế thừa tất cả các thành phần của lớp cơ sở có liên quan. Ví dụ trong sơ đồ 1 thì lớp C được kế thừa các thành phần của lớp A và B. Trong sơ đồ 3 lớp A được kế thừa các thành phần của lớp B, C, D.

1.4.3 Sự đa kế thừa (multiple inheritance)

Sự đa kế thừa là hiện tượng một lớp được xây dựng bằng cách kế thừa từ nhiều lớp khác. Trong sơ đồ 3 lớp A kế thừa từ 2 lớp B, C, D là hiện tượng đa kế thừa.

1.5 Tính đa hình (polymorphism)

Đa hình thái trong lập trình hướng đối tượng đề cập đến khả năng quyết định trong lúc thi hành (runtime) mã nào sẽ được chạy, khi có nhiều phương thức trùng tên nhau nhưng ở các lớp có cấp bậc khác nhau.

Chú ý: khả năng đa hình thái trong lập trình hướng đối tượng còn được gọi với nhiều cái tên khác nhau như: tương ứng bội, kết ghép động,..

Đa hình thái cho phép các vấn đề khác nhau, các đối tượng khác nhau, các phương thức khác nhau, các cách giải quyết khác nhau theo cùng một lược đồ chung.

Các bước để tạo đa hình thái:

(1). Xây dựng lớp cơ sở (thường là lớp cơ sở trừu tượng, hoặc là một giao diện – trong java hay C#), lớp này sẽ được các lớp con mở rộng(đối với lớp thường, hoặc lớp trừu tượng), hoặc triển khai chi tiết (đối với giao diện).

(2). Xây dựng các lớp dẫn xuất từ lớp cơ sở vừa tạo. Trong lớp dẫn xuất này ta sẽ ghi đè các phương thức của lớp cơ sở(đối với lớp cơ sở thường), hoặc triển khai chi tiết nó (đối với lớp cơ sở trừu tượng hoặc giao diện).

(3). Thực hiện việc tạo khuôn xuống, thông qua lớp cơ sở, để thực hiện hành vi đa hình thái

Khái niệm về tạo khuôn lên, tạo khuôn xuống

- Hiện tượng một đối tượng của lớp cha tham trỏ đến một đối tượng của lớp con thì được gọi là tạo khuôn xuống, việc tạo khuôn xuống luôn được java chấp thuận, do vậy khi tạo khuôn xuống ta không cần phải ép kiểu tường minh.
- Hiện tượng một đối tượng của lớp con tham trỏ tới một đối tượng của lớp cha thì được gọi là tạo khuôn lên, việc tạo khuôn lên là an toàn, vì một đối tượng của lớp con cũng có đầy đủ các thành phần của lớp cha, tuy nhiên việc tạo khuôn lên sẽ bị báo lỗi nếu như ta không ép kiểu một cách tường minh.

1.6 Gói (package)

Là một cách tổ chức các thành phần, phần tử trong hệ thống thành các nhóm. Nhiều gói có thể được kết hợp với nhau để trở thành một hệ thống con (subsystem).

Như vậy:

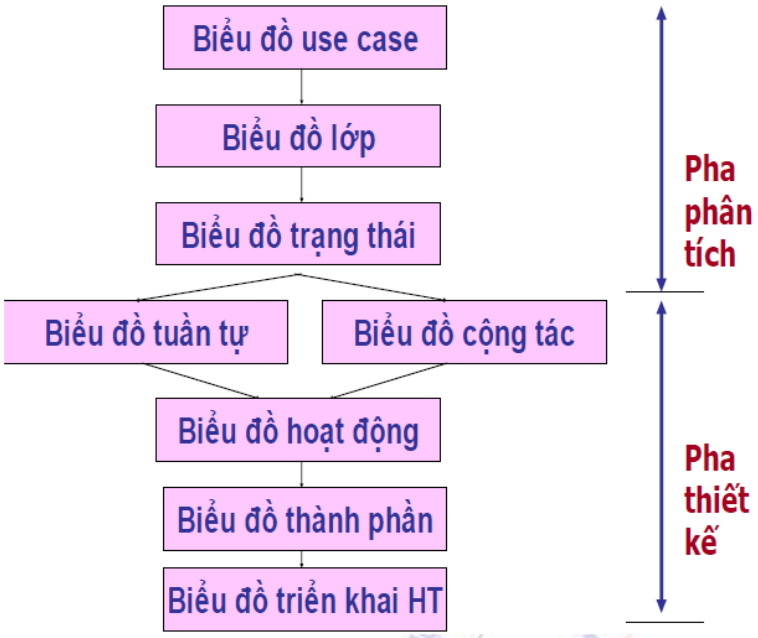
- Phân tích hướng đối tượng: xây dựng một mô hình chính xác để mô tả hệ thống cần xây dựng là gì. Thành phần của mô hình này là các đối tượng gắn với hệ thống thực.

- Thiết kế hướng đối tượng: Là giai đoạn tổ chức chương trình thành các tập hợp đối tượng cộng tác, mỗi đối tượng trong đó là thực thể của một lớp. Kết quả của pha thiết kế cho biết hệ thống sẽ được xây dựng như thế nào qua các bản thiết kế kiến trúc và thiết kế chi tiết.

- Lập trình và tích hợp: Thực hiện bản thiết kế hướng đối tượng bằng cách sử dụng các ngôn ngữ lập trình hướng đối tượng (C++, Java, C#…).

2. Các bước phân tích thiết kế hướng đối tượng

Các bước phân tích thiết kế hướng đối tượng được xây dựng dựa trên biểu đồ các ký hiệu UML. Đó là ngôn ngữ mô hình hoá thống nhất được xây dựng để mô hình hoá quá trình phát triển hệ thống phần mềm hướng đối tượng. Các vấn đề cơ bản về UML sẽ được giới thiệu chi tiết trong mục 3. Mục này chỉ nhằm giới thiệu một cách khái quát các bước trong phân tích và thiết kế hướng đối tượng.



Hình 7. Các bước phân tích thiết kế theo UML

Pha phân tích:

- Xây dựng Biểu đồ use case: Dựa trên tập yêu cầu ban đầu, người phân tích tiến hành xác định các tác nhân, use case và các quan hệ giữa các use case để mô tả lại các chức năng của hệ thống. Một thành phần quan trọng trong biểu đồ use case là các kịch bản mô tả hoạt động của hệ thống trong mỗi use case cụ thể.

- Xây dựng Biểu đồ lớp: Xác định tên các lớp, các thuộc tính của lớp, một số phương thức và mối quan hệ cơ bản trong sơ đồ lớp.

- Xây dựng biểu đồ trạng thái: Mô tả các trạng thái và chuyển tiếp trạng thái trong hoạt động của một đối tượng thuộc một lớp nào đó.

Trong Pha thiết kế:

- Xây dựng các biểu đồ tương tác (gồm biểu đồ cộng tác và biểu đồ tuần tự): mô tả chi tiết hoạt động của các use case dựa trên các scenario đã có và các lớp đã xác định trong pha phân tích.
- Xây dựng biểu đồ lớp chi tiết: tiếp tục hoàn thiện biểu đồ lớp bao gồm bổ sung các lớp còn thiếu, dựa trên biểu đồ trạng thái để bổ sung các thuộc tính, dựa trên biểu đồ tương tác để xác định các phương thức và mối quan hệ giữa các lớp.

- Xây dựng biểu đồ hoạt động: mô tả hoạt động của các phương thức phức tạp trong mỗi lớp hoặc các hoạt động hệ thống có sự liên quan của nhiều lớp. Biểu đồ hoạt động là cơ sở để cài đặt các phương thức trong các lớp.
- Xây dựng biểu đồ thành phần: xác định các gói, các thành phần và tổ chức phần mềm theo các thành phần đó.
- Xây dựng biểu đồ triển khai hệ thống: xác định các thành phần và các thiết bị cần thiết để triển khai hệ thống, các giao thức và dịch vụ hỗ trợ.

2. Giới thiệu ngôn ngữ mô hình hóa hợp nhất UML (Unified Modeling Language)

2.1 UML là gì ?

UML (Unified Modeling Language) là ngôn ngữ trực quan được dùng trong quy trình phát triển các hệ thống phần mềm. UML là *ngôn ngữ đặc tả hình thức* (formal specication language), được dùng để *đặc tả*, *trực quan hóa*, và *tư liệu hóa* phần mềm hướng đối tượng, nó không phải là một tiến trình và cũng không mô tả một tiến trình hay một phương pháp, do đó UML phải được sử dụng kết hợp với một tiến trình phương pháp luận. Ví dụ: Công tu Rational Software đã đề xuất một quy trình phát triển phần mềm – tiến trình RUP (Rational Unified Process) được xem như là một phương pháp luận phát triển hệ thống và có ngôn ngữ mô hình hóa là UML.

Chúng ta cần chú ý đến thuật ngữ *"ngôn ngữ*", ngôn ngữ ở đây không giống với ngôn ngữ tự nhiên của con người hay ngôn ngữ lập trình. Tuy nhiên nó cũng có một tập các quy luật xác định cách sử dụng. Các ngôn ngữ lập trình có một hệ thống các ký hiệu, các quy tắc cú pháp, các lệnh dùng để xây dựng các chương trình; nói cách khác, ngôn ngữ lập trình bao gồm một tập các *phần tử* và một tập các *quy luật* cho phép chúng ta tổ hợp các phần tử lại với nhau để tạo các chương trình hợp lệ. Các ngôn ngữ mô hình hóa hợp nhất cũng có một tập các phần tử và một tập các quy luật riêng. Các phần tử trong UML hầu hết là các đối tượng đồ họa như đường thẳng, hình chữ nhật, hình oval, …. Chúng thường được đặt nhãn để cung cấp thêm thông tin.

Mô hình hóa mang lại sự hiểu biết về một hệ thống. Một mô hình không thể giúp chúng ta hiểu rõ một hệ thống, thường là phải xây dựng một số mô hình xét từ những góc độ khác nhau. Các mô hình này có quan hệ với nhau.

UML sẽ cho ta biết cách tạo ra và đọc hiểu được một mô hình được cấu trúc tốt, nhưng nó không cho ta biết những mô hình nào nên tạo ra và khi nào tạo ra chúng. Đó là nhiệm vụ của quy trình phát triển phần mềm.

a) UML là ngôn ngữ dùng để trực quan hóa

Đối với nhiều lập trình viên, không có khoảng cách nào giữa ý tưởng để giải quyết một vấn đề và việc thể hiện điều đó thông qua các đoạn mã. Họ nghĩ ra và họ viết mã. Trên thực tế, điều này gặp một số vấn đề. Thứ nhất, việc trao đổi về các ý tưởng giữa những người lập trình sẽ gặp khó khăn, trừ khi tất cả đều nói cùng một ngôn ngữ. Thậm chí ngay cả khi không gặp trở ngại về ngôn ngữ thì đối với từng công ty, từng nhóm cũng có những "ngôn ngữ" riêng của họ. Điều này gây trở ngại cho một người mới vào để có thể hiểu được những việc đang được tiến hành. Hơn nữa, trong lĩnh vực phần mềm, nhiều khi khó có thể hiểu được nếu chỉ xem xét các đoạn mã lệnh. Ví dụ như

sự phân cấp của các lớp, ta có thể phải duyệt rất nhiều đoạn lệnh để hiểu được sự phân cấp của các lớp. Và nếu như người lập trình không mô tả các ý tưởng mà anh ta đã xây dựng thành mã lệnh thì nhiều khi cách tốt nhất là xây dựng lại trong trường hợp một người khác đảm nhận tiếp nhiệm vụ khi anh ta rời khỏi nhóm.

Xây dựng mô hình sử dụng ngôn ngữ UML đã giải quyết được các khó khăn trên. Khi trở thành một chuẩn trong việc lập mô hình, mỗi kí hiệu mang một ý nghĩa rõ ràng và duy nhất, một nhà phát triển có thể đọc được mô hình xây dựng bằng UML do một người khác viết.

Những cấu trúc mà việc nắm bắt thông qua đọc mã lệnh là khó khăn nay đã được thể hiện trực quan. Một mô hình rõ ràng, sáng sủa làm tăng khả năng giao tiếp, trao đổi giữa các nhà phát triển.

b) UML là ngôn ngữ dùng để chi tiết hóa

Có nghĩa là xây dựng các mô hình một cách tỉ mỉ, rõ ràng, đầy đủ ở các mức độ chi tiết khác nhau. Đặc biệt là UML thực hiện việc chi tiết hoá tất cả các quyết định quan trọng trong phân tích, thiết kế và thực thi một hệ thống phần mềm.

c) UML là ngôn ngữ dùng để sinh ra mã ở dạng nguyên mẫu

Các mô hình xây dựng bởi UML có thể ánh xạ tới một ngôn ngữ lập trình cụ thể như : Java, C++... thậm chí cả các bảng trong một cơ sở dữ liệu quan hệ hay CSDL hướng đối tượng.

Việc các yêu cầu có khả năng thường xuyên thay đổi trong quá trình phát triển hệ thống dẫn đến việc các cấu trúc và hành vi của hệ thống được xây dựng có thể khác mô hình mà ta đã xây dựng. Điều này có thể làm cho một mô hình tốt trở nên vô nghĩa vì nó không còn phản ánh đúng hệ thống nữa. Cho nên phải có một cơ chế để đồng bộ hóa giữa mô hình và mã lệnh.

UML cho phép cập nhật một mô hình từ các mã thực thi (ánh xạ ngược). Điều này tạo ra sự nhất quán giữa mô hình của hệ thống và các đoạn mã thực thi mà ta xây dựng cho hệ thống đó.

d) UML là ngôn ngữ dùng để lập và cung cấp tài liệu

Một tổ chức phần mềm ngoài việc tạo ra các đoạn mã lệnh (thực thi) thì còn tạo ra các tài liệu sau:

- Ghi chép về các yêu cầu của hệ thống
- Kiến trúc của hệ thống
- Thiết kế
- Mã nguồn
- Kế hoạch dự án
- Tests
- Các nguyên mẫu

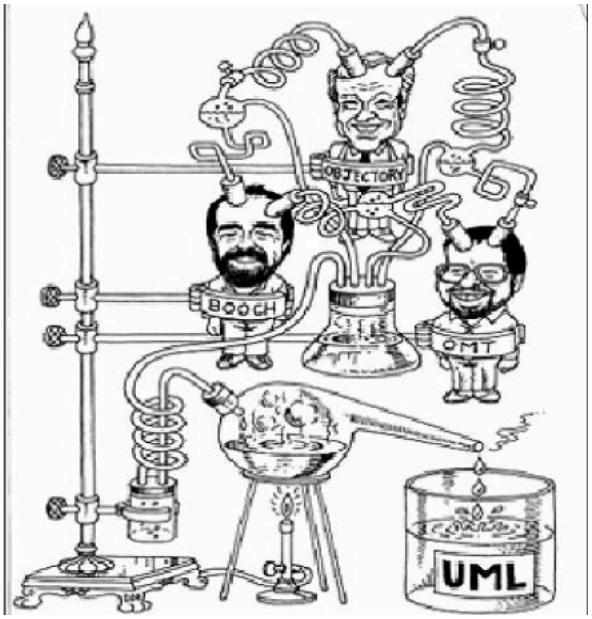
2.2 Lịch sử phát triển và tương lai của UML

Đầu những năm 1980, ngành công nghệ phần mềm chỉ có duy nhất một ngôn ngữ hướng đối tượng là Simula. Sang nửa sau của thập kỷ 1980, các ngôn ngữ hướng đối tượng như Smalltalk và C++ xuất hiện. Cùng với chúng, nảy sinh nhu cầu mô hình hoá các hệ thống phần mềm theo hướng đối tượng. Và một vài trong số những ngôn ngữ mô hình hoá xuất hiện những năm đầu thập kỷ 90 được nhiều người dùng là:

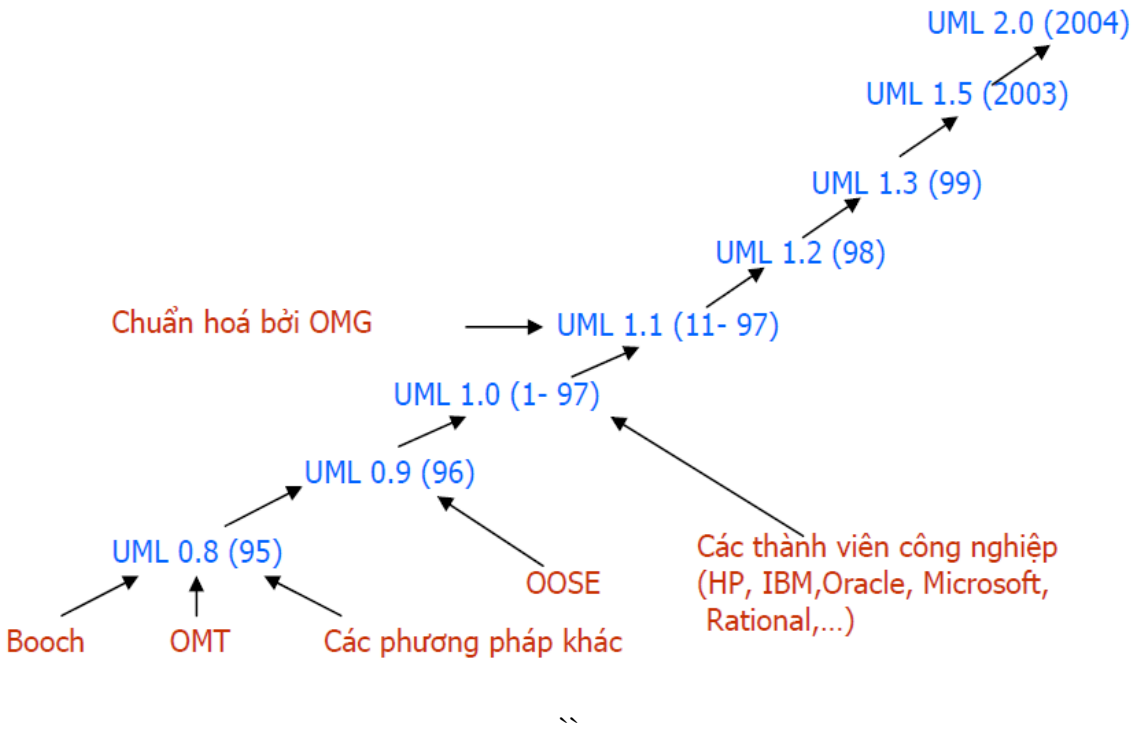
- Grady Booch's Booch Modeling Methodology
- James Rambaugh's Object Modeling Technique – OMT
- Ivar Jacobson's OOSE Methodology
- Hewlett- Packard's Fusion
- Coad and Yordon's OOA and OOD

Mỗi phương pháp luận và ngôn ngữ trên đều có hệ thống ký hiệu riêng, phương pháp xử lý riêng và công cụ hỗ trợ riêng, khiến nảy ra cuộc tranh luận phương pháp nào là tốt nhất. Đây là cuộc tranh luận khó có câu trả lời, bởi tất cả các phương pháp trên đều có những điểm mạnh và điểm yếu riêng. Vì thế, các nhà phát triển phần mềm nhiều kinh nghiệm thường sử dụng phối hợp các điểm mạnh của mỗi phương pháp cho ứng dụng của mình. Trong thực tế, sự khác biệt giữa các phương pháp đó hầu như không đáng kể và theo cùng tiến trình thời gian, tất cả những phương pháp trên đã tiệm cận lại và bổ sung lẫn cho nhau. Chính hiện thực này đã được những người tiên phong trong lĩnh vực mô hình hoá hướng đối tượng nhận ra và họ quyết định ngồi lại cùng nhau để tích hợp những điểm mạnh của mỗi phương pháp và đưa ra một mô hình thống nhất cho lĩnh vực công nghệ phần mềm.

Trong bối cảnh trên, người ta nhận thấy cần thiết phải cung cấp một phương pháp tiệm cận được chuẩn hoá và thống nhất cho việc mô hình hoá hướng đối tượng. Yêu cầu cụ thể là đưa ra một tập hợp chuẩn hoá các ký hiệu (Notation) và các biểu đồ (Diagram) để nắm bắt các quyết định về mặt thiết kế một cách rõ ràng, rành mạch. Đã có ba công trình tiên phong nhắm tới mục tiêu đó, chúng được thực hiện dưới sự lãnh đạo của James Rumbaugh, Grady Booch và Ivar Jacobson. Chính những cố gắng này dẫn đến kết quả là xây dựng được một ***Ngôn ngữ mô hình hoá thống nhất*** (Unifield Modeling Language – UML).



Hình 8 Sự ra đời của UML



Hình 9. Các phiên bản UML

Những tài liệu đầu tiên về UML đã được trình làng vào năm 1996 và phiên bản 1.0 được công bố vào tháng giêng năm 1997.

Và hiện nay, phiên bản mới nhất của UML là 2.1.2 phát hành chính thức năm 2007. Tài liệu đặc tả được công bố vào tháng 11 năm 2007.

2.3 Tại sao dùng UML ?

2.3.1 Tại sao dùng biểu đồ trong phân tích thiết kế?

Khi thiết kế các sản phẩm chúng ta đều dùng hình ảnh hoặc biểu đồ để trợ giúp. Các nhà thiết kế thời trang, các kỹ sư, các kiến trúc sư và các nhà phân tích thiết kế - tất cả đều sử dụng biểu

đồ để hình dung công việc của họ. Các phân tích và thiết kế viên dùng các biểu đồ để hình dung hệ thống phần mềm mà họ đang thiết kế, mặc dù sự thật sản phẩm của quy trình thiết kế (tức là các chương trình máy tính) vốn không trực quan. Vậy việc sử dụng biểu đồ mang lại cho quy trình thiết kế những thuận lợi gì?

Có hai lợi ích chính khi sử dụng biểu đồ để tạo một thiết kế:

- Trừu tượng hóa các thuộc tính của bản thiết kế.
- Chi ra các mối quan hệ giữa các thành phần của bản thiết kế.

Khi kiến trúc sư thiết kế một tòa nhà, anh ta sẽ đưa ra một số bản vẽ với nhiều mục đích khác nhau. Bao gồm một biểu đồ cho một *cái nhìn về toàn bộ tòa nhà với rất ít chi tiết* nhằm chỉ ra các đặc điểm đặc biệt của bản thiết kế, chẳng hạn như vị trí của ống nước; các biểu đồ vẽ ra *chi tiết bản thiết kế*, chẳng hạn như hình dáng của các vật bằng gỗ hoặc màu và vân bề mặt bên ngoài. Không có biểu đồ nào có thể biểu hiện hết mọi phương diện của một đối tượng phức tạp, chẳng hạn như một tòa nhà, và không một ai có thể xử lý hết mọi thông tin của một bản thiết kế tòa nhà trong một lần. Điều này cũng giống với các hệ thống phần mềm: chúng rất phức tạp, và người thiết kế sẽ biểu diễn các khía cạnh khác nhau của một bản thiết kế bằng các biểu đồ khác nhau. *Mỗi biểu đồ chỉ biểu diễn một hoặc nhiều khía cạnh đặc trưng từ bản thiết kế tổng quát*. Mặc dù thế, mỗi biểu đồ không thể biểu diễn từng chi tiết của các khía cạnh của bản thiết kế. Một biểu đồ ống nước trong một tòa nhà chỉ có thể sử dụng các đường thẳng để biểu diễn cho các ống nước thay vì tìm cách vẽ độ rộng của ống nước theo tỷ lệ. Tương tự như vậy, một biểu đồ biểu diễn giao tiếp giữa các thành phần khác nhau trong một hệ thống phần mềm có thể dùng các đường thẳng để biểu diễn cho giao tiếp mà không cần tìm cách biểu diễn cách thức mà giao tiếp xảy ra.

Sử dụng biểu đồ để đơn giản hóa hệ thống và để biểu diễn các đặc điểm chính nào đó được gọi là *sự trừu tượng*. Trừu tượng hóa là một cơ chế được dùng để biểu diễn một sự vật phức tạp trở nên đơn giản hơn bằng cách dùng một số loại mô hình. Hơn nữa, nếu sự trừu tượng được biểu diễn ở mức vật lý, chẳng hạn như một biểu đồ trên giấy hoặc một đối tượng vật lý, thì chúng ta sẽ dùng thuật ngữ *mô hình*.

Trong phân tích thiết kế hệ thống, các mô hình được tạo ra để trừu tượng hóa các đặc điểm quan trọng của hệ thống thế giới thực. Một lớp UML mô tả một khách hàng chỉ gồm những đặc điểm của khách hàng liên quan đến hệ thống nghiệp vụ. Một lớp UML mô hình hành vi của chiếc máy bay trong hệ thống điều khiển không lưu sẽ mô hình chỉ các đặc điểm mà hệ thống điều khiển thời gian thực quan tâm. Trong cả hai trường hợp, người phân tích hoặc thiết kế quyết định đặc điểm nào là cần, đặc điểm nào là không cần.

Ví dụ: Trong hầu hết các hệ thống nghiệp vụ, các đặc điểm của khách hàng quan tâm bao gồm *tên*, *địa chỉ*, *số điện thoại*, *số fax* và *địa chỉ email*. Màu tóc, cân nặng và chiều cao không phải là các đặc điểm cần thiết.

2.3.2 Tại sao dùng đặc tả UML?

Trong dự án phát triển hệ thống hướng đối tượng, UML là ngôn ngữ mô hình được ưu tiên cho việc phân tích và thiết kế một sản phẩm. Lý do mạnh mẽ nhất để sử dụng UML bởi vì nó đã trở thành chuẩn thực tế đối với mô hình hướng đối tượng. Nếu cần thu hút một nhóm các nhà phát triển

hoặc cần chuyển thông tin trong mô hình cho những người khác, thì UML là sự lựa chọn hiển nhiên vì nó dễ dàng giao tiếp giữa các bên tham gia.

Lý do thứ hai UML là ngôn ngữ mô hình hợp nhất. Nó là sản phẩm kết hợp từ ý tưởng của ba nhà dẫn đầu trong việc lập mô hình hướng đối tượng và kết hợp chúng thành một mô hình duy nhất. Từ các phiên bản đầu tiên, một số tổ chức liên quan đến phát triển UML cũng cố gắng hợp tác các đặc điểm tốt nhất của ngôn ngữ mô hình khác, vì thế UML có thể được xem là một ngôn ngữ tốt nhất trong lĩnh vực này. Tuy nhiên, có một điều nguy hiểm ỏ đay là việc cố gắng hợp nhất nhiều quan điểm trên mô hình hướng đối tượng làm cho UML phình ra với nhiều ký hiệu thừa. RTF của OMG đã cố gắng tránh điều này bằng cách giữ cho phần cốt lõi của UML đơn giản, đồng thời dùng *profile* và cơ các chế mở rộng khác để mở rộng nó.

Lý do thứ ba mà chúng ta nên sử dụng UML đó là các *profile* đặc biệt đã tồn tại để giúp người sử dụng áp dụng cho một vấn đề đặc trưng, hiện một số loại *profile* khác cũng đang được xây dựng. Nếu một vấn đề nào đó chưa có *profile* tương ứng, thì ta có thể mở rộng ký hiệu để áp dụng.

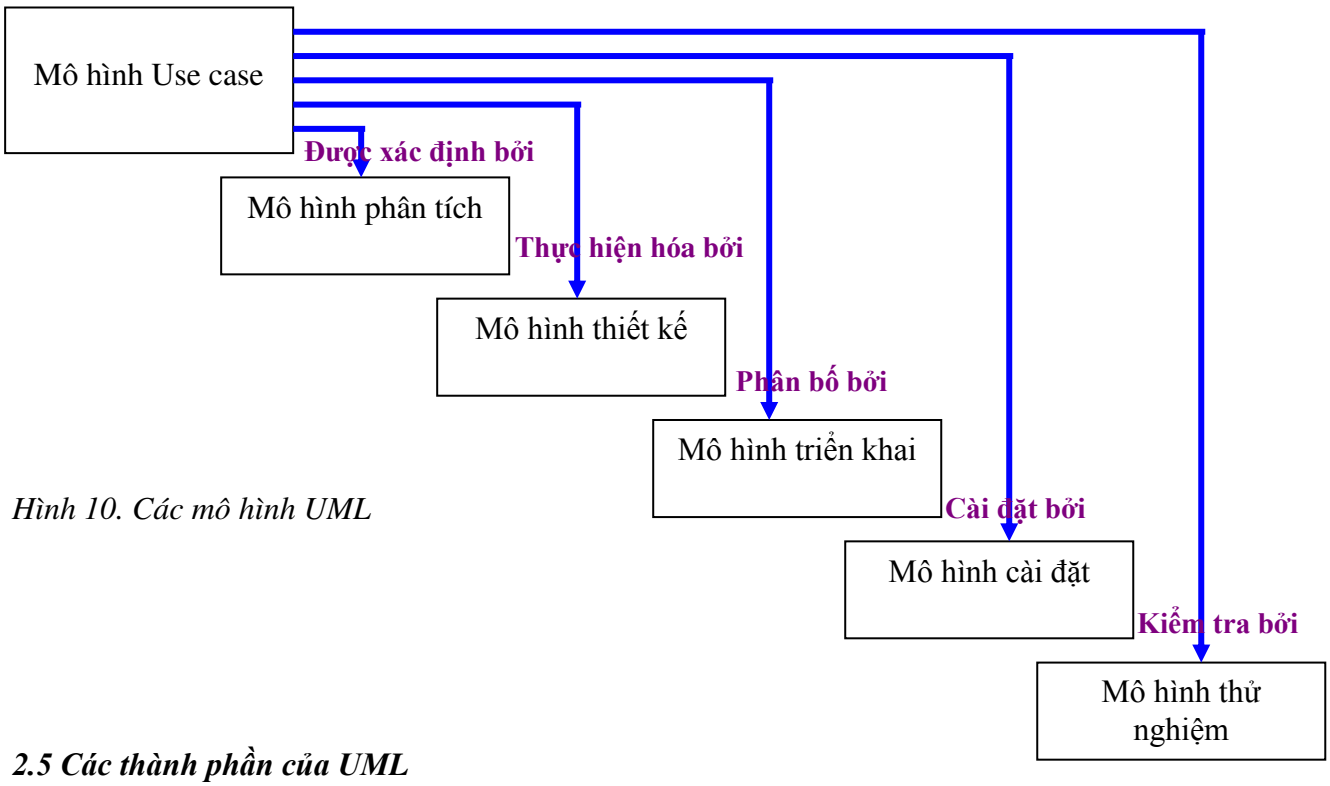
2.4 Một số đặc điểm chính của phương pháp phân tích thiết kế bằng UML

Trước đây, quy trình tuần tự được xem là phương pháp hợp lý nhất để phát triển hệ thống. Tuy nhiên, trải qua vài thập niên, đã cho thấy các dự án sử dụng quy trình tuần tự thường ít thành công bởi những lý do sau đây:

- Sự giả định ban đầu khi phát triển hệ thống có sai sót.
- Thất bại trong việc kết hợp các nhân tố con người.
- Các hệ thống ngày càng lớn và thường hay thay đổi.
- Chúng ta vẫn còn đang trong giai đoạn thăm dò của công nghệ phần mềm và chưa có nhiều kinh nghiệm.

Như đã trình bày ở trên, UML không phải là một tiến trình và cũng không mô tả một tiến trình hay một phương pháp, do đó UML phải được sử dụng kết hợp với một tiến trình phương pháp luận hay còn gọi lần tiến trình lặp. Tính chất lặp gồm các đặc trưng cơ bản sau:

- *Tính lặp (iterative):* Dự án sẽ được chia thành các dự án nhỏ trong mỗi bước lặp. Mỗi dự án nhỏ cũng sẽ được phân tích, thiết kế, cài đặt và kiểm tra.
- *Gia tăng (incremental):* Hệ thống tiến hóa thông qua một tập các sự gia tăng, mỗi bước được chia ở trên là một incremental.
- *Tập chung kiến trúc (architecture centric)*: Kiến trúc của hệ thống phải được phát triển nhằm đáp ứng các yêu cầu của use case chính, trong giới hạn của chuẩn phần cứng mà hệ thống sẽ chạy.



Hình 10. Các mô hình UML

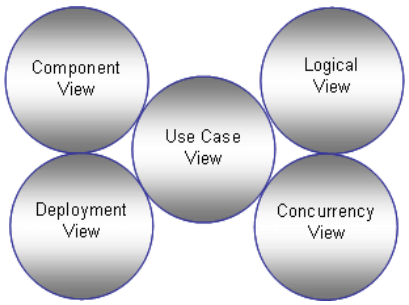
2.5 Các thành phần của UML

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

- *Hướng nhìn (View):* Hướng nhìn chỉ ra những khía cạnh khác nhau của hệ thống cần phải được mô hình hóa. Một hướng nhìn không phải là một bản vẽ, mà là một sự trừu tượng hóa bao gồm một loạt các biểu đồ khác nhau. Chỉ qua việc định nghĩa một loạt các hướng nhìn khác nhau, mỗi hướng nhìn chỉ ra một khía cạnh riêng biệt của hệ thống, người ta mới có thể tạo dựng nên môt bức tranh toàn diện về hệ thống. Cũng chính các hướng nhìn này nối kết ngôn ngữ mô hình hóa với quy trình được chọn cho giai đoạn phát triển.
- *Biểu đồ (diagram):* Biểu đồ là các hình vẽ miêu tả nội dung trong một hướng nhìn. UML có tất cả 9 loại biểu đồ khác nhau được sử dụng trong những sự kết hợp khác nhau để cung cấp tất cả các hướng nhìn của một hệ thống.
- *Phần tử mô hình hóa (model element):* Các khái niệm được sử dụng trong các biểu đồ được gọi là các phần tử mô hình, thể hiện các khái niệm hướng đối tượng quen thuộc. Ví dụ như lớp, đối tượng, thông điệp cũng như các quan hệ giữa các khái niệm này, bao gồm cả liên kết, phụ thuộc, khái quát hóa. Một phần tử mô hình thường được sử dụng trong nhiều biểu đồ khác nhau, nhưng nó luôn luôn có chỉ một ý nghĩa và một kí hiệu.
- *Cơ chế chung:* Cơ chế chung cung cấp thêm những lời nhận xét bổ sung, các thông tin cũng như các quy tắc ngữ pháp chung về một phần tử mô hình; chúng còn cung cấp thêm các cơ chế để có thể mở rộng ngôn ngữ UML cho phù hợp với một phương pháp xác định (một quy trình, một tổ chức hoặc một người dùng).

a) Hướng nhìn (View)

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



Hình 11. Các View trong UML

Mỗi một hướng nhìn được miêu tả trong một loạt các biểu đồ, chứa đựng các thông tin nêu bật khía cạnh đặc biệt đó của hệ thống. Trong thực tế khi phân tích và thiết kế rất dễ xảy ra sự trùng lặp thông tin, cho nên một biểu đồ trên thực tế có thể là thành phần của nhiều hướng nhìn khác nhau. Khi nhìn hệ thống từ nhiều hướng nhìn khác nhau, tại một thời điểm có thể người ta chỉ tập trung vào một khía cạnh của hệ thống. Một biểu đồ trong một hướng nhìn cụ thể nào đó cần phải đủ độ đơn giản để tạo điều kiện giao tiếp dễ dàng, để dính liền với các biểu đồ khác cũng như các hướng nhìn khác, làm sao cho bức tranh toàn cảnh của hệ thống được miêu tả bằng sự kết hợp tất cả các thông tin từ tất cả các hướng nhìn. Một biểu đồ chứa các kí hiệu hình học mô tả các phần tử mô hình của hệ thống. UML có tất cả các hướng nhìn sau:

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

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

- Hướng nhìn Use case mang tính trung tâm, bởi nó đặt ra nội dung thúc đẩy sự phát triển các hướng nhìn khác. Mục tiêu chung của hệ thống là cung cấp các chức năng miêu tả trong hướng nhìn này – cùng với một vài các thuộc tính mang tính phi chức năng khác – vì thế hướng nhìn này có ảnh hưởng đến tất cả các hướng nhìn khác. Hướng nhìn này

cũng được sử dụng để thẩm tra (verify) hệ thống qua việc thử nghiệm xem hướng nhìn Use case có đúng với mong đợi của khách hàng (Hỏi: "Đây có phải là thứ bạn muốn") cũng như có đúng với hệ thống vừa được hoàn thành (Hỏi: "Hệ thống có hoạt động như đã đặc tả?").

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

- Hướng nhìn logic miêu tả phương thức mà các chức năng của hệ thống sẽ được cung cấp. Chủ yếu nó được sử dụng cho các nhà thiết kế và nhà phát triển. Ngược lại với hướng nhìn Use case, hướng nhìn logic nhìn vào phía bên trong của hệ thống. Nó miêu tả kể cả cấu trúc tĩnh (lớp, đối tượng, và quan hệ) cũng như sự tương tác động sẽ xảy ra khi các đối tượng gửi thông điệp cho nhau để cung cấp chức năng đã định sẵn.
- Cấu trúc tĩnh được miêu tả bằng các ***biểu đồ lớp*** (class diagram) và ***biểu đồ đối tượng*** (object diagram). Quá trình mô hình hóa động được miêu tả trong các biểu đồ trạng thái (state diagram), biểu đồ trình tự (sequence diagram), biểu đồ tương tác (collaboration diagram) và biểu đồ hoạt động (activity diagram).

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

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

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

- Hướng nhìn song song nhắm tới sự chia hệ thống thành các qui trình (process) và các bộ xử lý (processor). Khía cạnh này, vốn là một thuộc tính phi chức năng của hệ thống, cho phép chúng ta sử dụng một cách hữu hiệu các nguồn tài nguyên, thực thi song song, cũng như xử lý các sự kiện không đồng bộ từ môi trường. Bên cạnh việc chia hệ thống thành các tiểu trình có thể được thực thi song song, hướng nhìn này cũng phải quan tâm đến vấn đề giao tiếp và đồng bộ hóa các tiểu trình đó.
- Hướng nhìn song song giành cho nhà phát triển và người tích hợp hệ thống, nó bao gồm các ***biểu đồ động*** (trạng thái, trình tự, tương tác và hoạt động) cùng các biểu đồ thực thi (biểu đồ thành phần và biểu đồ triển khai).

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

- Hướng nhìn triển khai chỉ cho chúng ta sơ đồ triển khai về mặt vật lý của hệ thống, ví dụ như các máy tính cũng như các máy móc và sự liên kết giữa chúng với nhau. Hướng nhìn triển khai giành cho các nhà phát triển, người tích hợp cũng như người thử nghiệm hệ thống và được thể hiện bằng các biểu ***đồ triển khai***. Hướng nhìn này cũng bao gồm sự ánh xạ các thành phần của hệ thống vào cấu trúc vật lý; ví dụ như chương trình nào hay đối tượng nào sẽ được thực thi trên máy tính nào.

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

Ngoài các hướng nhìn kể trên, ngành công nghiệp phần mềm còn sử dụng cả các hướng nhìn khác, ví dụ hướng nhìn tĩnh-động, hướng nhìn logic-vật lý, quy trình nghiệp vụ (workflow) và các hướng nhìn khác. UML không yêu cầu chúng ta phải sử dụng các hướng nhìn này, nhưng đây cũng chính là những hướng nhìn mà các nhà thiết kế của UML đã nghĩ tới, nên có khả năng nhiều công cụ sẽ dựa trên các hướng nhìn đó.

b) Biểu đồ (Diagram)

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

1. *Biểu đồ Use Case* mô tả sự tương tác giữa các tác nhân ngoài và hệ thống thông qua các Use Case. Các Use Case là những nhiệm vụ chính, các dịch vụ, những trường hợp sử dụng cụ thể mà hệ thống cung cấp cho người sử dụng và ngược lại.
2. *Biểu đồ lớp* mô tả cấu trúc tĩnh, mô tả mô hình khái niệm bao gồm các lớp đối tượng và các mối quan hệ của chúng trong hệ thống hướng đối tượng.
3. *Biểu đồ trình tự* thể hiện sự tương tác của các đối tượng với nhau, chủ yếu là trình tự gửi và nhận thông điệp để thực thi các yêu cầu, các công việc *theo thời gian*.
4. *Biểu đồ cộng tác* tương tự như biểu đồ trình tự nhưng nhấn mạnh vào sự tương tác của các đối tượng trên cơ sở cộng tác với nhau bằng cách trao đổi các thông điệp để thực hiện các yêu cầu *theo ngữ cảnh công việc*.
5. *Biểu đồ trạng thái* thể hiện chu kỳ hoạt động của các đối tượng, của các hệ thống con và của cả hệ thống. Nó là một loại ôtômát hữu hạn trạng thái, mô tả các trạng thái, các hành động mà đối tượng có thể có và các sự kiện gắn với các trạng thái theo thời gian.

6. *Biểu đồ hành động* chỉ ra dòng hoạt động của hệ thống, bao gồm các trạng thái hoạt động, trong đó từ một trạng thái hoạt động sẽ chuyển sang trạng thái khác sau khi một hoạt động tương ứng được thực hiện. Nó chỉ ra trình tự các bước, tiến trình thực hiện cũng như các điểm quyết định và sự rẽ nhánh theo luồng sự kiện.
7. *Biểu đồ thành phần* chỉ ra cấu trúc vật lý của các thành phần trong hệ thống, bao gồm: các thành phần mã nguồn, mã nhị phân, thư viện và các thành phần thực thi.
8. *Biểu đồ triển khai* chỉ ra cách bố trí vật lý các thành phần theo kiến trúc được thiết kế của hệ thống.

C) Cơ chế chung

Trang trí:

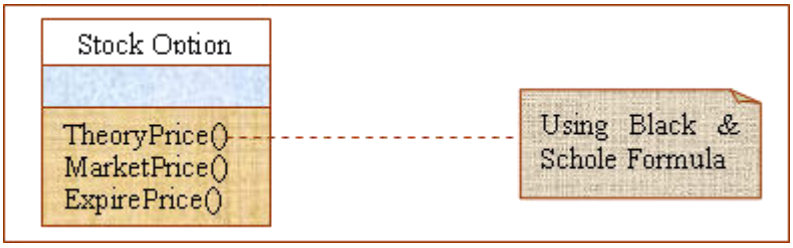
Các sự trang trí trực quan có thể được sử dụng kèm thêm vào các phần tử mô hình trong biểu đồ. Động tác trang trí bổ sung thêm ngữ nghĩa cho phần tử. Một ví dụ là kỹ thuật được sử dụng để phân biệt một loại thực thể (lớp) và một thực thể. Khi thể hiện một loại, tên phần tử sẽ được in đậm. Khi cũng chính phần tử đó thể hiện chỉ một thực thể của loại này, tên phần tử sẽ được gạch dưới và có thể được coi là cả tên của thực thể lẫn tên của loại đó. Một hình chữ nhật thể hiện lớp với tên được in đậm sẽ thể hiện một lớp và tên được gạch dưới sẽ thể hiện một đối tượng, đây là một ví dụ tiêu biểu của adornment. Cũng nguyên tắc đó được áp dụng cho các nút mạng, khi ký hiệu nút được in đậm là thể hiện một loại nút, ví dụ như máy in (**Printer**), khi ký hiệu được gạch dưới là thể hiện một thực thể của lớp nút mạng này ví dụ John's HP 5MP-printer. Các kiểu trang trí khác là các lời đặc tả về số lượng trong quan hệ (multiplicity), nơi số lượng là một số hay một khoảng số chỉ ra bao nhiêu thực thể của các loại thực thể được nối với nhau sẽ có thể tham gia trong một quan hệ.

Hình 12. Phân biệt lớp và đối tượng bằng trang trí

Ghi chú:

Cho dù một ngôn ngữ mô hình hóa có được mở rộng đến bao nhiêu chăng nữa, nó cũng không thể định nghĩa tất cả mọi việc. Nhằm tạo điều kiện bổ sung thêm cho một mô hình những thông tin không thể được thể hiện bằng phần tử mô hình, UML cung cấp khả năng kèm theo lời ghi chú. Một lời ghi chú có thể được để bất kỳ nơi nào trong bất kỳ biểu đồ nào, và nó có thể chứa bất kỳ loại thông tin nào. Dạng thông tin của bản thân nó là chuỗi ký tự (string), không được UML diễn giải. Lời ghi chú thường đi kèm theo một số các phần tử mô hình trong biểu đồ, được nối bằng một đường chấm chấm, chỉ ra phần tử mô hình nào được chi tiết hóa hoặc được giải thích (hình 2.4).

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

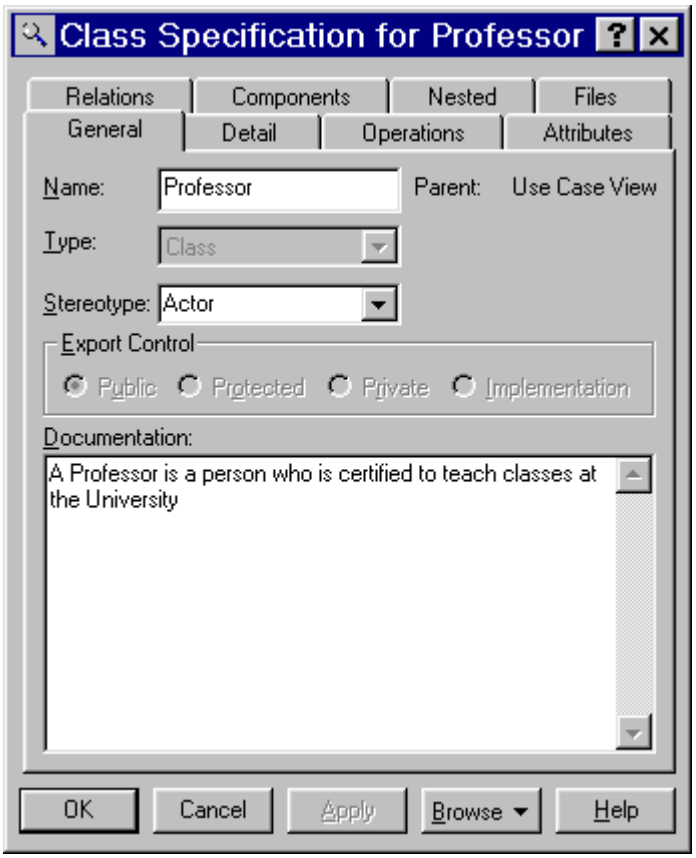


Hình 13. Một ví dụ về ghi chú

Đặc tả (Specification)

Các phần tử mô hình có thuộc tính (Property) chứa các giá trị dữ liệu về phần tử này. Một thuộc tính được định nghĩa với một tên và một giá trị đính kèm (tagged value), thường chúng ở trong một dạng thông tin được xác định trước, ví dụ như số nguyên hay chuỗi kí tự. Có một loạt thuộc tính đã được định nghĩa trước, ví dụ như tài liệu (docement), trách nhiệm (Responsibility), sự trường tồn (Persistence) và tính song song (Conccurency).

Thuộc tính được sử dụng để thêm các đặc tả bổ sung về một phần tử, những thông tin bình thường ra không được thể hiện trong biểu đồ. Ví dụ tiêu biểu là một lớp sẽ được miêu tả bằng một tài liệu văn bản nhất định, cung cấp nhiều thông tin hơn về trách nhiệm cũng như khả năng của lớp này. Loại đặc tả này bình thường ra không được chỉ ra trong các biểu đồ, nhưng thường thì trong đa phần các công cụ mô hình hóa chúng sẽ có thể được truy cập qua hành động nhấp nút vào một phần tử nào đó, hiệu quả là một cửa sổ chứa đặc tả với tất cả các thuộc tính sẽ được chỉ ra (Hình 2.5).



Hình 14. Một cửa sổ đặc tả thể hiện các đặc tính của class

2.6 Mở rộng UML

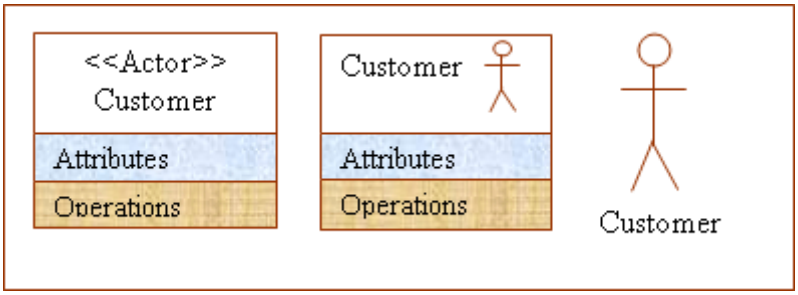
UML có thể được mở rộng hoặc có thể được sửa đổi để phù hợp với một phương pháp đặc biệt, một tổ chức cụ thể hay một người dùng cụ thể. Chúng ta sẽ bàn luận sơ qua đến ba cơ chế mở rộng UML: khuôn mẫu (stereotype), giá trị đính kèm (tagged value) và hạn chế (constraint).

2.6.1 Khuôn mẫu (Stereotype)

Cơ chế mở rộng khuôn mẫu định nghĩa một loại phần tử mô hình mới dựa trên một phần tử mô hình đã tồn tại. Khuôn mẫu có thể được coi là "tương tự" như một phần tử đã có sẵn, cộng thêm phần quy định ngữ nghĩa (semantic) riêng biệt không có trong phần tử gốc kia. Khuôn mẫu của một phần tử có thể được sử dụng trong cùng tình huống như phần tử căn bản. Khuôn mẫu dựa trên tất cả các loại phần tử mô hình sẵn có - lớp, nút mạng, thành phần, cũng như các mối quan hệ như liên kết, khái quát hóa, sự phụ thuộc. Ngôn ngữ UML có chứa một số lượng lớn các khuôn mẫu được định nghĩa sẵn và chúng được sử dụng để sửa đổi các phần tử mô hình sẵn có, thay cho việc phải định nghĩa hoàn toàn mới. Cơ chế này giúp gìn giữ tính đơn giản của nền tảng ngôn ngữ UML.

Khuôn mẫu được miêu tả qua việc đưa tên của chúng vào trong một cặp ký tự ngoặc nhọn <<>>, theo ví dụ hình 2.6. Ký tự ngoặc nhọn này được gọi là guillements. Khuôn mẫu cũng có thể có ký hiệu hình học riêng. Một phần tử của một loại khuôn mẫu cụ thể có thể được kết hợp với tên khuôn mẫu đi kèm ký hiệu hình học mô tả phần tử căn bản, hay là sự kết hợp của hai yếu tố này. Bất kỳ khi nào một phần tử mô hình được nối kết với một ký hiệu khuôn mẫu, ta sẽ đọc "đây là loại phần tử thuộc loại khuôn mẫu …". Ví dụ, một lớp với <<Window>> sẽ được gọi là "Một lớp trong dạng khuôn mẫu cửa sổ", ý nghĩa của nó là một dạng lớp cửa sổ. Những thuộc tính cụ thể mà một lớp cửa sổ cần phải có sẽ được định nghĩa khi khuôn mẫu này được định nghĩa.

Như đã nói, khuôn mẫu là một cơ chế mở rộng xuất sắc, là một cơ chế ngăn cho ngôn ngữ UML không trở nên quá phức tạp, mặc dù vẫn cho phép thực hiện sự mở rộng và sửa đổi cần thiết. Đa phần các phần tử mô hình mới mà bạn cần đến đều có một khuôn mẫu nền tảng trong ngôn ngữ UML. Một khuôn mẫu sau đó có thể được sử dụng để cộng thêm các ngữ nghĩa cần thiết, nhằm mục đích định nghĩa nên các phần tử mô hình còn thiếu.

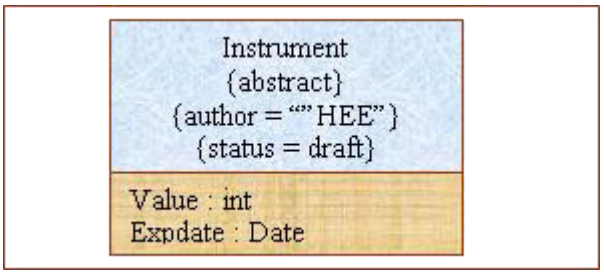


Hình 15. Customer là một khuôn mẫu của <<actor>>

2.6.2 Giá trị đính kèm (Tagged value)

Như đã nói, các phần tử mô hình có thể có các thuộc tính chứa một cặp tên-giá trị về bản thân chúng (hình 2.7). Các thuộc tính này cũng còn được gọi là các giá trị đính kèm. UML chứa một loạt các thuộc tính được định nghĩa trước, nhưng kể cả người sử dụng cũng có thể định nghĩa ra các thuộc tính mới để chứa các thông tin bổ sung về các phần tử mô hình. Mọi hình dạng thông tin đều có thể được đính kèm vào phần tử: các thông tin chuyên biệt về phương pháp, các thông tin của nhà quản trị về tiến trình mô hình hóa, các thông tin được sử dụng bởi các công cụ khác, ví dụ như

các công cụ tạo code hay bất kỳ một loại thông tin nào mà người sử dụng muốn đính kèm vào phần tử mô hình.



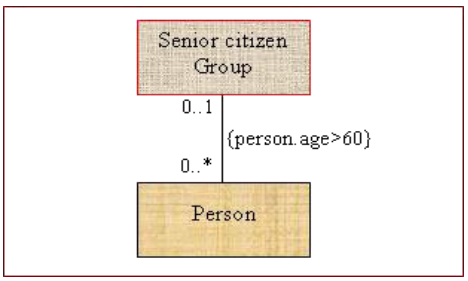
Hình 2.7 Một ví dụ về tagged value

2.6.3 Hạn chế (Constraint)

Một sự hạn chế là một sự giới hạn về sự sử dụng hoặc ý nghĩa của một phần tử. Sự hạn chế hoặc sẽ được khai báo trong công cụ và được sử dụng nhiều lần trong rất nhiều biểu đồ khác nhau, hay được định nghĩa và sử dụng trong chỉ một biểu đồ, theo như nhu cầu

Hình 2.8 chỉ ra mối quan hệ nối kết giữa nhóm các công dân lớn tuổi và lớp con người, chỉ ra rằng nhóm công dân có thể có nhiều liên quan. Mặc dù vậy, để miêu tả rằng chỉ những người nào lớn hơn 60 tuổi mới có thể tham gia vào nhóm này, người ta định nghĩa một sự hạn chế, thu hẹp tiêu chuẩn tham gia đối với chỉ những người nào mà thuộc tính tuổi tác có giá trị lớn hơn 60. Định nghĩa này sẽ hạn chế số lượng những người được sử dụng trong mối quan hệ. Nếu không có nó, người ta rất dễ hiểu lầm khi diễn tả biểu đồ. Trong trường hợp tồi tệ, nó có thể dẫn đến sự thực thi sai trái của hệ thống.

Trong trường hợp này, hạn chế được định nghĩa và ứng dụng trực tiếp trong chính biểu đồ mà nó được cần tới. Nhưng nhìn chung thì hạn chế cũng có thể được định nghĩa với tên cùng lời đặc tả riêng, ví dụ: "công dân già" và "người lớn hơn 60 tuổi" và hạn chế này sẽ được sử dụng trong nhiều biểu đồ khác nhau. UML có chứa một loạt các hạn chế được định nghĩa sẵn.



Hình 2.8 Một ràng buộc hạn chế đối với Person góp phần vào quan hệ kết hợp

3. Công cụ hỗ trợ Visual Paradigm

3.1 Cài đặt phần mềm

Môi trường cài đặt:

Cặt đặt Visual paradigm for UML:

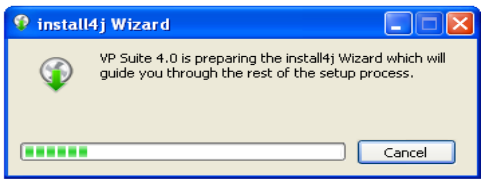
a) Download bộ cài Visual Paradigm Suite

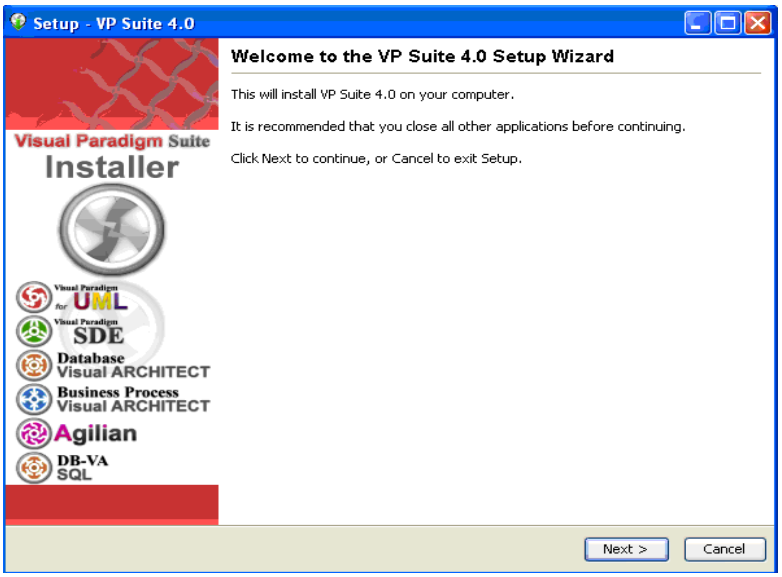
Để thực hiện cài đặt trước hết bạn cần download bộ cài tại địa chỉ http://www.visual-paradigm.com/download/

b) Installing VP-UML

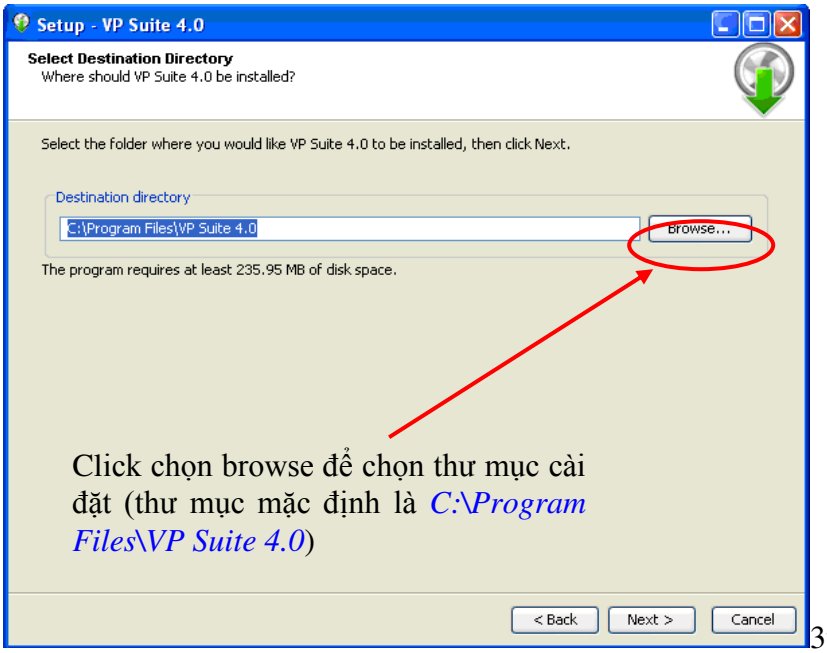
Visual Paradigm Suite installer cung cấp từng bước với giao diện đồ họa để cài đặt VP-UML vào hệ thống.

- Double click vào file setup để thực hiện cài đặt, màn hình xuất hiện



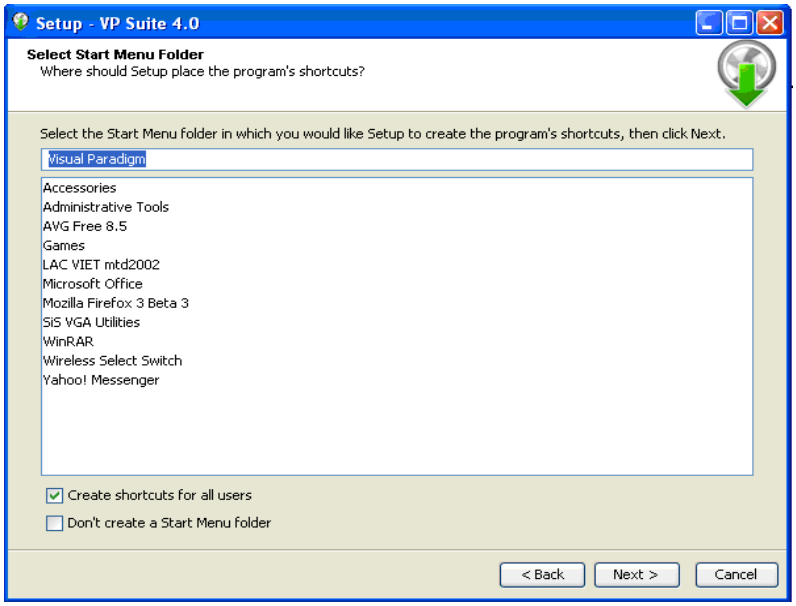


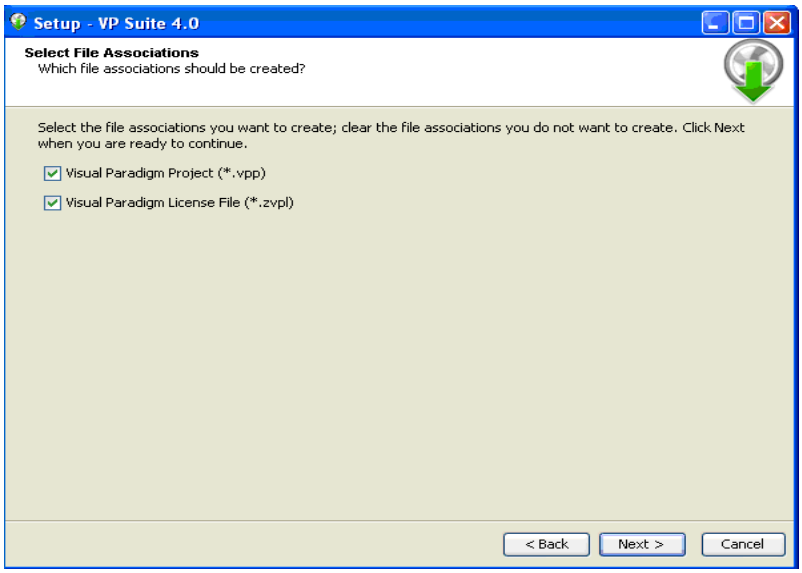
- Click next:



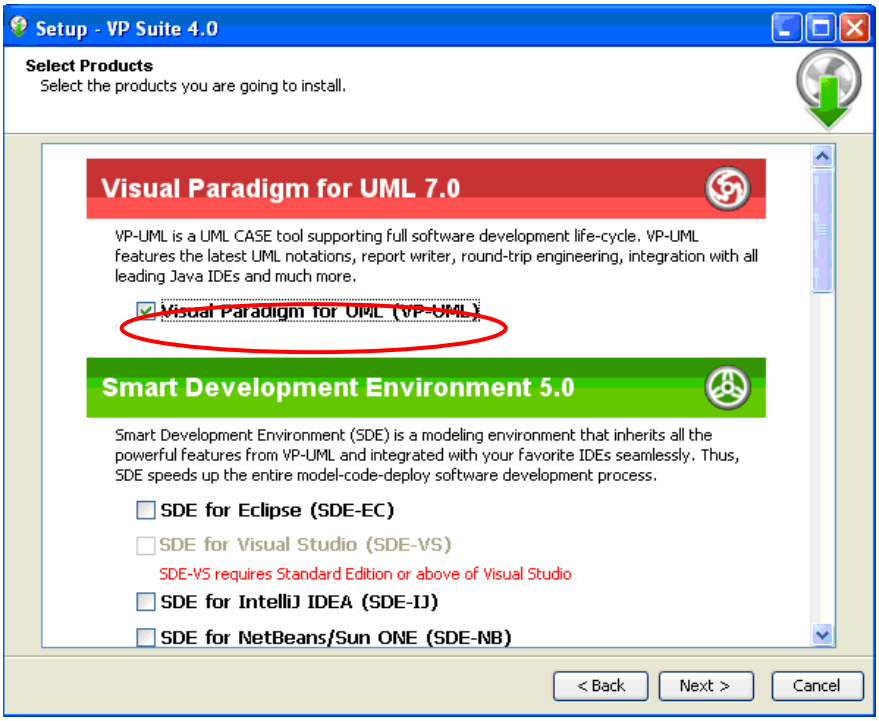
3wr

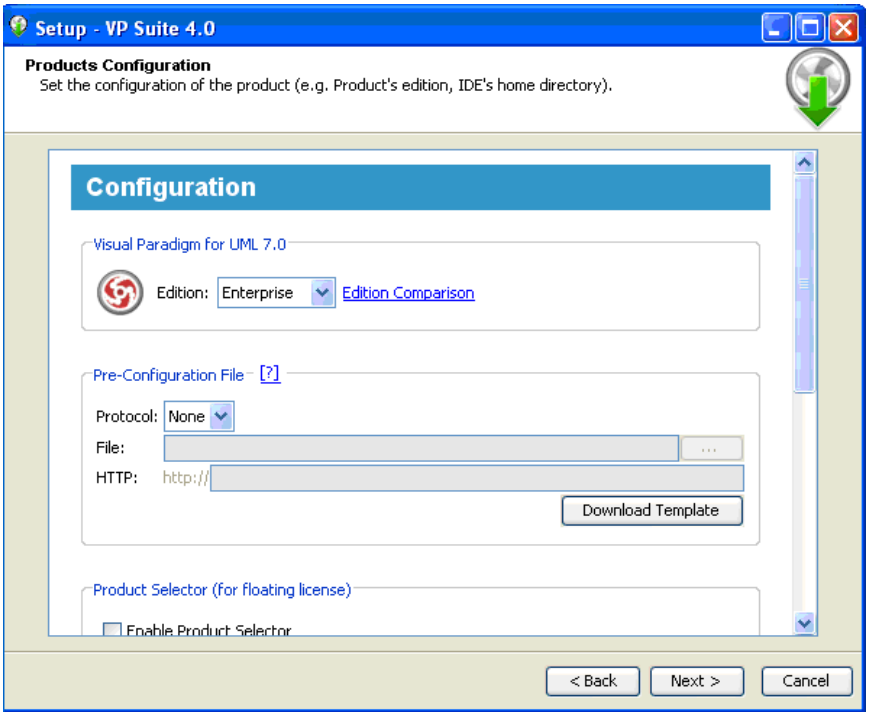
Chọn thư mục cài đặt cho VP-UML sau đó click next:



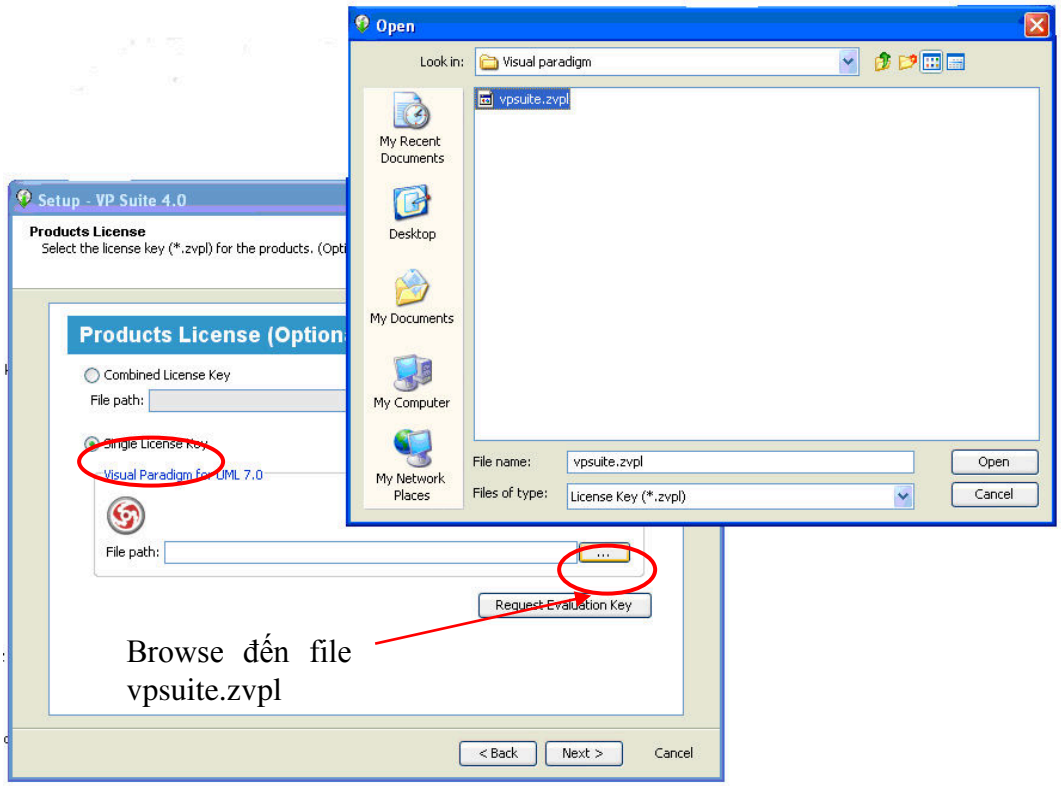


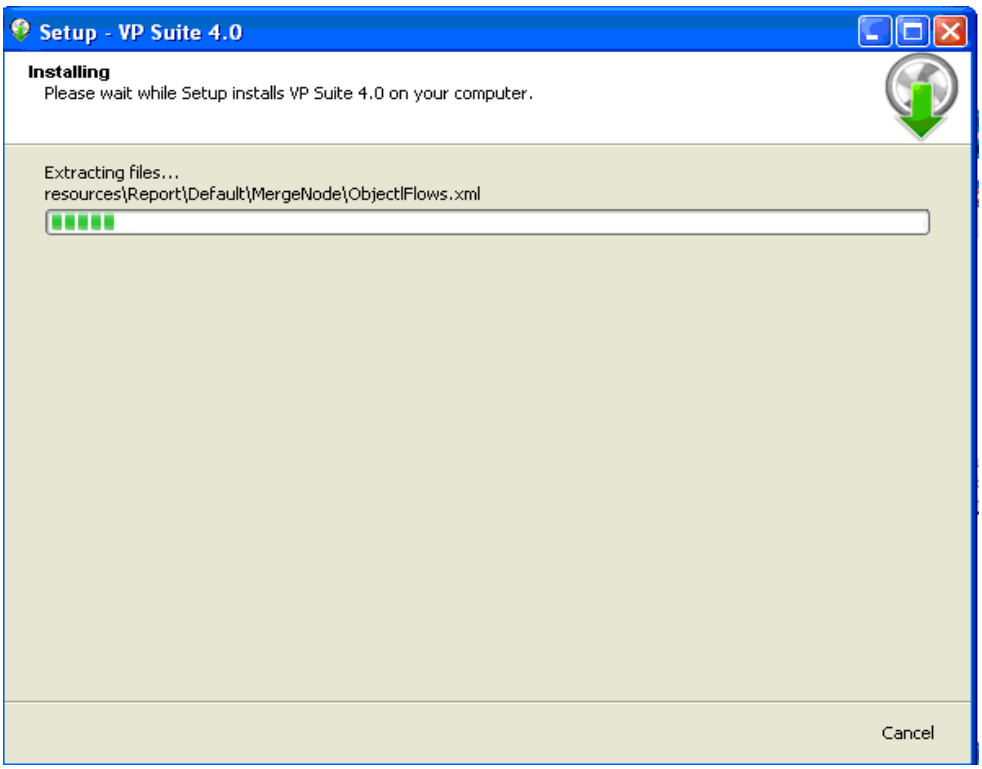
- Check chọn Visual Paradigm for UML (VP-UML) rồi click next:

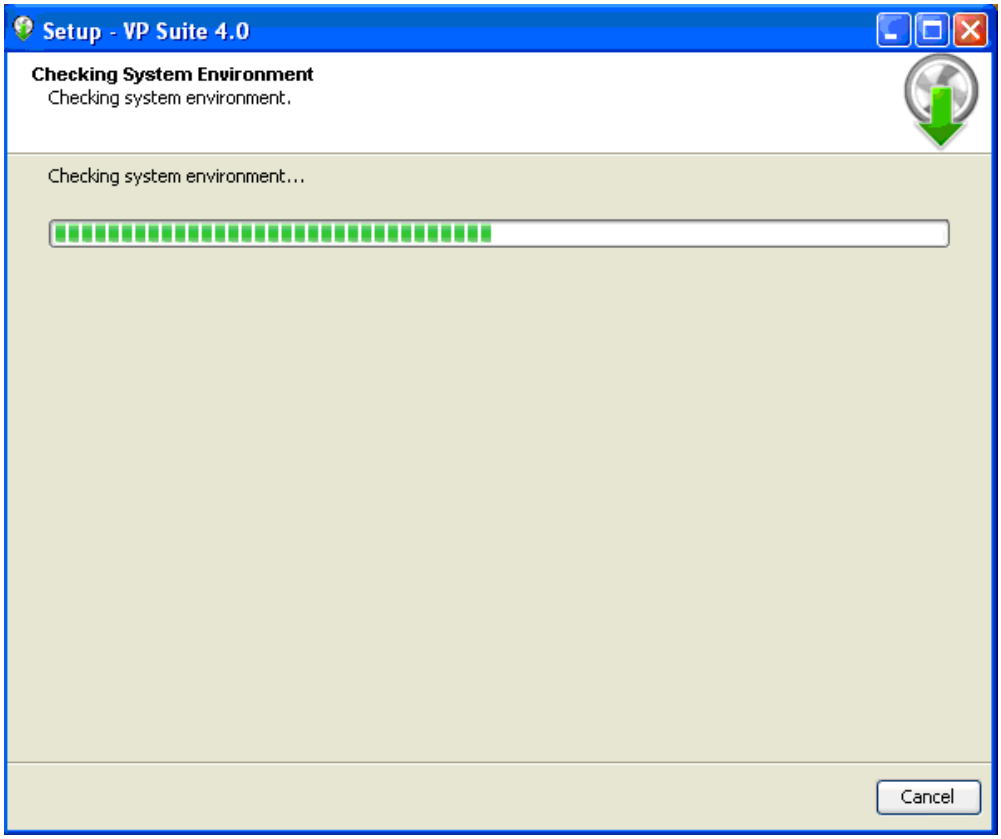




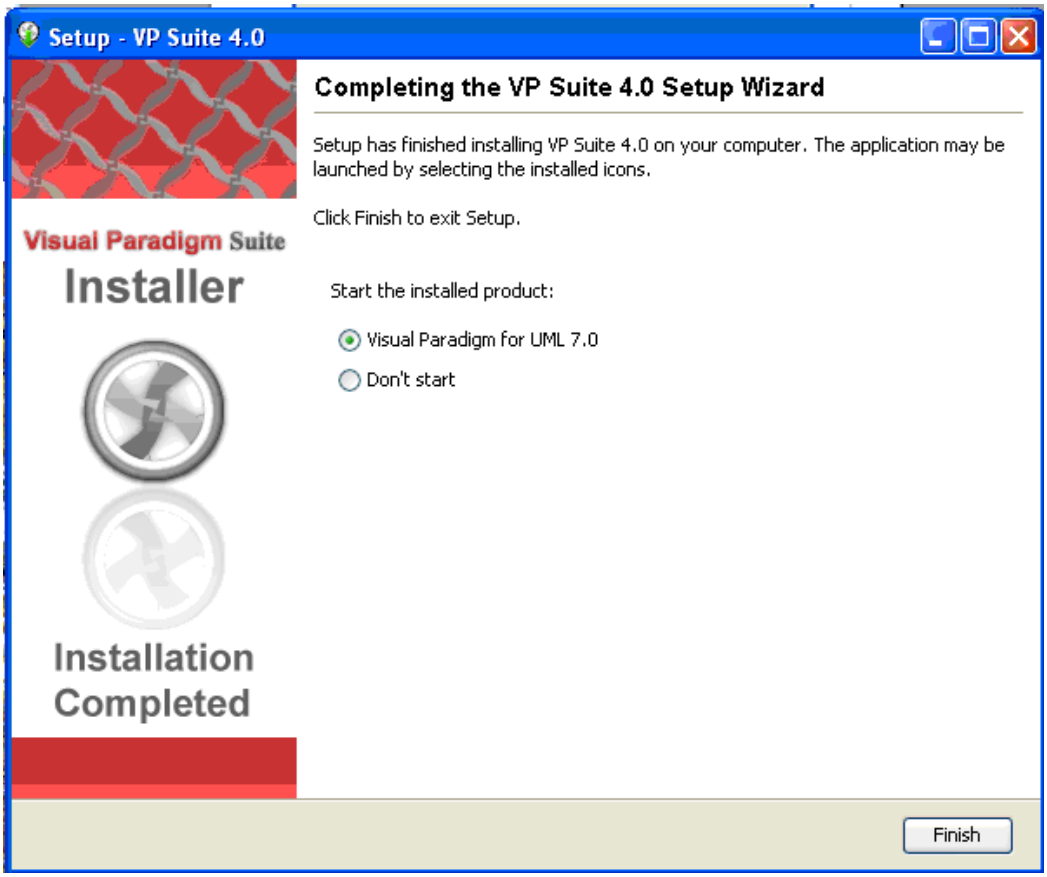
- Check chọn **Signle License key,** rồi click next:

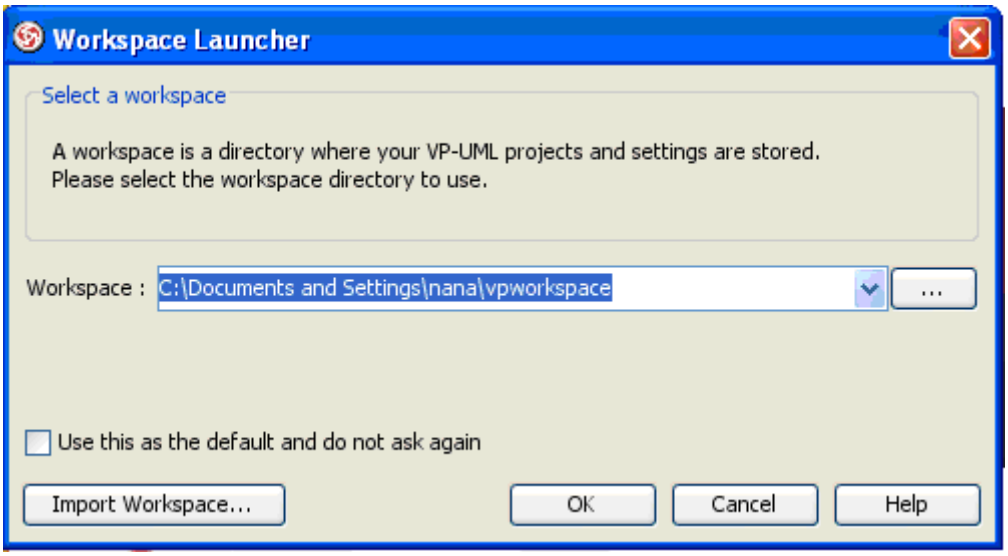

Open
Look in:
Visual paradigm
vpsuite.zvpl
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
File name:
vpsuite.zvpl
Open
Files of type:
License Key (*.zvpl)
Cancel
Setup - VP Suite 4.0
Products License
Select the license key (*.zvpl) for the products.
Products License (Option
Combined License Key
File path:
Visual Paradigm for UML 7.0
File path:
Request Evaluation Key
Browse đến file vpsuite.zvpl
< Back
Next >
Cancel


Setup - VP Suite 4.0
Installing
Please wait while Setup installs VP Suite 4.0 on your computer.
Extracting files...
resources\Report\Default\MergeNode\ObjectIFlows.xml
Cancel



- Click finish để hoàn thành cài đặt:

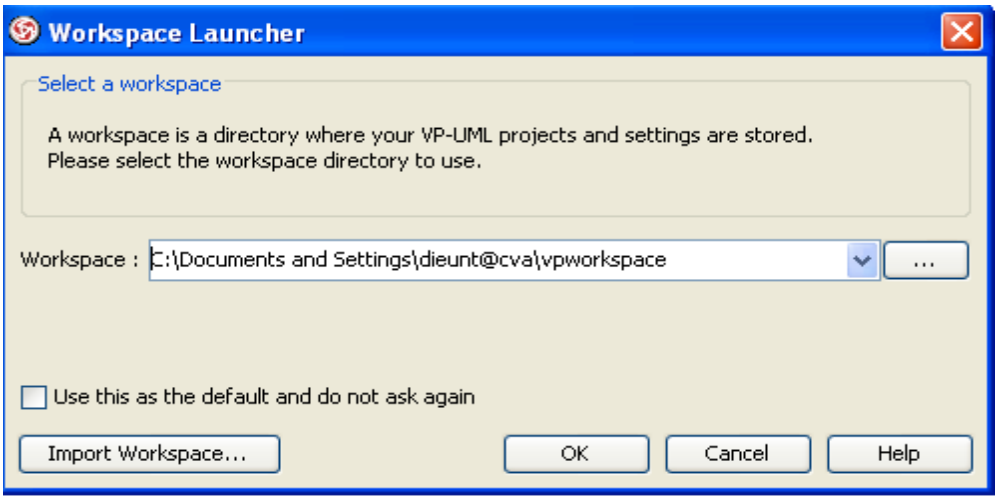



Workspace Launcher
Select a workspace
A workspace is a directory where your VP-UML projects and settings are stored.
Please select the workspace directory to use.
Workspace :
C:\Documents and Settings\nana\vpworkspace
...
Use this as the default and do not ask again
Import Workspace...
OK
Cancel
Help

3.2 Màn hình khởi động Visual Paradigm

Sau khi hoàn tất cài đặt, click vào biểu tượng ***Visual Paradigm for UML 7.0 Enterprise Edition*** hoặc vào **Start → Visual Paradigm → Visual Paradigm for UML 7.0 Enterprise Edition** để khởi động phần mềm:

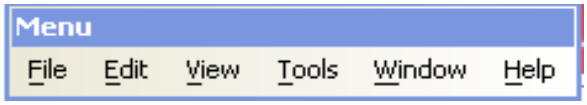
Khi khởi động phần mềm Visual Paradigm, hộp thoại đầu tiên xuất hiện yêu cầu người dùng chọn thư mục Workspace Launcher, thư mục mặc định là thư mục nằm trong Documents and Settings và có tên trùng với tên user đăng nhập máy (Hình 2.9).



Hình 2.9 Hộp thoại chọn thư mục Workspace

Mặc định khi khởi động Visual Paradigm màn hình boa gồm năm phần chính:

- Phần đầu là thanh Menu và các thanh công cụ (Tool bar)
  - Thanh **Menu** bao gồm các menu: File, Edit, View, Tools, Window, Help.



Hình 2.10 Thanh menu

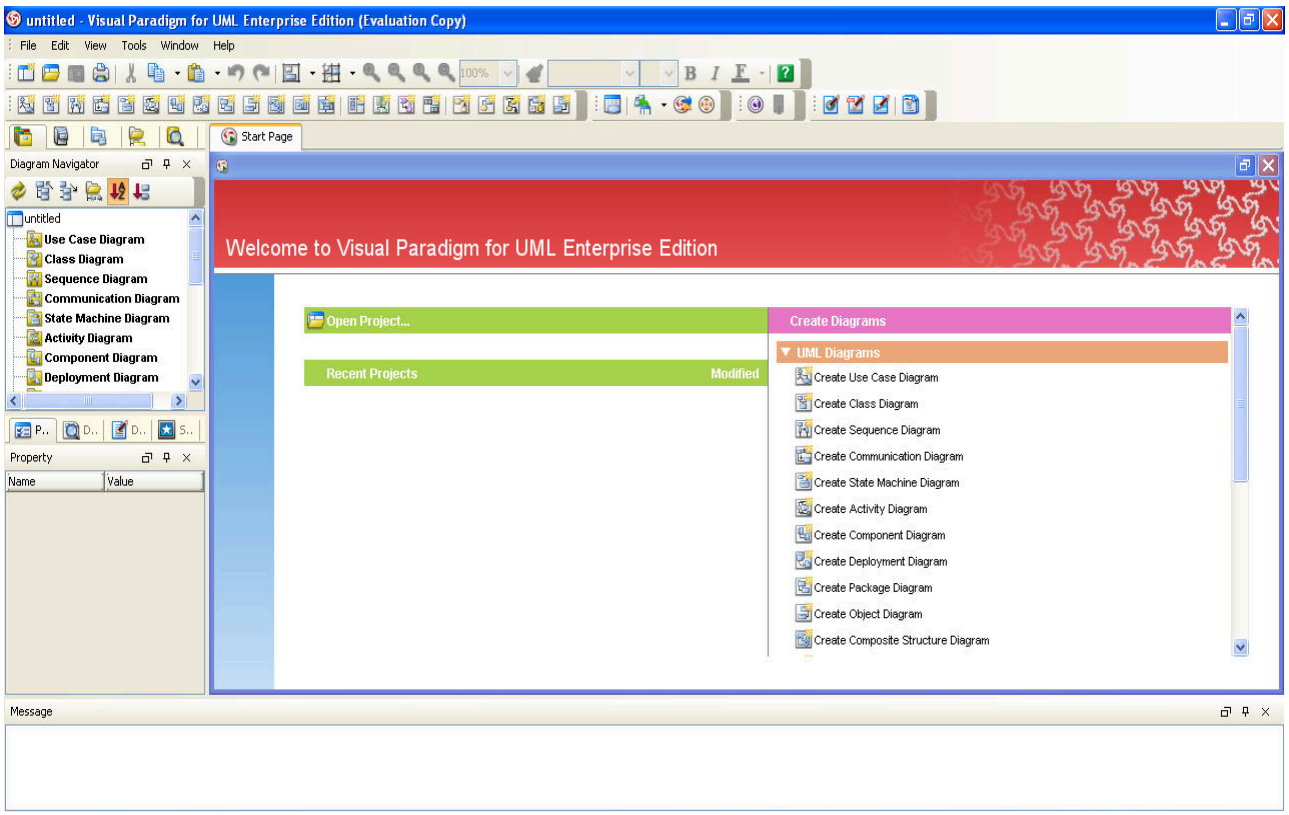
  - Thanh công cụ bao gồm các thanh: Standard, Diagram, ORM, Teamwork, Navigate, Report, Import and Export, Tools.
    - Thanh ***Standard*** là thanh công cụ chuẩn, dùng cho các thao tác cơ bản như Tạo một project mới (New project), Mở một project đã tồn tại (Open project), Lưu project, Copy, Paste, Undo, Redo, các công cụ room màn hình, …

Hình 2.11 Thanh standard

    - Thanh ***Diagram*** là thanh dùng để tạo các biểu đồ trong UML, như: *Biểu đồ Use case* (Use case diagram), *Biểu đồ lớp* (Class diagram), *Biểu đồ tuần tự* (Sequence diagram), *Biểu đồ truyền thông* (Communication diagram), *Biểu đồ trạng thái*

(State diagram), *Biểu đồ hoạt động* (Activity diagram), *Biểu đồ thành phần* (Component diagram), *Biểu đồ triển khai* (Deployment diagram), …

Hình 2.12 Thanh diagram



Hình 2.19 Giao diện khởi động của Visual Paradigm

TỔNG KẾT CHƯƠNG 2

Chương 2 đã giới thiệu ngôn ngữ mô hình hoá thống nhất UML và công cụ Visual Paradigm cho phát triển phần mềm hướng đối tượng. Các nội dung chính cần ghi nhớ:

- UML ra đời từ sự kết hợp các phương pháp luận phát triển phần mềm hướng đối tượng khác nhau đã có trước đó. UML hiện nay đã được coi là ngôn ngữ mô hình hoá chuẩn cho phát triển các phần mềm hướng đối tượng
- UML được chia thành nhiều hướng nhìn, mỗi hướng nhìn quan tâm đến hệ thống phần mềm từ một khía cạnh cụ thể.
- Nếu xét theo tính chất mô hình thì UML có hai dạng mô hình chính là mô hình tĩnh và mô hình động. Mỗi mô hình lại bao gồm một nhóm các biểu đồ khác nhau.
- Mỗi biểu đồ UML có một tập ký hiệu riêng để biểu diễn các thành phần của biểu đồ đó. Quá trình biểu diễn các biểu đồ cũng phải tuân theo các quy tắc được định nghĩa trong UML.

- Hiện nay có rất nhiều công cụ hỗ trợ phân tích thiết kế hệ thống hướng đối tượng sử dụng UML trong đó bộ công cụ Visual Paradigm là bộ công cụ được sử dụng khá rộng rãi với nhiều tính năng ưu việt. Các ví dụ trong tài liệu này đều được xây dựng và biểu diễn trên Visaul Paradigm.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

Câu 1. Ba tác giả gia nhập công ty phần mềm *Rational* để phát triển UML là ai?

Câu 2. Tổ chức nào hiện nay chịu trách nhiệm về chuẩn UML?

Câu 3. Lý do sử dụng UML?

Câu 4. Hướng nhìn là gì? UML bao gồm những hướng nhìn nào?

Câu 5. Liệt kê các biểu đồ của UML?

Câu 6. Phân biệt mô hình tĩnh và mô hình động trong UML?

CHƯƠNG 3. PHÂN TÍCH HỆ THỐNG

1. Giới thiệu Case study

Trong giáo trình này, chúng ta sẽ sử dụng case study: *"Phân tích và thiết kế hệ thống quản lý Thư viện "* toàn bộ giáo trình sẽ chủ yếu xoay quanh phân tích thiết kế case study này.

a) Định nghĩa bài toán

Đây là bước mở đầu của quá trình phát triển hệ thống, nhiệm vụ chủ yếu là xác định các nhu cầu của bài toán. "*Nhu cầu là mẹ của mọi sáng tạo*", cho nên để sáng tạo ra một hệ thống mới, người phát triển trước hết phải làm quen, thâm nhập vào chuyên môn, nghiệp vụ mà hệ thống đó phải đáp ứng và tìm hiểu, tập hợp các nhu cầu của hệ thống

Với hệ thống ***Quản lý thư viện***: Hệ thống cần quản lý toàn bộ đầu sách, sách, tạp chí và tài liệu liên quan của thư viện.

- Khi độc giả đến mượn sách phải ghi phiếu mượn, sau khi cán bộ thư viện nhận phiếu mượn sách của độc giả thì kiểm tra thẻ thư viện của độc giả đó. Nếu độc giả không có thẻ thì yêu cầu làm thẻ thư viện. Nếu độc giả có thẻ thì kiểm tra tài liệu còn nợ của độc giả. Nếu độc giả còn nợ tài liệu thì yêu cầu trả tài liệu còn nợ mới được mượn tài liệu tiếp còn nếu độc giả không nợ tài liệu thì kiểm tra xem khả năng cho mượn hiện tại của thư viện. Nếu không có khả năng cho mượn thì thông báo cho độc giả còn nếu thấy có khả năng cho mượn được thì tìm tài liệu và giao tài liệu cho độc giả.

- Khi độc giả trả tài liệu thì cán bộ thư viện kiểm tra trình trạng của tài liệu và kiểm tra hạn trả tài liệu đó. Nếu tài liệu bị rách thì lập biên bản bồi thường theo nguyên tắc và nếu sách quá hạn trả thì hệ thống cho phép khoá thẻ thư viện của độc giả đó theo nguyên tắc. Ngược lại, hệ thống cho phép cập nhật số lượng sách cũng như tình trạng mượn của độc giả.

- Hệ thống cho phép người quản lý thực hiện in thẻ độc giả khi độc giả đăng ký cấp thẻ cũng như đăng ký cấp lại thẻ trong trường hợp có nhu cầu làm lại.

- Khi cần báo cáo thống kê định kì về danh mục các loại sách có trong thư viện, tình hình bạn đọc.

b) Mục đích của hệ thống

Từ định nghĩa bài toán, chúng ta đưa ra mục đích mà hệ thống phải đạt được để đáp ứng nhu cầu của khách hàng như sau:

- Lên kế hoạch bổ sung tài liệu.
- Nhận sách mới về.
- Phân loại sách.
- Cho độc giả mượn sách.
- Nhận trả sách từ độc giả.
- Tra cứu sách.
- Thông báo sách quá hạn.

- Huỷ sách quá hạn lưu trữ
- Thống kê sách: Thống kê tần suất mượn sách theo thời gian.
- Làm báo cáo hoạt động của thư viện.
- Cấp thẻ cho đọc giả.
- Thống kê sách cho mượn.
- Thống kê đọc giả.
- Quản lý kho sách.
- Quản lý nhà xuất bản.

c) Thuộc tính của hệ thống

Thuộc tính của hệ thống là những yêu cầu *phi chức năng* (Non-functional Requirement), đó là những ràng buộc, những giới hạn và những yêu cầu kỹ thuật mà người phát triển hệ thống phải tuân theo.

Hệ thống HTTN phải có các thuộc tính sau:

- *Thời gian xử lý và trả lời nhanh*, ví dụ: khi chọn danh sách các câu hỏi để trộn n đề thì thời gian xử lý phải nhanh, kết quả trả ra chính xác như mong muốn.
- *Độ chính xác tuyệt đối:* khi tổ chức thi, kết quả chấm bài của thí sinh yêu cầu phải chính xác.
- *Dễ sử dụng với những giao diện đồ hoạ thân thiện phù hợp với người dùng*: giao diện quản lý đặc biệt là giao diện thi của thí sinh phải dễ sử dụng, thân thiện, như các window và các hộp thoại, v.v.
- *Bảo mật:* bảo mật là yêu cầu đầu tiên trong thi cử, đề thi phải được đảm bảo an toàn cho đến khi quá trình tổ chức thi diễn ra. Một điều vô cùng quan trọng là phải tránh được việc thi hộ hay tráo đề thi.
- *Hệ thống thực hiện trên những hệ điều hành phổ dụng* như Microsoft Window 95, 98, 2000, NT, XP, Unix, Linux, v.v.

2. Xác định yêu cầu của hệ thống

Như đã trình bày ở trên, xác định yêu cầu của hệ thống là giai đoạn rất quan trọng trong chu trình phát triển phần mềm. Từ các yêu cầu của khách hàng chúng ta xác định được các mục tiêu của phần mềm cần phát triển. Thường đó là các yêu cầu chức năng về *những gì mà hệ thống phải thực hiện*, nhưng chưa cần chỉ ra các chức năng đó thực hiện như thế nào. Việc xác định đúng và đầy đủ các yêu cầu của bài toán là nhiệm vụ rất quan trọng, nó làm cơ sở cho tất cả các bước tiếp trong dự án phần mềm. Muốn vậy, thì phải đặc tả được các yêu cầu của hệ thống.

UML cung cấp biểu đồ Use case để đặc tả các yêu cầu của hệ thống, và tài liệu đặc tả được sử dụng để:

- Làm cơ sở để trao đổi với người sử dụng, để thảo luận giữa các nhóm thành viên trong dự án phát triển phần mềm về những gì mà hệ thống sẽ phải thực hiện (và những gì nó không cần thực hiện).
- Làm căn cứ cơ bản cho các bước tiếp theo trong quá trình phát triển phần mềm.

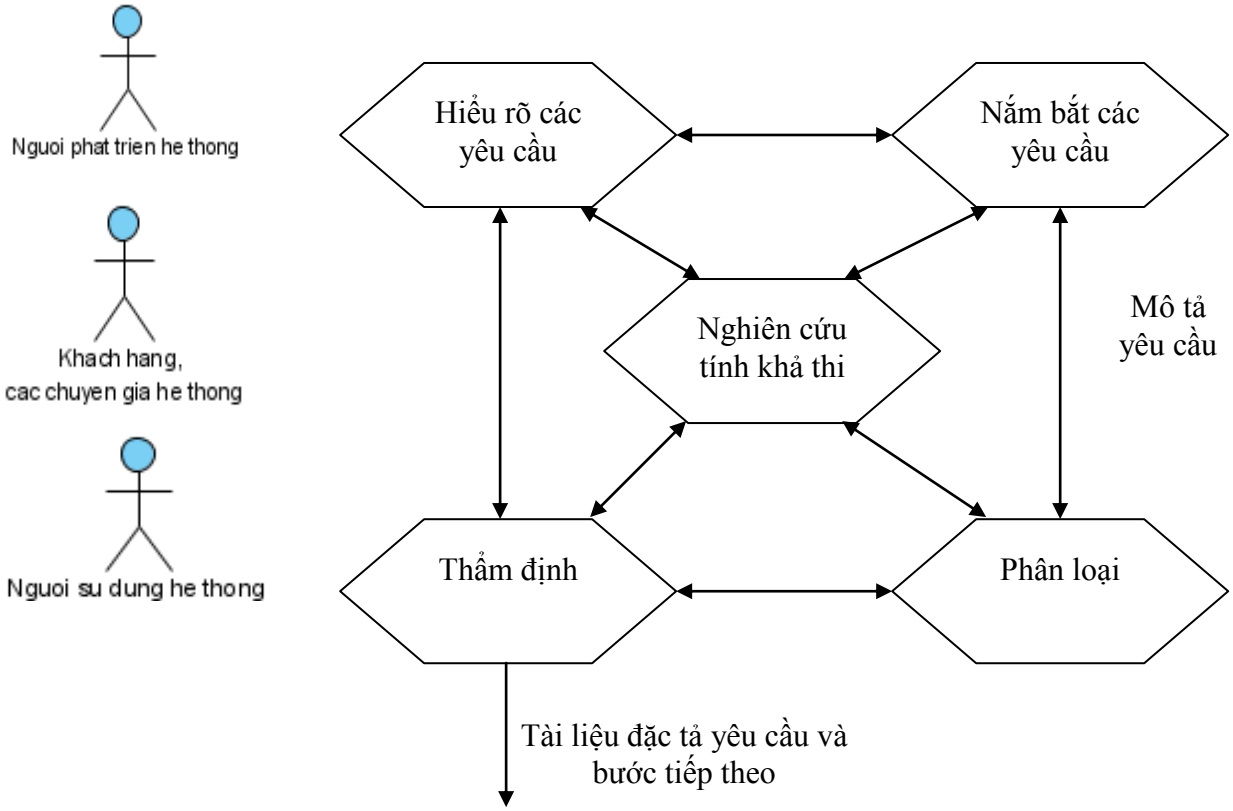
Muốn đạt được các mục tiêu trên thì quá trình phải thực hiện:

- *Hiểu rõ miền xác định của bài toán (Domain Understanding):* Những người phát triển sẽ xây dựng hệ thống theo sự hiểu biết của họ như thế nào về những yêu cầu của khách hàng và những khái niệm cơ sở của bài toán ứng dụng.
- *Nắm bắt các yêu cầu (Requirement Capture):* Người phân tích phải nắm bắt được tất cả các nhu cầu của khách hàng bằng cách phải trao đổi với mọi người có liên quan đến dự án, tham khảo các tài liệu liên quan. Thông qua việc thảo luận, trao đổi với khách hàng, các chuyên gia của lĩnh vực ứng dụng và những người đã, đang sử dụng những hệ thống có sẵn ta có thể phát hiện và nắm bắt được các yêu cầu của họ. Phương pháp trừu tượng hoá giúp ta dễ dàng nắm bắt được các yêu cầu của hệ thống.
- *Phân loại (Classification):* Vấn đề quan trọng nhất trong giai đoạn này là phải hiểu rõ các yêu cầu đã được xác định. Muốn vậy, ta phải tìm cách phân loại chúng theo tầm quan trọng, hay chức năng chính của những người sử dụng và của khách hàng.
- *Thẩm định (Validation):* Kiểm tra xem các yêu cầu có thống nhất với nhau và đầy đủ không, đồng thời tìm cách giải quyết các mối mâu thuẫn giữa các yêu cầu nếu có.
- *Nghiên cứu tính khả thi (Feasibility Study):* Tính khả thi của một dự án tin học phải được thực hiện dựa trên các yếu tố bao gồm các khía cạnh tài chính, chiến lược, thị trường, con người, đối tác, kỹ thuật, công nghệ và phương pháp mô hình hoá, v.v.

Nói chung không có các qui tắc hướng dẫn cụ thể để biết khi nào công việc phân tích các yêu cầu sẽ kết thúc và quá trình phát triển có thể chuyển sang bước tiếp theo. Nhưng có thể dựa vào các câu trả lời cho những câu hỏi sau để chuyển sang bước tiếp theo.

- Khách hàng, người sử dụng (NSD) và những người phát triển đã hiểu hoàn toàn hệ thống chưa?
- Đã nêu được đầy đủ các yêu cầu về chức năng (dịch vụ), đầu vào/ra và những dữ liệu cần thiết chưa?

Bức tranh chung trong pha phân tích yêu cầu của hệ thống có thể mô tả như trong hình 3.1 sau đây:



Hình 3.1 Mối quan hệ giữa các công việc trong pha phân tích yêu cầu hệ thống

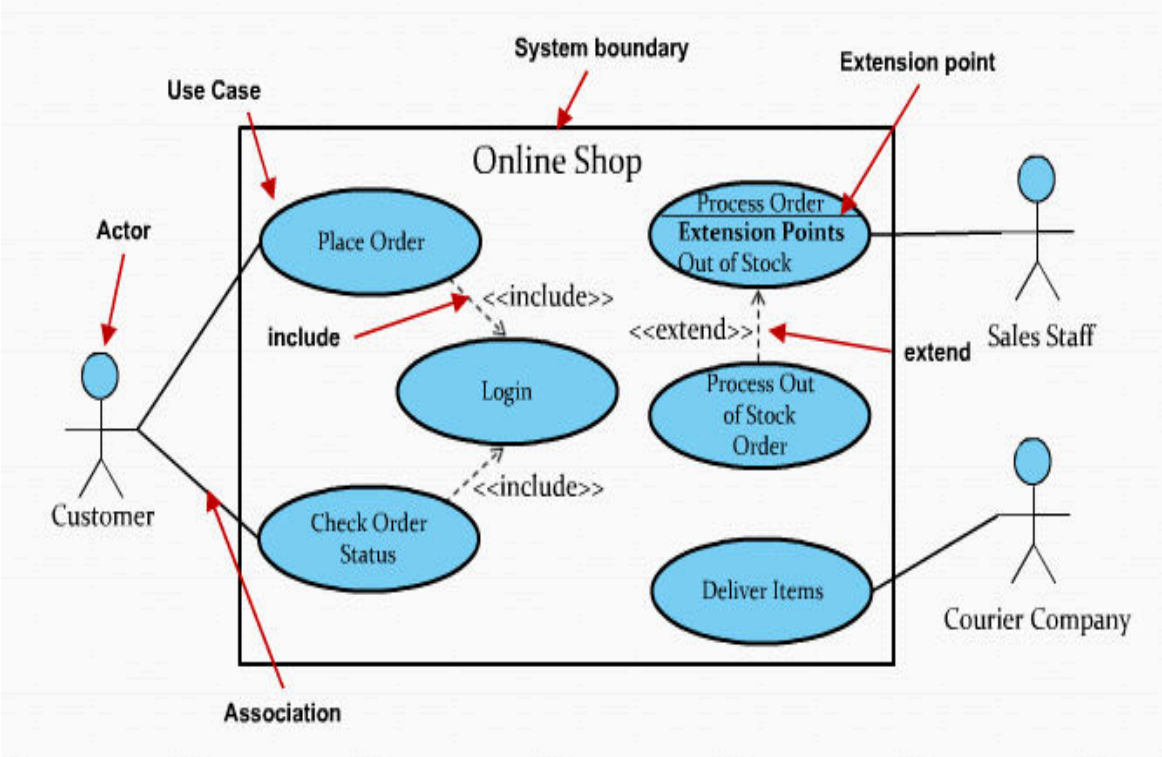
Vấn đề xác định đúng và đầy đủ các yêu cầu của hệ thống là rất quan trọng, nó ảnh hưởng rất lớn tới chất lượng của sản phẩm sau này. Theo *Finkelstein* (1989) khi khảo sát, đánh giá về các pha thực hiện trong quá trình phát triển phần mềm thì trên 55% các lỗi của phần mềm là do các yêu cầu của hệ thống chưa được phát hiện đầy đủ và chi phí cho việc sửa các lỗi đó (để bảo trì hệ thống) chiếm tới trên 80% chi phí của cả dự án.

3. Mô hình hóa Use case

3.1 Giới thiệu

Use case cung cấp một bức tranh toàn cảnh về những gì xảy ra trong hệ thống hiện tại hoặc những gì sẽ xảy ra trong hệ thống mới, do đó có rất nhiều dự án tin học được bắt đầu từ các use case. *Biểu đồ use case* rất đơn giản với rất ít ký hiệu. Đây là một phương tiện giao tiếp hữu hiệu với người dùng về hệ thống, về những gì hệ thống dự định sẽ làm. *Use case* cũng có thể được dùng làm cơ sở cho các *đặc tả kiểm tra* (test specification) sau này.

Biểu đồ use case đưa ra các use case (tình huống sử dụng), các actor (tác nhân) và các association (quan hệ kết hợp) giữa chúng. Hình 3.2 cho thấy sự đơn giản của một biểu đồ use case. Use case biểu diễn chuỗi hành động mà hệ thống thực hiện, actor biểu diễn người hoặc hệ thống khác tương tác với hệ thống đang được mô hình hóa. Các biểu đồ use case được hỗ trợ bởi các đặc tả hành vi (behaviour specification), nhằm định nghĩa các tương tác trong một use case nào đó.



Hình 3.2 Biểu đồ Use case trong Hệ thống bán hàng trực tuyến

Các use case đã được Jacobson đưa vào UML (1992), ban đầu được thực hiện trong công ty Ericsson, Thụy Điển. Trong tiếp cận của Jacobson, use case là điểm bắt đầu cho việc phát triển hệ thống mới. Trong thuật ngữ UML, gói *use case* là một gói con của gói *phần tử hành vi* (behavioural element). Nghĩa là nó được sử dụng để xác định hành vi của một thực thể, chẳng hạn một hệ thống hoặc một hệ thống con. Use case không xác định chi tiết cách mà hành vi được thực hiện như thế nào, chi tiết này sẽ được thảo luận trong các mô hình khác của quy trình phát triển hệ thống. Thông thường, cách thực hiện một use case được định nghĩa trong một hoặc nhiều *biểu đồ cộng tác* (Collaboration diagram) nhằm biểu diễn sự tương tác giữa các đối tượng cùng hoạt động.

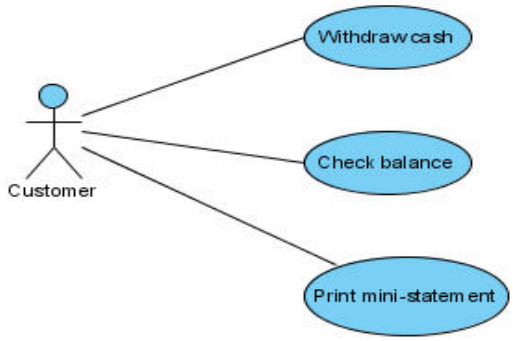
Ví dụ 1: Trong một hệ thống ngân hàng, các use case định nghĩa tương tác giữa khách hàng và *máy rút tiền tự động ATM* (Automated Teller Machines). Hình 3.3 là một biểu đồ use case đơn giản. *Customer* biểu diễn *lớp* (class) của tất cả các khách hàng sử dụng. Khi bạn dùng ATM để rút tiền, thì bạn có một *thể hiện* (instance) của *Customer* đang dùng một *thể hiện* nào đó của use case *withdraw cash*. Một người khác cũng đến rút tiền, anh ta là một thể hiện khác của lớp Customer. Bạn có thể rút tiền thành công, còn anh ta thì không chẳng hạn vì không còn tiền, và withdraw cash xử lý tình huống này khác với của bạn, đó là từ chối yêu cầu. Một số người khác có thể dùng một thể hiện khác của use case *Check balance* hoặc use case *Print mini-statement*.

Thuật ngữ *scenario* (kịch bản) thường được dùng để ám chỉ đến các diễn biến khác nhau mà các thể hiện của cùng một use case sẽ thực hiện. Biểu đồ use case chỉ đặt tên cho các use case, các tài liệu hoặc biểu đồ kèm theo sẽ chỉ ra các *scenario*.

Ví dụ 2. Các *scenario* của use case *withdraw cash* có thể là:

- Thẻ của khách hàng không được nhận biết và bị từ chối.
- Khách hàng nhập số PIN sai và được yêu cầu nhập lại.

- Khách hàng nhập số PIN sai 3 lần thẻ bị ATM nuốt.
- Khách hàng nhập số lượng tiền không hợp lệ.
- ATM cố gắng kết nối với hệ thống ngân hàng nhưng không được do nó bị trục trặc hoặc mạng có vấn đề.
- ATM không đủ tiền mặt để đáp ứng nhu cầu của khách hàng.
- Tài khoản của khách hàng còn ít hơn số muốn rút.
- Khách hàng hủy bỏ việc rút tiền.



Hình 3.3 Biểu đồ use case của hệ thống con ATM

Chúng ta có thể có thêm vài *scenario* khác, use case *withdraw cash* sẽ biểu diễn tất cả. Cuối cùng use case này có thể xác định thật chi tiết, đủ để đối tượng tạo ra hệ thống xử lý một cách chính xác. Ở giai đoạn đầu, chúng ta nên xác định diễn biến đơn giản (khách hàng rút thành công lượng tiền yêu cầu) và các *scenario* gần gũi nhất.

3.2 Mục đích và các ký hiệu trong Use case

3.2.1 Mục đích của biểu đồ Use case

Các use case được tạo ra ở giai đoạn đầu của một dự án. Tạo ra các biểu đồ use case và các tài liệu là công việc thiên về kỹ thuật phân tích hơn kỹ thuật thiết kế. Các use case cũng có thể được dùng ở giai đoạn sau của quy trình phát triển dự án. Ví dụ để đặc tả các tình huống kiểm tra. Các use case được dùng để mô hình hóa cái gì xảy ra trong hệ thống hiện tại, hoặc có thể mô hình hóa hệ thống sắp phát triển.

Các mục đích sử dụng chính:

- Dùng để mô hình hóa các chuỗi hành động mà hệ thống sẽ thực hiện và nhằm cung cấp một kết quả có ý nghĩa cho một người nào đó hoặc một hệ thống bên ngoài mà chúng ta gọi chung là actor.
- Cung cấp một cái nhìn tổng thể về những gì mà hệ thống sẽ làm và ai dùng nó.
- Đưa ra cơ sở để xác định giao tiếp giữa người-máy đối với hệ thống.
- Dùng để mô hình hóa các scenario cho một use case.
- Để người dùng cuối có thể hiểu được và có thể giao tiếp với hệ thống ở mức tổng thể.
- Làm cơ sở cho việc phác thảo các đặc tả kiểm tra.

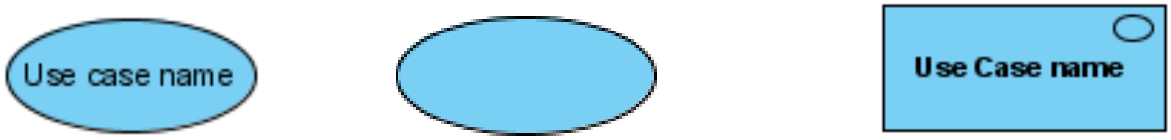
3.2.2 Các ký hiệu trong biểu đồ Use case

Use case mô tả một chuỗi các hành động mà hệ thống sẽ thực hiện để đạt được kết quả có ý nghĩa đối với một tác nhân. Mô tả có thể được biểu diễn bằng văn bản có cấu trúc (chẳng hạn như mã giả), hoặc thông qua một *đặc tả hành vi* được biểu diễn bởi một liên kết đến một biểu đồ khác (chẳng hạn như biểu đồ cộng tác). Cái nhìn ở mức cao của các use case sẽ được biểu diễn bởi biểu đồ use case.

Các ký hiệu cơ bản: use case, actor và relationship

Use case, được biểu diễn bằng đồ thị trong *biểu đồ use case*, cho phép phân tích viên hình dung từng use case trong ngữ cảnh của các use case khác, và cho thấy mối qua hệ của nó với các actor và các use case khác.

Trong biểu đồ, các use case được biểu diễn bằng hình elipse (Hình 3.4), tên use case được đặt bên trong hoặc bên dưới, chúng ta phải biểu diễn nhất quán, nghĩa là trong cùng một biểu đồ không nên biểu diễn use case bằng cả hai cách.



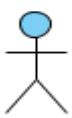
Use case name

Hình 3.4 Ký hiệu Use case

Tên use case là một chuỗi gồm các ký tự, các con số và hầu hết các dấu phân cách, ngoại trừ dấu hai chấm bởi dấu hai chấm được dùng để phân tách tên use case với tên package. Quy ước đặt tên cho use case như sau: *Trước hết là một động từ và sau đó là cụm danh từ mô tả hành vi*. Ví dụ: ”Kiểm tra đơn hàng”, “Đặt hàng”, “Trả nợ”, “Xử lý thiếu tiền”, …Xem Hình 3.5.

Hình 3.5 Các tên use case hợp lệ

Actor là người hoặc hệ thống tương tác với các use case. Thường thì actor là người dùng hệ thống. Trong biểu đồ use case, mỗi actor được vẽ bằng một biểu tượng hình người với *tên vai trò* (role name) đặt bên dưới (Hình 3.6).



Actor role name

Hình 3.6 Ký hiệu actor

Khi actor là người, thì tên actor là tên vai trò mà actor đảm nhiệm chứ không phải là tên công việc.

Ví dụ, trong *Hệ thống quản lý thư viện*, *"Độc giả"* là một Actor của hệ thống.

Hình 3.7 Actor "Độc giả"

Đoạn thẳng nối actor và use case mô tả quan hệ giữa chúng là quan hệ (relationship) tương tác giữa actor và use case:

Hình 3.8 Actor và use case

Quan hệ này cho biết một kết hợp giữa một actor và một use case. Nghĩa là con người hoặc hệ thống, trong vai trò actor này, sẽ giao tiếp với các thể hiện của use case, tham gia vào chuỗi các biến cố được use case biểu diễn. Trong thực tế, use case sẽ được cài đặt thành chức năng của hệ thống máy tính và actor là người nhập thông tin vào hệ thống và nhận thông tin ra.

Một actor có thể kết hợp với một hoặc nhiều use case và một use case cũng có thể kết hợp với một hoặc nhiều actor.

Đặc tả hành vi (behaviour specification)

Mỗi use case biểu diễn một chuỗi hoạt động đưa ra một số kết quả đối với một hoặc nhiều actor. Chuỗi hoạt động này được sưu liệu trong một đặc tả hành vi. Một công cụ CASE có thể biểu diễn đặc tả này bằng một biểu đồ. Biểu đồ này có thể là *biểu đồ tuần tự* (sequence diagram), *biểu đồ cộng tác* (collaboration diagram), *biểu đồ trạng thái* (statechart diagram) hoặc có thể là *văn bản* (mã của ngôn ngữ lập trình). Thông thường thì một đặc tả không hình thức được cungg cấp như một mô tả use case. Trong CASE, có thể dùng cả hai cách để mô tả use case: Đặc tả không hình thức và đặc tả hình thức bằng cách liên kết đến biểu đồ.

Một số biểu đồ UML có các quy tắc cú pháp để biểu diễn thông tin văn bản. Mục đích là để nó có thể thực thi mô hình và mô phỏng hệ thống, thậm chí chuyển mô hình thành mã chương trình của một ngôn ngữ lập trình nào đó, như C++ hoặc Java. Tuy nhiên, các mô tả use case không có bất kỳ cú pháp nào cả bạn có thể mô tả bằng bất kỳ dạng nào bạn thích. Tất nhiên, khi làm việc trong một công ty, bạn sẽ phải tuân thủ quy tắc của công ty khi nhập sưu liệu cho use case.

Có hai hướng tiếp cận thông dụng để viết các mô tả use case:

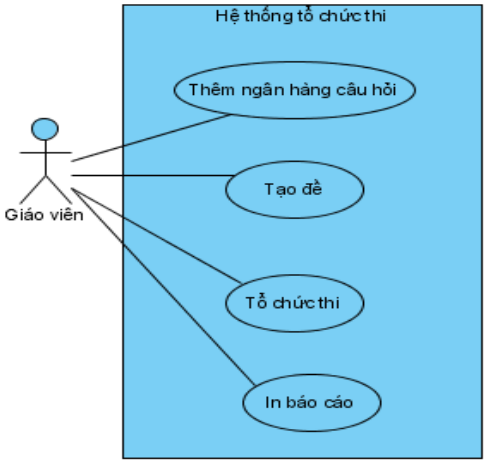
- Thứ nhất là viết ngắn gọn một hoặc vài phát biểu hoặc đoạn văn để mô tả chuỗi hoạt động tiêu biểu của use case.

- Thứ hai là liệt kê thành hai cột, một cột là hoạt động của tác nhân và cột còn lại là đáp ứng của hệ thống.

Tác nhân	Hệ thống

Bảng 3.1 Đặc tả actor

Các use case trong biểu đồ có thể được đặt trong hình chữ nhật. Hình chữ nhật này hoặc ánh xạ vào use case hoàn chính chứa tất cả các use case và actor, hoặc ánh xạ vào hệ thống hay hệ thống con chứa chúng. Hình 3.9 các use case trong hệ thống con.



Hình 3.9 Các use case trong một hệ thống con

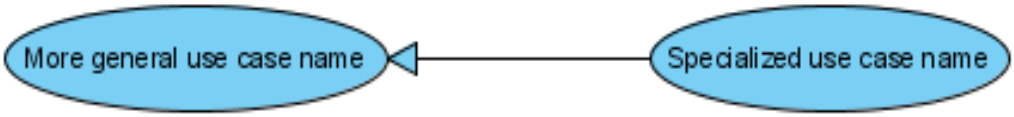
3.3 Các kiểu kết hợp (association) và quan hệ (relationship)

Có bốn kiểu kết hợp và quan hệ trong một biểu đồ use case:

- Kết hợp *generalization* (tổng quát hóa) giữa các use case.
- Kết hợp *generalization* giữa các actor.
- Quan hệ *include* (bao gồm) giữa các use case.
- Quan hệ *extend* (mở rộng) giữa các use case

3.3.1 Kết hợp generalization giữa các use case

Đôi khi chúng ta gặp trường hợp cùng một use case lại có nhiều phiên bản khác nhau, các phiên bản này có cùng một số hành động giống nhau và một số hành động không giống nhau. Hãy khảo sát hai use case cùng chức năng thêm ngân hàng câu hỏi vào cơ sở dữ liệu như sau: *"Nhập câu hỏi"* và *"Thêm câu hỏi từ file word"* hai use case cùng một chức năng nhưng hoạt động lại không hoàn toàn giống nhau, ta có thể xem chúng như hai use case đặc biệt của use case *"Thêm ngân hàng câu hỏi"*, lúc này ta gọi *"Thêm ngân hàng câu hỏi"* là một use case tổng quát. Điều này được biểu diễn trong biểu đồ thông qua việc dùng *generalization*, ký hiệu là một mũi tên hình tam giác và đường nối hướng đến use case tổng quát. Xem hình 3.10.



Hình 3.10 Ký hiệu kết hợp generalization *giữa các use case*

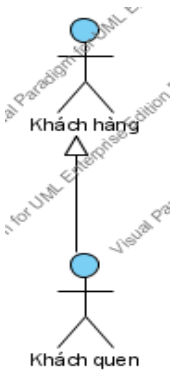
Đôi khi một use case tổng quát ở mức cao là use case không bao giờ tồn tại trong hệ thống thực, nó được dùng định nghĩa các chức năng chung cho các use case cụ thể. Trong trường hợp này chúng ta gọi nó là *abstract use case* (use case trừu tượng), tên của use case trừu tượng sẽ được in nghiêng. Các use case đặc biệt thường được gọi là *concrete use case* (use case cụ thể) hay real use case (use case thực). *Generalization* thường được cài đặt bằng kỹ thuật kế thừa (inheritance).

3.3.2 Kết hợp generalization giữa các actor

Actor cũng có mối quan hệ kế thừa. Ví dụ như có thể có hai actor là "nhân viên trả lương tháng", "nhân viên làm hợp đồng". Cả hai đều thuộc một kiểu là "Nhân viên". Actor "Nhân viên" là một actor trừu tượng vì nó không có một thể hiện nào trong thực tế, nó được dùng để chỉ ra rằng có một số điểm chung giữa hai actor trên, xem hình 3.13.

Nói chung việc mô tả quan hệ kế thừa giữa các Actor là không cần thiết, trừ trường hợp chúng thực hiện những tương tác khác nhau đối với hệ thống, ký hiệu xem hình 3.12.

Hình 3.12 Quan hệ generalization giữa hai actor

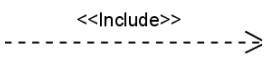


Hình 3.13 Quan hệ generalization giữa các actor trong hệ thống con

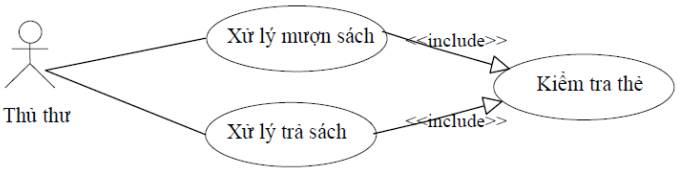
3.3.3 Quan hệ Include giữa các use case

- Ý nghĩa: được thành lập khi chúng ta có các use case mà tìm thấy một vài use case có những dòng hoạt động chung, và để tránh mô tả dòng hoạt động chung đó lặp lại trên những use case này, chúng ta có thể tách những dòng hoạt động chung đó ra thành một use case khi đó tồn tại quan hệ ***include***

- Ký hiệu:



- Ví dụ:

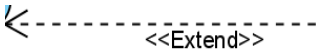


Hình 3.14. Biểu diễn quan hệ Include

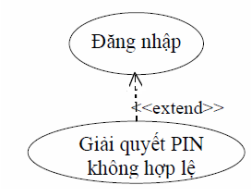
3.3.4 Quan hệ Extend giữa các use case

- Ý nghĩa: được dùng khi chúng ta có một use case tương tự như use case khác nhưng có nhiều hơn một vài xử lý đặc biệt.

- Ký hiệu:



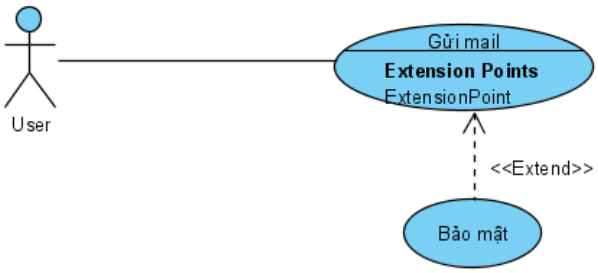
- Ví dụ:



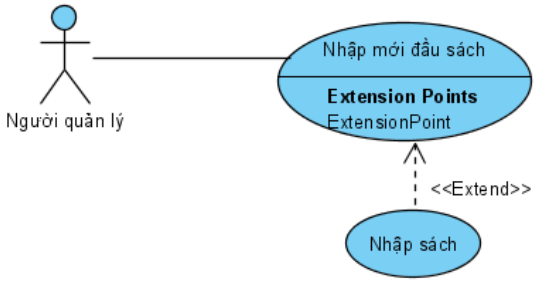
Hình 3.15. Biểu diễn quan hệ Extend

Quan hệ extend dùng để chỉ:

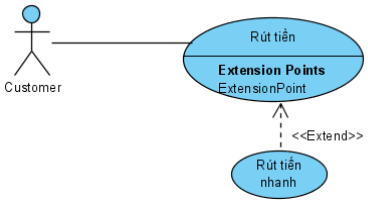
- Các hành vi tùy chọn (Có thể thực hiện hay không). Ví dụ: khi gửi email có thể thực hiện các thao tác bảo mật nội dung thư hoặc là không. Ta có Use case "Bảo mật" có quan hệ extends với Use case "Gửi email":



- Các hành vi mà chỉ thực hiện trong một số điều kiện nhất định. Ví dụ: trong bài toán quản lý thư viện khi thực hiện "Thêm mới đầu sách" người dùng có thể chọn "Thêm mới sách" hay không tùy chọn sách đã có trong thư viện hay chưa.



- Một số hành vi khác sẽ được thực hiện phụ thuộc vào sự lựa chọn của người dùng.Ví dụ như: người dùng của hệ thống rút tiền tự động có thể chọn Rút tiền nhanh hoặc Rút tiền theo cách bình thường. Ta có Use case "Rút tiền nhanh" có quan hệ extends với Use case "Rút tiền".



3.4 Sơ đồ Use case

Làm thế nào để đưa các use case? Chúng ta xác định được các use case và actor của hệ thống đang phân tích dựa trên các tài liệu nguồn, chẳng hạn như các ghi chú và các bản ghi chép lại từ những cuộc phỏng vấn, đó chính là các tài liệu phân tích yêu cầu người sử dụng.

3.4.1 Xác định actor

Actor không phải là một phần của hệ thống. Nó thể hiện một người hay một hệ thống khác tương tác với hệ thống. Vai trò của một Actor có thể:

- Chỉ cung cấp thông tin cho hệ thống.
- Chỉ lấy thông tin từ hệ thống.
- Nhận thông tin từ hệ thống và cung cấp thông tin cho hệ thống

Thông thường, các actor được tìm thấy trong phát biểu bài toán bởi sự trao đổi giữa phân tích viên với khách hàng và các chuyên gia trong lĩnh vực(domain expert). Các câu hỏi thường được sử dụng để xác định actor cho một hệ thống là:

- Ai là người sẽ dùng hệ thống để *nhập* thông tin?
- Ai là người sẽ dùng hệ thống để *nhận* thông tin?
- Hệ thống có tương tác với các hệ thống nào khác không?

Có ba loại actor chính:

- *Người dùng*. Ví dụ: giáo viên, sinh viên, nhân viên, khách hàng...
- *Hệ thống khác*.
- *Sự kiện thời gian*. Ví dụ: Kết thúc tháng, đến hạn...

Điều gì tạo nên một tập hợp Actor tốt?

Cần phải cân nhắc kỹ lưỡng khi xác định actor của hệ thống. Công việc này thường được thực hiện lặp đi lặp lại. Danh sách đầu tiên về các actor hiếm khi là danh sách cuối cùng.

Ví dụ như trong bài toán đăng kí các môn học của một trường đại học, có một câu hỏi là liệu các sinh viên mới vào trường là một actor và sinh viên cũ là một actor khác? Giả sử câu trả là có thì bước tiếp theo là xác định xem cách thức mà hai actor này tương tác với hệ thống. Nếu chúng sử dụng hệ thống theo những cách khác nhau thì chúng là hai actor ngược lại sẽ chỉ là một actor mà thôi.

Mô tả Actor:

Việc mô tả một cách ngắn gọn về mỗi actor cần thêm vào mô hình. Mô tả này cần chỉ rõ vai trò của actor khi tương tác với hệ thống.

Ví dụ: *"Sinh viên: là những người đăng kí học các lớp ở trường đại học"*.

Ví dụ xác định actor trong Bài toán quản lý thư viện:

Sau đây là cuộc phỏng vấn nhỏ giữa nhà phân tích (NPT) và Người quản lý thư viện (NQL):

NPT: Ai sẽ là người trực tiếp thực hiện quá trình xử lý khi sinh viên đến đăng ký làm thẻ thư viện?

NQL: Khi sinh viên muốn làm thẻ thì cần phải điền đầy đủ thông tin vào phiếu đăng ký và nộp lại cho người quản lý thư viện. Căn cứ vào phiếu đăng ký, thư viện sẽ cấp phát thẻ cho sinh viên.

NPT: Ai sẽ là người trực tiếp xử lý khi độc giả đến mượn trả sách?

NQL: Thủ thư sẽ trực tiếp thực hiện các thao tác trên.

Để xác định Actor của hệ thống chúng ta thực hiện trả lời các câu hỏi sau:

- Ai sẽ là người dùng hệ thống để *nhập* thông tin?

Trong hệ thống Quản lý thư viện, sẽ phải có người chịu trách nhiệm quản lý, nhập vào danh mục sách, đầu sách trong thư viện. Ta sẽ xác định được một Actor là "***Người quản lý"***.

- Ai sẽ là người dùng hệ thống để *nhận* thông tin?

Dữ liệu kết xuất ra có thể là các kết xuất báo cáo hàng kỳ, hàng năm học. Người nhận những dữ liệu này có thể là Thủ thư hoặc người quản lý. Như vậy ta có tác nhân nữa là "***Thủ thư"***.

- Hệ thống có tương tác với hệ thống nào khác không? Hệ thống là hoạt động độc lập và không nhận hay trả dữ liệu cho hệ thống khác (không tương tác với hệ thống khac).

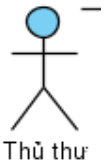
Ngoài ra việc xác định tác nhân của hệ thống có thể thực hiện bằng cách trả lời các câu hỏi sau:

1. Ai sẽ sử dụng các chức năng chính của hệ thống (HT)?
2. Ai cần sự hỗ trợ của HT để thực hiện các công việc hàng ngày?
3. Ai quản trị, bảo dưỡng để đảm bảo cho HT hoạt động thường xuyên?

4. Hệ thống quản lý, sử dụng những thiết bị nào?
5. Hệ thống cần tương tác với những bộ phận, hệ thống nào khác?
6. Ai hay cái gì quan tâm đến kết quả xử lý của hệ thống?

Như vậy ta xác định được ba actor cho hệ thống:

Đặc tả: Trực tiếp sử dụng hệ thống. Đóng vai trò quản trị hệ thống, thực hiện phân loại và quản lý, lên kế hoạch bổ sung tài liệu, quản lý thông tin về độc giả, thẻ độc giả. Quản lý kho sách cũng như các thống kê thường xuyên.



Đặc tả: Thủ thư là người trực tiếp sử dụng hệ thống. Có vai trò xử lý mượn trả tài liệu khi độc giả có yêu cầu. Xử lý khi độc giả vi phạm quy chế thư viện.

Đặc tả: Không trực tiếp sử dụng hệ thống, đóng vai trò là một tác nhân bên ngoài. Độc giả có quyền đăng ký làm thẻ với người quản lý. Thực hiện mượn trả sách.

3.4.2 Xác định Use case

Định nghĩa: Use case là một khối chức năng được thực hiện bởi hệ thống để mang lại một kết quả có giá trị đối với một actor nào đó.

Mô tả:

Use case mô tả sự tương tác đặc trưng giữa người dùng và hệ thống. Nó thể hiện ứng xử của hệ thống đối với bên ngoài, trong một hoàn cảnh nhất định, xét từ quan điểm của người sử dụng. Nó mô tả các yêu cầu đối với hệ thống, có nghĩa là những gì hệ thống phải làm chứ không phải mô tả hệ thống làm như thế nào. Tập hợp tất cả Use case của hệ thống sẽ mô tả tất cả các trường hợp mà hệ thống có thể được sử dụng.

Một Use case có thể có những biến thể. Mỗi một biến thể được gọi là một kịch bản (scenario). Phạm vi của một Use case thường được giới hạn bởi các hoạt động mà người dùng thực hiện trên hệ thống trong một chu kì hoạt động để thực hiện một sự kiện nghiệp vụ.

Một Use case mô tả một nghiệp vụ thông thường. Nghiệp vụ này bao gồm các bước riêng rẽ, còn được gọi là các hoạt động. Khi các bước được mô tả dưới dạng văn bản thì việc chỉ ra sự phụ thuộc giữa các bước là một việc mất nhiều thời gian. Việc thể hiện các bước dưới dạng kí hiệu là dễ dàng và dễ hiểu hơn. Do đó Use case thường được mô tả chi tiết thông qua các biểu đồ mô tả hành vi (behavior) như biểu đồ hoạt động (activity diagram), biểu đồ trình tự (sequence diagram), biểu đồ hợp tác(collaboration diagram).

Use case cũng có thể được mô tả thông qua các thiết kế nguyên mẫu màn hình, các ví dụ về biểu mẫu báo cáo. Điều này giúp cho người dùng dễ dàng mường tượng hệ thống sẽ làm việc như thế nào, qua đó có thể kiểm tra tính đúng đắn của Use case.

Các câu hỏi thường được sử dụng để xác định Use Case cho một hệ thống là:

- Nhiệm vụ của mỗi actor là gì?
- Có actor nào sẽ tạo, lưu trữ, thay đổi, xóa hoặc đọc thông tin trong hệ thống?
- Có actor nào cần báo tin cho hệ thống về một thay đổi đột ngột từ bên ngoài?
- Có actor nào cần được thông báo về một sự việc cụ thể xảy ra trong hệ thống?
- Trường hợp sử dụng nào sẽ hỗ trợ và bảo trì hệ thống?
- Tất cả các yêu cầu về mặt chức năng có được thể hiện hết thông qua các trường hợp sử dụng chưa?

Chú ý: Tên Use case phải bắt đầu bằng một động từ.

Điều gì tạo nên một Use Case tốt?

Có một câu hỏi thường xuyên được đặt ra về mức độ chi tiết của Use case. Nó nên ở mức độ nào là tốt. Có lẽ không có câu trả lời hoàn toàn đúng, nhưng có một số nhận xét như sau: "Một Use case thường biểu hiện một chức năng được thực hiện trọn vẹn (không ngắt quãng) từ đầu đến cuối. Một Use case phải mang lại một điều gì đó có giá trị đối với actor".

Mô tả Use case:

Use case cần có một vài câu ngắn gọn mô tả mục đích của Use case, cho ta biết chức năng do Use case cung cấp.

Luồng sự kiện cho một Use case:

Use case chỉ cung cấp một khung nhìn ở mức cao, tổng quát. Để hiểu rõ hơn hệ thống cần phải làm gì thì cần phải mô tả chi tiết hơn, gọi là luồng sự kiện. Nó là một tài liệu mô tả các hoạt động cần thiết để đạt được ứng xử mong đợi của Use case.

Tuy là mô tả chi tiết nhưng luồng sự kiện vẫn được viết sao cho có thể chỉ ra những gì hệ thống cần làm chứ không phải chỉ ra hệ thống làm như thế nào.

Ví dụ: trong luồng sự kiện chúng ta nói *"Kiểm tra mã của người dùng"* chứ không nói rằng việc đó phải thực hiện bằng cách xem xét ở trong một bảng nào đó trong cơ sở dữ liệu. Nó mô tả chi tiết những gì người dùng của hệ thống sẽ làm và những gì hệ thống sẽ làm. Nó cần phải đề cập tới:

- Use case bắt đầu và kết thúc khi nào và như thế nào?
- Có những sự tương tác nào giữa Use case và actor để thực hiện chức năng đó.
- Những dữ liệu nào cần thiết cho Use case?
- Thứ tự thực hiện thông thường của các sự kiện.
- Các mô tả về các luồng ngoại lệ hoặc rẽ nhánh.

Mỗi dự án cần có một mẫu chuẩn cho việc tạo tài liệu về luồng sự kiện. Có thể dùng theo mẫu đơn giản như sau:

X. Luồng sự kiện cho Use case ABC

X1. Điều kiện bắt đầu: danh sách những điều kiện phải thỏa mãn trước khi Use case được thực hiện. Ví dụ như: một Use case khác phải thực hiện trước khi Use case này được thực hiện hay người dùng phải có đủ quyền để thực hiện Use case này. Không nhất thiết mọi Use case đều phải có điều kiện bắt đầu.

X2. Luồng sự kiện chính: mô tả những bước chính sẽ xảy ra khi thực hiện Use case.

X3. Các luồng phụ (luồng con).

X4. Các luồng rẽ nhánh.

Trong đó X là số thự tự của Use case trong hệ thống.

Ví dụ: Luồng sự kiện mô tả Use case cho hệ thống rút tiền tự động như sau:

1.1 Điều kiện bắt đầu.

1.2 Luồng chính:

1.2.1 Người dùng đưa thẻ vào máy.

1.2.2. Máy hiển thông báo chào mừng và yêu cầu nhập mã số

1.2.3 Người dùng nhập mã số

1.2.4 Máy xác nhận mã số đúng. Nếu nhập sai mã số, luồng rẽ nhánh E-1 được thực hiện.

1.2.5 Máy hiện ra ba lựa chọn:

Rút tiền: luồng con A-1

Chuyển tiền: luồng con A-2

Thêm tiền vào tài khoản: luồng con A-3

1.2.6 Người dùng chọn rút tiền

1.3. Luồng con:

1.3.1 Luồng con A-1

1.3.1.1 Máy hỏi số lượng tiền cần rút

1.3.1.2 Người dùng nhập số tiền cần rút

Máy kiểm tra trong tài khoản có đủ tiền không. Nếu không đủ luồng rẽ nhánh E-2 được thực hiện

....

14. Luồng rẽ nhánh:

1.4.1 E-1: Người dùng nhập sai mã số

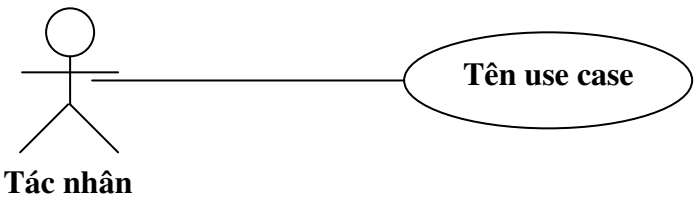
Máy thông báo là người dùng đã nhập sai mã số yêu cầu người dùng nhập lại hoặc hủy bỏ giao dịch.

1.4.2 E-2: Không đủ tiền trong tài khoản...

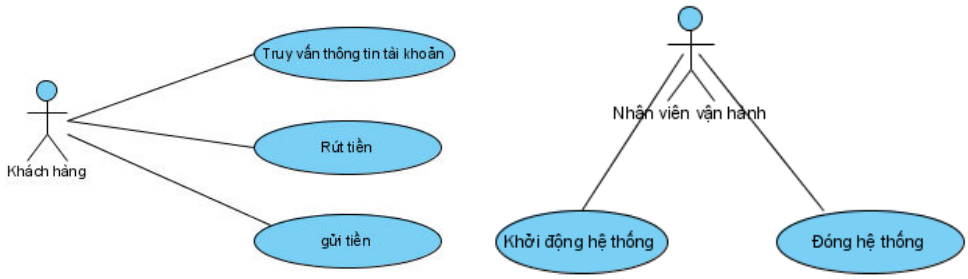
3.4.3 Xác định mối quan hệ

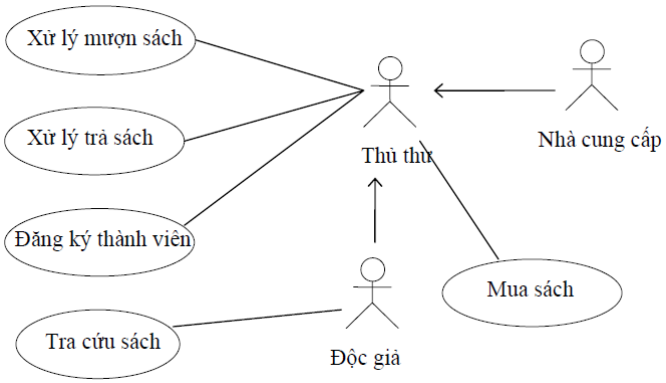
- **Quan hệ tác nhân và use case**: Quan hệ này cho biết tác nhân sẽ tương tác với use case. Một use case luôn luôn khởi tạo bằng một tác nhân và có thể tương tác với nhiều tác nhân.

Ký hiệu:



Ví dụ:



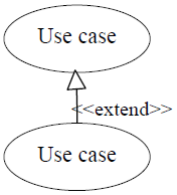


- **Mối quan hệ giữa các use case:** Việc mô tả use case sẽ khó hiểu nếu use case này chứa đựng nhiều luồng phụ hoặc luồng ngoại lệ chỉ xử lý cho những sự kiện trong những điều kiện đặc biệt. Để làm đơn giản mô tả này chúng ra sử dụng các mỗi quan hệ <<extend>> và <<include>>.

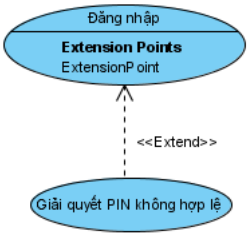
a) Quan hệ mở rộng <<extend>>: được dùng khi chúng ta có một use case tương tự như use case khác nhưng có nhiều hơn một vài xử lý đặc biệt. Giống như liên kết tổng quát – chuyên biệt, trong đó, use case chuyên biệt là một mở rộng của use case tổng quát bằng việc đưa thêm vào các hoạt động hoặc ngữ nghĩa mới vào use case tổng quát, hoặc bỏ qua hoạt động của use case tổng quát.

Ví dụ: giả sử Đăng nhập là một use case cơ bản. Use case này sẽ đại diện cho tất cả những gì được xem là thực hiện đăng nhập một cách xuyên suốt.Tuy nhiên, nhiều vấn đề có thể tác động đến dòng sự kiện chính. Ví dụ, mã số PIN không hợp lệ, hoặc thẻ không đọc được do bị hư,…. Do đó, chúng ta không phải luôn luôn thi hành các hoạt động thường xuyên của một use case được cho và như vậy, cần thiết tạo ra các use case mới để giải quyết những tình huống mới. Tất nhiên, chúng ta có thể đưa vào use case cơ bản các nội dung xử lý đặc biệt đó. Tuy nhiên, điều này có thể dẫn đến sự phức tạp với nhiều luận lý riêng biệt và sẽ làm giảm vai trò của dòng chính.

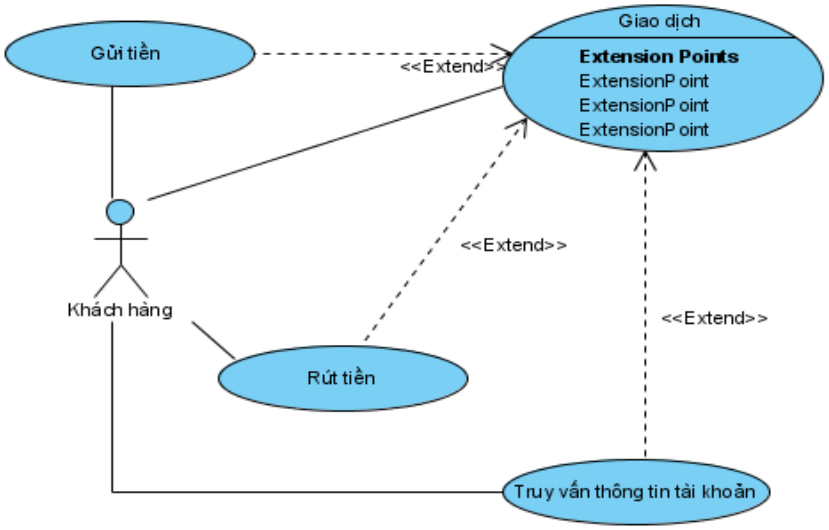
Để giải quyết vấn đề này chúng ta có thể sử dụng quan hệ <<extend>>. Ở đây chúng ta gom các xử lý cơ bản hoặc bình thường vào trong một use case (cơ bản). Các xử lý đặc biệt vào những use case (chuyên biệt) khác. Rồi tạo một liên kết <<extend>> giữa use case cơ bản tới các use case chuyên biệt để khai báo rằng: ngoài xử lý dòng chính (cơ bản), use case cơ bản có mở rộng đến các tình huống xử lý đặc biệt được giải quyết trong các use case chuyên biệt.



Ví dụ:



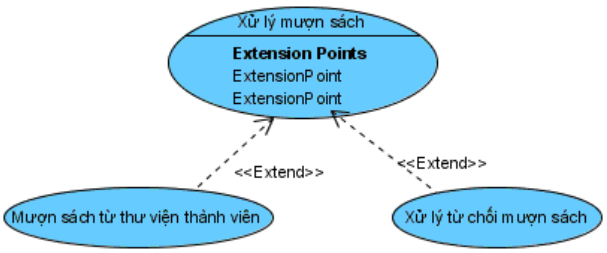
Tạo một use case tổng quát có tên là Giao dịch của các use case Rút tiền, Gửi tiền và Truy vấn thông tin tài khoản. Tạo các liên kết <<extend>> từ use case Giao dịch đến các use case này. Như vậy, một rút tiền, hoặc gửi tiền, hoặc truy vấn thông tin tài khoản là một loại giao dịch mà khách hàng có thể sử dụng trên máy ATM. Có nghĩa rằng, các xử lý trong use case Giao dịch sẽ cung cấp một dòng chung và khi khách hàng chọn một loại giao dịch đặc biệt nào đó thì use case này sẽ mở rộng việc giải quyết thông qua các use case chuyên biệt.



Giao dịch: khách hàng tương tác với hệ thống bắt đầu bằng việc đăng nhập hệ thống. Sau khi đăng nhập, khách hàng có thể thực hiện các giao dịch. Sau đây là các bước:

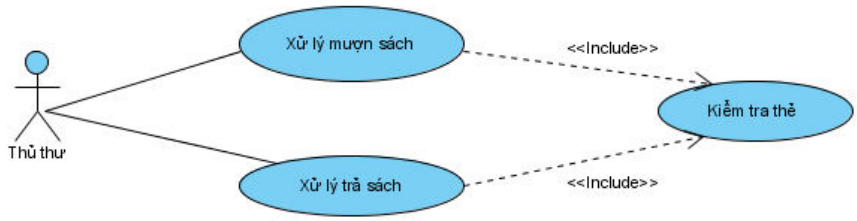
- Đưa thẻ vào máy
- Thực hiện đăng nhập
- Yêu cầu loại giao dịch
- Nhập loại giao dịch
- Thực hiện giao dịch
- Đẩy thẻ ra
- Yêu cầu lấy thẻ
- Lấy thẻ

Trong hệ thống quản lý thư viện, use case Mượn sách ngoài dòng sự kiện chính còn có các dòng phụ. Dòng phụ này sẽ được kích hoạt để giải quyết vấn đề khi một độc giả đến mượn tài liệu nhưng không có trong thư viện và thư viện sẽ mượn tài liệu đó từ những thư viện khác có liên kết. Hoặc do đọc giả không thoả các điều kiện để được mượn (mượn sách quá hạn chưa trả của lần mượn trước). Do đó, chúng ta tách dòng phụ này và use case "Mượn sách từ thư viện thành viên" và " "và tạo một liên kết <<extend>> từ use case này đến use case Xử lý mượn sách.



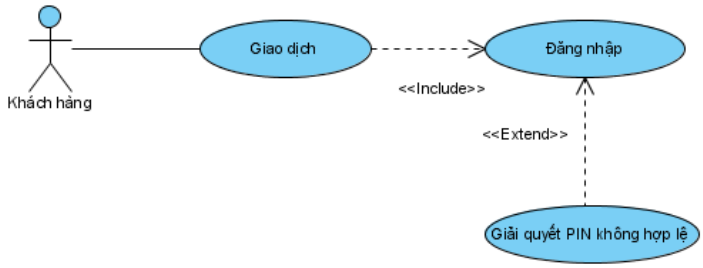
b) Quan hệ bao gồm <<include>>: được thành lập khi chúng ta có các use case mà tìm thấy một vài use case có những dòng hoạt động chung, và để tránh mô tả dòng hoạt động chung đó lặp lại trên những use case này, chúng ta có thể tách những dòng hoạt động chung đó ra thành một use case. Use case mới này có thể sử dụng bởi những use case khác. Quan hệ giữa những use case với

use case được trích ra này gọi là quan hệ <<include>>. Quan hệ sử dụng giúp chúng ta tránh sự trùng lắp bằng cách cho phép một use case có thể được chia sẽ.



Trong ví dụ trên, use case *mượn sách* và *trả sách* đều phải thực hiện công việc kiểm tra thẻ thư viện của đọc giả, do đó chúng ta phát sinh một use case mới là *kiểm tra thẻ* bằng cách trích ra hoạt động kiểm tra thẻ thư viện từ hai use case trên và tạo một liên kết <<include>> tới use case từ hai use case đó tới use case mới. Các use case *Xử lý mượn sách* và *Xử lý trả sách* đều thừa hưởng tất cả hoạt động của use case của use case *kiểm tra thẻ*.

Trong hệ thống ATM, use case Giao dịch sẽ có mối liên kết <<include>> với use case đăng nhập.



Đăng nhập: khách hàng nhập vào mã số PIN gồm bốn ký số. Nếu mã số PIN hợp lệ, tài khoản của khách hàng sẽ sẵn sàng cho các giao dịch. Các bước như sau:

- Yêu cầu password
- Nhập password
- Kiểm tra password

Giải quyết PIN không hợp lệ: nếu mã số PIN không hợp lệ, hệ thống sẽ hiển thị một thông báo tới khách hàng.

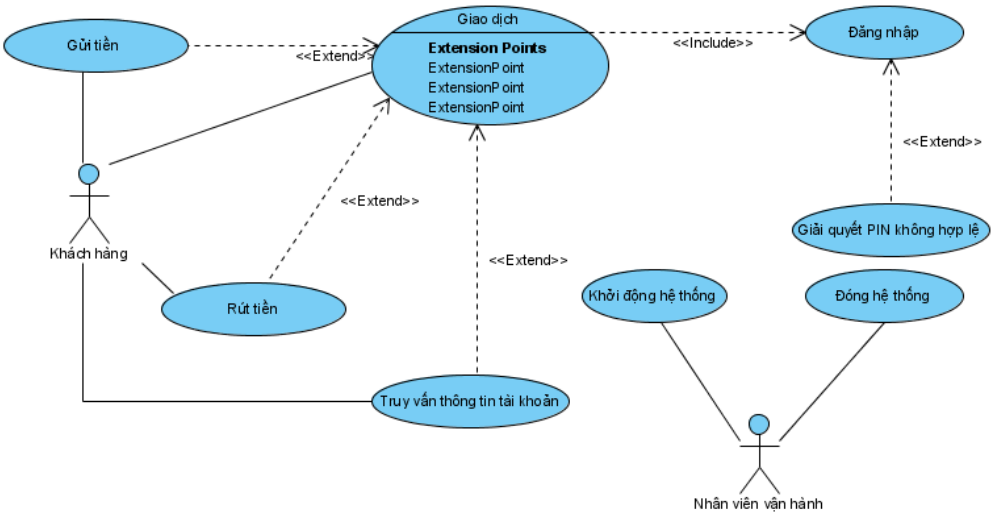
Sự giống nhau giữa liên kết <<extend>> và liên kết <<include>> là tất cả đều được xem như là một loại kế thừa. Khi chúng ta muốn chia sẽ một số hoạt động chung trong nhiều use case, dùng liên kết <<include>> bằng cách trích các hoạt động chia sẽ đó thành một use case mới. Khi chúng ta muốn thêm vào một ít khác biệt cho một use case để mô tả một tình huống đặc biệt trong một tình huống chung, chúng ta sẽ tạo một use case mới có liên kết <<extend>> với use case chung đó.

Dựa vào các liên kết được thiết lập cho các use case chúng ta phân use case thành hai loại:

- *Use case trừu tượng:* là use case chưa hoàn hảo nghĩa là không tương tác với bất kỳ một tác nhân nào mà được sử dụng bởi một use case khác. Use case trừu tượng cũng có thể

có liên kết <<extend>> hoặc liên kết <<include>> trong những mức độ khác. Ví dụ: các use case *Kiểm tra thẻ*, *Xử lý từ chối mượn sách*,… là các use case trừu tượng.

- *Use case cụ thể:* là use case có tương tác trực tiếp với một tác nhân. Ví dụ: các use case *Xử lý mượn sách*, *Xử lý trả sách*, hoặc *Khởi động máy*, *Đóng máy*,….

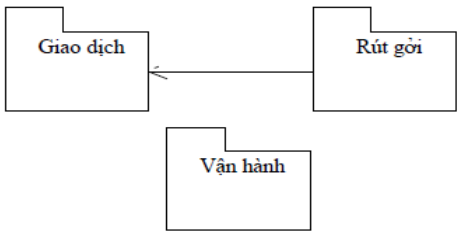


Hình 3.15 Mô hình Use case của hệ thống ATM

3.4.4 Phân chia các use case thành các gói (package)

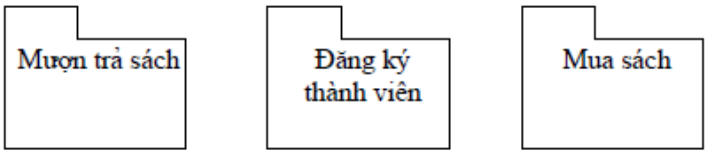
Mỗi use case minh hoạ một kịch bản trong hệ thống. Khi gặp những hệ thống tương đối phức tạp thì chúng ta nên thu hẹp tiêu điểm của các kịch bản trong hệ thống bằng cách phân chia thành các gói. Mỗi gói phản ánh một phạm vi của hệ thống mà chúng ta chỉ muốn quản lý nó khi chúng ta truy cập gói đó.

Ví dụ, có thể chia các se case của hệ thống máy ATM thành ba gói: Giao dịch, Rút gửi và Vận hành



Trong đó, gói Giao dịch gồm các use case: Giao dịch, Đăng nhập, Giải quyết PIN không hợp lệ; gói Rút gởi gồm các use case: Gởi tiền, Rút tiền, Truy vấn thông tin tài khoản; và gói Vận hành gồm các use case: Khởi động hệ thống, Đóng hệ thống.

Hệ thống quản lý thư viện được chia thành ba gói như sau: Mượn trả sách, đăng ký thành viên, và Mua sách.

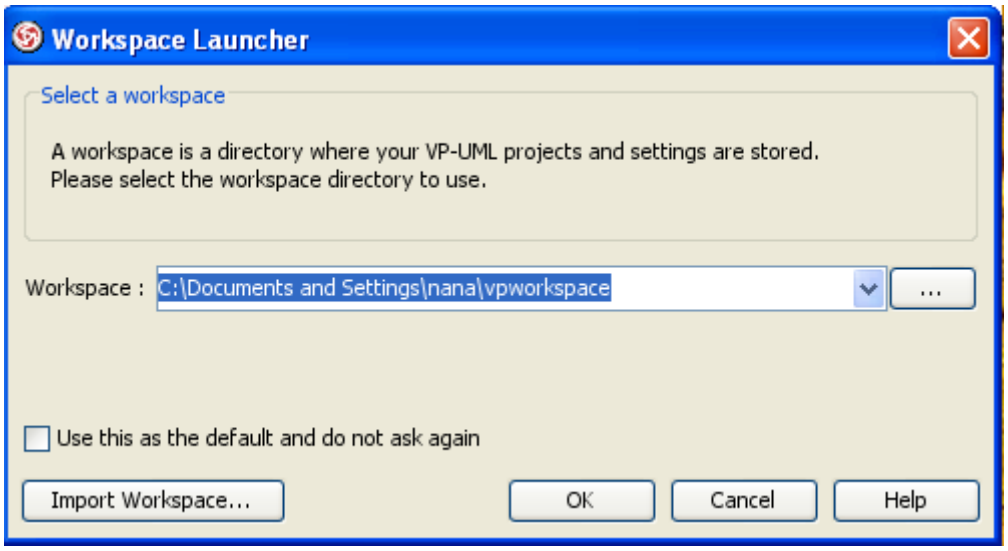


Trong đó, gói Mượn trả sách gồm các use case: Xử lý mượn sách, Xử lý trả sách, Kiểm tra thẻ, Mượn sách từ thư viện thành viên, Xử lý từ chối mượn sách; gói Đăng ký thành viên gồm use case: Đăng ký thành viên; gói mua sách gồm use case: Mua sách.

3.5 Tạo lập biểu đồ use case trong Visual Paradigm (VP)

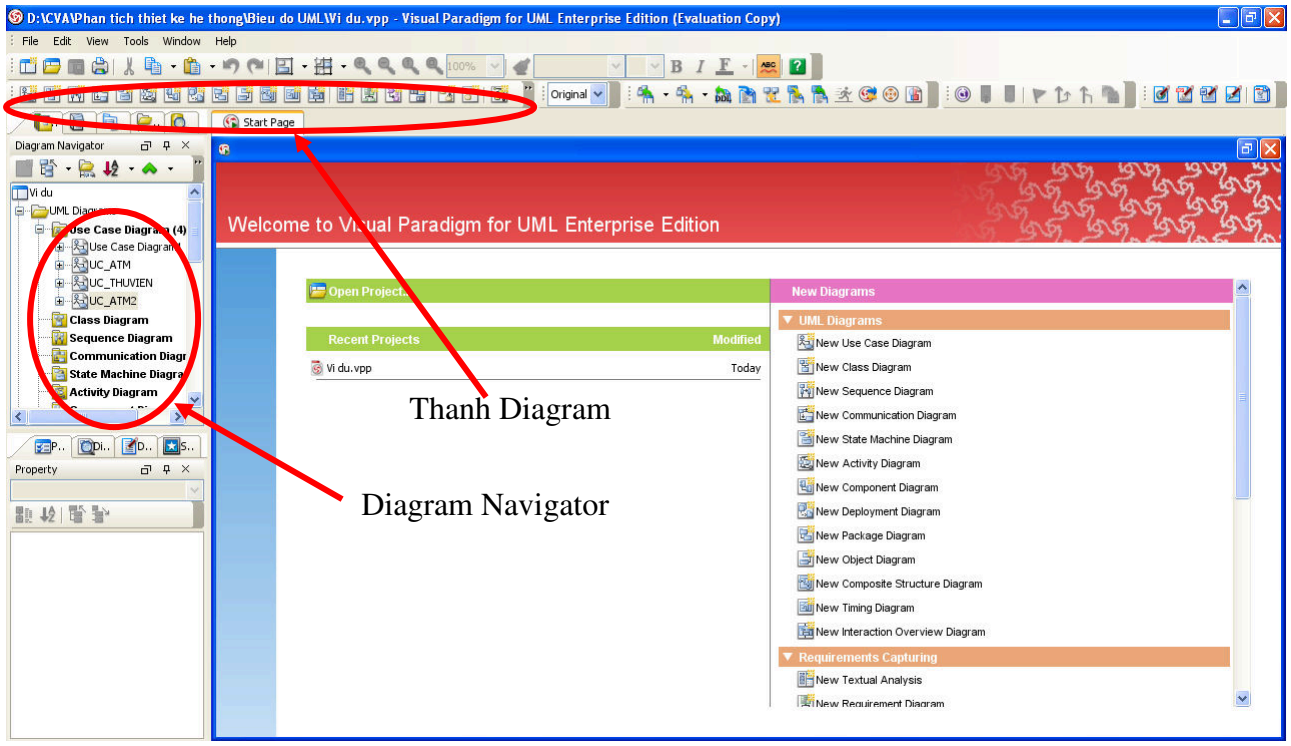
Để vẽ biểu đồ use case trong Visual Paradigm chúng ta thực hiện như sau:

- Trước hết phải khởi động Visual Paradigm: vào start → program → Visual Paradigm → Visual Paradigm for UML 7.0 Enterprise Edition. Hoặc double-click vào biểu tượng Visual Paradigm for UML 7.0 Enterprise Edition ở màn hình desktop. Chương trình khởi động hộp thoại đầu tiên xuất hiện cho phép bạn chọn thư mục workspace (thư mục làm việc của VP):



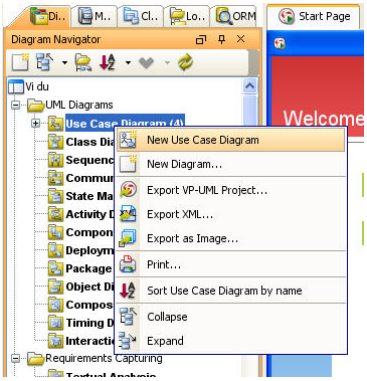
Hình 3.16 Hộp thoại chono workspace

Màn hình chính khi khởi động VP như sau:



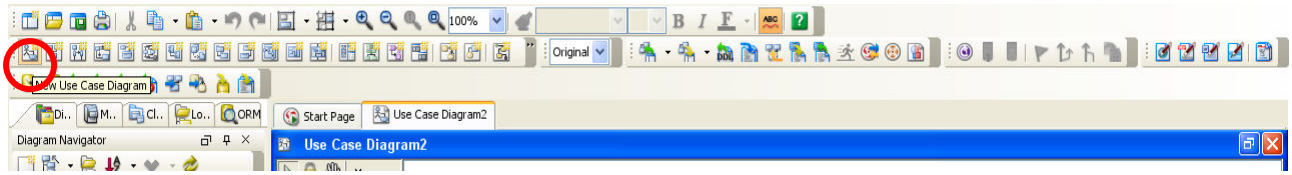
Như vậy, để tạo một biểu đồ Use case trong Visual Paradigm chúng ta thực hiện:

Cách 1: Click chuột phải vào Use case diagram ở mục Diagram Navigator, chọn New Use Case Diagram hoặc chọn New Diagram rồi chọn Use case diagram → đặt tên cho biểu đồ Use case, chẳng hạn: UC_ATM.



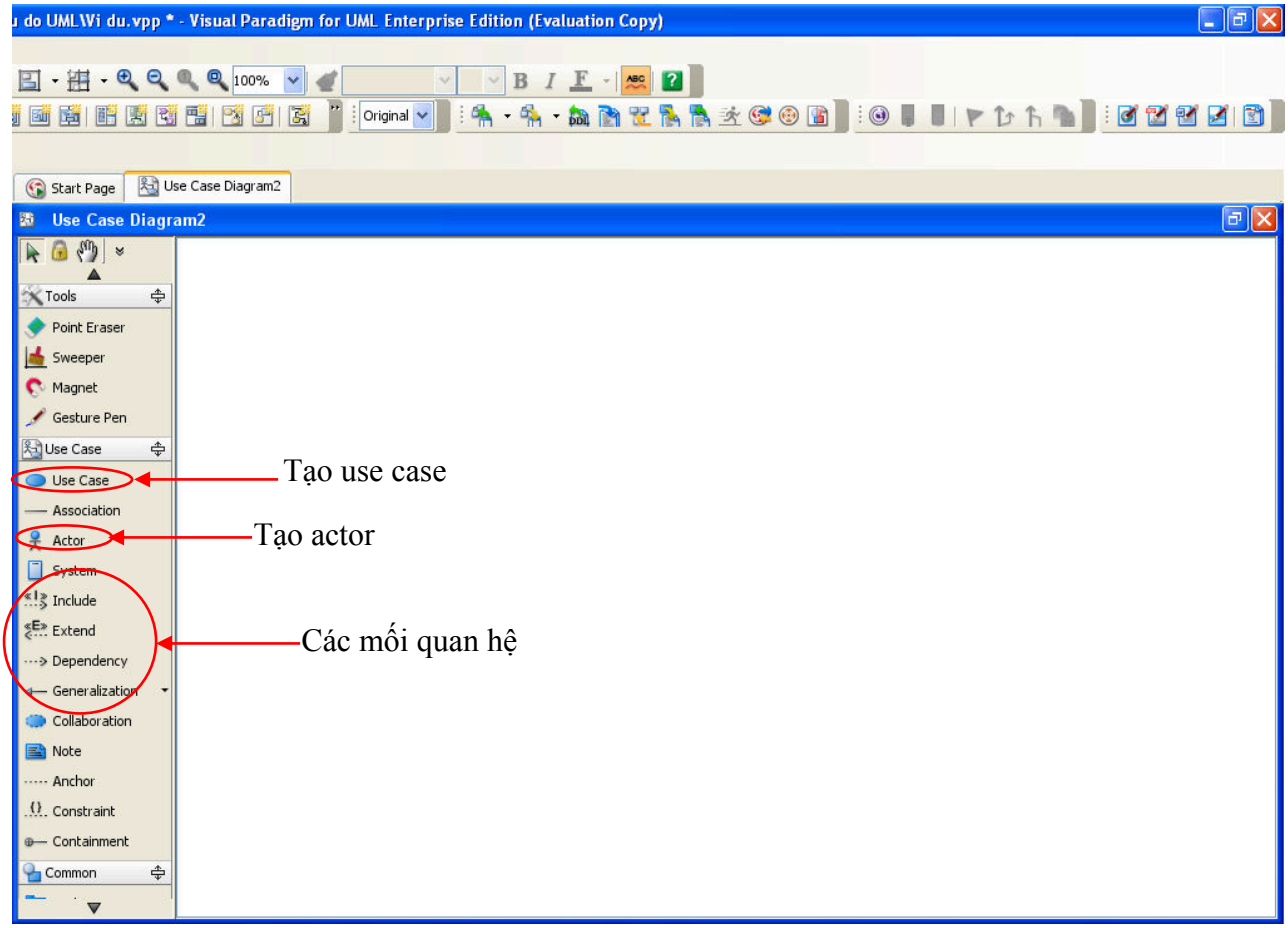
Hình 3.17 Tạo biểu đồ Use case trong VP

Cách 2: Chọn biểu tượng New Use Case Diagram tại thanh Diagram (xem hình 3.18) → đặt tên cho biểu đồ Use case:

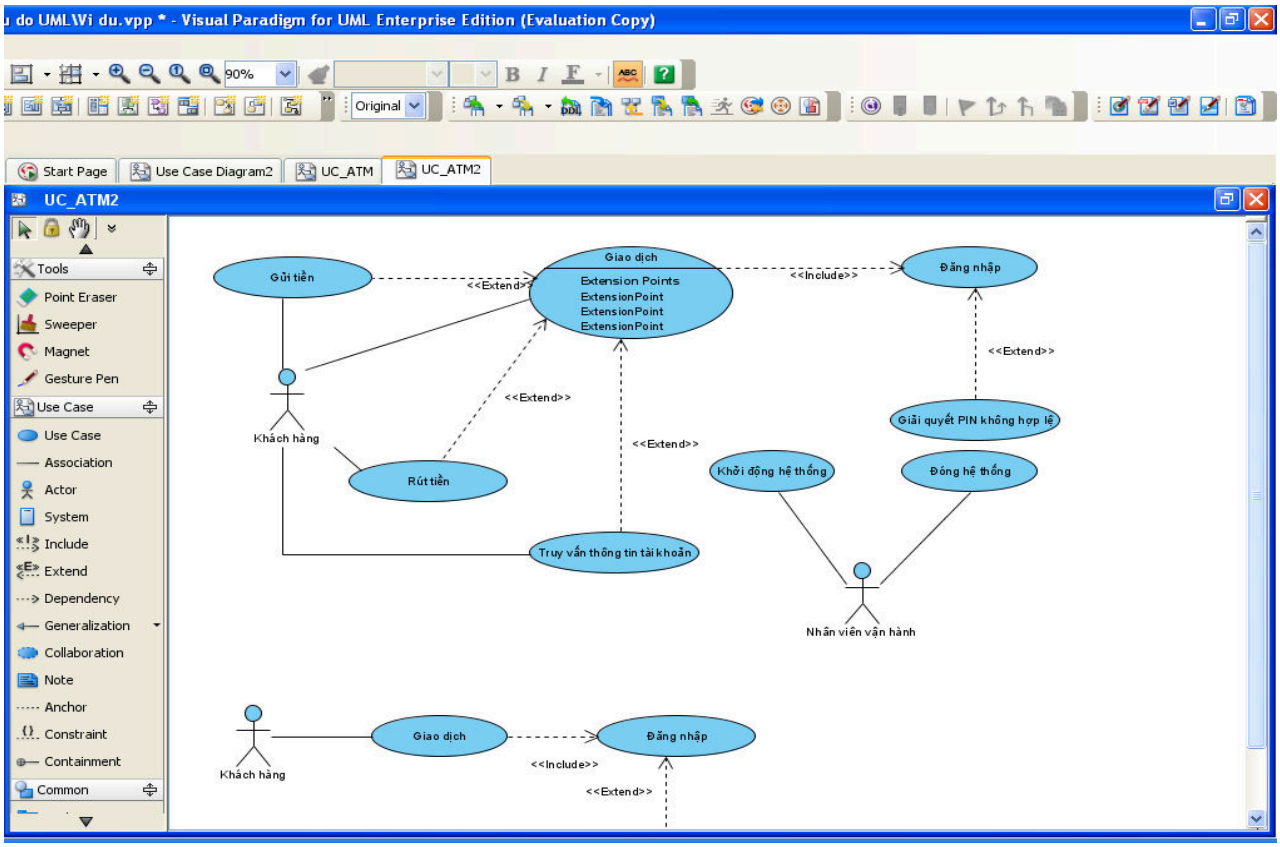


Hình 3.18 Tạo biểu đồ Use case trong VP cách 2

Sau đó chúng ta có thể tạo các Actor, Use case, các mỗi quan hệ giữa chúng như mong muốn:



Hình 3.19 Tạo các thành phần trong biểu đồ use case



Hình 3.20 Biểu đồ use case hoạt động hệ thống ATM

4. Mô hình hóa nghiệp vụ (Business modeling)

Học xong phân này sinh viên có thể hiểu:

- Như thế nào là mô hình hoá nghiệp vụ, mục tiêu và quy trình của mô hình hoá nghiệp vụ
- Các hoạt động trong phân tích, thiết kế qui trình nghiệp vụ
- Áp dụng UML vào mô hình hoá nghiệp vụ. Đặc biệt, sử dụng sơ đồ use case biểu diễn nội dung của hệ thống nghiệp vụ trong giai đoạn phân tích. Sử dụng sơ đồ đối tượng trong việc thiết kế hiện thực hoá nghiệp vụ.
- Chuyển dịch các yêu cầu tự động hoá từ hệ thống nghiệp vụ sang hệ thống phần mềm tự động hoá

Giới thiệu:

Mô hình hóa nghiệp vụ là một kỹ thuật để tìm hiểu quy trình nghiệp vụ của một tổ chức. Mô hình nghiệp vụ xác định các quy trình nghiệp vụ nào được hỗ trợ bởi hệ thống. Tóm lại, song song với quá trình khảo sát tìm hiểu về vấn đề hệ thống thì cách tiếp cận nghiệp vụ là phương pháp có hệ thống nhất để nắm bắt các yêu cầu của các ứng dụng nghiệp vụ.

Khi những hệ thống ngày càng phức tạp, việc mô hình hóa trực quan và cách vận dụng các kỹ thuật mô hình hóa ngày càng trở nên quan trọng hơn. Có nhiều nhân tố bổ sung cho sự thành công của một dự án, nhưng việc có một tiêu chuẩn ngôn ngữ mô hình hóa chặt chẽ là nhân tố quan trọng nhất. Một trong những mục đích đầu tiên của mô hình hoá nghiệp vụ là tạo ra các "đối tượng" (mô hình) nhằm để dễ hiểu hơn và dể có thể thiết kế những chương trình máy tính bằng cách thông qua hiện tượng thế giới thực như: người, nguyên liệu làm việc và cách thức chúng thực hiện những nhiệm vụ của họ. Như vậy, việc mô hình hóa nghiệp vụ là lập mô hình những tổ chức thế giới thực.

Phạm vi ảnh hưởng của việc mô hình hóa nghiệp vụ có thể biến đổi tuỳ theo nhu cầu và hệ thống nghiệp vụ cụ thể. Có thể đơn giản chúng ta chỉ nhằm vào việc tăng năng suất bằng cách cải tiến những quy trình đã tồn tại, hoặc là đang tạo ra những sự cải tiến có ảnh hưởng lớn bằng cách thay đổi đáng kể những qui trình nghiệp vụ dựa trên sự phân tích kỹ lưỡng các mục tiêu và các khách hàng của tổ chức. Cho dù là bất kỳ trường hợp nào, những hệ thống thông tin hỗ trợ cho hệ thống nghiệp vụ đều bị ảnh hưởng.

Tại sao phải mô hình hoá nghiệp vụ?

Trong quá trình phát triển hệ thống phần mềm một vấn đề tồn tại rất lớn là đội ngũ phát triển hệ thống thường hiếm khi có một kiến thức hiểu biết đầy đủ về nghiệp vụ của tổ chức mà chính họ là người xây dựng hệ thống phầm mềm thực hiện trong môi trường nghiệp vụ đó. Trong khi đó, người sử dụng phần mềm chính là các đối tượng xử lý nghiệp vụ thường không am hiểu tường tận về các công nghệ và các kỹ thuật của phần mềm nhằm chọn lựa và áp dụng nó một cách phù hợp và hiệu quả với nhu cầu của mình. Khoảng cách này là một vấn đề rất lớn dẫn đến nhiều sự thất bại của quá trình tin học hoá hệ thống. Do đó, làm thế nào để các đối tượng này có thể hiểu và thống nhất được tốt nhất về cách giải quyết hệ thống trong quá trình tin học hoá. Những mô hình nghiệp vụ đưa ra các cách thức diễn tả những qui trình nghiệp vụ dưới dạng những đối tượng và hành động tương tác giữa chúng. Nếu không mô hình hóa nghiệp vụ thì ta có thể gặp nhiều rủi ro do những người phát triển không có thông tin đầy đủ về cách thức mà nghiệp vụ được thực hiện. Họ chỉ làm

những gì mà họ hiểu rõ, như là thiết kế và tạo ra phần mềm, mà không quan tâm đến những qui trình nghiệp vụ. Điều này gây ra một sự lãng phí do trước đó đã xây dựng các qui trình nghiệp vụ tốn kém. Rủi ro do những hệ thống được xây dựng không hỗ trợ các nhu cầu thực sự của tổ chức cũng có thể xảy ra rất cao.

Việc hiểu rõ những qui trình nghiệp vụ là quan trọng để có thể xây dựng những hệ thống đúng. Việc mô hình hóa nghiệp vụ có thể có mục tiêu chính là sự phát triển hệ thống, trong đó công việc thực sự là xác định đúng các yêu cầu hệ thống.

Cơ sở để xây dựng hệ thống là sử dụng những vai trò và trách nhiệm của con người cũng như định nghĩa những gì được xử lý bởi nghiệp vụ. Điều này được thể hiện trong một mô hình đối tượng nghiệp vụ, mà qua đó có thể thấy rõ các vai trò đối tượng sẽ được làm rõ.

Với sự xuất hiện của e-business, mô hình hóa nghiệp vụ cũng trở nên quan trọng hơn. Các ứng dụng e-business được xây dựng để tự động hóa những qui trình nghiệp vụ. Một khi xác định được các mô hình nghiệp vụ, chúng ta cần phải thiết lập những mối quan hệ giữa các use case hệ thống và những mô hình nghiệp vụ. Điều này sẽ cho phép các nhà phân tích được thông báo khi có những thay đổi ở trong hệ thống. Tóm lại, mục đích của mô hình hóa nghiệp vụ là:

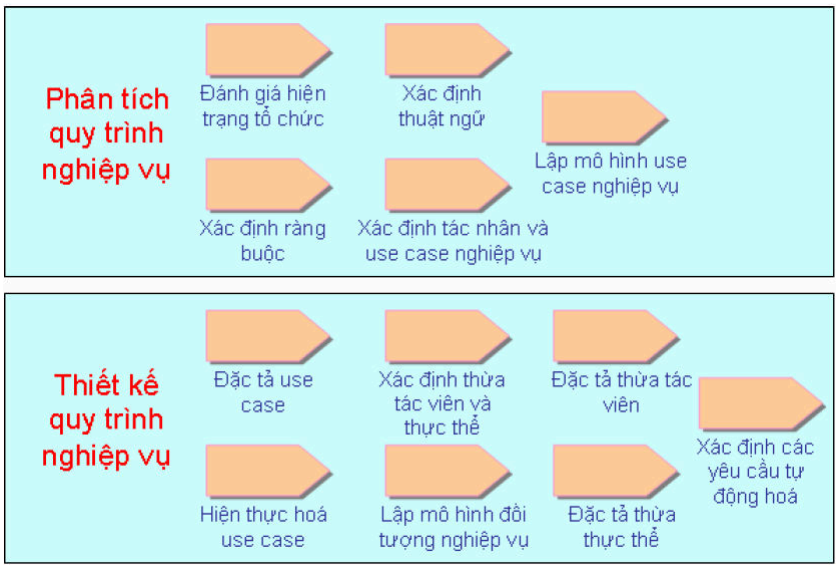
- Hiểu được cấu trúc và các hoạt động của tổ chức được triển khai hệ thống.
- Hiểu được các vấn đề hiện tại trong tổ chức và xác định các vấn đề cần cải tiến.
- Bảo đảm rằng các khách hàng, người dùng cuối, và các nhà phát triển có sự hiểu biết chung về tổ chức.
- Thiết lập các yêu cầu hệ thống nhằm hỗ trợ tổ chức.

Để đạt được những mục đích trên, luồng công việc mô hình hóa nghiệp vụ mô tả một bức tranh tổng quát về tổ chức, từ đó xác định các qui trình (process), các vai trò (role), và các trách nhiệm của tổ chức này trong mô hình use-case nghiệp vụ (business use-case model) và mô hình đối tượng nghiệp vụ (business object model).

Luồng công việc trong mô hình hoá nghiệp vụ

Hệ thống nghiệp vụ là một loại hệ thống do đó quá trình tiếp cận mô hình hoá cũng tuân theo quy trình chung qua nhiều giai đoạn. Tài liệu này sẽ giới thiệu hai giai đoạn mô hình hoá nghiệp vụ sử dụng UML.

- ***Phân tích quy trình nghiệp vụ:*** đây là giai đoạn đầu tiên của mô hình hóa nghiệp vụ giúp cho các nhà quản lý dự án hiểu rõ tình trạng tổ chức hiện tại và hoạt động của tổ chức, nắm bắt yêu cầu của người dùng và khách hàng từ đó phác thảo và giới hạn hệ thống phát triển.
- ***Thiết kế quy trình nghiệp vụ:*** đây là giai đoạn đặc tả chi tiết một bộ phận của tổ chức bằng cách mô tả luồng công việc của một hay nhiều nghiệp vụ, xác định các dối tượng làm việc và các thực thể nghiệp vụ trong biểu diễn hiện thực hóa nghiệp vụ và sắp xếp các hành vi của nghiệp vụ đồng thời xác định các trách nhiệm, thao tác, thuộc tính và mối quan hệ giữa các người làm việc và các thực thể trong nghiệp vụ.



Phân tích quy trình nghiệp vụ

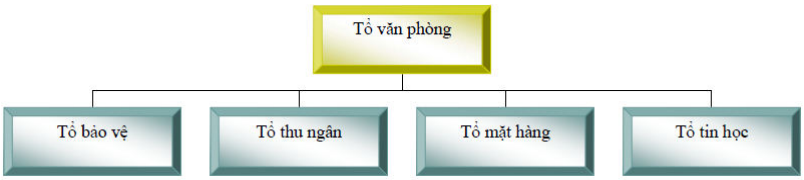
Các công việc của quy trình phân tích nghiệp vụ bao gồm:

- Đánh giá và nắm bắt thông tin về tổ chức.
- Xác định các đối tượng liên quan (stakeholder) và khách hàng của hệ thống.
- Định nghĩa phạm vi của việc mô hình hóa nghiệp vụ.
- Tán thành những tiềm năng cải tiến và các mục tiêu mới của tổ chức.
- Mô tả những mục tiêu chính của tổ chức.

Nắm bắt thông tin về tổ chức

Để thiết kế hệ thống phù hợp với nhu cầu của khách hàng thì việc hiểu rõ thông tin về cấu trúc tổ chức sẽ được triển khai hệ thống là điều quan trọng. Tất cả các thành viên trong dự án đều cần phải nắm bắt rõ ràng các thông tin này. Chúng ta có thể mô tả ngắn gọn các bộ phận cấu thành tổ chức và mối quan hệ giữa các bộ phận này thông qua các sơ đồ tổ chức và trình bày ngắn gọn các thông tin liên quan.

Ví dụ: Sơ đồ tổ chức của siêu thị Co-Op Mart Cống Quỳnh



Tổ văn phòng: Gồm 1 Giám Đốc và 2 phó Giám Đốc có nhiệm vụ điều phối toàn bộ hoạt động của siêu thị. Tổ phải nắm được tình hình mua bán, doanh thu của siêu thị để báo cáo lại cho ban giám đốc hợp tác xã Sài Gòn Co-Op. Việc báo cáo được thực hiện hàng tháng, hàng quý hoặc cũng có khi báo cáo đột xuất theo yêu cầu.

Tổ bảo vệ: Kiểm tra, bảo vệ an ninh của Siêu Thị, ghi nhận Hàng Hóa đổi lại của khách hàng.

Tổ thu ngân: Thực hiện việc bán hàng và lập hóa đơn cho khách hàng đồng thời ghi nhận lại số hàng hoá bán được của mỗi loại để báo cáo cho tổ quản lý sau mỗi ca làm việc.

Tổ mặt hàng: Nhiệm vụ của tổ là kiểm tra chất lượng hàng hoá và nắm tình trạng hàng hoá của siêu thị, đảm bảo hàng hoá luôn ở trong tình trạng tốt nhất khi đến tay khách hàng. Khi phát hiện hàng hư hỏng phải kịp thời báo ngay cho tổ văn phòng để có biện pháp giải quyết và điều phối hàng. Ngoài ra, thường xuyên thống kê số lượng hàng tồn trên quầy, báo cáo về tổ văn phòng

Tổ tin học: Thực hiện việc nhập liệu, kết xuất các báo cáo cần thiết phục vụ cho tổ Văn Phòng.

Xác định các đối tượng có liên quan và khách hàng

Việc tin học hóa công tác quản lý trong một tổ chức tạo ra những biến đổi, phần do việc tự động hóa công việc hành chính, phần do cấu trúc lại tổ chức và sự vận hành của hệ thống. Những thay đổi quan trọng phát sinh từ việc thiết kế hệ thống thông tin, chủ yếu tập trung vào việc tin học hóa, nếu không biết thực hiện dần dần sẽ có nguy cơ lớn chạm đến con người trong tổ chức dẫn đến thất bại ngay từ đầu. Bản thân công việc thiết kế hệ thống thông tin đã được thực hiện dưới nhiều góc độ khác nhau, thậm chí kể cả tâm lý, với những đặc thù riêng biệt và có độ phức tạp cao. Chính vì thế, cần phải tìm hiểu những đối tượng có liên quan và khách hàng của hệ thống là ai, đồng thời nắm bắt được nhu cầu của họ.

Nếu đánh giá tình trạng của tổ chức, ta nên xác định những đối tượng có liên quan trong nghiệp vụ. Nhưng khi xác định các mục tiêu của hệ thống thì cần xác định những đối tượng liên quan trong phạm vi dự án và điều đó cũng phụ thuộc vào phạm vi mô hình hóa nghiệp vụ, cũng như những phạm vi nào cần xác định đối với việc mô hình hóa.

Ví dụ: thể hiện thông tin các đối tượng liên quan và người dùng hệ thống qua bảng sau trong hệ thống siêu thị như sau:

Bảng mô tả tóm tắt các đối tượng liên quan:

Tên	Đại diện	Vai trò
Người quản lý	Giám đốc, người quản lý siêu thị	Theo dõi tiến trình phát triển của dự án và theo dõi tình hình hoạt động của siêu thị.
Nhân viên bán hàng	Người nhập các thông tin trong hệ thống.	Chịu trách nhiệm trong khâu bán hàng ở siêu thị, duy trì hoạt động của siêu thị.

Bảng mô tả tóm tắt các người dùng:

Tên	Mô tả	Đối tượng liên quan
Người quản lý	Đáp ứng các nhu cầu quản lý siêu thị như hàng hóa, khách hàng, doanh số.	Người quản lý
Nhân viên bán hàng	Đảm bảo rằng hệ thống sẽ đáp ứng các nhu cầu của công việc bán hàng.	Nhân viên bán hàng
Khách hàng	Đáp ứng nhu cầu tra cứu thông tin về hàng hóa có trong siêu thị.	

Nắm bắt nhu cầu của các đối tượng liên quan

Nhiệm vụ trao cho họ là những công việc thực sự như là xử lý thông tin, chứ không chỉ đơn thuần là thao tác với máy tính và các thiết bị, vì vậy ta không được phép bỏ qua các ý kiến, nhu cầu của họ đối với hệ thống tin học tương lai. Hãy liệt kê danh sách các nhu cầu chính bằng cách điền đủ thông tin vào bảng sau:

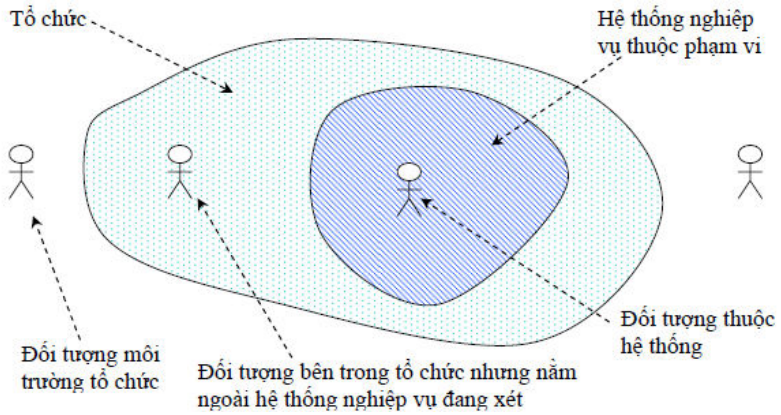
Tên đối tượng liên quan/ khách hàng	Độ ưu tiên	Nhu cầu	Giải pháp hiện hành	Giải pháp đề xuất

Ví dụ:

Tên đối tượng liên quan/ khách hàng	Độ ưu tiên	Nhu cầu	Giải pháp hiện hành	Giải pháp đề xuất
Người quản lý	Cao	Xem các báo cáo thống kê theo các yêu cầu khác nhau	Báo cáo thống kê doanh thu	Hiển thị báo cáo theo nhiều tiêu chí khác nhau, thông tin bố trí dễ nhìn và đơn giản nhưng đầy đủ.

Giới hạn hệ thống phát triển

Cần phải đạt được sự thỏa thuận về những thực thể chính nằm ngoài hệ thống với các đối tượng liên quan và với nhau. Trong trường hợp mô hình hóa nghiệp vụ để xác định các yêu cầu cho một hệ thống cụ thể, có thể có những phần trong tổ chức sẽ không bị ảnh hưởng bởi hệ thống này, những phần đó có thể được xem như các thực thể nằm bên ngoài.



Những ranh giới đặt ra cho hệ thống có thể khác rất nhiều so với những gì có thể được xem là ranh giới của tổ chức.

Nếu mục đích là xây dựng một hệ thống mới để hỗ trợ bán hàng, ta không cần quan tâm đến bất cứ việc gì trong kho hàng, nhưng cần xem nó như là một tác nhân bởi vì chúng ta cần phải làm rõ ranh giới giữa chúng. Trong ví dụ này, các thực thể bên trong tổ chức được xem như là bên ngoài hệ thống đang xét và được mô hình hóa thành tác nhân nghiệp vụ.

Nếu mục tiêu xây dựng hệ thống là nhằm nâng cao khả năng trao đổi thông tin với các đối tác hay các nhà cung cấp (ứng dụng business-to-business) như quản lý đặt hàng thì các đối tác hay các nhà cung cấp này của tổ chức được mô hình hóa cần phải được quan tâm. Trong trường hợp này, các thực thể bên ngoài tổ chức sẽ nằm trong tổ chức. Điều này chỉ xảy ra khi sự cộng tác giữa các bên ảnh hưởng sâu sắc đến phương thức hoạt động của nhau. Nhưng nếu sự ảnh hưởng này không quá lớn hay nghiêm trọng thì các đối tác được xem như là các thực thể bên ngoài và được mô hình hóa thành tác nhân nghiệp vụ.

Nếu mục đích là xây dựng các ứng dụng chung, tùy biến (như là ứng dụng kế toán tài chính) thì chúng ta cần trình bày cách thức khách hàng sẽ sử dụng sản phẩm cuối như thế nào và nó là một thực thể trừu tượng.

Xác định và trình bày các vấn đề của hệ thống

Trong quá trình khảo sát hệ thống, có thể thu thập rất nhiều nhu cầu cần thay đổi của khách hàng. Đây được xem là các vấn đề của khách hàng cần chúng ta giải quyết trong hệ thống tương lai. Vì thế ta cần phải hiểu và trình bày rõ ràng các vấn đề này để mọi thành viên trong dự án nắm bắt tốt. Có thể áp dụng mẫu như sau:

Vấn đề	mô tả vấn đề
Đối tượng chịu tác động	các stakeholder bị ảnh hưởng bởi vấn đề
Ảnh hưởng của vấn đề	tác động ảnh hưởng của vấn đề
Một giải pháp thành công	liệt kê một vài lợi ích của một giải pháp thành công

Ví dụ:

Vấn đề	Cơ sở dữ liệu của các khách hàng thân thiết được lưu trữ ở nhiều nơi và không có sự đồng bộ .
Đối tượng chịu tác động	Khách hàng, người quản lý
Ảnh hưởng của vấn đề	Dịch vụ khách hàng thân thiết chỉ thiết lập được ở từng siêu thị của hệ thống Co-Op. Điều này là bất hợp lý, làm rắc rối trong việc nâng cao dịch vụ khách hàng, làm giảm khả năng cạnh tranh của siêu thị.
Một giải pháp thành công	Nhân viên có thể sử dụng chung một account cấp cho mỗi khách hàng được dùng ở tất cả siêu thị thuộc hệ thống Co-Op. Nâng cao khả năng chăm sóc khách hàng của siêu thị tốt hơn từ đó thu hút được khách hàng nhiều hơn, tăng doanh thu của siêu thị. Giúp người quản lý có thể làm tốt công tác quản lý khách hàng, theo dõi tình hình phục vụ khách hàng một cách dễ dàng.

Xác định những lãnh vực cần ưu tiên

Cần phải thảo luận và đạt được sự nhất trí về những lãnh vực cần được ưu tiên trong mô hình hóa nghiệp vụ. Sự thảo luận này có thể theo nhiều hướng khác nhau, tùy vào phạm vi của mô hình hóa nghiệp vụ.

Nếu mục đích mô hình hóa nghiệp vụ là tạo một mô hình hay để thực hiện sự cải tiến đơn giản, thì chỉ cần mô tả nghiệp vụ hiện tại. Khi đó, những lĩnh vực nào cần cải tiến phải xác định rõ.

Nếu mục đích là tạo một nghiệp vụ mới hay thay đổi hoàn toàn nghiệp vụ hiện tại, thì phạm vi mô hình hóa sẽ lớn hơn. Lúc này, công việc tái cấu trúc các use case nghiệp vụ của một nghiệp vụ đã tồn tại hay thêm các use case nghiệp vụ mới - để tái thiết kế nghiệp vụ (business reengineering) hay thiết kế mới nghiệp vụ (business creation) là cần thiết.

Ví dụ: Trong Hệ thống quản lý nghiệp vụ bán hàng tại siêu thị Co-Op Mart, việc mô hình hóa nghiệp vụ nhằm mục đích để cải tiến nghiệp vụ nên chúng ta chỉ cần xác định những nghiệp vụ cần cải tiến.

Để cải tiến nghiệp vụ, một số câu hỏi được đặt ra như sau:

- Cấu trúc của tổ chức có thể được cải tiến không? Đó là cách thức tổ chức nhân viên làm việc trong các qui trình nghiệp vụ. Ta có thể xây dựng các nhóm nhân viên có nhiều năng lực khác nhau để thực hiện những công việc chính, giảm số lượng người giữ vai trò một công việc dẫn đến giảm chi phí, giảm các sai sót và để cho các nhân viên có nhiều trách nhiệm hơn, khi đó họ không phải chờ người khác quyết định.
- Có công việc nào không cần thiết không? Xác định những công việc không cần thiết trong tổ chức như: viết báo cáo mà không có ai đọc, lưu trữ những thông tin không bao giờ được sử dụng ....
- Có công việc nào giống hoặc tương tự nhau được thực hiện ở những nơi khác nhau không? Như công việc được làm lại, do người ta không tin tưởng vào kết quả hoặc không biết trước đó đã làm gì hay các kết quả được kiểm tra và chấp thuận nhiều lần.
- Có vấn đề nào về thời gian và chi phí không? Thời gian thực hiện có thế là một vấn đề thậm chí nếu mỗi thứ đều hoạt động tốt. Để xác định công việc nào có thời gian quá cấp bách, hãy phân tích mỗi use case nghiệp vụ sử dụng thời gian. Xác định mối quan hệ giữa thời gian sản xuất, thời gian chờ, và thời gian truyền.

Kết quả chính của hoạt động này là một bản mô tả tầm nhìn nghiệp vụ (Business Vision), trong đó mô tả tầm nhìn (vision) của hệ thống tương lai. Business Vision xác định một tập hợp các mục tiêu của công việc mô hình hóa nghiệp vụ, cung cấp đầu vào cho qui trình kiểm chứng dự án, có liên quan mật thiết với trường hợp nghiệp vụ (Business Case), cũng như tài liệu Vision của công nghệ phần mềm. Nó được sử dụng bởi các nhà quản lý, những người có thẩm quyền về ngân quỹ, những người làm việc trong mô hình hóa nghiệp vụ, và các nhà phát triển nói chung.

Sưu liệu này phải bảo đảm rằng:

- Nó phải được cập nhật và được phân phối.
- Nó phải giải quyết được đầu vào từ tất cả các đối tượng có liên quan.

Xác định và mô tả các thuật ngữ nghiệp vụ

Một trong những khó khăn của dự án phần mềm quản lý hệ thống thông tin là sự bất đồng ngôn ngữ diễn đạt vấn đề giữa khách hàng và quản trị dự án hay giữa các thành viên tham gia trong dự án. Điều này gây ra các khó khăn trong việc tìm hiểu hay hiểu lầm các quy trình nghiệp vụ trong tổ chức của các thành viên trong dự án. Nhằm tránh những rủi ro này, chúng ta cần phải xác định và thống nhất những thuật ngữ trong các quy trình nghiệp vụ của tổ chức.

Sưu liệu thuật ngữ này chỉ thực sự hữu ích khi cần phân biệt rõ những từ chuyên môn của nghiệp vụ được dùng trong việc mô hình hóa nghiệp vụ với các từ chuyên môn của nghiệp vụ được dùng trong quá trình phát triển phần mềm.

Thông thường, mỗi thuật ngữ được mô tả như một danh từ với định nghĩa của nó. Tất cả các bên tham gia phải thống nhất với nhau về định nghĩa của các thuật ngữ này.

Ví dụ: Bảng thuật ngữ của hệ thống quản lý siêu thị như sau

Thuật ngữ	Diễn giải
Người quản lý	Người quản lý siêu thị và cũng là người quản trị hệ thống. Nguoiquanly được gọi chung cho những người được cấp quyền là "Quản lý", có thể bao gồm giám đốc, phó giám đốc, kế toán, nhân viên tin học, ...
Nhân viên bán hàng	Là nhân viên làm việc trong siêu thị. Nhân viên bán hàng, đứng ở quầy thu tiền và tính tiền cho khách hàng. Thông qua các mã vạch quản lý trên từng mặt hàng được nhân viên bán hàng nhập vào hệ thống thông qua một đầu đọc mã vạch.
Tên đăng nhập	Tên đăng nhập của người sử dụng hệ thống. Mỗi nhân viên khi vào làm trong siêu thị sẽ được đăng ký một tên đăng nhập nhằm để quản lý. Khi đăng nhập vào hệ thống, nhân viên đó sẽ sử dụng tên này để đăng nhập. Người quản lý chịu trách nhiệm quản lý tên đăng nhập của nhân viên. Tồn tại duy nhất.
Mật khẩu	Mật khẩu đăng nhập của người sử dụng hệ thống. Mỗi nhân viên khi sử dụng tên đăng nhập sẽ được đăng ký kèm theo một mật khẩu đăng nhập. Mỗi nhân viên chỉ được biết duy nhất một mật khẩu của mình. Mật khẩu có thể rỗng.
Quyền đăng nhập	Quyền đăng nhập vào hệ thống. Tùy theo quyền và chức vụ trong công ty, nhân viên có quyền đăng nhập tương ứng.
Khách hàng thân thiết	Khách hàng thân thiết của siêu thị hay khách hàng đăng ký tham gia chương trình khách hàng thân thiết của siêu thị.
Điểm thưởng	Số điểm của khách hàng thân thiết trong siêu thị được thưởng do mua vượt mức thanh toán của siêu thị.
Ngày cấp thẻ	Ngày cấp thẻ khách hàng thân thiết cho khách hàng khi họ đăng ký chương trình khách hàng thân thiết của siêu thị.
Hóa đơn thanh toán	Hóa đơn tính tiền của siêu thị khi khách hàng mua hàng tại siêu thị
Chủng loại hàng	Chủng loại hàng hóa trong siêu thị, được phân chia tương ứng theo quầy hàng trưng bày trong siêu thị.
Loại hàng	Loại hàng trong siêu thị được phân chia theo tiêu chí công ty sản xuất, đơn vị tính....
Hàng hóa	Hàng hóa được bày bán trong siêu thị.
Hàng tồn	Số lượng hàng hóa còn lại trong siêu thị chưa bán được cho khách hàng.
Mức giảm	Tỉ lệ phần trăm giảm đối với khách hàng thân thiết
Thống kê doanh thu	Báo cáo thống kê tình hình kinh doanh của siêu thị theo tiêu chí nào đó như: hàng hóa, quý, khoảng thời gian....
Thống kê hàng hóa	Báo cáo thống kê số lượng hàng hóa của siêu thị theo tiêu chí nào đó như: hàng hóa, quý, khoảng thời gian....

Xác định actor và use case nghiệp vụ

Mục đích:

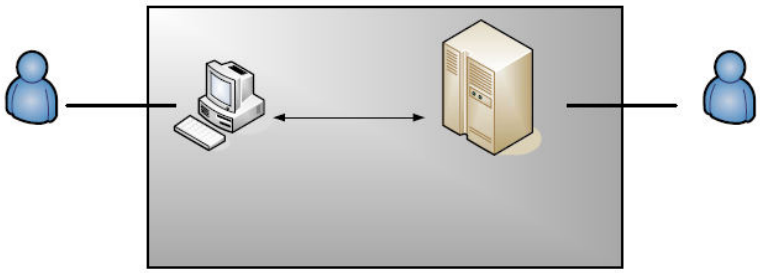
- Phác thảo các qui trình trong nghiệp vụ.
- Xác định ranh giới của nghiệp vụ cần được mô hình hóa.
- Xác định những gì sẽ tương tác với nghiệp vụ.
- Tạo ra các lược đồ của mô hình use-case nghiệp vụ.

Tác nhân (actor) trong môi trường nghiệp vụ

Để hiểu rõ được mục tiêu của nghiệp vụ, cần phải biết nghiệp vụ tương tác với những ai; nghĩa là ai đang yêu cầu hay quan tâm đến đầu ra của nó. Những ai đó này được biểu diễn như là các business actor.

Thuật ngữ actor (tác nhân) ám chỉ vai trò mà một người hay một thứ gì đó nắm giữ trong khi tương tác với nghiệp vụ. Những loại người dùng nghiệp vụ sau đây có khả năng được xem là những tác nhân nghiệp vụ: khách hàng, nhà cung cấp, đối tác, đồng nghiệp ở những nghiệp vụ không được mô hình hóa ...

Như vậy, một tác nhân thường tương ứng với một người sử dụng. Tuy nhiên, có những tình huống, chẳng hạn như một hệ thống thông tin đóng vai trò của một tác nhân. Ví dụ, ngân hàng có thể quản lý hầu hết các giao dịch trực tuyến từ một máy tính thì các use case của hệ thống sẽ tương tác với ngân hàng, khi đó ngân hàng được xem là một tác nhân, điều đó có nghĩa tác nhân lúc này là một hệ thống thông tin.



Một actor biểu diễn một loại người dùng cụ thể hơn là một người dùng thực tế. Nhiều người dùng thực tế của một nghiệp vụ có thể chỉ giữ một vai trò của tác nhân; nghĩa là, họ được xem như là các thể hiện của cùng một tác nhân. Hoặc một người dùng có thể giữ nhiều vai trò tác nhân khác nhau; nghĩa là cùng một người có thể là thế hiện của các tác nhân khác nhau.

Cách thức đặt tên các tác nhân nghiệp vụ: Tên của một tác nhân nghiệp vụ cần phản ánh vai trò nghiệp vụ của nó đồng thời nó có thể áp dụng được với bất cứ ai - hay bất cứ hệ thống thông tin nào - đóng vai trò ấy.

Tiêu chí đánh giá những thừa tác viên chuẩn:

- Mỗi thứ tương tác trong môi trường nghiệp vụ - cả con người và máy móc - đều được mô hình hóa bởi các tác nhân. Không thể chắc chắn tìm thấy tất cả tac nhân cho đến khi tất cả use case được tìm ra và được mô tả đầy đủ.
- Mỗi tác nhân "người" diễn tả một vai trò, chứ không phải một người cụ thể. Chúng ta phải chỉ rõ ít nhất hai người có thể có vai trò của mỗi tác nhân. Nếu không, ta có thể đang mô hình hóa một người, chứ không phải một vai trò. Dĩ nhiên là có những tình huống chỉ tìm thấy một người có thể đóng một vai trò.
- Mỗi tác nhân mô hình hóa một thứ gì đó ở bên ngoài nghiệp vụ.
- Mỗi tác nhân có liên quan đến ít nhất một use case. Nếu một tác nhân không tương tác với ít nhất một use case, thì nên loại bỏ nó đi.
- Một tác nhân cụ thể không tương tác với nghiệp vụ theo nhiều cách khác nhau hoàn toàn. Nếu một tác nhân tương tác theo nhiều cách khác nhau hoàn toàn, thì một tác nhân

có thể có nhiều vai trò khác nhau. Trong trường hợp đó, tác nhân đó được chia thành nhiều actor, mỗi cái biểu diễn cho một vai trò khác nhau.

- Mỗi tác nhân có một cái tên và mô tả rõ ràng. Tên của tác nhân cần trình bày vai trò nghiệp vụ của nó, tên này phải dễ hiểu cho những người không nằm trong nhóm mô hình hóa nghiệp vụ.

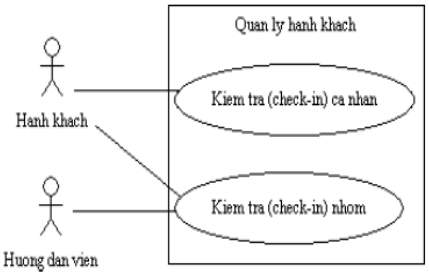
Xác định use case nghiệp vụ

Các qui trình của một nghiệp vụ được xác định thành một số các use case nghiệp vụ khác nhau, mỗi cái biểu diễn một luồng công việc cụ thể trong nghiệp vụ. Một use case nghiệp vụ xác định những gì xảy ra trong nghiệp vụ khi nó được thực hiện; nó mô tả sự thực thi một chuỗi các hành động nhằm tạo ra một kết quả có giá trị cho một tác nhân cụ thể.

Tên của use case cần diễn tả những gì xảy ra khi một thể hiện use case được thực hiện. Do đó, tên cần ở dạng chủ động, thông thường là một động từ kết hợp với một danh từ.

Tên có thể mô tả các hoạt động trong use case từ góc nhìn bên ngoài hoặc bên trong, ví dụ: đặt hàng hay nhận đặt hàng. Cho dù một use case mô tả những gì xảy ra bên trong nghiệp vụ, cách tự nhiên nhất vẫn là đặt tên use case từ góc nhìn của tác nhân chủ chốt trong use case đó. Một khi đã quyết định theo phong cách nào, ta nên áp dụng cùng một quy tắc cho tất cả use case trong mô hình nghiệp vụ.

Ví dụ:



Một hành khách hoặc có thể đi du lịch riêng lẻ hoặc cùng với một nhóm. Khi đi du lịch cùng với một nhóm, sẽ có một hướng dẫn viên du lịch cùng đi và việc check-in có thể được thực hiện cho một đoàn bởi hướng dẫn viên hoặc bởi một hành khách đại diện

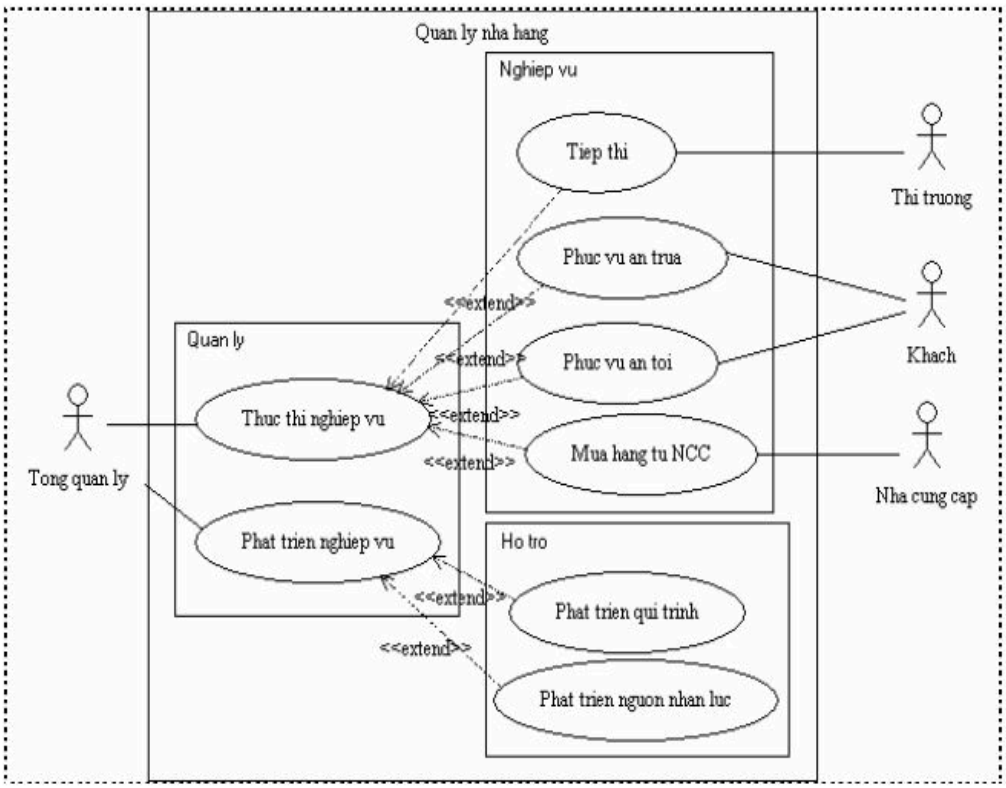
Phân loại use case nghiệp vụ

Khi nhìn vào các hoạt động trong một nghiệp vụ, ta có thể xác định tối thiểu ba loại công việc tương ứng với ba loại use case sau:

- Các hoạt động liên quan đến công việc của tổ chức, thường được gọi là các qui trình nghiệp vụ.
- Nhiều hoạt động không liên quan đến công việc của tổ chức, nhưng phải được thực hiện theo một cách nào đó để làm cho nghiệp vụ hoạt động. Ví dụ như quản trị hệ thống, dọn dẹp, an ninh. Các use case này mang đặc điểm hỗ trợ.
- Công việc quản lý. Các use case có đặc điểm quản lý cho thấy những loại công việc ảnh hưởng đến cách thức quản lý các use case khác và các mối quan hệ của nghiệp vụ với những chủ nhân của nó.

Thông thường, một use case quản lý mô tả tổng quan về các mối quan hệ giữa nhà quản lý với những nhân viên làm việc trong các use case. Nó cũng mô tả cách thức phát triển và khởi tạo các use case.

Ví dụ: các loại use case nghiệp vụ của một tổ chức nhà hàng



Lưu ý rằng một use case nghiệp vụ quan trọng đôi khi có thể là một use case nghiệp vụ hỗ trợ trong một nghiệp vụ khác. Ví dụ: phát triển phần mềm là một use case nghiệp vụ quan trọng của một công ty phát triển phần mềm, trong khi đó nó được phân loại thành một use case nghiệp vụ hỗ trợ trong một ngân hàng hay một công ty bảo hiểm.

Qui mô của một use case nghiệp vụ

Đôi khi khó quyết định được một dịch vụ là một, hay nhiều use case nghiệp vụ. Áp dụng định nghĩa của một use case nghiệp vụ cho qui trình đăng ký chuyến bay. Một hành khách đưa vé và hành lý cho nhân viên đăng ký, nhân viên này sẽ tìm một chổ ngồi cho hành khách, in ra thẻ lên máy bay và bắt đầu xử lý hành lý. Nếu hành khách có một hành lý thông thường, nhân viên đăng ký sẽ in ra thẻ đánh dấu hành lý và thẻ kiểm soát hành khách, cuối cùng kết thúc use case nghiệp vụ bằng cách gắn thẻ đánh dấu cho hành lý, đưa thẻ kiểm soát cùng với thẻ lên máy bay cho hành khách. Nếu hành lý là một dạng đặc biệt hay chứa những thứ đặc biệt không thể vận chuyển một cách bình thường, hành khách phải mang nó đến một quầy hành lý đặc biệt. Nếu hành lý quá nặng, hành khách phải tiếp tục đến văn phòng vé máy bay để trả tiền, bởi vì các nhân viên đăng ký không xử lý việc đóng tiền.

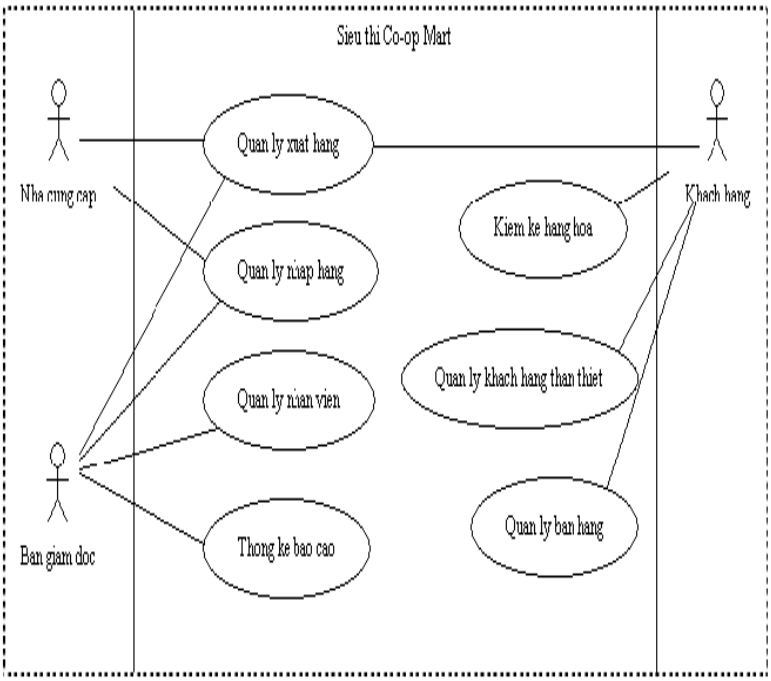
Câu hỏi đặt ra là có cần một use case nghiệp vụ tại quầy đăng ký, một use case nghiệp vụ khác tại quầy hành lý đặc biệt và cái thứ ba ở văn phòng vé? Hay là chỉ cần một use case nghiệp vụ duy nhất? Chắc chắn là sự giao dịch này có liên quan đến ba loại hành động khác nhau. Nhưng câu hỏi ở đây là có một hành động nào đó sẽ có ý nghĩa đối với hành khách mang hành lý đặc biệt nếu

hành khách này không thực hiện những hành động còn lại? Câu trả lời là không có, nó chỉ là một thủ tục hoàn chỉnh - từ lúc hành khách đến quầy đăng ký đến khi ông ta trả thêm phí phụ thu (chỉ có giá trị hay có ý nghĩa đối với hành khách). Như vậy, thủ tục hoàn chỉnh có liên quan đến ba quầy khác nhau chính là một trường hợp sử dụng hoàn chỉnh, tức là một use case nghiệp vụ.

Ngoài tiêu chí này, điều quan trọng là cần giữ mô tả của các dịch vụ có liên quan mật thiết này cùng với nhau, để sau này có thể xem lại chúng cùng một lúc, điều chỉnh, kiểm tra và viết hướng dẫn cho chúng, và nói chung là quản lý chúng như một đơn vị.

Kết quả của quá trình tiếp cận phân tích nghiệp vụ là (các) sơ đồ use case nghiệp vụ và các mô tả của use case.

Ví dụ: mô hình use case mô tả nghiệp vụ của siêu thị Co-op Mart như sau:



Thiết kế qui trình nghiệp vụ

Đặc tả các use case nghiệp vụ:

Bước đầu tiên trong giai đoạn thiết kế này chính là đặc tả các use case nghiệp vụ nhằm làm rõ nội dung của các use case này. Chú ý rằng chúng ta chỉ mô tả các nội dung xử lý thứ tự luận lý giữa các xử lý này, về kết quả, cách mô tả này độc lập tương đối với một một trường thực tế xử lý nó có nghĩa rằng chúng ta chỉ làm rõ phần nội dung của use case mà chưa mô tả rõ các vai trò thực hiện và các đối tượng bị tác động bởi use case.

Hãy bắt đầu mô tả luồng công việc bình thường trong business use case, xác định sự tương tác giữa các tác nhân và use case. Sau đó, khi luồng công việc bình thường ổn định, ta bắt đầu mô tả các luồng công việc thay thế khác. Một luồng công việc use-case nghiệp vụ được trình bày theo các cách thức đã nhất trí và tham khảo bảng chú giải chung khi viết những văn bản mô tả.

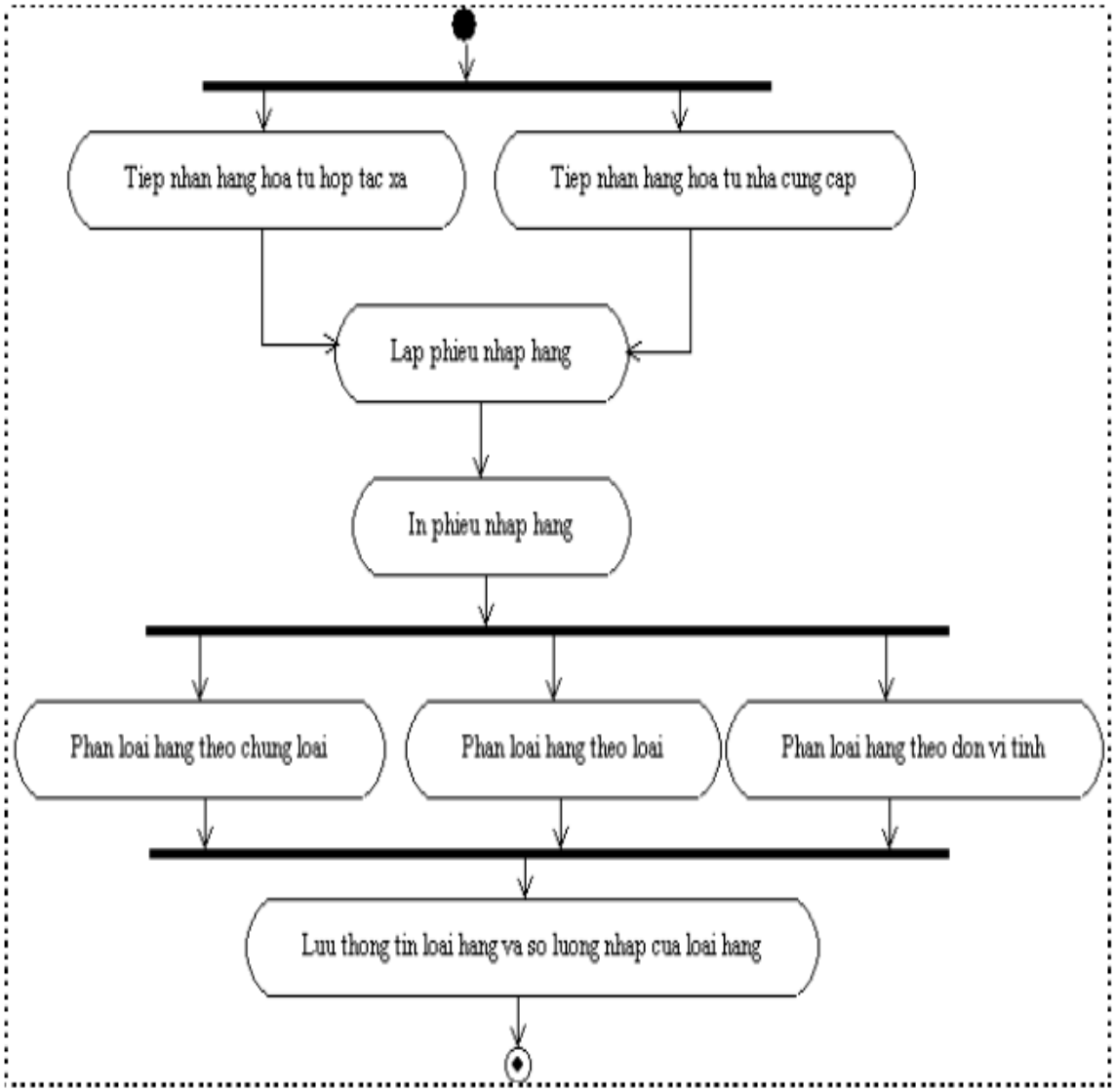
Mô tả tất cả những luồng sự kiện bất thường và luồng sự kiện tùy chọn. Mô tả một luồng sự kiện con trong phần bổ sung của luồng công việc đối với các trường hợp:

- Những luồng sự kiện con tham gia phần lớn luồng công việc chính.

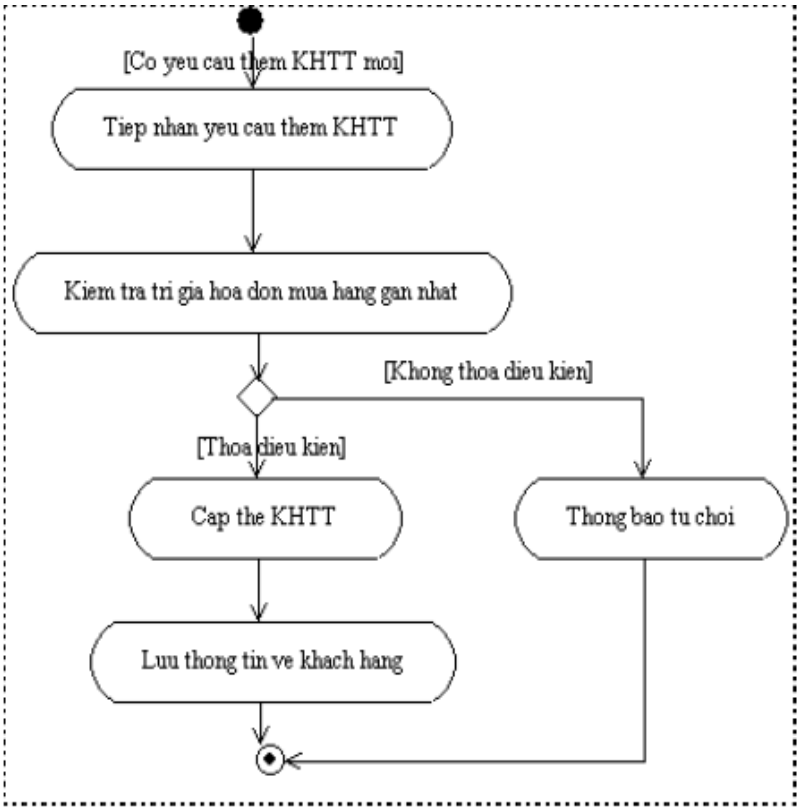
- Những luồng công việc bất thường giúp luồng công việc chính rõ ràng hơn.
- Những luồng sự kiện con xảy ra ở những khoảng thời gian khác nhau trong cùng một luồng công việc và chúng có thể được thực thi.

Ngoài ra, có thể minh họa cấu trúc luồng công việc trong một lược đồ activity.

Ví dụ: sơ đồ hoat động của use case nhập hàng



Ví dụ: sơ đồ hoạt động đặc tả use case Quản lý khách hàng thân thiết



Xác định các thừa tác viên nghiệp vụ (business worker) và các thực thể chịu tác động bởi nghiệp vụ (business entity)

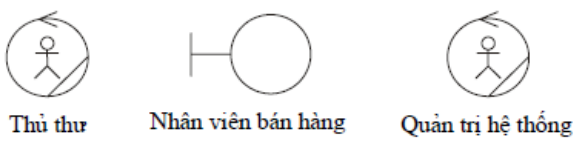
Xác định thừa tác viên nghiệp vụ

Một thừa tác viên biểu diễn sự trừu tượng của một người hoạt động trong nghiệp vụ. Một đối tượng thừa tác viên tương tác với các đối tượng thừa tác viên khác đồng thời thao tác với các đối tượng thực thể để hiện thực hóa một thể hiện use-case.

Một thừa tác viên được khởi tạo khi luồng công việc của thể hiện use-case tương ứng bắt đầu hay ngay vào lúc người có vai trò của thừa tác viên bắt đầu thực hiện vai trò đó trong thể hiện use-case. Một đối tượng thừa tác viên thường "sống" khi use case thực thi.

UML phân chia thừa tác viên thành hai loại: thừa tác viên thực hiện các công việc bên trong hệ thống và thừa tác viên tương tác trực tiếp với các tác nhân bên ngoài hệ thống. Thực sự việc phân chia này chỉ muốn chuyên biệt hơn nữa vai trò của các thừa tác viên trong việc giao tiếp với tác nhân bên ngoài.

Ký hiệu:



Xác định các thực thể nghiệp vụ

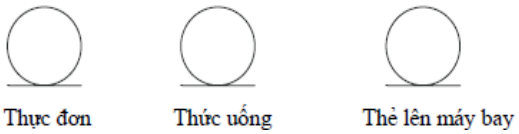
Các business entity biểu diễn những thứ được xử lý hoặc sử dụng bởi các thừa tác viên khi chúng thực thi một use case nghiệp vụ. Một thực thể thường biểu diễn một thứ gì đó có giá trị cho một số thể hiện use case hoặc thể hiện use case, vì vậy đối tượng thực hể sống lâu hơn. Nói chung, thực thể không nên giữ thông tin nào về cách thức nó được sử dụng bởi ai.

Một thực thể biểu diễn một tài liệu hoặc một phần thiết yếu của sản phẩm. Đôi khi nó là một thứ gì đó mơ hồ, như kiến thức về một thị trường hay một khách hàng. Ví dụ về các thực thể tại nhà hàng là Thực đơn và Thức uống; tại phi trường, Vé và thẻ lên máy bay (Boarding Pass) là những thực thể quan trọng.

Mô hình hóa các hiện tượng thành những thực thể chỉ khi những lớp khác trong mô hình đối tượng phải tham chiếu đến các hiện tượng này. Những thứ khác có thể được mô hình hóa thành các thuộc tính của các lớp thích hợp, hay chỉ cần được mô tả bằng văn bản trong những lớp này.

Tất cả mỗi thứ trong nghiệp vụ, như sản phẩm, tài liệu, hợp đồng, ... đều được mô hình hóa thành các thực thể nghiệp vụ, và nó tham gia vào tối thiểu một use case nghiệp vụ.

Ký hiệu:

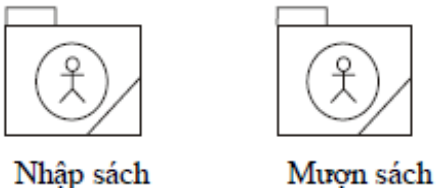


Các khái niệm UML hỗ trợ thêm cho quá trình mô hình hoá nghiệp vụ

Ngoài ra, UML (phiên bản 1.5) còn bổ sung thêm một số stereotype cho phép mô hình hoá đầy đủ hơn về hệ thống nghiệp vụ:

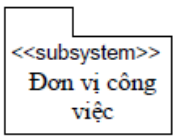
Đơn vị tổ chức (organization unit)

Mô tả : tập hợp các thừa tác viên, thực thể, use case nghiệp vụ, các sơ đồ, và các đơn vị tổ chức khác. Dối tượng này được dùng để phân chia mô hình nghiệp vụ thành nhiều phần khác nhau.



Đơn vị công việc (WorkUnit)

Mô tả: là một loại hệ thống con có thể chứa một hoặc nhiều thực thể. Nó là một tập đối tượng hướng nhiệm vụ nhằm hình thành một tổng thể có thể nhận thức được bởi người dùng cuối và có thể có một giao diện xác định cách nhìn các thực thể công việc thích hợp tới nhiệm vụ đó.



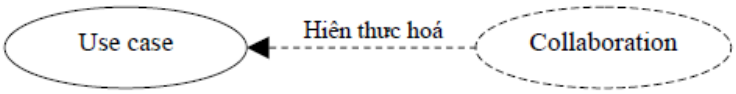
Hiện thực hoá use case nghiệp vụ

Trong một dự án mô hình hóa nghiệp vụ hướng use-case, hãy phát triển hai khung nhìn nghiệp vụ.

Use case nghiệp vụ trình bày khung nhìn bên ngoài của nghiệp vụ, qua đó xác định những gì thiết yếu cần thực hiện cho nghiệp vụ để phân phối các kết quả mong muốn cho tác nhân. Nó cũng xác định trong nghiệp vụ cần có những tương tác nào với tác nhân khi use case được thực thi. Khung nhìn này được phát triển khi đang lựa chọn và nhất trí về những gì cần được thực hiện trong mỗi use case. Một tập hợp các use case cung cấp một cái nhìn tổng quan về nghiệp vụ, nó rất hữu ích để thông báo cho các nhân viên về những thay đổi, những điểm khác biệt của nghiệp vụ đang thực hiện, và những kết quả nào được mong muốn.

Mặt khác, một hiện thực hóa use-case cung cấp một khung nhìn bên trong về use case, qua đó xác định cách thức công việc cần được tổ chức và thực hiện như thế nào nhằm đạt được những kết quả mong muốn như trên. Một hiện thực hóa bao gồm các thừa tác viên và thực thể có liên quan đến sự thực thi một use case và các mối quan hệ giữa chúng. Những khung nhìn như vậy cần thiết cho công việc lựa chọn và thống nhất về cách thức tổ chức các công việc trong mỗi use case nhằm đạt được những kết quả mong muốn.

Cả hai khung nhìn của use case đều chủ yếu dành cho những nhân viên bên trong nghiệp vụ - khung nhìn bên ngoài dành cho những người hoạt động bên ngoài use case, khung nhìn bên trong dành cho những người hoạt động bên trong use case.



Mô hình hoá hiện use case hiện thực hoá qua việc lập cấu trúc mô hình đối tượng nghiệp vụ (business object)

Sơ đồ đối tượng nghiệp vụ là một tập các sơ đồ nhằm trình bày sự hiện thực hóa của các use case nghiệp vụ. Nó mô tả trừu tượng cách thức các thừa tác viên và thực thể liên kết và cộng tác với nhau để thực hiện nghiệp vụ.

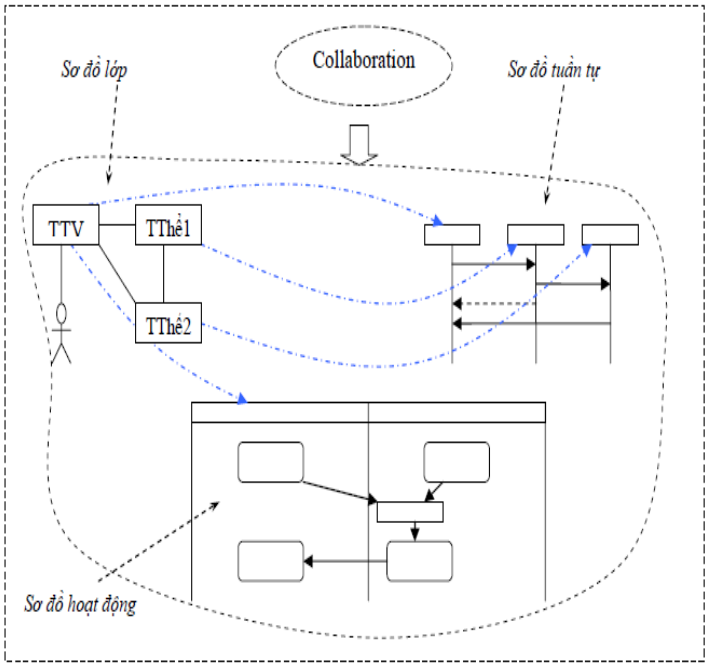
Giải thích

Sơ đồ đối tượng xác định các use case từ góc nhìn bên trong của các thừa tác viên. Mô hình định nghĩa cách thức các nhân viên nghiệp vụ với những gì họ xử lý liên hệ với nhau để tạo ra các kết quả mong muốn. Nó nhấn mạnh vào các vai trò được thực hiện trong lĩnh vực nghiệp vụ và các trách nhiệm của nhân viên. Các đối tượng của các lớp trong mô hình cần có khả năng thực hiện tất cả use case nghiệp vụ.

Các thành phần chính của mô hình đối tượng nghiệp vụ là:

- Các thừa tác viên (worker): cho thấy các trách nhiệm của một nhân viên
- Các thực thể (entity): biểu diễn đầu ra, tài nguyên, sự kiện được sử dụng
- Các hiện thực hóa use-case nghiệp vụ: cho thấy các thừa tác viên cộng tác và các thực thể thực hiện luồng công việc như thế nào. Các hiện thực hóa use-case nghiệp vụ được đặc tả với:

- Các lược đồ lớp: là các thừa tác viên và thực thể tham gia
- Các lược đồ hoạt động: trong đó các swimlane cho thấy các trách nhiệm của các thừa tác viên, các luồng đối tượng cho thấy cách sử dụng các thực thể trong luồng công việc.
- Các lược đồ tuần tự: mô tả chi tiết sự tương tác giữa các thừa tác viên, tác nhân, và cách truy xuất các thực thể khi thực hiện một use case nghiệp vụ.



Mục đích của mô hình đối tượng nghiệp vụ

Nó là một artifact trung gian để làm rõ các ý kiến về nghiệp vụ theo cách suy nghĩ của các nhà phát triển phần mềm, mà vẫn giữ được nội dung nghiệp vụ. Nó là một sự thống nhất về những gì ta biết về lĩnh vực nghiệp vụ được mô tả dưới dạng các đối tượng, thuộc tính, trách nhiệm.

Nó khảo sát bản chất của lĩnh vực nghiệp vụ nhằm chuyển tiếp lối tư duy về các vấn đề nghiệp vụ sang lối tư duy về các ứng dụng phần mềm.

Nó làm rõ những yêu cầu được hỗ trợ bởi hệ thống thông tin đang xây dựng.

Nó thống nhất các định nghĩa về đối tượng nghiệp vụ, các mối quan hệ giữa các đối tượng, tên các đối tượng và quan hệ. Qua đó, cho phép trình bày chính xác các kiến thức về lĩnh vực nghiệp vụ sao cho các chuyên gia về lĩnh vực nghiệp vụ có thể hiểu được.

Lập cấu trúc mô hình đối tượng nghiệp vụ:

Phân tích chu kỳ sống của mỗi thực thể. Mỗi thực thể nên được tạo ra và hủy đi bởi một người nào đó trong đời sống của nghiệp vụ. Hãy bảo đảm rằng mỗi thực thể được truy xuất và sử dụng bởi một thừa tác viên hay một thực thể khác.

Cần giảm bớt số lượng các thừa tác viên. Khi phát triển các mô hình, có thể ta sẽ thấy có quá nhiều thừa tác viên. Hãy bảo đảm rằng mỗi thừa tác viên tương ứng với một tập hợp các tác vụ mà một người thường thực hiện.

Mỗi thực thể nên có một người chịu trách nhiệm cho nó. Điều này có thể được mô hình hóa bằng một mối kết hợp từ thừa tác viên đến các thực thể mà thừa tác viên đó chịu trách nhiệm.

Một số thực thể có thể do những người ngoài nghiệp vụ chịu trách nhiệm. Mô tả điều này trong bản mô tả vắn tắt của thực thể đó.

Lược đồ lớp (class diagram)

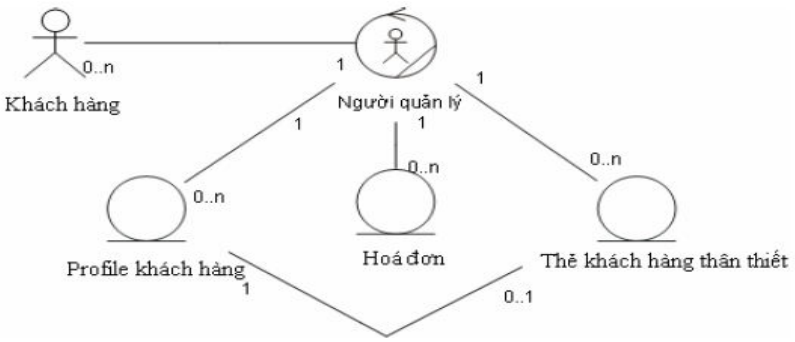
Một lược đồ lớp cho thấy một tập hợp các thành phần (tĩnh) của mô hình, như lớp, gói, nội dung của chúng và các mối quan hệ.

Các lược đồ lớp cho thấy các mối kết hợp, kết tập và tổng quát hóa giữa thừa tác viên và thực thể. Những lược đồ lớp có thể được quan tâm:

- Các hệ thống phân cấp kế thừa
- Các mối kết tập của thừa tác viên và thực thể.
- Cách thức các thừa tác viên và thực thể liên quan đến nhau thông qua các mối kết hợp.

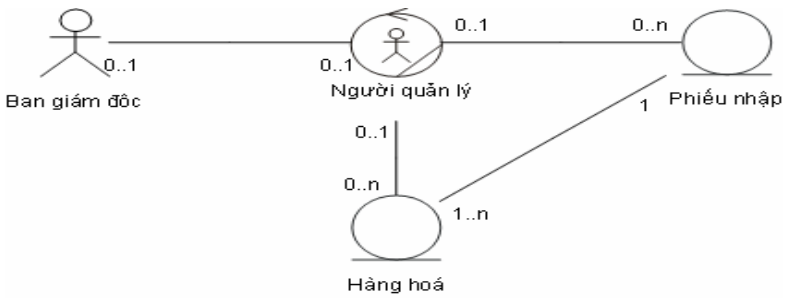
Các lược đồ lớp cho thấy các cấu trúc chung trong mô hình đối tượng nghiệp vụ, nhưng cũng có thế là một phần của tài liệu mô tả một hiện thực hóa use case bằng cách cho thấy các thừa tác viên và thực thể tham gia.

Ví dụ: sơ đồ lớp cho use case Quản lý khách hàng thân thiết cho biết các thừa tác viên, các thực thể và tác nhân liên kết với nhau trong việc thực hiện của use case này.



Trong đó, Người quản lý là thừa tác viên thực hiện use case. Profile khách hàng và Hoá đơn là hai thực thể được sử dụng trong use case này bởi thừa tác viên.

Hoặc sơ đồ lớp cho use case Quản lý nhập hàng



Trong đó, Người quản lý là thừa tác viên thực hiện use case. Phiếu nhập và hàng hoá là các thực thể bởi thừa tác viên này trong việc thực hiện hoạt động của use case.

Đặc tả luồng công việc hiện thực hoá use case nghiệp vụ

Sử dụng sơ đồ hoạt động

Đầu tiên để lập tài liệu cho hiện thực hóa của một use case nghiệp là vẽ một lược đồ hoạt động, trong đó các luồng (swimlane) biểu diễn các thừa tác viên tham gia. Đối với mỗi hiện thực hóa use-case, có thể có một hoặc nhiều lược đồ hoạt động để minh họa luồng công việc.

Một cách phổ biến là sử dụng một lược đồ tổng quan không có các swimlane để mô tả toàn bộ luồng công việc, trong đó trình bày các "hoạt động vĩ mô" ở mức cao. Sau đó, đối với mỗi hoạt động vĩ mô sẽ có một lược đồ hoạt động chi tiết, trình bày các luồng (swimlane) và các hoạt động ở cấp độ thừa tác viên. Mỗi lược đồ nên được gói gọn trong một trang giấy.

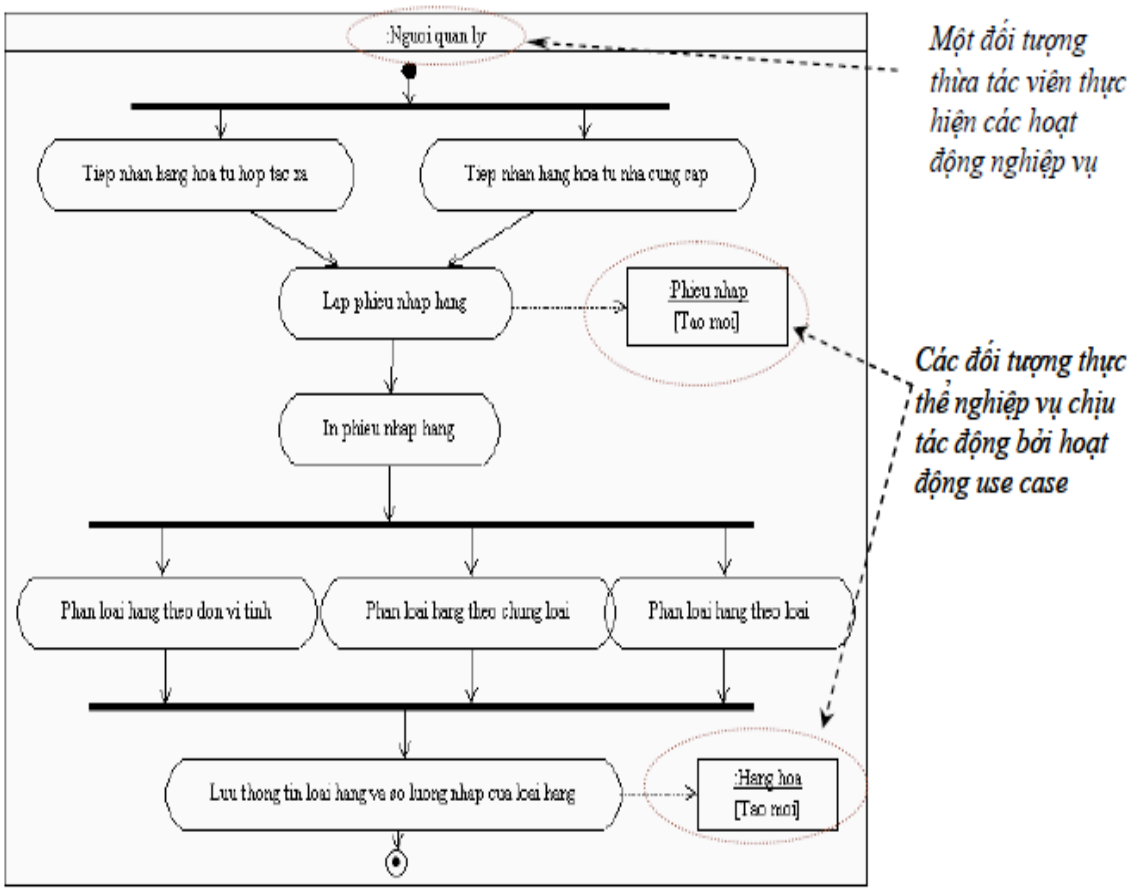
Lược đồ hoạt động trong mô hình đối tượng minh họa luồng công việc của một hiện thực hóa use-case nghiệp vụ. Lược đồ hoạt động của một hiện thực hóa use-case khảo sát việc sắp xếp các công việc theo một thứ tự nhằm đạt được các mục tiêu của nghiệp vụ, cũng như thỏa mãn nhu cầu giữa các tác nhân bên ngoài và các thừa tác viên bên trong. Một hoạt động có thế là một công việc thủ công hoặc tự động hóa để hoàn thành một đơn vị công việc.

Các lược đồ hoạt động giúp:

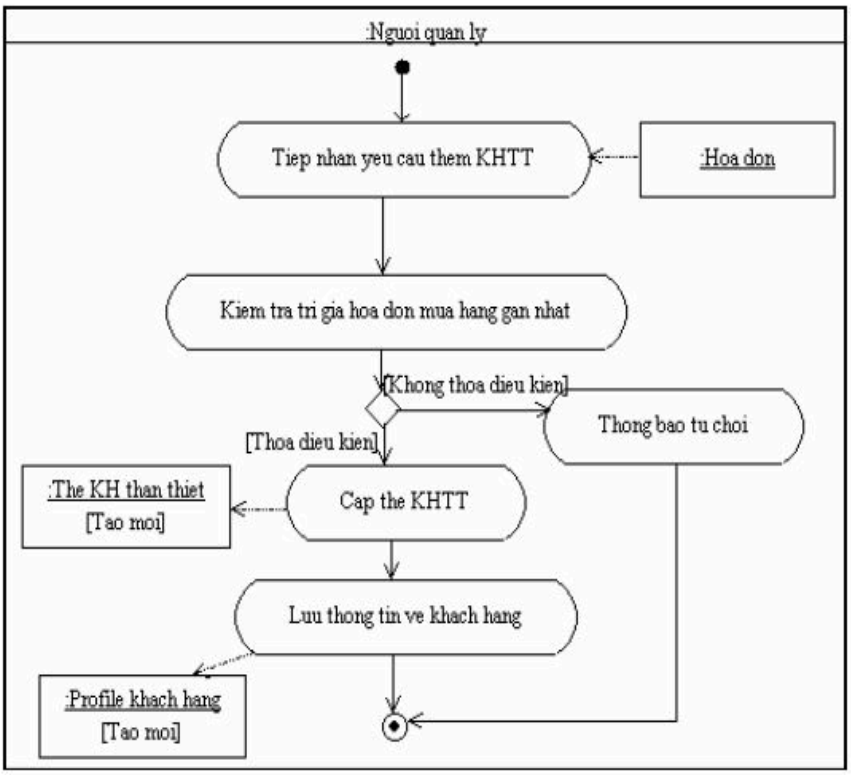
- Cung cấp cơ sở để giới thiệu các hệ thống thông tin đến doanh nghiệp một cách dễ hiểu hơn.
- Thiết lập các mục tiêu cho các dự án phát triển hệ thống nhằm cải tiến nghiệp vụ.
- Điều chỉnh mức độ đầu tư vào việc tự động hóa quy trình dựa trên các thông tin đo lường qui trình nghiệp vụ đó.

So sánh với lược đồ tuần tự có cùng mục đích, lược đồ hoạt động thì tập trung mô tả cách thức phân chia trách nhiệm thành các lớp, trong khi lược đồ tuần tự mô tả cách thức các đối tượng tương tác theo trình tự. Các lược đồ hoạt động tập trung vào luồng công việc, trong khi các lược đồ tuần tự tập trung vào việc xử lý các thực thể. Chúng bổ sung cho nhau, như lược đồ tuần tự cho thấy những gì xảy ra trong một trạng thái hoạt động.

Ví dụ: sơ đồ hoạt động hiện thực hoá use case Quản lý nhập hàng



Sơ đồ hoạt động hiện thực hoá use case Quản lý khách hàng thân thiết



Sử dụng các swimlane (làn bơi)

- Nếu các swimlane được sử dụng và được nhóm thành các lớp (chủ yếu là các thừa tác viên) trong mô hình đối tượng, thì ta đang sử dụng lược đồ hoạt động để trình bày các hiện thực hóa use-case nghiệp vụ, hơn là các use case nghiệp vụ.
- Lược đồ hoạt động cung cấp chi tiết về những gì xảy ra trong nghiệp vụ bằng cách khảo sát những người có các vai trò cụ thể (các thừa tác viên) và các hoạt động mà họ thực hiện. Đối với các dự án phát triển ứng dụng, các lược đồ này giúp ta hiểu một cách chi tiết về lĩnh vực nghiệp vụ sẽ được hỗ trợ hay chịu tác động của ứng dụng mới. Các lược đồ hoạt động giúp ta hình dung hệ thống mới được đề nghị rõ ràng hơn đồng thời xác định các use case của hệ thống đó.

Sử dụng các luồng đối tượng

- Trong ngữ cảnh này, các luồng đối tượng được sử dụng để cho thấy cách thức các thực thể được tạo ra và sử dụng trong một luồng công việc. Các luồng đối tượng trình bày các đầu vào và đầu ra từ các trạng thái hoạt động trong một biểu đồ hoạt động. Có hai thành phần ký hiệu sau:
  - Trạng thái luồng đối tượng (object flow state): biểu diễn một đối tượng của một lớp tham gia vào luồng công việc được biểu diễn trong biểu đồ hoạt động. Đối tượng này có thể là đầu ra của một hoạt động và là đầu vào của nhiều hoạt động khác.
  - Luồng đối tượng (object flow) là một kiểu luồng điều khiển với một trạng thái luồng đối tượng làm đầu vào/đầu ra.
- Ký hiệu luồng đối tượng biểu diễn sự tồn tại của một đối tượng trong một trạng thái cụ thể, chứ không là chính đối tượng đó. Cùng một đối tượng này có thể được thao tác bởi một số các hoạt động kế tiếp nhau làm thay đổi trạng thái của đối tượng. Sau đó, nó có thể được hiển thị nhiều lần trong một biểu đồ hoạt động, mỗi lần xuất hiện sẽ biểu diễn một trạng thái khác nhau trong đời sống của nó. Trạng thái của đối tượng tại mỗi thời điểm có thể được đặt trong ngoặc và viết thêm vào tên của lớp.
- Một trạng thái luồng đối tượng có thể xuất hiện như là trạng thái kết thúc của một luồng đối tượng (sự chuyển tiếp) và là trạng thái bắt đầu của nhiều luồng đối tượng (những sự chuyển tiếp).
- Các luồng đối tượng có thể được so sánh với các luồng dữ liệu bên trong luồng công việc của một use case. Không giống như các luồng dữ liệu truyền thống, các luồng đối tượng tồn tại ở một thời điểm xác định trong một biểu đồ hoạt động.

Sử dụng các lược đồ hợp tác (collaboration) và tuần tự (sequence)

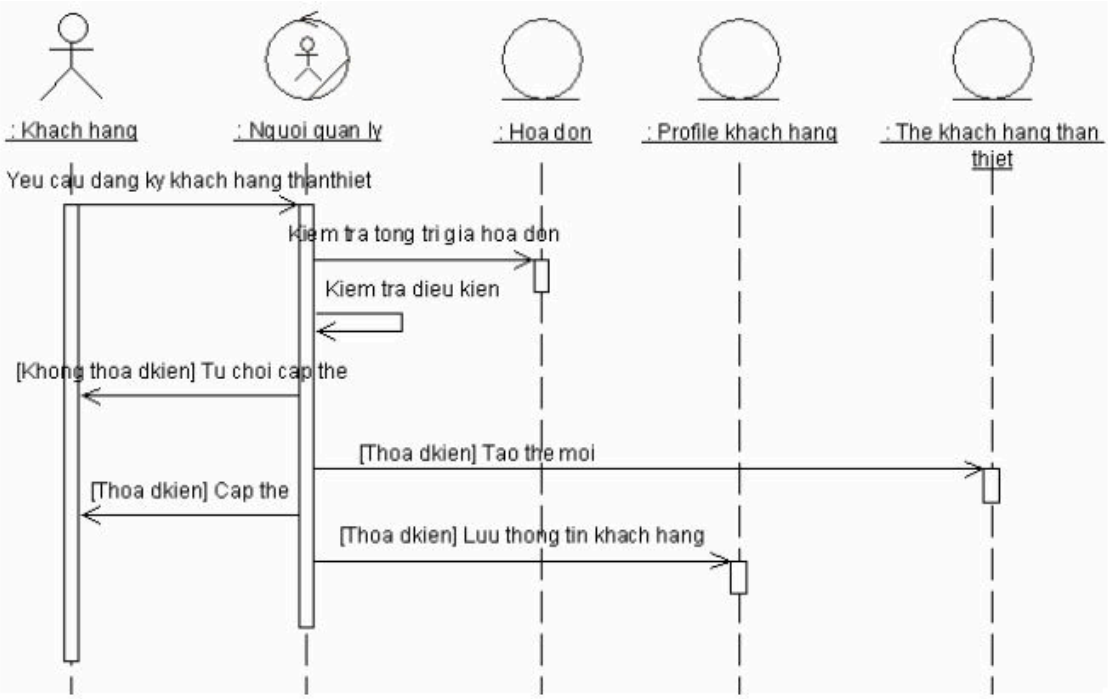
Đối với mỗi hiện thực hóa use-case, có thể có một hoặc nhiều lược đồ tương tác để mô tả các thừa tác viên và thực thể tham gia, cùng với những tương tác của chúng. Có 2 loại lược đồ tương tác là: lược đồ tuần tự và lược đồ hợp tác. Chúng diễn tả những thông tin tương tự nhau, nhưng trình bày những thông tin này theo những cách khác nhau:

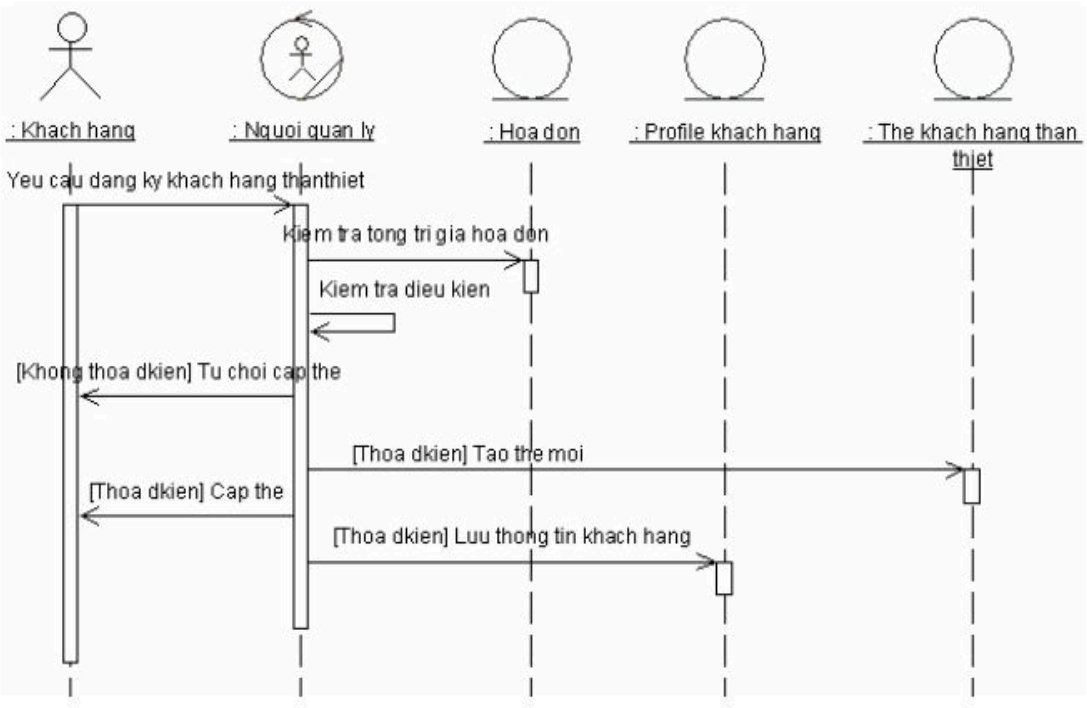
- Các lược đồ tuần tự mô tả rõ ràng trình tự các sự kiện. Với các kịch bản phức tạp, các lược đồ tuần tự thích hợp hơn so với các lược đồ hoạt động.

- Các lược đồ hợp tác trình bày các mối liên kết giao tiếp và những thông điệp giữa các đối tượng. Chúng phù hợp hơn trong việc giúp ta hiểu được tất cả các hiệu quả trên một đối tượng cho trước.
- Nếu ít có các luồng thay thế, nhưng có nhiều thực thể liên quan, các lược đồ tương tác thường là một sự lựa chọn tốt hơn so với lược đồ hoạt động, nhằm để trình bày hiện thực hóa của luồng công việc.

Một lược đồ tuần tự mô tả một mẫu tương tác giữa các đối tượng, được sắp xếp theo thứ tự thời gian; nó cho thấy các đối tượng tham gia vào sự tương tác theo những "lifeline" và những thông điệp mà chúng gửi cho nhau.

Về mặt đồ họa, một lược đồ tuần tự mô tả chi tiết sự tương tác giữa các thừa tác viên, tác nhân, và cách thức các thực thể được truy xuất khi một use case được thực thi. Một lược đồ sequence mô tả vắn tắt các thừa tác viên tham gia làm những gì, và cách thức các thực thể được thao tác thông qua những sự kích hoạt, và cách thức chúng giao tiếp bằng cách gửi thông điệp cho nhau.



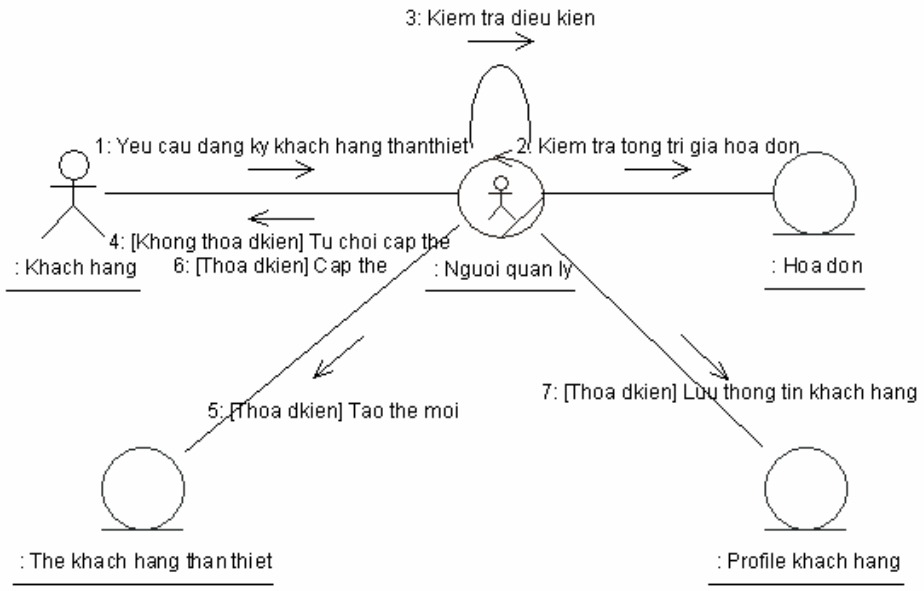


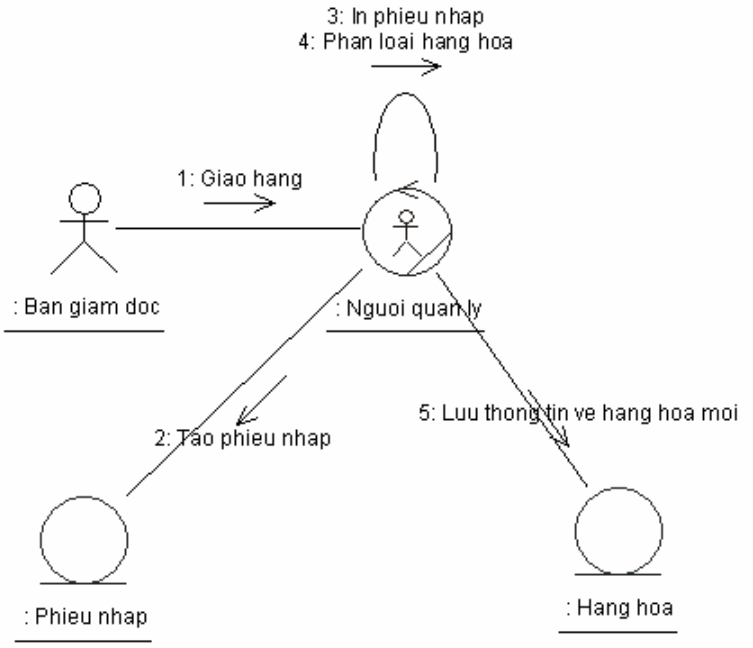
Những thông tin được tìm thấy trong một lược đồ tuần tự cũng có thể được biểu diễn trong một lược đồ hợp tác.

Một lược đồ hợp tác mô tả một mẫu tương tác giữa các đối tượng; nó cho thấy các đối tượng tham gia vào sự tương tác thông qua những mối liên kết giữa chúng và những thông điệp mà chúng gửi cho nhau.

Một lược đồ hợp tác về mặt ngữ nghĩa cũng tương tự như một lược đồ tuần tự, nhưng tập trung chủ yếu vào các đối tượng, trong khi lược đồ tuần tự tập trung vào các tương tác. Một lược đồ tuần tự trình bày một tập con các đối tượng có liên quan đến chuỗi công việc bị ảnh hưởng, bao gồm các mối liên kết giữa chúng, các thông điệp và các chuỗi thông điệp.

Ví dụ:



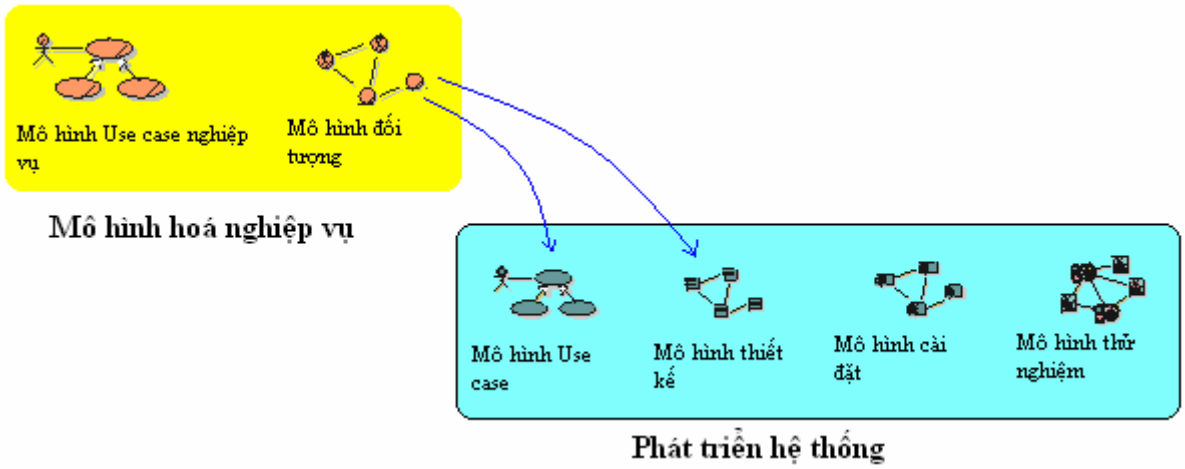


Xác định yêu cầu tự động hoá

Mục đích:

- Hiểu được cách thức sử dụng các công nghệ mới cải thiện hoạt động hiệu quả của tổ chức.
- Xác định mức độ tự động hóa trong tổ chức.
- Thiết lập các yêu cầu hệ thống từ những kết quả mô hình hóa nghiệp vụ.

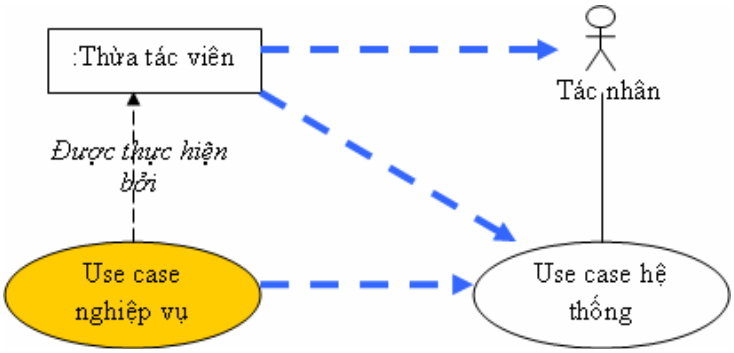
Xác định tác nhân và use case hệ thống phần mềm



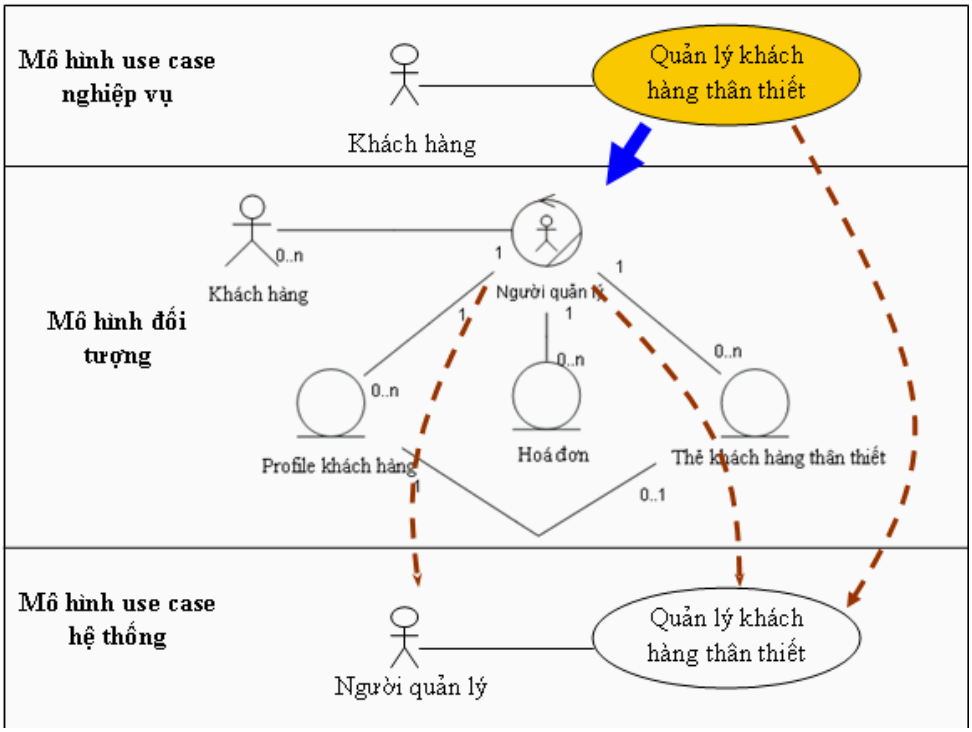
Để xây dựng các hệ thống, cần phải hiểu rõ các qui trình nghiệp vụ. Thậm chí sẽ hữu ích hơn nếu sử dụng các vai trò và trách nhiệm của nhân viên, cũng như những gì được xử lý bởi nghiệp vụ làm nền tảng để xây dựng hệ thống. Điều này được nắm bắt từ góc nhìn bên trong nghiệp vụ dựa vào mô hình đối tượng, trong đó có thể thấy được mối liên kết chặt chẽ nhất đến hình thức thể hiện các mô hình của hệ thống

Để xác định các use case trong hệ thống thông tin, hãy bắt đầu từ các thừa tác viên trong mô hình đối tượng nghiệp vụ. Đối với mỗi thừa tác viên, thực hiện những bước sau đây:

- Xác định xem thừa tác viên sẽ sử dụng hệ thống thông tin không?
- Nếu có, xác định một tác nhân cho thừa tác viên đó của hệ thống thông tin trong mô hình use-case của hệ thống thông tin. Đặt tên tác nhân với tên của thừa tác viên.
- Đối với mỗi use case nghiệp vụ mà thừa tác viên tham gia, tạo một use case hệ thống và mô tả vắn tắt.
- Xem xét các mục tiêu về tốc độ thực thi hay những thông tin bổ sung cho thừa tác viên cần được chú thích như là một yêu cầu đặc biệt của use case hệ thống, hoặc nhập vào sưu liệu đặc tả bổ sung của hệ thống.
- Lập lại những bước này cho tất cả các thừa tác viên.



Ví dụ: Xác định tác nhân và use case hệ thống phần mềm cho use case nghiệp vụ Quản lý khách hàng thân thiết



Trong trường hợp này, khách hàng giao tiếp với người quản lý để giải quyết các yêu cầu về khách hàng thân thiết. Người quản lý sẽ sử dụng phần mềm như là một công cụ nhằm trợ giúp trong các xử lý đáp ứng cho yêu cầu của khách hàng. Do đó, người quản lý lúc này sẽ là tác nhân của hệ thống phần mềm và các chức năng được tự động hoá trong use case nghiệp vụ Quản lý khách hàng thân thiết sẽ trở thành use case mô tả chức năng phần mềm hệ thống.

Nếu mục đích của việc xây dựng một hệ thống là tự động hóa hoàn toàn các qui trình nghiệp vụ (chẳng hạn như việc xây dựng một ứng dụng thương mại điện tử) thì thừa tác viên sẽ không trở thành tác nhân hệ thống nữa. Thay vào đó, chính tác nhân của mội trường nghiệp vụ giao tiếp trực tiếp với hệ thống và hoạt động như một tác nhân hệ thống.

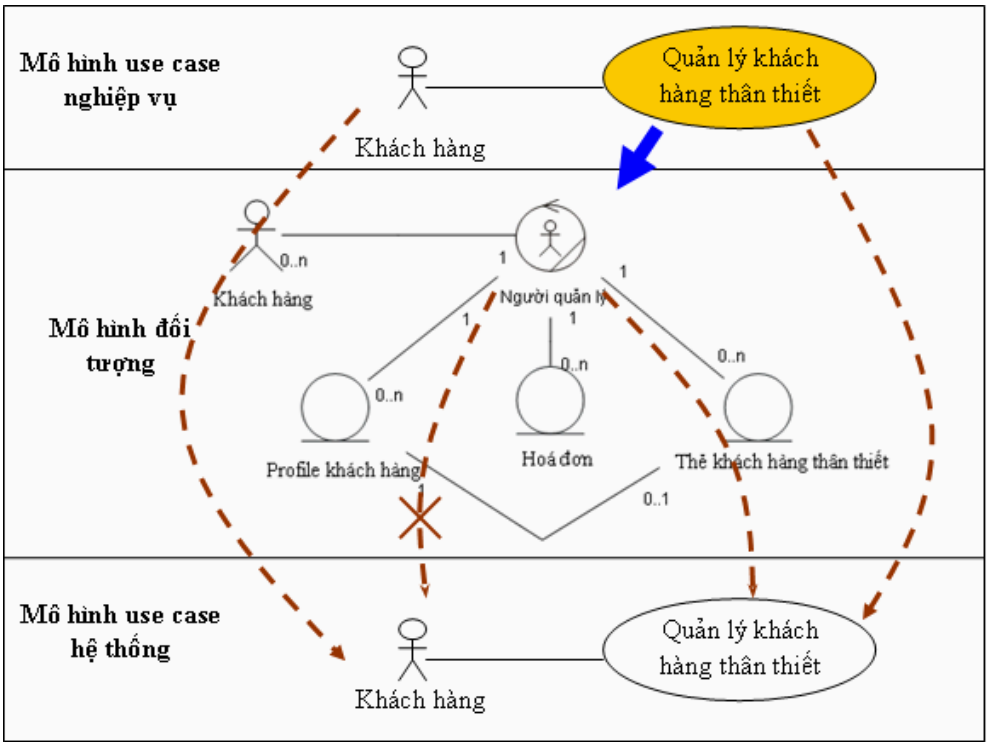
Khi đó, cách thức thực hiện nghiệp vụ sẽ bị thay đổi khi xây dựng một ứng dụng thuộc loại này. Các trách nhiệm của thừa tác viên sẽ chuyển sang tác nhân nghiệp vụ.

Ví dụ: giả sử xây dựng một site thương mại điện tử cho một việc quản lý khách hàng thân thiết, ta sẽ thay đổi cách thức mà qui trình được hiện thực hóa. Lúc này, Khách hàng sử dụng trực tiếp hệ thống thông qua việc truy cập ứng dụng web

Các trách nhiệm của thừa tác viên Người quản lý sẽ chuyển sang tác nhân nghiệp vụ Khách hàng

Tạo ra tác nhân hệ thống Khách hàng tương ứng với tác nhân nghiệp vụ Khách hàng.

Loại bỏ đi Người quản lý.

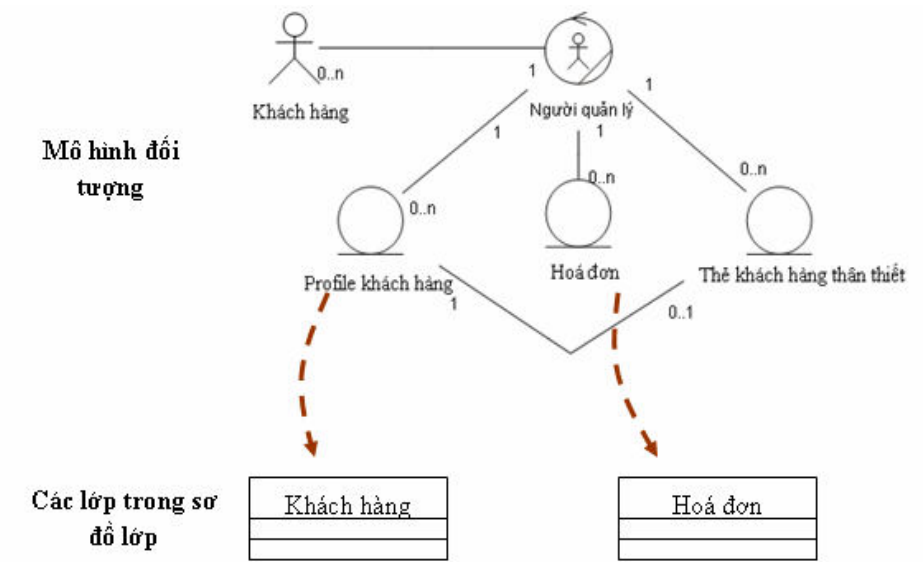


Xác định các lớp đối tượng tông tin trong hệ thống phần mềm

Một thực thể nghiệp vụ được quản lý bởi hệ thống thông tin sẽ tương ứng với một thực thể trong mô hình phân tích của hệ thống thông tin. Tuy nhiên, trong một số trường hợp, sẽ thích hợp nếu để các thuộc tính của thực thể nghiệp vụ tương ứng với các thực thể trong mô hình hệ thống thông tin.

Nhiều thừa tác viên có thể truy xuất một thực thể nghiệp vụ. Do đó, các thực thể tương ứng trong hệ thống có thể tham gia vào một số use case hệ thống thông tin.

Ví dụ: các lớp đối tượng hệ thống phần mềm cho từ sơ đồ lớp của use case nghiệp vụ "Quản lý khách hàng thân thiết"



5. Xây dựng sơ đồ lớp, đối tượng của hệ thống

Mục tiêu:

- Các khái niệm về sự phân loại
- Tìm các lớp đối tượng với các phương pháp: cụm danh từ, phân loại đối tượng và sử dụng sơ đồ use case
- Xác định liên kết giữa các lớp
- Xác định thuộc tính và phương thức của lớp

5.1 Giới thiệu

Phân tích hướng đối tượng là một tiến trình mà qua đó chúng ta có thể định dạng được các lớp đóng một vai trò quan trọng nhằm đạt được mục tiêu và yêu cầu hệ thống. Mô hình hoá đối tượng là một tiến trình mà trong đó, các đối tượng trong một hệ thống thực được thể hiện bởi các đối tượng luận lý trong các sơ đồ và trong chương trình. Sự thể hiện trực quan này của các đối tượng và quan hệ giữa chúng cho phép dễ dàng hiểu về đối tượng của hệ thống. Tuy nhiên, việc xác định lớp là một công việc khó nhất bởi vì không có một cấu trúc lớp nào hoàn hảo cũng như không có một cấu trúc nào hoàn toàn đúng.

Trong phần dưới đây sẽ trình bày về cách để phát triển các mô hình đối tượng bằng cách xây dựng các sơ đồ lớp mô tả việc phân loại đối tượng hệ thống. Trước tiên, chúng ta sẽ ôn lại các khái niệm cơ bản của sơ đồ lớp. Sau đó, chúng ta sẽ được giới thiệu xây ựng sơ đồ lớp thông qua việc giới thiệu lần lượt về các cách tiếp cận để phân loại đối tượng và xác định lớp, cách xác định liên kết giữa các lớp cũng như thuộc tính và phương thức của lớp.

5.2 Sơ đồ lớp (Class diagram)

a) Đối tượng

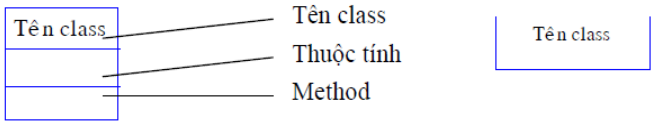
Trong tiếp cận hướng đối tượng, chúng ta mô hình hoá hệ thống bằng các đối tượng, nghĩa là nhìn hệ thống như là một đối tượng . Do đó, trước khi tiếp cận để mô hình hoá hệ thống. Chúng ta cần phải hiểu như thể nào là một đối tượng (object). Có nhiều nguồn mô tả hoặc định nghĩa về đối tượng, tuy nhiên trong tài liệu này chúng ta có thể tổng hợp lại như sau: một đối tượng là một thực thể có một vai trò xác định rõ ràng trong lãnh vực ứng dụng, có trạng thái, hành vi và định danh. Một đối tượng là một khái niệm, một sự trừu tượng hoá hoặc một sự vật có ý nghĩa trong phạm vi ngữ cảnh của hệ thống.

Đối tượng được có thể là một thực thể hữu hình, trực quan (như là: con người, vị trí, sự vật,…); có thể là một khái niệm, sự kiện (ví dụ: bộ phận, đặng ký, …); có thể là một khái niệm trong tiến trình thiết kế (như là: User interface, Controller, Scheduler,…)

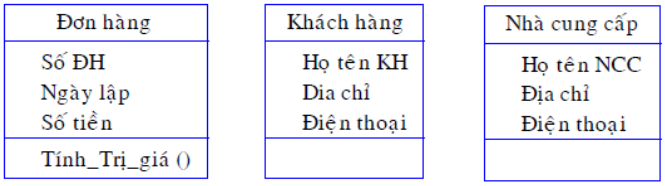
b) Lớp (class)

Là một tập hợp các đối tượng chia sẽ chung một cấu trúc và hành vi (cùng thuộc tính, hoạt động, mối quan hệ và ngữ nghĩa). Cấu trúc được mô tả bởi các thuộc tính và các mối quan hệ, còn hành vi được mô tả bởi các hoạt động. Một lớp là một sự trừu tượng hoá của các đối tượng thế giới thực, và các đối tượng tồn tại trong thế giới thực được xem như là các thể hiện của lớp.

Ký hiệu: lớp được trình bày gồm 3 phần: tên lớp, danh sách các thuộc tính (attribute), danh sách các hoạt động (operation). Trong đó, phần thuộc tính và phần hoạt động có thể bị che dấu đi trong mức độ trình bày tổng quan.



Ví dụ: biểu diễn tập hợp các đơn hàng NGK, khách hàng mua NGK, nhà cung cấp NGK,… cùng chia sẽ chung thuộc tính, hoạt động, mối quan hệ và ngữ nghĩa thành các lớp:



c) Mối kết hợp (association)

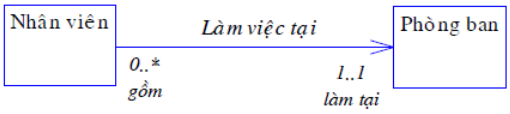
Mối kết hợp nhị phân: là quan hệ ngữ nghĩa được thiết lập giữa hai hay nhiều lớp, biểu diễn bởi những thành phần sau:

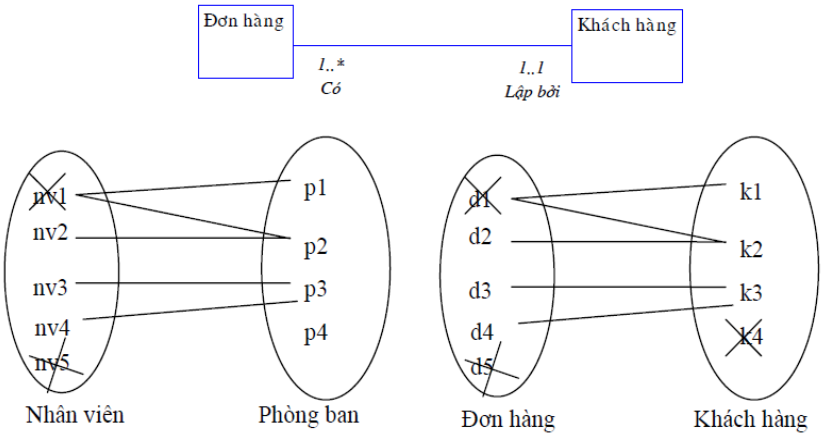
- ***Tên quan hệ***: thường là cụm động từ phản ánh mục đích của mối kết hợp
- ***Vai trò quan hệ (role)***: là một phần của mối kết hợp dùng để mô tả ngữ nghĩa tham gia của một lớp vào mối kết hợp đó (không phải một phần của lớp). Mỗi quan hệ có thể có 2 vai trò (quan hệ nhị phân) hoặc nhiều hơn (quan hệ đa phân).
    - *Tên vai trò:* dùng động từ hoặc danh từ (cụm danh từ) để biểu diễn vai trò của các đối tượng. Trong mối kết hợp làm việc tại có hai vai trò, làm tại và gồm cho biết: nhân viên làm việc tại phòng ban và phòng ban gồm có các nhân viên trực thuộc.
    - *Bản số:* là cặp giá trị (mincard, maxcard) xác định khoảng giá trị cho phép một đối tượng của một lớp có thể tham gia bao nhiêu lần vào mối kết hợp với các đối tượng của các lớp khác.

Giá trị mincard: qui định về ràng buộc tối thiểu của một đối tượng tồn tại trong lớp phải tham gia vào mối kết hợp với một số lượng lớn hơn hoặc bằng.

Giá trị maxcard: qui định số lượng tối đa mà một đối tượng của lớp nếu tồn tại trong lớp đó không được tham gia vào mối kết hợp vượt giá trị này.

Bản số mối kết hợp dưới cho biết một nhân viên phải thuộc ít nhất và nhiều nhất (duy nhất) một phòng ban, tuy nhiên mỗi đối tượng phòng ban có thể tồn tại mà không có nhân viên làm việc thuộc phòng. Các mẫu bản số thường là: 0..1, 1..1, 3..5, 0..*, 1..*, 2..*





Tổng quát, cho hai lớp C1, C2 và mối kết hợp A giữa chúng. Tùy theo giá trị bản số tối thiểu chúng ta có những trường hợp sau:

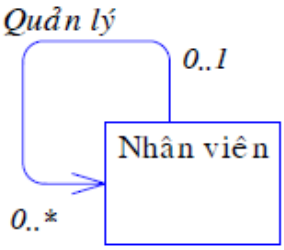
- Nếu mincard (C1,A) = 0 thì chúng ta nói rằng lớp C1 có sự tham gia tùy ý trong mối kết hợp bởi vì một đối tượng của lớp C1 có thể không tham gia kết hợp với đối tượng lớp C2 trong mối kết hợp A.
- Nếu mincard (C1,A) >0 thì chúng ta nói rằng lớp C1 có sự tham gia bắt buộc vào mối kết hợp bởi vì một đối tượng của lớp C1 phải bắt buộc tham gia kết hợp với ít nhất một phần tử của lớp C2 trong mối kết hợp A.

Tùy theo giá trị của bản số tối đa mà chúng ta có các trường hợp sau:

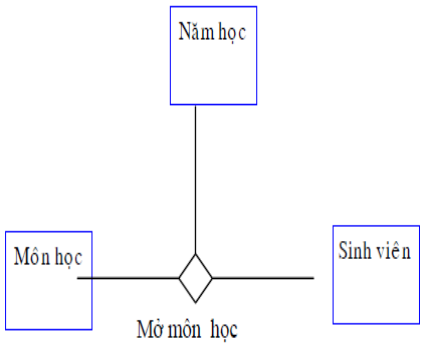
- Nếu maxcard(C1,A) = 1 và maxcard(C2,A) = 1 thì ta gọi là mối kết hợp một - một (one- to – one)
- Nếu maxcard(C1,A) = 1 và maxcard(C2,A) = n thì ta gọi là mối kết hợp một - nhiều (one- to – many)
- Nếu maxcard(C1,A) = n và maxcard(C2,A) = 1 thì ta gọi là mối kết hợp nhiều - một (many- to – one)
- Nếu maxcard(C1,A) = n và maxcard(C2,A) = n thì ta gọi là mối kết hợp nhiều - nhiều (many- to – many)

  - Tính khả điều hướng (navigability): được mô tả bởi một mũi tên chỉ ra hướng truy xuất trong mối kết hợp từ một đối tượng của lớp đến một đối tượng của lớp còn lại. Tính khả điều hướng có thể là không có, hoặc chỉ một, hoặc cả hai. Ví dụ trên cho thấy chiều mũi tên trong mối kết hợp làm việc tại cho biết chúng ta có thể truy cập lớp phòng ban từ mối kết hợp, tuy nhiên, chúng ta không thể truy xuất tới lớp nhân viên từ mối kết hợp này. Hoặc trong mối kết hợp giữa lớp Đơn hàng và Khách hàng, hướng truy xuất là có thể cho cả hai lớp (không có chiều mũi tên)

- ***Mối kết hợp phản thân:*** một mối kết hợp có thể được thiết lập từ một lớp đến chính nó. Ví dụ mối kết hợp quản lý được thiết lập giữa lớp Nhân viên tới chính nó cho biết một nhân viên quản lý những nhân viên khác.

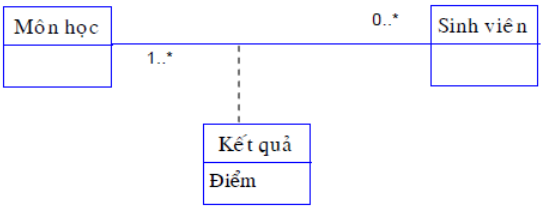


- ***Mối kết hợp đa phân:*** là mối kết hợp được thiết lập từ ba lớp trở lên. Ký hiệu mối kết hợp đa phân là một hình thoi với hơn ba vai trò nối tới các lớp tham gia. Đây là mối kết hợp được thừa hưởng từ cách tiếp cận truyền thống của mô hình thực thể - kết hợp (ER). Tuy nhiên, mối kết hợp đa phân không cho phép quan hệ thu nạp, bản số phức tạp. Do đó, trong cách sử dụng chúng ta thường thay thế nó bằng một lớp và đưa mối kết hợp nhị phân.

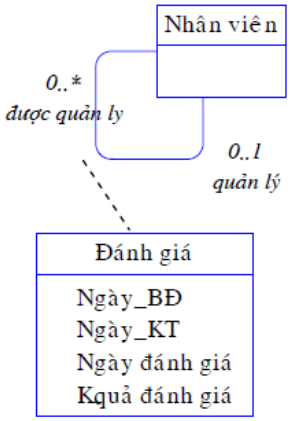


d) Lớp kết hợp

Khi một mối kết hợp có các đặc trưng (thuộc tính, hoạt động, và các mối kết hợp), chúng ta tạo một lớp để chứa các thuộc tính đó và kết nối với mối kết hợp, lớp này được gọi là lớp kết hợp. Tên của lớp này chính là tên của mối kết hợp, trong trường hợp lớp này có thuộc tính nhưng không có hoạt động hoặc bất kỳ mối kết hợp nào khác, thì tên của mối kết hợp vẫn duy trì trên mối kết hợp và để trống phần tên của lớp này để duy trì tính tự nhiên của nó.

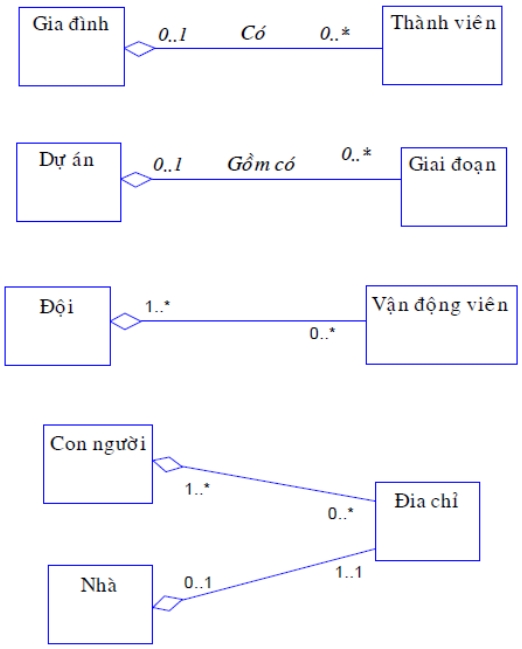


Trường hợp phổ biến nhất của lớp kết hợp là mối kết hợp nhiều - nhiều. Ví dụ trên cho thấy, sinh viên tham gia các môn học khác nhau và mỗi lần đăng ký học, sinh viên sẽ có một kết quả được ghi nhận bởi điểm thi. Vậy điểm thi là một thuộc tính được hình thành thông qua việc tham gia học tập của một sinh viên trên một môn học nên nó là thuộc tính của mối kết hợp.



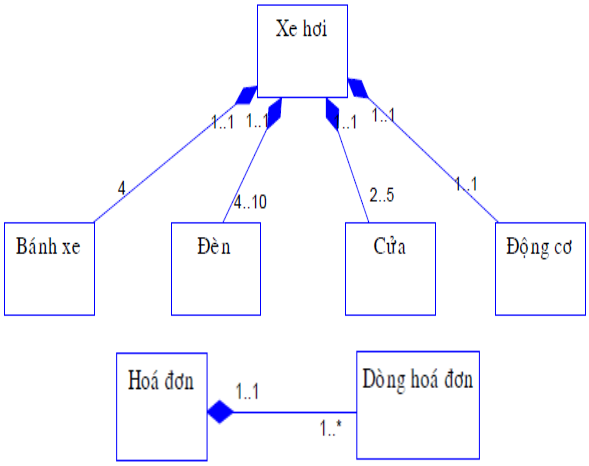
e) Quan hệ thu nạp (aggregation) và quan hệ thành phần (composition, a-part-of)

- Quan hệ thu nạp (aggregation): mô tả mối quan hệ giữa một đối tượng lớn hơn được tạo ra từ những đối tượng nhỏ hơn. Một loại quan hệ đặc biệt này là quan hệ "có", nó có nghĩa là một đối tượng tổng thể có những đối tượng thành phần. Ví dụ dưới đây cho thấy, Gia đình là một đối tượng tổng thể có những Thành viên trong gia đình.



Một đối tượng thành phần cũng có thể tham gia kết hợp với nhiều đối tượng tổng thể khác nhau, trường hợp này gọi là chia sẽ. Ví dụ một vận động viên có quan hệ tới một đội với ý nghĩa là một phần tử của đội, tuy nhiên vận động viên này cũng có thể thành viên của một đội khác, trường hợp này gọi là sự chia sẽ. Do đó, nếu một đội bị hủy bỏ, thì không nhất thiết phải huỷ bỏ vận động viên này.

- Quan hệ thành phần (composition) là một loại đặc biệt của quan hệ thu nạp, nó có một sự liên hệ mạnh mẽ hơn để trình bày thành phần của một đối tượng phức hợp. Quan hệ thành phần cũng được xem như là quan hệ thành phần - tổng thể (partwhole), và đối tượng tổng hợp sẽ quản lý việc tạo lập và huỷ bỏ của những đố tượng thành phần của nó.

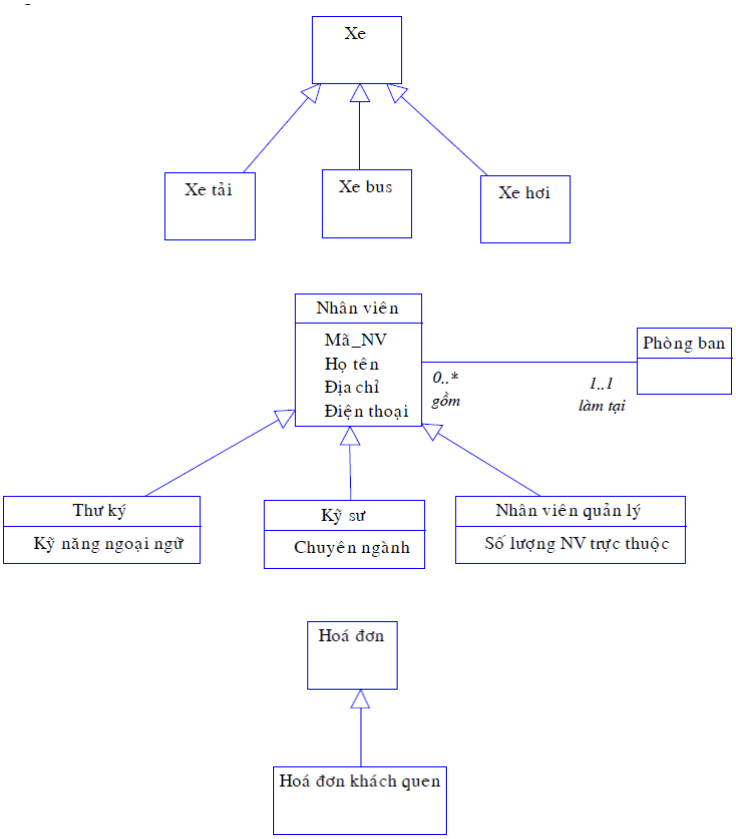


Như vậy, quan hệ thành phần mô tả sự phụ thuộc rất chặt chẽ giữa lớp tổng thể đến lớp thành phần về sự phụ thuộc. Nghĩa là các lớp thành phần là một bộ phận cấu tạo nên lớp tổng thể và thể hiện vật lý của nó là nằm trong lớp tổng thể.

Ví dụ trên cho thấy, mộ chiếc xe hơi được làm nên bởi những bánh xe, đèn, cửa, động cơ, … việc tạo thành một chiếc xe hơi là việc lắp ráp các thành phần này. Cũng như hoá đơn chứa các dòng hoá đơn trong đó, một hoá đơn bị huỷ nghĩa là các đòng của hóa đơn đó cũng sẽ bị huỷ theo.

f) Quan hệ tổng quát hóa

Là quan hệ được thiết lập giữa một lớp tổng quát hơn đến một lớp chuyên biệt. Quan hệ này dùng để phân loại một tập hợp đối tượng thành những loại xác định hơn mà hệ thống cần làm rõ ngữ nghĩa.



Ví dụ trên đây chỉ ra rằng, tất cả các lớp chuyên biệt Thư ký, Kỹ sư, Nhân viên quản lý đều có thể kế thừa các thuộc tính (Mã_NV, Họ tên, Địa chỉ, Điện thoại) của lớp tổng quát Nhân iên và mối kết hợp giữa lớp Nhân viên với Phòng ban.

Trong mối kết hợp tổng quát hoá, một thể hiện của lớp chuyên biệt cũng là một thể hiện của lớp tổng quát. Ví dụ trên cho thấy một đối tượng Kỹ sư, hoặc Thư ký, hoặc Nhân viên quản lý đều là một đối tượng của lớp Nhân viên. Vì lý do đó, đặc trưng của loại kết hợp này là tính kế thừa, một lớp chuyên biệt có thể kế thừa tất cả các đặc trưng (thuộc tính, mối kết hợp, hoạt động) của lớp tổng quát.

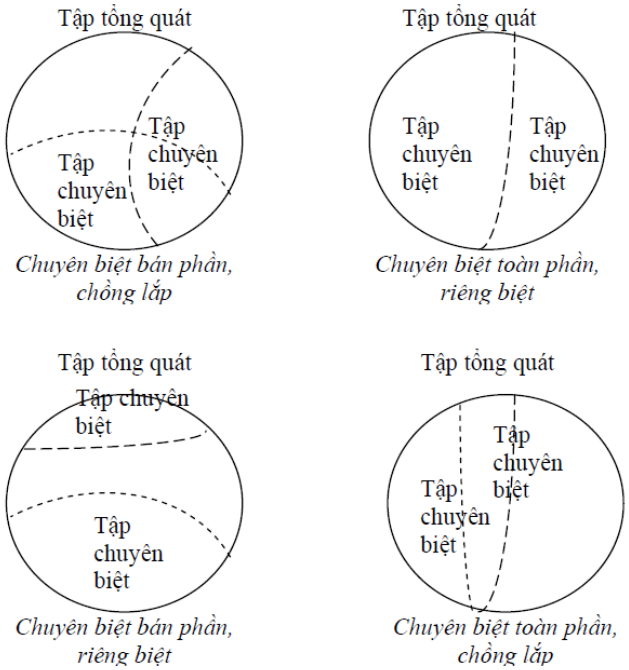
Sự tương quan của các lớp trong quan hệ tổng quát hoá

Sự tương quan giữa các lớp chuyên biệt với lớp tổng quát:

- Tập hợp các đối tượng của tất cả các lớp chuyên biệt phủ toàn bộ tập đối tượng của lớp tổng quát thì gọi là toàn phần (complete).
- Tập hợp các đối tượng của tất cả các lớp chuyên biệt không phủ toàn bộ tập đối tượng của lớp tổng quát thì gọi là bán phần (incomplete).

Sự tương quan giữa các lớp chuyên biệt:

- Không tồn tại một đối tượng của lớp tổng quát thuộc hai lớp chuyên biệt trở lên thì gọi là riêng biệt (disjoint).
- Tồn tại một đối tượng của lớp tổng quát thuộc hai lớp chuyên biệt trở lên thì gọi là chồng lắp (overlapping).



Hình 3. Sự tương quan giữa các lớp trong quan hệ tổng quát hoá

Như vậy, trong quan hệ tổng quát hoá, sự tương quan giữa các lớp được biểu diễn quan bốn trường hợp (bán phần - chồng lắp, bán phần - riêng biệt, toàn phần - chồng lắp, toàn phần – riêng biệt). Sự tương quan này phản ánh ràng buộc ngữ nghĩa trong tập hợp các đối tượng của quan hệ:

một đối tượng của lớp chuyên biệt này có thể là đối tượng trong lớp chuyên biệt khác hay không? Và một đối tượng trong lớp tổng quát có thể không thuộc một lớp chuyên biệt nào hay không?.

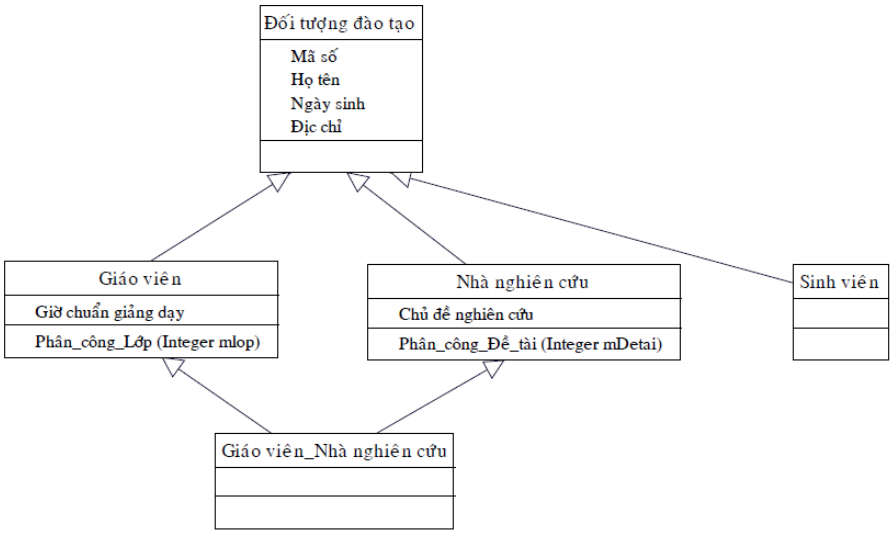
Ví dụ, quan hệ tổng quát hoá giữa Xe – Xe tải, Xe bus, Xe hơi có sự tương quan là bán phần – riêng biệt (incomplete, disjoint). Quan hệ giữa Nhân viên – Thư ký, Kỹ sư, Nhân viên quản lý có sự tương quan là bán phần - chồng lắp (incomplete, overlapping).

Đa kế thừa

Đa số các trường hợp trong quan hệ tổng quát hoá là đơn kế thừa, nơi mà một lớp là chuyên biệt duy nhất cho một lớp tổng quát. Trong một số trường hợp đặc biệt chúng ta cũng thấy một lớp chuyên biệt có thể kế thừa từ hai hoặc nhiều lớp tổng quát. Trường hợp này gọi là đa kế thừa.

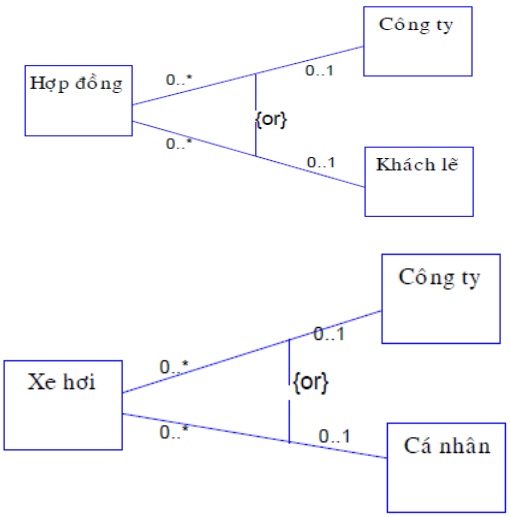
Ví dụ sau cho thấy, lớp Giáo viên_Nhà nghiên cứu là lớp đa kế thừa từ hai lớp Giáo viên và lớp Nhà nghiên cứu. Lớp này sẽ thừa kế tất cả các đặc trưng như: Giờ chuẩn giảng dạy, Chủ đề nghiên cứu, Phân_công_Lớp và Phân_công_Đề_tài của hai lớp trên.

Tuy nhiên, theo lời khuyên của các chuyên gia thì không nên sử dụng đa kế thừa vì tính chất phức tạp của nó. Do đó, đa kế thừa không được đưa vào ngôn ngữ UML gốc và một số ngôn ngữ hướng đối tượng khác.



G) Quan hệ hoặc (OR)

Là mối quan hệ xác định một tình huống mà trong đó hai (hoặc nhiều) lớp tham gia mối kết hợp với một lớp thứ ba với ràng buộc loại trừ. Một thể hiện của lớp thứ ba sẽ tham gia kết hợp loại trừ với các đối tượng của hai lớp kia (hoặc là không kết hợp, hoặc kết hợp chỉ các đối tượng của một trong hai lớp) tại một thời điểm. Ví dụ dưới đây cho thấy, một hợp đồng có thể được lập bởi một công ty hoặc bởi một khách hàng lẽ. Hoặc một chiếc xe hơi thì được sở hữu bởi một cá nhân hoặc bởi một công ty, không sở hữu một lúc bởi cả hai.



Thực chất của mối kết hợp OR cũng chính là một ràng buộc ngữ nghĩa giữa các lớp tham gia kết hợp với một lớp thứ ba: sự hiện diện tham gia của đối tượng này sẽ không cho phép sự tham gia của đối tượng kia và ngược lại.

Thuộc tính (attribute)

Thuộc tính dùng để mô tả đặc trưng của đối tượng, người ta có thể chia thuộc tính thành ba loại sau:

- Thuộc tính đơn trị: là thuộc tính chỉ có một giá trị duy nhất cho một đối tượng, đây là thuộc tính phổ biến nhất. Ví dụ: họ tên, ngày sinh, lương,…
- Thuộc tính đa trị: là thuộc tính có thể có nhiều giá trị cho một đối tượng. Ví dụ: nếu chúng ta muốn lưu nhiều số điện thoại của một nhân viên, chúng ta có thể đặt thuộc tính số điện thoại trong lớp nhân viên là đa trị.
- Thuộc tính tham chiếu.

Biểu diễn một thuộc tính

Thuộc tính được biểu diễn gồm những thành phần như sau:

<phạm vi> <tên thuộc tính> : <biểu thức kiểu> = <giá trị khởi tạo>

<phạm vi>: nhận một trong những giá trị sau:

+ toàn cục (có thể truy cập bởi tất cả lớp)

bảo vệ (có thể truy cập bởi lớp và lớp chuyên biệt của nó)

- riêng (chỉ được truy cập bởi lớp)

Bản số: là một cặp (số tối thiểu, số tối đa) mà thuộc tính có thể có giá trị.

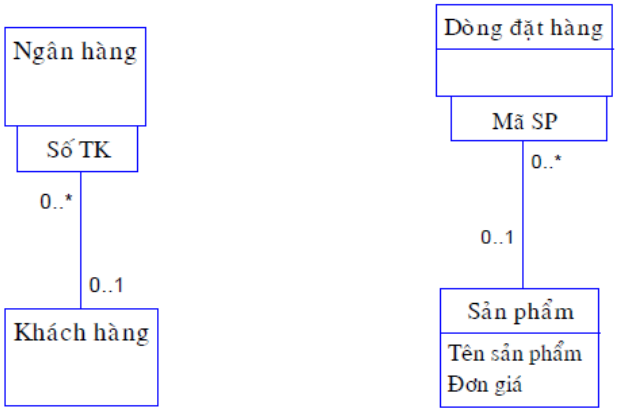
- số tối thiểu= 0 → thuộc tính được gọi là không bắt buộc.
- số tối thiểu = 1 → thuộc tính được gọi là bắt buộc.
- số tối đa = 1 → thuộc tính đơn trị.
- số tối đa > 1 → thuộc tính đa trị

Ví dụ: Số điện thoại[0..*]: string, Địa chỉ[0..1]: string,…

Trong giai đoạn phân tích việc xác định thuộc tính thường chỉ bao gồm xác định tên thuộc tính (có thể thêm kiểu dữ liệu), các đặc điểm khác của thuộc tính sẽ được xác định lại trong giai đoạn thiết kế.

Thuộc tính quan hệ (Qualifier)

Là một thuộc tính quan hệ (không phải của lớp). Nghĩa là thuộc tính này sẽ hình thành từ một quan hệ giữa các lớp. Nhằm thực hiện việc thiết lập mối liên kết giữa một tập thể hiện với một thể hiện khác. Một đối tượng và một giá trị của thuộc tính qualifier sẽ xác lập một định danh duy nhất, hình thành nên khoá phức hợp.



Xem xét mối kết hợp giữa lớp Sản phẩm và lớp Dòng đặt hàng của đơn đặt hàng. Một dòng trong đơn hàng sẽ liên quan đến một sản phẩm sẽ được đặt, mỗi dòng đơn hàng tham chiếu đến duy nhất một sản phẩm, trong khi đó mỗi sản phẩm có thể được đặt trong nhiều dòng đơn hàng của những đơn hàng khác nhau. Mối liên kết qualifier có thuộc tính Mã SP cho biết: mỗi sản phẩm có duy nhất một Mã SP và Dòng đơn hàng liên kết với Sản phẩm sử dụng Mã SP.

Định danh (identifier)

Định danh là một hoặc nhiều thuộc tính của lớp có giá trị xác định duy nhất cho một đối tượng của lớp.

5.3 Các cách tiếp cận xác định lớp đối tượng

Gần như không có một phương thức chung để xác định các lớp của một hệ thống. Thông thường đây là một quá trình sáng tạo và lặp lại qua nhiều vòng lặp và cần phải có sự thống nhất với các chuyên gia trong lãnh vực ứng dụng nghiệp vụ. Có nhiều phương pháp tiếp cận để xác định lớp. Có phương pháp đề nghị tiến hành mô hình hoá nghiệp vụ, chỉ ra phạm vi bài toán nghiệp vụ sẽ được tự động hoá mà kết quả của nó sẽ cung cấp các lớp cho hệ thống tương ứng với việc tự động hoá việc quản lý, lưu trữ các đối tượng thông tin (thực thể) đã được thống nhất trên các yêu cầu với người dùng xử lý nghiệp vụ. Có phương pháp để xuất xác định tất cả các lớp thuộc phạm vi bài toán, mối quan hệ của chúng. Sau đó, sẽ phân tích use case và phân bổ trách nhiệm các lớp theo use case. Có phương pháp đề xuất lấy mô hình use case làm nền tảng để tìm lớp (use case - driven), và trong quá trình xác định trách nhiệm thực hiện của use case thì các lớp xẽ được xác định. Vì quá trình xác định lớp trong giai đoạn này là một quá trình lặp lại mà kết quả của bước sau có thể làm

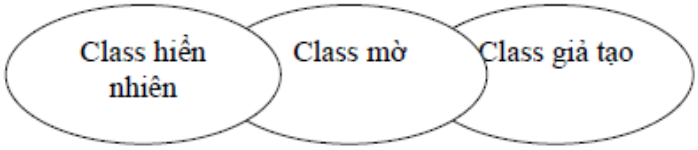
thay đổi các kết quả của bước trước, cho nên các lớp được tìm thấy thường được gọi là lớp ứng viên (candidate class).

Dưới đây chúng ta đề cập đến một số kỹ thuật tiếp cận để xác định lớp: tiếp cận theo cụm danh từ; tiếp cận theo mẫu chung; tiếp cận theo hướng gia tăng; và tiếp cận theo use case.

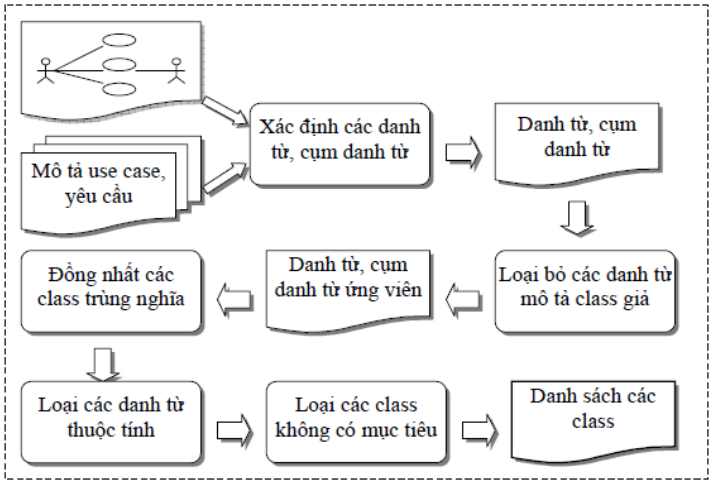
a) Tiếp cận theo cụm danh từ (noun phrase)

Phương pháp tiếp cận theo cụm danh từ được đề xuất bởi Rebecca Wirfs-Brock, Brian Wilkerson, và Lauren Wiener. Phương pháp đề xuất việc xác định các lớp thông qua việc đọc trong các văn bản mô tả use case hoặc các mô tả yêu cầu để tìm kiếm và trích lọc các cụm danh từ. Các cụm danh từ có thể được xem là các ứng viên của các lớp và các động từ là các ứng viên của phương thức (method) của lớp. Tất cả danh từ hoặc cụm danh từ tìm được sẽ được phân thành ba loại:

- Các lớp hiển nhiên
- Các lớp mờ
- Các lớp giả tạo



Đầu tiên tất cả lớp thuộc loại lớp giả sẽ bị loại bỏ, vì nó không có mục đích hoặc không cần thiết để sử dụng. Các lớp thuộc hai loại còn lại sẽ trở thành các ứng viên. Quy trình xác định như sau:



Khởi tạo danh sách các lớp ứng viên

- Tìm các danh từ hoặc các cụm danh từ trong các mô tả use case, yêu cầu
- Tất cả các lớp phải có ý nghĩa trong lãnh vực ứng dụng, tránh đưa vào các lớp cài đặt được mô tả trong giai đoạn thiết kế.
- Đặt tên cho lớp

Trích lọc trong use case và mô tả use case của hệ thống ATM, chúng ta có những danh từ và cụm danh từ sau:

Tài khoản	Bao thư	PIN không hợp lệ
Số dư tài khoản	Bốn ký số	Ngân hàng
Số tiền	Ngân quỹ	Khách hàng ngân hàng
Tiến trình đăng nhập	Tiền	Thẻ
Thẻ ATM	PIN	Tiền mặt
Máy ATM	Lịch sử giao dịch	Khách hàng
Thông điệp	Loại bỏ các lớp giả	Tài khoản khách hàng
Mật khẩu	Bước	VND
Mã PIN	Hệ thống	
Mẫu tin	Giao dịch	

Loại bỏ các lớp giải

Các lớp ứng viên phải thuộc loại lớp hiển nhiên và lớp mờ. Các lớp giả sau đây sẽ bị loại khỏi danh sách: Bao thư, Bốn ký số, Bước.

Tài khoản	~~Bao thư~~	PIN không hợp lệ
Số dư tài khoản	Bốn ký số	Ngân hàng
Số tiền	Ngân quỹ	Khách hàng ngân hàng
Tiến trình đăng nhập	Tiền	Thẻ
Thẻ ATM	PIN	Tiền mặt
Máy ATM	Lịch sử giao dịch	Khách hàng
Thông điệp	Loại bỏ các lớp giả	Tài khoản khách hàng
Mật khẩu	Bước	VND
Mã PIN	Hệ thống	
Mẫu tin	Giao dịch	

Đồng nhất các lớp ứng viên trùng lắp

Cần rà soát lại danh sách để tìm kiếm các danh từ, cụm danh từ trùng lắp về ý nghĩa mặc dù cách dùng từ có khác nhau. Chúng ta chọn lựa danh từ, hoặc cụm danh từ chứa đầy ngữ nghĩa nhất và loại những danh từ, cụm danh từ khác.

Khách hàng, Khách hàng ngân hàng	= Khách hàng
Tài khoản, Tài khoản khách hàng	= Tài khoản

PIN, Mã PIN	= PIN
Tiền, Ngân quỹ	= Ngân quỹ
Thẻ ATM, Thẻ	= Thẻ ATM
Tài khoản	Bao thư
Số dư tài khoản	Bốn ký số
Số tiền	Ngân quỹ
Tiến trình đăng nhập	Tiền
Thẻ ATM	PIN
Máy ATM	PIN không hợp lệ
Ngân hàng	Thông điệp
Khách hàng ngân hàng	Mật khẩu
Thẻ	Mã PIN
Tiền mặt	Mẫu tin
Khách hàng	Bước
Tài khoản khách hàng	Hệ thống
VND	Giao dịch
	Lịch sử giao dịch

Xác định các danh từ, cụm danh từ có thể là các thuộc tính

Các danh từ hoặc cụm danh từ là các thuộc tính khi:

- Chỉ được sử dụng như là giá trị
- Không có nhiều hơn một đặc trưng riêng, hoặc chỉ mô tả một đặc trưng của đối tượng khác.

Xem xét các danh từ, cụm danh từ có thể là thuộc tính của danh sách trên ta có:

Số tiền: một giá trị, không phải một lớp

Số dư tài khoản: thuộc tính của lớp Tài khoản

PIN không hợp lệ: một giá trị, không phải một lớp

Mật khẩu: một thuộc tính (có thể của lớp Khách hàng)

Lịch sử giao dịch: một thuộc tính (có thể của lớp Giao dịch)

PIN: một thuộc tính (có thể của lớp Khách hàng)

Sau đây là danh sách các ứng viên còn lại:

Tài khoản Bao thư

Số dư tài khoản	Bốn ký số
Số tiền	Ngân quỹ
Tiến trình đăng nhập	Tiền
Thẻ ATM	PIN
Máy ATM	PIN không hợp lệ
Ngân hàng	Thông điệp
Khách hàng ngân hàng	Mật khẩu
Thẻ	Mã PIN
Tiền mặt	Mẫu tin
Khách hàng	Bước
Tài khoản khách hàng	Hệ thống
VND	Giao dịch
	Lịch sử giao dịch

Loại bỏ các lớp ứng viên không có mục tiêu hoặc không thuộc phạm vi hệ thống

Mỗi lớp phải có một mục tiêu khi thuộc hệ thống, mục tiêu này phải thật rõ ràng trong ngữ cảnh mục tiêu chung hệ thống. Nếu chúng ta không thể diễn đạt mục tiêu của lớp trong hệ thống thì loại ra khỏi danh sách. Hoặc các lớp mặc dù có tham gia vào hoạt động của hệ thống, tuy nhiên nó không thuộc phạm vi quản lý của hệ thống sẽ bị loại ra. Các lớp ứng viên là:

Máy ATM: cung cấp một giao diện tới ngân hàng

Thẻ ATM: cung cấp một khách hàng với một khoá tới một tài khoản

Khách hàng: một khách hàng là một cá nhân sử dụng máy ATM, có một tài khoản.

Ngân hàng: các khách hàng phụ thuộc vào ngân hàng. Nó là một nơi tập trung các tài khoản và xử lý các giao dịch tài khoản.

Tài khoản: nó mô hình hoá một tài khoản của khách hàng và cung cấp các dịch vụ về tài khoản cho khách hàng

Giao dịch: mô tả một giao tác của khách hàng khi sử dụng thẻ ATM. Một giao tác được lưu trữ với thời gian, ngày, loại, số tiền, và số dư.

Các danh từ, cụm danh từ không có mục đích hoặc không thuộc phạm vi quản lý của hệ thống:

Thông điệp

Hệ thống

Mẫu tin

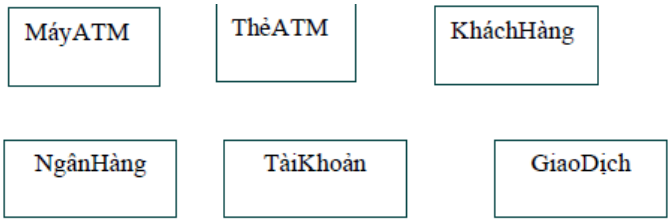
Ngân quỹ

VND

Tiền mặt

Tiến trình đăng nhập

Kết quả của quá trình chọn lựa gồm các lớp ứng viên sau hệ thống ATM:



Nhận xét: một hạn chế chính của cách tiếp cận cụm danh từ là nó phụ thuộc vào tính đúng và đầy đủ của các tài liệu mô tả. Điều này trên thực tế để có được những tài liệu này thì quả là khó. Hoặc chăng một văn bản lớn của hệ thống có thể dẫn đến quá nhiều lớp ứng viên! Dầu vậy, cách tiếp cận này rất có tính sư phạm và hữu dụng khi kết hợp với các cách tiếp cận khác.

b) Tiếp cận phân loại

Phương pháp thứ hai được gọi là phương pháp sử dụng mẫu lớp chung, phương pháp này dựa trên một cơ sở tri thức về việc phân loại lớp theo những mẫu chung. Các mẫu chung đó là:

Lớp khái niệm (concept)

Một khái niệm là một quan niệm hoặc sự hiểu biết riêng biệt về thế giới. Lớp khái niệm bao gồm các nguyên lý được dùng để tổ chức hoặc để lưu trữ các hoạt động và các trao đổi về mặt quản lý. Thông thường các khái niệm là các ý tưởng, sự hiểu biết được chia sẽ trong công đồng và dùng để trao đổi.

Ví dụ: phương pháp, phương pháp luận, mô hình,… là ví dụ của đối tượng lớp khái niệm.

Lớp sự kiện (event)

Lớp sự kiện là các điểm thời gian cần được lưu trữ. Các sự việc xảy ra tại một thời điểm, hoặc một bước trong một dãy tuần tự các bước. Liên quan tới các sự việc được lưu trữ là các thuộc tính (và các đối tượng chứa thuộc tính) như là: ai, cái gì, khi nào, ở đâu, như thế nào, hoặc tại sao.

Ví dụ: đăng ký, kết quả, hoá đơn,…

Lớp tổ chức (organization)

Một lớp tổ chức là một tập hợp con người, tài nguyên, phương tiện, hoặc những nhóm xác định chức năng người dùng,….

Ví dụ: đơn vị, bộ phận, phòng ban, chức danh,…

Lớp con người (people)

Lớp con người thể hiện các vai trò khác nhau của người dùng trong việc tương tác với ứng dụng. Những đối tượng này thường là người dùng hệ thống hoặc những người không sử dụng hệ

thống nhưng thông tin về họ được lưu trữ bởi hệ thống (đa số là những đối tượng mà hệ thống có trao đổi thông tin nhưng không sử dụng hệ thống)

Ví dụ: sinh viên, khách hàng, giáo viên, nhân viên,…

Lớp vị trí (place)

Các vị trí vật lý mà hệ thống cần lưu trữ thông tin về nó.

Ví dụ: toà nhà, kho, văn phòng, chi nhánh, đại lý, …

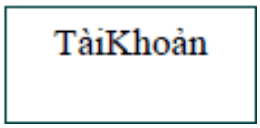
Sự vật hữu hình và lớp thiết bị

Các đối tượng vật lý hoặc các nhóm của đối tượng hữu hình mà có thể cảm nhận trực quan và các thiết bị mà hệ thống tương tác.

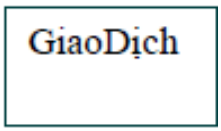
Ví dụ: xe hơi, máy bay, … là các sự vật hữu hình; thiết bị cảm ứng nhiệt là một lớp thiết bị.

Ví dụ: chúng ta cố gắng xác định lại các lớp trong hệ thống ATM dùng phương pháp tiếp cận:

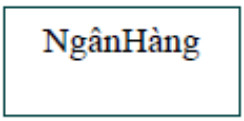
- Các lớp khái niệm:



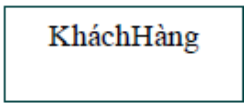
- Các lớp sự kiện:



- Các lớp tổ chức:



- Các lớp con người:



- Các lớp sự vật hữu hình và thiết bị:

c) Cách tiếp cận theo use case

Như chúng ta đã được giới thiệu, use case được dùng để mô hình hoá các kịch bản trong hệ thống và xác định cách thức các tác nhân tương tác với kịch bản. Kịch bản có thể được mô tả bằng

văn bản hoặc thông qua một thứ tự các bước. Một khi hệ thống được mô tả trong ngữ nghĩa các kịch bản. Chúng ta có thể kiểm tra đoạn mô tả văn bản hoặc các bước của mỗi kịch bản để xác định các đối tượng nào cần thiết để cho kịch bản được thực hiện. Chúng ta có thể mô hình hoá các kịch bản của use case sử dụng sơ đồ tuần tự (sequence diagram) hoặc sơ đồ hợp tác (collaboration diagram). Các mô hình này cho phép chúng ta mô hình hoá một cách xác định hơn ở giai đoạn phân tích và trợ giúp trong việc thiết kế hệ thống qua việc mô hình hoá sự tương tác giữa các đối tượng trong hệ thống.

Tuy nhiên, việc mô hình hoá kịch bản của use case một cách quá cụ thể sẽ dễ dẫn đến mô tả hoạt động phần mềm hệ thống nơi mà các đối tượng phần mềm có thể sẽ được xác định (mà đúng ra nó phải được xác định ở giai đoạn thiết kế). Do đó, cách tiếp cận này nên kết hợp với cách tiếp cận phân tích cụm danh từ hoặc cách tiếp cận phân loại để xác định đúng các đối tượng trong giai đoạn phân tích.

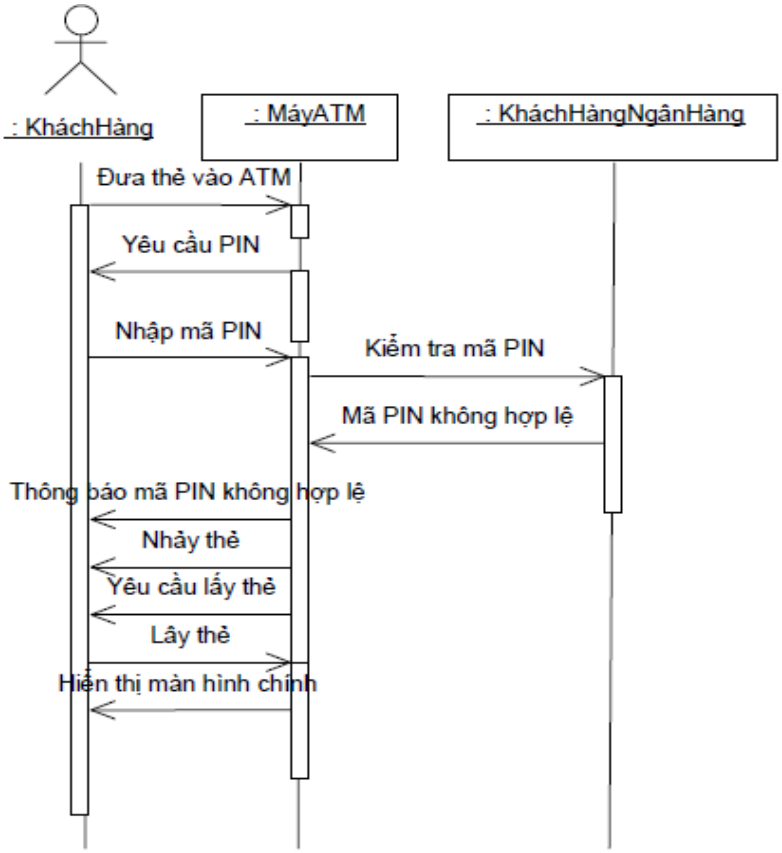
Trước tiên, chúng ta xác định các dòng tương tác của tác nhân với hệ thống trong một use case. Sau đó, chúng ta sẽ đặt câu hỏi "đối tượng nào của hệ thống sẽ chịu trách nhiệm tiếp nhận sự tương tác này?". Trả lời câu hỏi này giúp chúng ta tìm ra đối tượng đầu tiên của use case. Nếu đối tượng này chuyển giao toàn bộ hoặc một phần trách nhiệm xử lý cho đối tượng khác nào đó, thì chúng ta tiếp tục tiếp tục xác định đối tượng đó. Quá trình này cứ tiếp tục cho đến khi hết tất cả các dòng tương tác đã được kiểm tra.

Ví dụ: trong hệ thống ATM chúng ta xem hoạt động của use case "Giải quyết PIN không hợp lệ". Ở đây chúng ta cần nghĩ về tuần tự các hoạt động mà một khách hàng có thể thực hiện:

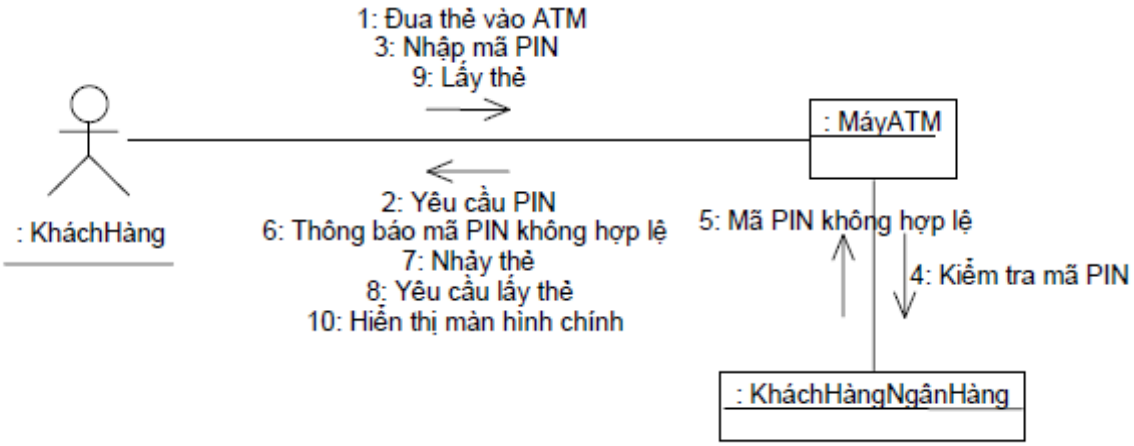
- Đưa vào thẻ ATM
- Nhập mã PIN
- Rút thẻ ATM

Dựa trên các hoạt động này, phản ứng của hệ thống hoặc chấp nhận quyền truy cập của tài khoản tương ứng hoặc từ chối. Kế tiếp chúng ta cần xác định một cách tường minh hơn về hệ thống: Chúng ta đang tương tác với cái gì (của hệ thống)? Máy ATM. Tiếp tục với kịch bản tiếp theo: máy ATM sẽ sử dụng đối tượng nào để kiểm tra mã PIN? Khách hàng ngân hàng.

Một khách hàng trong trường hợp này là bất kỳ người nào muốn truy cập đến một tài khoản thông qua máy ATM, và có thể có hoặc có thể không có tài khoản. Ngược lại, một khách hàng ngân hàng có một tài khoản.

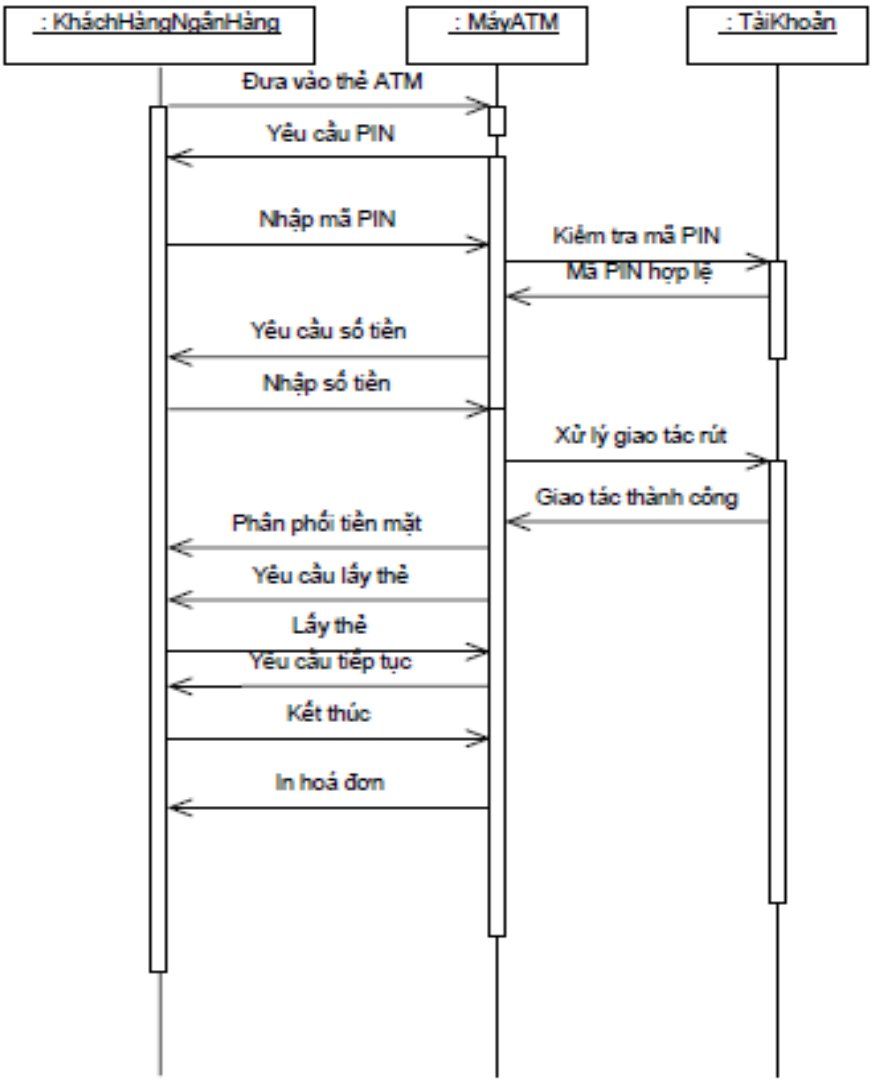

: KháchHàng
: MáyATM
: KháchHàngNgânHàng
Đưa thẻ vào ATM
Yêu cầu PIN
Nhập mã PIN
Kiểm tra mã PIN
Mã PIN không hợp lệ
Thông báo mã PIN không hợp lệ
Nhảy thẻ
Yêu cầu lấy thẻ
Lấy thẻ
Hiển thị màn hình chính

Sơ đồ hợp tác

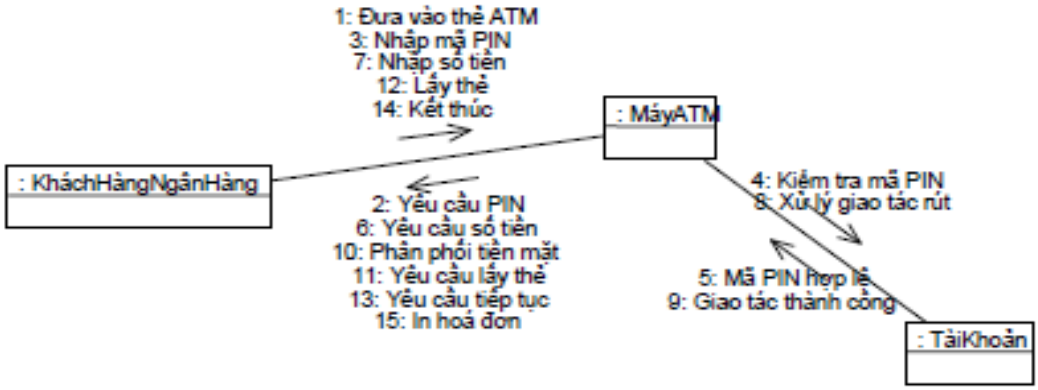

1: Đưa thẻ vào ATM
3: Nhập mã PIN
9: Lấy thẻ
: MáyATM
: KháchHàng
2: Yêu cầu PIN
6: Thông báo mã PIN không hợp lệ
7: Nhảy thẻ
8: Yêu cầu lấy thẻ
10: Hiển thị màn hình chính
5: Mã PIN không hợp lệ
4: Kiểm tra mã PIN
: KháchHàngNgânHàng

Hoặc xét use case "Rút tiền"

Sơ đồ tuần tự



Sơ đồ hợp tác



Như vậy, dựa vào hai sơ đồ tuần tự của hai use case chúng ta đã xác định được các lớp: MáyATM, KháchHàngNgânHàng (KháchHàng), TàiKhoản. Tiếp tục mô hình hoá với sơ đồ tuần tự hoặc hợp tác với các use case còn lại của hệ thống ATM, chúng ta sẽ xác định được các lớp còn lại.

5.4 Xác định mối quan hệ giữa các lớp

Trong môi trường hướng đối tượng, đối tượng đảm nhiệm một vai trò chủ động trong một hệ thống. Đối tượng không tồn tại một cách độc lập mà luôn tương tác với những đối tượng khác. Sự tương tác này thể hiện thông qua mối kết hợp bao gồm luôn các hoạt động và phương thức của các đối tượng. Sau đây chúng ta sẽ được giới thiệu các hướng dẫn giúp xác định ba loại liên kết: mối kết hợp, mối kết hợp thu nạp (thành phần) và mối kết hợp tổng quát hoá.

a) Xác định mối kết hợp (association)

Xác định mối kết hợp bắt đầu bằng việc phân tích sự tương tác giữa các lớp. Thông thường thì bất kỳ có một liên kết phụ thuộc nào giữa hai hay nhiều lớp đều có thể thiết lập mối kết hợp. Một cách xác định chính là kiểm tra trách nhiệm của đối tượng để thiết lập mối kết hợp, nói cách khác, nếu một đối tượng được thiết lập để thi hành một nhiệm vụ và lại thiếu thông tin để thi hành nhiệm vụ đó, thì đối tượng này phải ủy quyền thực hiện lại cho đối tượng khác sở hữu thông tin đó. Có nhiều cách tiếp cận để xác định mối kết hợp, đầu tiên trích lọc các mối kết hợp ứng viên từ việc tham khảo mô tả hệ thống và yêu cầu hệ thống và những tài liệu khác liên quan đến hệ thống. Sau đó, tinh chế dần để chọn ra mối kết hợp có ý nghĩa nhất.

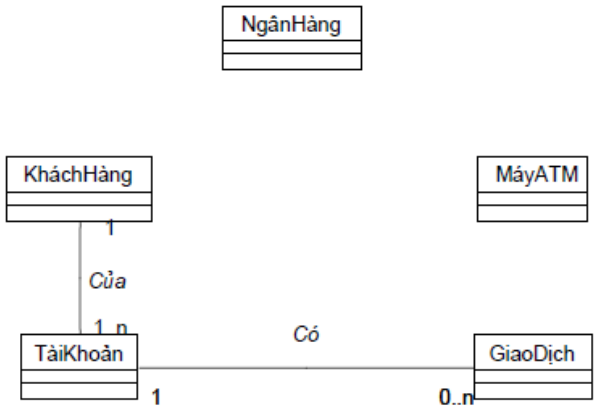
Chú ý rằng mối kết hợp và mối kết hợp thành phần có ngữ nghĩa rất gần nhau, do đó có những trường hợp khó phân biệt, mối kết hợp thành phần chỉ là một trường đặc biệt của mối kết hợp. Tùy theo lãnh vực ứng dụng nếu chúng ta tìm được một cách tự nhiên để biểu diễn mối kết hợp thành phần thì hãy chọn nó, ngược lại hãy chọn mối kết hợp để biểu diễn. Sau đây là các hướng dẫn và các mẫu chung cho phép xác định mối kết hợp:

Hướng dẫn xác định mối kết hợp

- Một sự phụ thuộc giữa hai hay nhiều lớp có thể thiết lập thành mối kết hợp. Mối kết hợp thường tương ứng với một động từ hoặc cụm giới từ như là thành phần của, làm việc cho, chứa trong, …
- Một tham chiếu từ một lớp đến một lớp khác là một mối kết hợp.

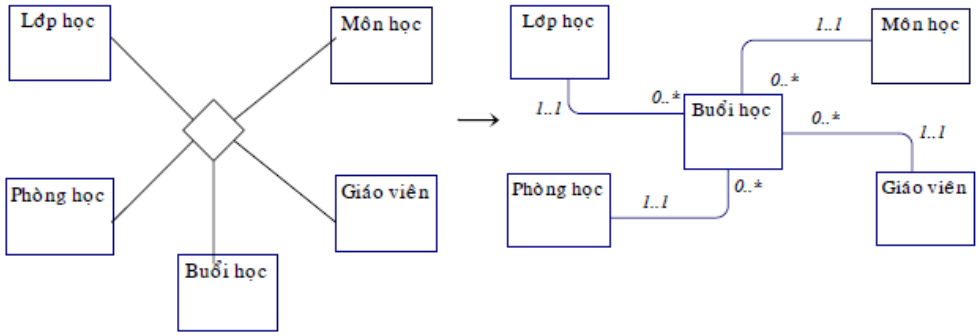
Mẫu chung xác định mối kết hợp

- Mối kết hợp vị trí (location): liên kết tới, thành phần của, chứa trong, làm việc tại, ….
- Mối kết hợp sở hữu: của, có, thuộc,… Mối kết hợp có giữa lớp TàiKhoản và lớp GiaoDịch, mối kết hớp của giữa lớp TàiKhoản và lớp KháchHàng
- Mối kết hợp truyền thông, liên lạc (communication): đặt tới, trao đổi với, gởi cho, tiếp nhận từ,…

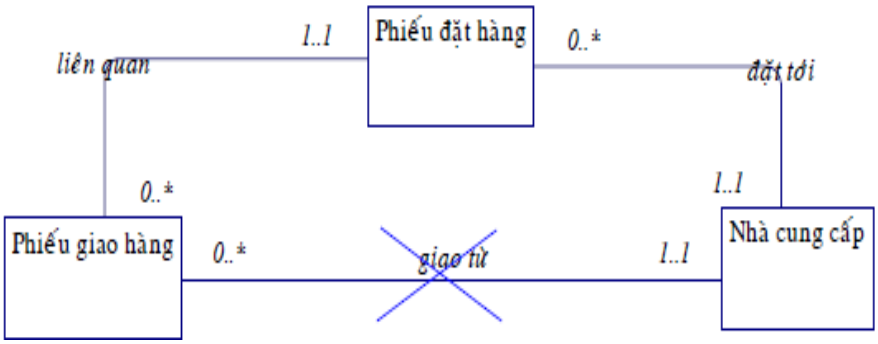


Loại bỏ những mối kết hợp không cần thiết

- *Mối kết hợp cài đặt:* mối kết hợp cài đặt nên được đưa vào trong quá trình thiết kế và cài đặt hệ thống. Mối kết hợp cài đặt là mối kết hợp mô tả sự liên quan giữa các lớp trong giai đoạn thiết kế cài đặt hệ thống bên trong môi trường phát triển hoặc ngôn ngữ lập trình cụ thể và không phải là môi liên kết giữa các đối tượng mô tả nghiệp vụ.
- *Mối kết hợp đa phân:* là mối kết hợp giữa ba lớp trở lên, mối kết hợp này phức tạp trong cách thể hiện. Nếu có thể, phát biểu lại nó dùng mối kết hợp nhị phân.



- *Mối kết hợp trực tiếp dư thừa (directed action):* là các mối kết hợp được định nghĩa trong ngữ nghĩa của những mối kết hợp khác (còn gọi là mối kết hợp suy diễn hoặc bắc cầu). Những mối kết hợp loại này là dư thừa do đó cần loại bỏ nó khỏi mô hình. Trong các mối kết hợp dưới đây liên quan, đặt tới, giao từ thì mối kết hợp giao từ có thể được suy ra từ mối kết hợp liên quan và đặt tới. Vậy mối kết hợp giao từ là dư thừa và có thể loại bỏ nó khỏi mô hình.

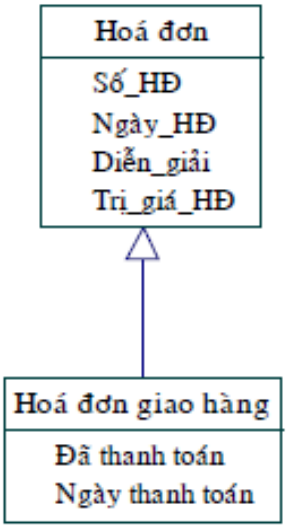


b) Xác định mối kết hợp tổng quát – chuyên biệt (generalization)

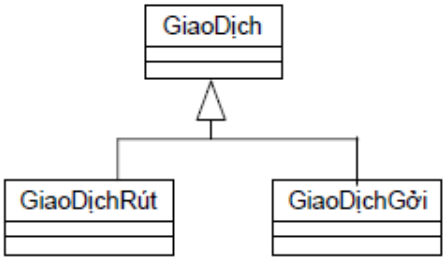
Mối kết hợp tổng quát hoá – chuyên biệt thể hiện quan hệ kế thừa giữa các lớp và một cấu trúc phân cấp xác định những dòng kế thừa này. Quan hệ này cho phép các đối tượng có thể xây dựng từ những đối tượng khác. Một thuận lợi chính khi sử dụng mối kết hợp này là chúng ta có thể xây dựng trên những gì đang có và quan trọng hơn là sử dụng lại cái đã có. Sau đây là các tiếp cận hướng dẫn xác định mối kết hợp tổng quát hoá:

- *Tiếp cận trên xuống (top-down):* Từ một lớp chúng ta tìm kiếm cụm danh từ chứa tên lớp và tính từ (hoặc danh từ). Đánh giá xem cụm danh từ này có thể là một trường hợp đặc biệt cần được quản lý trong hệ thống không; tìm kiếm xem có những đặc trưng riêng của lớp này mà hệ thống cần chỉ ra hay không. Nếu quyết định là một lớp thì nó là một loại chuyên biệt của lớp ban đầu.

  Ví dụ: từ lớp Hoá đơn chúng ta tìm thấy một cụm danh từ Hoá đơn giao hàng có chứa từ Hoá đơn, và trường hợp dưới đây quyết định Hoá đơn giao hàng là một lớp chuyên biệt của lớp



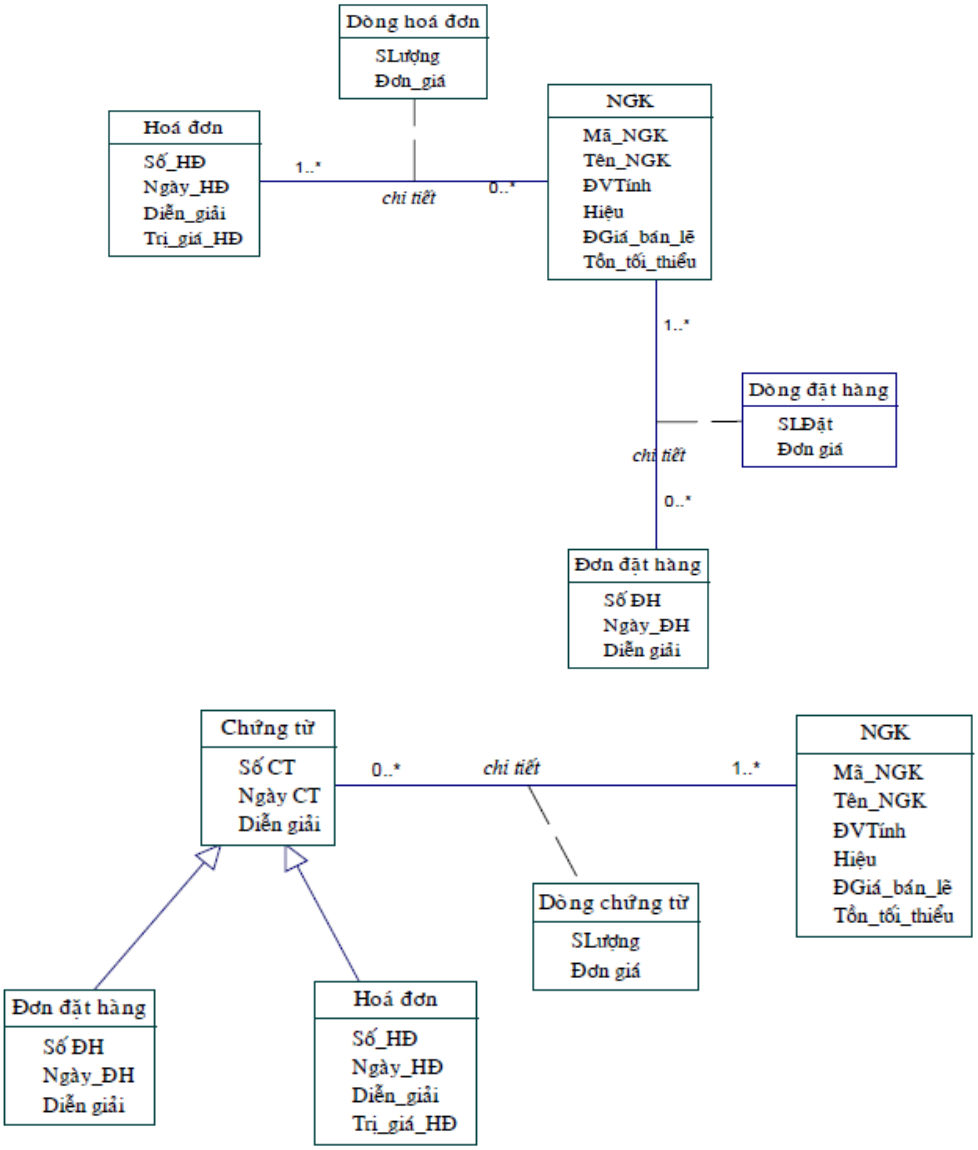
Hoặc trong hệ thống ATM, từ lớp GiaoDịch chúng ta có thể chuyên biệt hoá xuống hai lớp con GiaoDịchRút và GiaoDịchGởi.



Trong quá trình tạo cấu trúc phân cấp tổng quát – chuyên biệt, chúng ta không nên quá lạm dụng việc chuyên biệt hóa mà đưa vào tất cả các lớp chuyên biệt không cần thiết. Các lớp chuyên biệt cần thiết là các lớp có những đặc trưng (thuộc tính, hành vi và mối kết hợp) riêng được biểu diễn trong hệ thống. Ví dụ, có rất nhiều loại hoá đơn có thể tạo lớp chuyên biệt như là: hoá đơn bán lẽ, hoá đơn bán sĩ, hoá đơn giao hàng. Trong khi đó, chỉ có Hoá đơn giao hàng là có những thuộc

tính riêng nên nó được biểu diễn trong hệ thống, còn hai lớp còn lại không cần thiết vì không tìm ra đặc trưng riêng của nó.

- *Tiếp cận dưới lên (bottom-up):* tìm kiếm trong các lớp để xác định xem có các thuộc tính và phương thức giống nhau. Sau đó chúng ta có thể gom nhóm và đưa các thuộc tính và phương thức chung này lên một lớp tổng quát (trừu tượng). Tạo mối kết hợp tổng quát hoá từ các lớp này đến lớp tổng quát mới xác định. Mô hình dưới đây cho thấy, các lớp Hoá đơn và Đơn đặt hàng có ba thuộc tính giống nhau (Hoá đơn: Số HĐ, Ngày HĐ, Diễn giải. Đơn đặt hàng: Số ĐH, Ngày ĐH, Diễn giải) và mối kết hợp với lớp NGK giống nhau. Do đó, chúng ta tạo ra một lớp tổng quát Chứng từ và di chuyển cả ba thuộc tính và mối kết hợp lên lớp này và tạo ra mối kết hợp tổng quát hoá từ lớp



- Tiếp cận tái sử dụng: di chuyển các thuộc tính và phương thức lên càng cao càng tốt trong cấu trúc phân cấp.
- Đa kế thừa: tránh sử dụng quá mức đa kế thừa trong mô hình vì đa kế thừa sẽ phức tạp trong việc xác định đường dẫn kế thừa một hành vi từ những lớp nào.

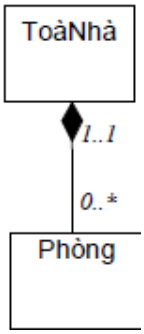
c) Xác định mối quan hệ thành phần (a-part-of, aggregation)

Mối quan hệ thành phần được sử dụng trong tình huống một lớp hình thành bao gồm những lớp thành phần. Hai đặc trưng chính của mối quan hệ thành phần là tính bắc cầu và tính phản đối xứng.

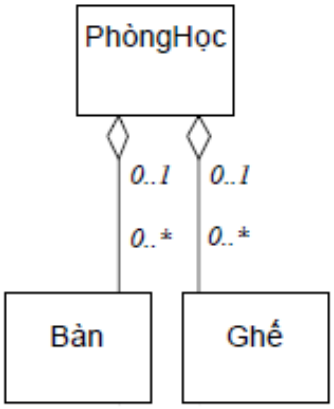
- Tính bắc cầu: nếu lớp A là một thành phần của lớp B và lớp B là thành phần của lớp C, thì lớp A là thành phần của lớp C.
- Tính phản đối xứng: nếu lớp A là thành phần của lớp B thì lớp B không phải là thành phần của lớp A.

Việc xác định mối quan hệ thành phần có thể dựa trên các mẫu chỉ dẫn sau:

- Tập hợp: một đối tượng vật lý được hình thành từ các đối tượng vật lý thành phần khác.

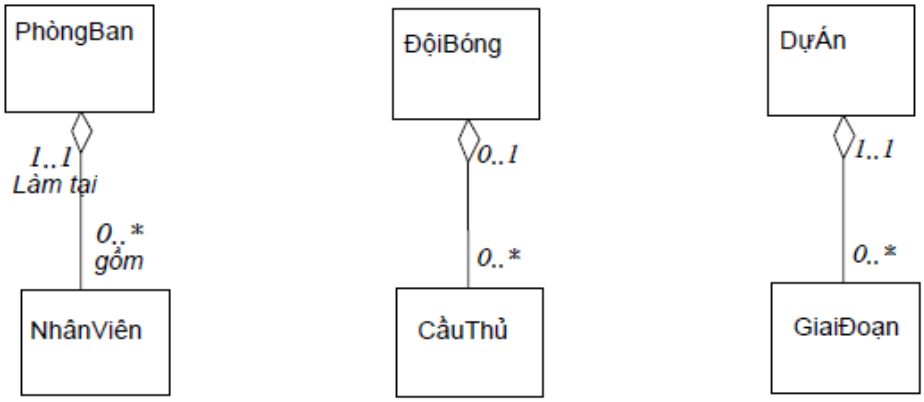


- Vật chứa: một đối tượng vật lý chứa đựng các thành phần nhưng không được cấu tạo bởi các thành phần.



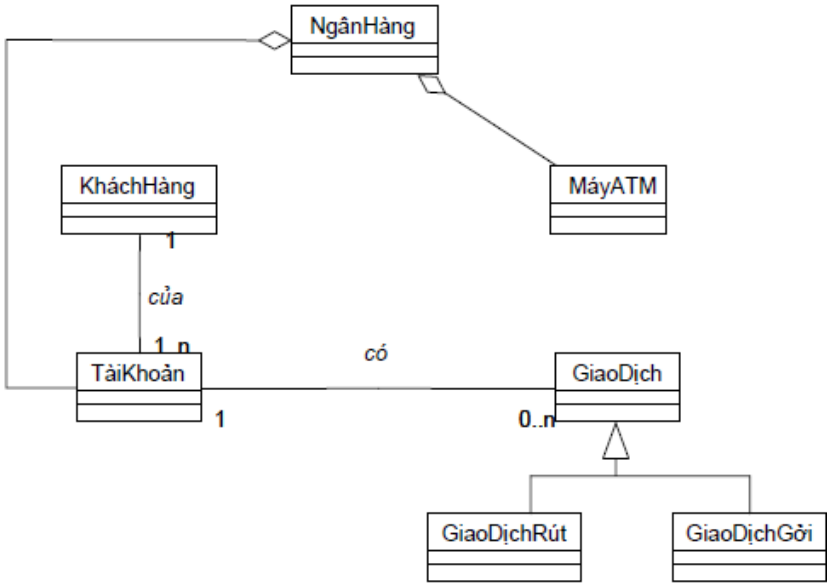
Một phòng học chứa đựng tất cả các đối tượng bàn ghế nhưng không được cấu tạo từ những đối tượng này.

- Tập hợp – thành viên: một đối tượng quan niệm chứa các thành phần có thể vật lý hoặc quan niệm.



Các lớp PhòngBan, ĐộiBóng, DứAn, GiaiĐoạn là các lớp quan niệm; các lớp NhânViên, CầuThủ là các lớp vật lý.

Trong hệ thống ATM, một ngân hàng bao gồm các máy ATM, các tài khoản, các toà nhà, các nhân viên,.v.v… Tuy nhiên, các đối tượng toà nhà, nhân viên,… không thuộc phạm vi hệ thống đang xét. Do đó, chúng ta định nghĩa mối kết hợp thành phần giữa lớp NgânHàng và các lớp: MáyATM, TàiKhoản.



d) Xác định thuộc tính (attribute) và phương thức (method) của lớp

Xác định thuộc tính và phương thức cũng là một công việc khó như là xác định lớp, và đây cũng là một tiến trình lặp. Thông thường người ta cũng dựa vào use case và các sơ đồ UML khác để xác định các thuộc tính và phương thức của lớp.

Thuộc tính là các thành phần mà một đối tượng phải ghi nhớ như là màu sắc, trị giá, hảng sản xuất,… Xác định thuộc tính của một lớp hệ thống bắt đầu với việc tìm hiểu về các trách nhiệm của hệ thống. Và chúng ta đã xác định răng, trách nhiệm của hệ thống có thể được nhận định qua việc phát triển các use case và các đặc điểm mong muốn của ứng dụng như là xác định thông tin gì mà người dùng cần cho hệ thống. Các câu hỏi sau đây có thể giúp xác định nhiệm vụ của lớp và các thành phần dữ liệu mà hệ thống muốn lưu trữ.

- Thông tin gì về đối tượng sẽ được lưu trữ?

- Dịchvụ gì mà một lớp phải cung cấp?
- Trả lời câu hỏi thứ nhất giúp chúng ta xác định các thuộc tính của một lớp. Trả lời câu hỏi thứ hai giúp chúng ta xác định các phương thức của lớp.

Xác định thuộc tính

Bằng việc phân tích các use case, các yêu cầu, các mô tả và các sơ đồ chúng ta có thể bắt đầu hiểu trách nhiệm của lớp và cách thức mà các lớp tương tác để thi hành công việc. Mục tiêu chính ở đây là để hiểu những gì mà một lớp có trách nhiệm về tri thức.

Sau đây là một số hướng dẫn giúp xác định lớp trong các use case:

- Thuộc tính thường tương ứng tới các danh từ đi theo bởi các cụm phó từ như là: chi phí của sản phẩm. Các thuộc tính cũng có thể tương ứng tới các tính từ hoặc các phó từ.
- Giữ cho lớp đơn giản: chỉ dùng đủ thuộc tính để diễn đạt trạng thái đối tượng
- Các thuộc tính ít có thể được mô tả đầy đủ trong mô tả vấn đề. Do đó, chúng ta phải sử dụng tri thức về lãnh vực ứng dụng và thực tế để tìm chúng.
- Không nên quan tâm quá về việc phải khám phá hết thuộc tính. Chúng ta có thể bổ sung thêm các thuộc tính trong các vòng lặp tiếp theo.

Ví dụ: xác định các thuộc tính cho các lớp của hệ thống ATM.

Xác định thuộc tính cho lớp KháchHàng

Bằng việc phân tích use case, các sơ đồ tuần tự và hợp tác và sơ đồ hoạt động, rõ ràng rằng, với lớp KháchHàng, lãnh vực bài toán và hệ thống đưa ra một vài thuộc tính. Tìm kiếm trong sơ đồ tuần từ của use case "Xứ lý PIN không hợp lệ" chúng ta tìm thấy rằng lớp KháchHàng phải có một mã PIN (hay password) và số thẻ. Do đó, mãPIN và sốThẻ là hai thuộc tính thích hợp của lớp KháchHàng. Các thuộc tính khác của KháchHàng là các biểu diễn tri thức chung về khách hàng, do đó các thuộc tính của lớp KháchHàng là:

tênKháchHàng

họKháchHàng

mãPIN

sốThẻ

trong thời điểm này chúng ta chỉ quan tâm đến chức năng của đối tượng khách hàng mà không quan tâm đến các thuộc tính cài đặt

Xác định thuộc tính cho lớp TàiKhoản

Tương tự các thuộc tính của lớp TàiKhoản được xác định là:

sốTàiKhoản

loạiTàiKhoản

sốDư

Xác định thuộc tính cho lớp GiaoDịch

giaoDịchID

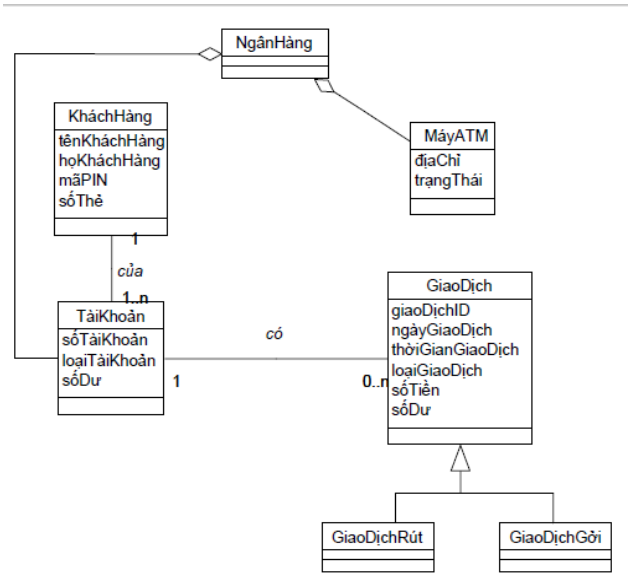
ngàyGiaoDịch

thờiGianGiaoDịch

loạiGiaoDịch

sốTiền

sốDư



Xác định thuộc tính cho lớp MáyATM

Máy ATM là một đối tượng vật lý và hữu hình, do đó thuộc tính của nó dùng để mô tả vị trí và trạng thái của máy.

địaChỉ

trạngThái

Xác định phương thức

Phương thức và thông điệp (message) là các thành phần gánh vác công việc xử lý hệ thống hướng đối tượng. Trong một môi trường hướng đối tượng, mỗi phần dữ liệu hoặc đối tượng được bao quanh bởi một tập hợp các thường trình gọi là các phương thức. Những phương thức này chính là các dịch vụ và các toán tử của một lớp định nghĩa để cài đặt hànhvi của các đối tượng thành viên của lớp. Các phương thức chính là cách thức mà đối tượng thực hiện tương tác với các đối tượng khác trong hệ thống. Phát hiện các phương thức để cài đặt các hành vi đối tượng là một hoạt động quan trọng trong giai đoạn phân tích.

Như là thuộc tính, một lớp sẽ có những phương thức nội bộ (private) và toàn cục (puplic). Trong giai đoạn phân tích, chúng ta chỉ quan tâm phát hiện các phương thức toàn cục mà không quan tâm đến các phương thức nội bộ của đối tượng. Các phương thức thường tương ứng với các truy vấn về các thuộc tính của đối tượng. Nói cách khác, các phương thức chịu trách nhiệm quản lý các trị của thuộc tính như là truy vấn, cập nhật, đọc và ghi. Chú ý rằng trong giai đoạn phân tích, chú ta đang làm việc ở mức cao của sử trừu tượng hoá. Do đó, các phương thức như là tạo (constructor) hoặc các phương thức mô tả chi tiết việc cài đặt sẽ được phát hiện trong giai đoạn thiết kế.

Trong phần này chúng chúng ta xem xét cách thức xác định phương thức sử dụng các sơ đồ UML như là: sơ đồ trạng thái, sơ đồ hoạt động, sơ đồ tuần tự/hợp tác và sơ đồ use case.

Xác định phương thức bằng việc phân tích các sơ đồ UML và use case

Trong sơ đồ tuần tự, các đối tượng được đặt trong sơ đồ với các đường đứt quảng thẳng đứng. Do đó, các sự kiện xảy ra giữa các đối tượng được đặt theo dòng nằm ngang. Một sự kiện

được xem như là một hành động để chuyển thông tin. Mặt khác, những hành động này là các toán tử mà đối tượng phải thực thi.

Ví dụ, để xác định các phương thức của lớp TàiKhoản, chúng ta xe xét các sơ đồ tuần tự ứng với các use case:

Rút tiền

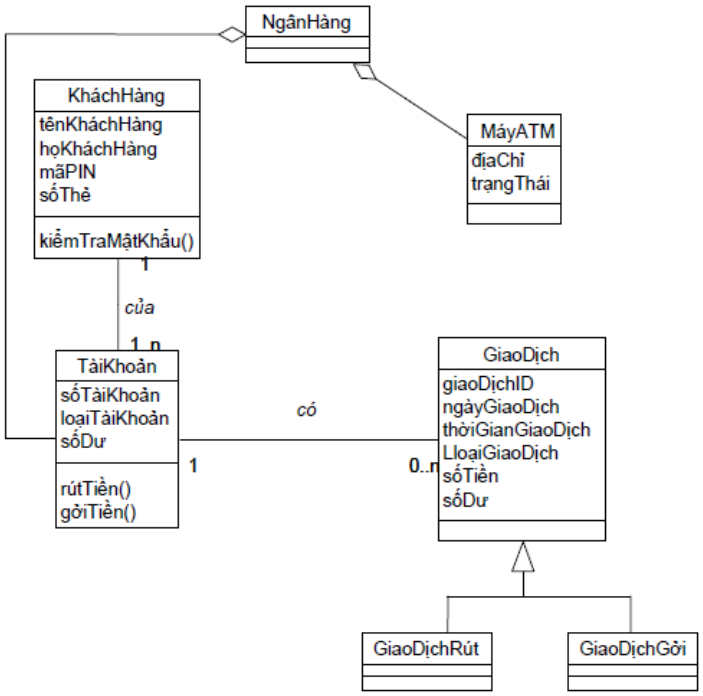
Gởi tiền

Xem thông tin tài khoản

Sơ đồ tuần tự có thể trợ giúp chúng ta xác định các dịch vụ mà các đối tượng phải cung cấp. Ví dụ, qua việc nghiên cứu sơ đồ tuần tự của use case Rút tiền, chúng ta thấy rằng lớp TàiKhoản phải cung cấp dịch vụ rútTiền. Qua việc nghiên cứu use case Gởi tiền, lớp TàiKhoản phải cung cấp dịch vụ gởiTiền.

Tương tự, các dịch vụ lớp KháchHàng:

kiểmTraMậtKhẩu (kiểm tra mật khẩu của khách hàng)



Chú ý, các dịch vụ được đưa vào lớp tổng quát sẽ được thừa kế trong các lớp chuyên biệt.

CHƯƠNG 4. THIẾT KẾ HỆ THỐNG

1. Thiết kế lớp

1.1 Các tiên đề và hệ luận trong thiết kế hướng đối tượng

a) Các tiên đề

Theo Suh, trong tiếp cận thiết kế hướng đối tượng có hai tiên đề được áp dụng, các tiên đề là:

Tiên đề 1: tiên đề độc lập.

Duy trì sự độc của các thành phần: Trong quá trình thiết kế, chúng ta đi từ yêu cầu và use case đến một thành phần hệ thống, mỗi thành phần phải thoã mãn yêu cầu mà không ảnh hưởng đến những thành phần khác.

Tiên đề 2: tiên đề thông tin.

Giảm tối đa nội dung thông tin thiết kế. Tiên đề này nói về tính đơn giản hoá trong thiết kế. Mục đích chính là giảm tối đa tính phức tạp, như vậy các đối tượng sẽ dễ bảo trì và nâng cấp hơn. Trong thiết kế hướng đối tượng, cách tốt nhất để giảm độ phức tạp là sử dụng tính thừa kế. (xem hệ luận 6)

b) Các hệ luận

Từ hai tiên đề trên, có nhiều hệ luận được trích dẫn. Những hệ luận này sẽ mang lại lợi ích hơn trong việc quyết định thiết kế các tình huống cụ thể, vì chúng có thể áp dụng tới các tình huống thực tế dễ dàng hơn so với tiên đề gốc ban đầu.

Hệ luận 1: thiết kế độc lập, giảm tối đa thông tin trao đổi (Uncouple Design with Less Information)

Sự cố kết giữa các đối tượng cao có thể làm giảm tính liên kết giữa chúng. Bởi vì chỉ một lượng nhỏ các thông tin cần thiết trao đổi giữa các đối tượng đó.

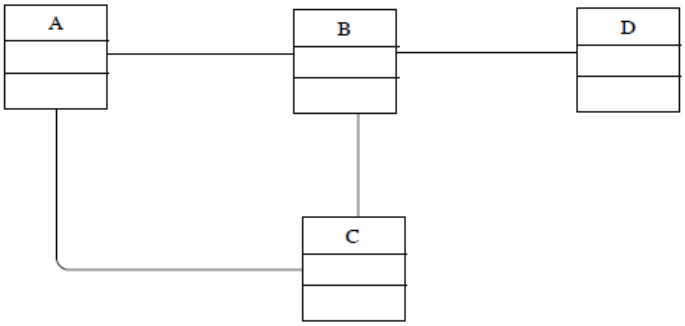
Coupling: Ở giai đoạn thiết kế, coupling được dùng để đo mức độ liên kết giữa các đối tượng hoặc giữa thành phần phần mềm. Coupling là một mối kết hợp nhị phân, và là một khái niệm quan trọng khi đánh giá một thiết kế bởi vì nó giúp chúng ta tập trung đúng vào vấn đề quan trọng của thiết kế. Ví dụ, một thành phần trong hệ thống thay đổi sẽ có một tác động tối thiểu đến những thành phần khác. Coupling trong các đối tượng càng mạnh càng mạnh thì hệ thống càng phức tạp. Mức độ của coupling có thể được đánh giá trên:

- Mức độ phức tạp của kết nối
- Kết nối tham chiếu đến chính bản thân đối tượng hoặc bên ngoài đối tượng
- Các thông điệp nhận và gửi đi

Mức độ coupling giữa hai thành phần được xác định bằng mức độ phức tạp của thông tin trao đổi giữa chúng. Coupling càng gia tăng thì càng làm gia tăng độ phức tạp hoặc mơ hồ, tối nghĩa của giao diện. Coupling càng giảm khi sự kết nối được thiết lập ở thành phần giao diện thay vì ở

thành phần bên trong. Trong thiết kế hướng đối tượng, có hai loại coupling là coupling tương tác và coupling thừa kế:

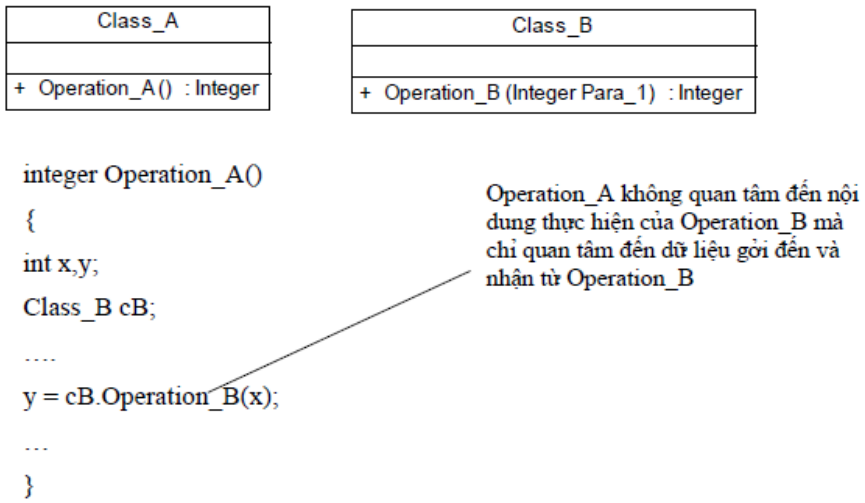
Coupling tương tác: thể hiện qua số lượng và độ phức tạp của các thông điệp giữa những thành phần, sự tương tác càng ít thì càng được ưu tiên. Coupling cũng áp dụng tới mức độ phức tạp của thông điệp. Một chỉ dẫn chung là giữ các thông điệp càng đơn giản và càng ít xảy ra thì càng tốt. Tổng quát, nếu một kết nối thông điệp gồm ba tham số trở lên thì kiểm tra xem có thể làm đơn giản hoá nó nữa được không. Một đối tượng được kết nối với nhiều thông điệp phức tạp thì gọi là liên kết chặt (tightly coupled), có nghĩa là bất kỳ có sự thay đổi nào có thể tác động đến sự thay đổi trong những cái khác.



Hình 4. Ví dụ về biểu diễn coupling và B là liên kết chặt

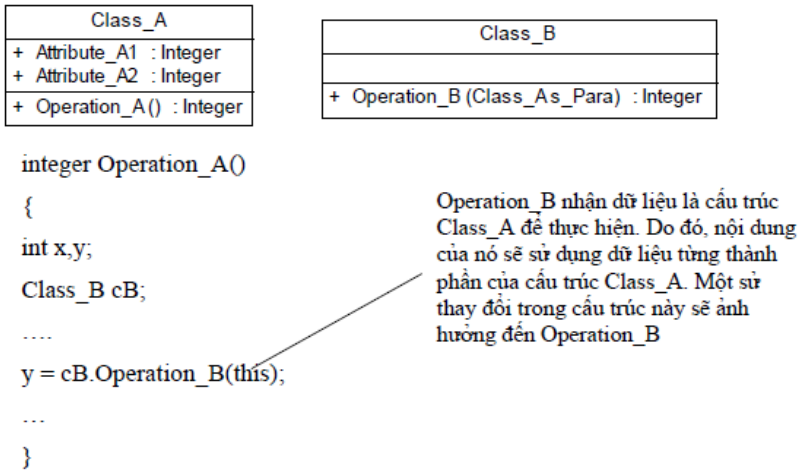
Năm mức độ coupling tương tác

Data coupling: kết nối giữa các thành phần hoặc là dữ liệu dạng nguyên tố hoặc là hoặc cấu trúc tổng hợp tất cả yếu tố được dùng bởi đối tượng nhận. Đây là loại coupling tốt nhất và là mục đích của thiết kế kiến trúc. Các đối tượng trao đổi với nhau thông qua các dữ liệu thành tố hoặc các cờ hiệu thông tin. Thành phần này không quan tâm đến bất cứ gì bên trong thành phần khác mà chỉ quan tâm đến dữ liệu gì nó cần và dữ liệu gì nó gởi trả.



Stamp coupling: trong stamp coupling, dữ liệu trao đổi là một phần của cấu trúc hoặc toàn bộ cấu trúc. Loại coupling này được đánh giá là thấp hơn so với data coupling bởi vì sự trao đổi là một cấu trúc thay vì một yếu tố đơn nên làm cho nó phức tạp hơn. Một thay đổi trong cấu trúc sẽ ảnh hưởng đến tất cả đối tượng sử dụng nó. Stamp coupling làm cho các thành phần phụ thuộc nhiều hơn đến

thành phần khác, bởi vì, để tránh những lỗi có thể xảy ra, những thành phần sẽ phải có hiểu biết về hoạt động bên trong của thành phần khác sử dụng cùng cấu trúc dữ liệu.



Control coupling: khi một thành phần thành phần gởi một thông tin điều khiển tới một thành phần khác, hai thành phần này được gọi là có control coupling. Thông tin điều khiển có thể xuất hiện dưới dạng cờ hiệu thông báo cho thành phần khác hành động sẽ thực hiện. Khi thông tin điều khiển được gởi đi, thành phần gởi phải biết về hoạt động bên trong của thành phần nhận, do đó nó tạo ra một sự phụ thuộc giữa các thành phần.

Common coupling: khi hai thành phần tham khảo đến cùng một vùng dữ liệu chung toàn cục thì có liên kết common coupling. Liên kết này cũng nên tránh bởi vì nó tạo ra một cơ hội lớn về lỗi trên toàn bộ hệ thống. Một lỗi phát sinh trong bất kỳ một thành phần nào sử dụng dữ liệu toàn cục này có thể gây ra lỗi trong bất kỳ thành phần khác sử dụng cùng dữ liệu này.

Content coupling: loại coupling được xếp thấp nhất là content coupling. Bởi vì loại này giới thiệu một thành phần tham khảo trực tiếp hoạt động bên trong của một thành phần khác. Ví dụ, một method có thể thay đổi dữ liệu trong một method khác hoặc thay đổi một đoạn code trong method khác. Hiện nay, các ngôn ngữ lập trình (đặc biệt là ngôn ngữ lập trình hướng đối tượng) đều không cho phép tạo ra content coupling.

Tên coupling	Xếp hạng phụ thuộc
Data coupling	Rất thấp
Stamp coupling	Thấp
Control coupling	Trung bình
Common coupling	Cao
Content coupling	Rất cao

Coupling thừa kế: là coupling giữa lớp tổng quát và lớp chuyên biệt. Một lớp chuyên biệt liên kết với lớp tổng quát của nó thông qua thuộc tính và method. Không như coupling tương tác, coupling thừa kế càng cao thì càng ưu tiên. Tuy nhiên, để đạt được mức độ cao coupling thừa kế trong một hệ thống. Mỗi lớp chuyên biệt không nên thừa kế những method và thuộc tính không liên quan hoặc không cần thiết. Ví dụ, nếu một lớp chuyên biệt chồng lên hầu hết các method hoặc không sử dụng nó từ lớp tổng quát, điều này cho thấy coupling thừa kế là thấp và lúc đó thiết kế viên phải thay đổi lại cách xây dựng tổng quát hoá và chuyên biệt hoá này.

Cohesion

Trong khi coupling đề cập đến mức độ tương tác giữa các đối tượng thành phần thì cohesion lại đề cập đến sự tương tác bên trong một đối tượng thành phần. Cohesion phản ánh tính "đơn mục đích" của một đối tượng. Tất cả các lệnh, chức năng trong một thành phần được định nghĩa gắn liền tới một nhiệm vụ. Trong hệ thống nếu các thành phần đạt được mức độ cohesion càng cao thì mức độ liên kết coupling giữa các thành phần càng yếu, do đó để đạt được đỉnh cao của cohesion thì chúng ta phải làm giảm mức độ coupling. Cohesion cũng trợ giúp trong thiết kế lớp bằng việc giúp xác định mục đích và mục tiêu của lớp một cách rõ ràng.

Method cohesion: một method chỉ nên đảm nhận một chức năng hoặc một nhiệm vụ. Thông thường cách đặt tên của method cũng phải ngụ ý nói lên chức năng liên quan của nó. Ví dụ: Chon_nha_cung_cap(), Tinh_ton(),…

Class cohesion: các method và các thuộc tính trong một lớp phải có mức độ cohesion cao, nghĩa là phải được dùng bởi các method bên trong lớp hoặc chính là method phục vụ cho mục đích của lớp.

Cohesion thừa kế: các lớp chuyên biệt và lớp tổng quát phải cùng mô tả về một loại đối tượng của hệ thống. Các thuộc tính và hành vi của lớp tổng quát phải được thừa hưởng tối đa trong lớp chuyên biệt theo cách hoặc là phục vụ cho hành vi của lớp chuyên biệt hoặc trở thành một hành vi của lớp chuyên biệt.

Hệ luận 2: đơn mục đích (single purpose)

Mỗi lớp phải có một mục đích xác định trong việc đạt được mục tiêu hệ thống. Nếu chúng ta mô tả một lớp phức tạp thì hãy phân chia nó thành những lớp nhỏ hơn, đơn giản và xác định hơn. Mỗi method chỉ cung cấp một dịch vụ, kích thước không quá lớn (không nên vượt quá một trang coding)

Hệ luận 3: nhiều lớp đơn. Tạo các lớp đơn để cho phép tái sử dụng

Kết quả thiết kế hệ thống tốt chính là tạo ra một tập lớn các lớp đơn giản hơn là một tập nhỏ lớp phức tạp. Các lớp càng ít chuyên biệt thì các vấn đề trong tương lai càng khó giải quyết hơn thông qua việc kết nối các lớp đang có và tái sử dụng chúng.

Thiết kế hướng đối tượng luôn đề xuất việc sản sinh thư viện các thành phần tái sử dụng và nhấn mạnh trên tính bao bọc (encapsulation), đơn vị hoá (modularization), và tính đa hình (polymorphism) để tái sử dụng khi xây dụng một đối tượng hơn là làm mới.

Hệ luận 4: ánh xạ kết quả giữa các giai đoạn phải chặt chẽ (strong mapping)

Giai đoạn phân tích và thiết kế dựa trên cùng mô hình, kết quả mô hình. Từ quá trình phân tích đến cài đặt, các chi tiết sẽ được đưa thêm vào nhưng vẫn duy trì về cơ bản giống nhau.

Ví dụ, trong giai đoạn phân tích chúng ta xác định lớp Hoá đơn. Trong giai đoạn thiết kế chúng ta thiết kế lớp Hoá đơn: method, dữ liệu,…

Hệ luận 5: chuẩn hoá (standardization).

Đề xuất sự chuẩn hoá bằng việc thiết kế các thành phần có thể thay thế và tái sử dụng các thành phần có sẵn.

Để tái sử dụng, chúng ta phải có một hiểu biết sâu về các lớp trong môi trường lập trình hướng đối tượng chúng ta đang dùng. Hầu hết các hệ thống hướng đối tượng đều có các các lớp thư viện xây dựng sẵn và ngày càng gia tăng khi chúng ta tạo các ứng dụng mới. Tri thức về các lớp tồn tại giúp chúng ta xác định những gì một lớp mới cần có do thừa kế hơn là phải xây dựng mới. Mặc dù vậy, tài liệu mô tả các lớp thư viện thường không được biên tập tốt hoặc ít được cập nhật thường xuyên khi có sự điều chỉnh, nâng cấp. Do đó, trong quá trình xây dựng các hệ thống chúng ta nên tạo ra các lớp thư viện và tích lũy dần nó từ việc xây dựng các hệ thống và luôn luôn xem xét việc thừa kế nó khi xây dựng một hệ thống mới.

Hệ luận 6: thiết kế thừa kế (design inheritance).

Các đặc tính chung phải được chuyển lên lớp tổng quát. Cấu trúc lớp tổng quát – lớp chuyên biệt phải tạo ra ý nghĩa luận lý.

1.2 Thiết kế lớp

Một tiến trình thiết kế lớp bao gồm:

- Tinh chế thuộc tính
- Thiết kế hành vi (method) và nghi thức (protocol) sử dụng sơ đồ trong UML
- Tinh chế quan hệ giữa các lớp
- Tinh chế sự phân cấp và thiết kế sự kế thừa

a) Phạm vi ảnh hưởng của lớp

Trong việc thiết kế hành vi và thuộc tính cho các lớp, chúng ta gặp phải hai vấn đề. Thứ nhất là nghi thức (protocol), hoặc giao diện tới các toán tử và sự có thể thấy (visibility) của nó; thứ hai là cách thức cài đặt nó. Nghi thức được xem là cách thức xác định các đặc trưng của lớp có thể "nhìn thấy" từ bên ngoài hoặc cỉ được xem là "nội bộ" bên trong. Các nghi thức đó là: toàn cục (public protocol), riêng (private protocol), và bảo vệ (protected protocol).

- pupblic protocol: định nghĩ các hành vi trạng thái của lớp
- private protocol: dùng để xác định các đặc trưng của lớp chỉ được truy cập bởi chính lớp đó
- Protected protocol: xác định đặc trưng của một lớp có thể sử dụng bởi chính nó và lớp con của nó.

b) Tinh chế thuộc tính

Các thuộc tính trong giai đoạn phân tích hướng đối tượng phải được tinh chế theo cách nhìn về việc cài đặt nó trong môi trường tin học. Trong giai doạn phân tích, chúng ta chỉ cần tên của thuộc tính là đủ. Tuy nhiên, trong giai đoạn thiết kế, thông tin chi tiết về thuộc tính đó phải được thêm vào. Mục tiêu chính của hoạt động này là tinh chế các thuộc tính tồn tại (được xác định trong

giai đoạn phân tích) và đưa thêm các thuộc tính dùng cho việc cài đặt nhằm đặc tả lớp như một thành phần của môi trường tin học.

Kiểu thuộc tính

Có ba kiểu cơ bản của thuộc tính:

- Thuộc tính đơn trị
- Thuộc tính đa trị
- Thuộc tính dùng để tham chiếu tới các đối tượng khác hoặc tới một thể hiện kết nối

Các thuộc tính thể hiện cho trạng thái của đối tượng. Khi một trạng thái của đối tượng thay đổi, sự thay đổi này được phản ánh qua giá trị của các thuộc tính. Thuộc tính đơn trị là loại phổ biến nhất, nó chỉ có một giá trị hoặc một trạng thái tương ứng với một đối tượng. Ví dụ: Tên, Ngày sinh, Mức lương,…

Thuộc tính đa trị có thể có một tập các giá trị tương ứng cho một đối tượng. Ví dụ: chúng ta muốn lưu trữ nhiều số điện thoại của một khách hàng, chúng ta có thể đặt thuộc tính Số điện thoại là đa trị.

Thuộc tính tham chiếu được dùng để cung cấp việc tham khảo cần thiết để một đối tượng đáp ứng các trách nhiệm của nó. Nói cách khác, thuộc tính tham chiếu hiện thực hoá mối kết hợp của các đối tượng.

Hiển thị thuộc tính

Trong UML việc trình bày thuộc tính được đề nghị như sau:

<Phạm vi> <tên> : <kiểu thuộc tính> = <giá trị khởi tạo>

Trong đó, *<phạm vi>* sẽ là:

+ : toàn cục (public protocol)

: bảo vệ (protected protocol)

- : cục bộ (private protocol)

<kiểu thuộc tính> là một đặc tả cài đặt thuộc tính độc lập ngôn ngữ.

<giá trị khởi tạo> là một biểu thức độc lập ngôn ngữ xác định giá trị khởi tạo khi một đối tượng được tạo mới. tham số này là tuỳ chọn.

Thuộc tính đa trị được xác định bằng việc thêm vào chỉ số mảng theo sau tên thuộc tính. Ví dụ:

Địa_chỉ[3]: string

Tập_hợp_điểm[2..*]: điểm

Ví dụ: tinh chế thuộc tính các lớp của hệ thống ATM

Lớp KháchHàng

#tênKháchHàng: String

#họKháchHàng: String

#mãPIN: String

#sốThẻ: String

#tàiKhoản: TàiKhoản (thuộc tính tham chiếu)

Trong đó, thuộc tính #tàiKhoản dùng để mô tả mối quan hệ giữa lớp KháchHàng và lớp

TàiKhoản. Việc thêm thuộc tính này cho phép chúng ta tham khảo đến một đối tượng tài khoản từ một đối tượng khách hàng. Tất cả thuộc tính đều được gán cho một phạm vi truy cập nội bộ dạng nghi thức protected nhằm đảm bảo tính bao bọc và có thể thừa kế nếu sau này các phát triển thêm các lớp con.

Lớp TàiKhoản

#sốTàiKhoản: String

#loạiTàiKhoản: String

#sốDư: float

#giaoTác: GiaoTác (cài đặt mối kết hợp giữa lớp TàiKhoản và lớp GiaoTác)

#kháchHàng: KháchHàng (cài đặt mối kết hợp giữa lớp TàiKhoản và lớp KhácHàng)

Lớp GiaoTác

#giaoDịchID: String

#ngàyGiaoDịch: Date

#thờiGianGiaoDịch: Time

#loạiGiaoDịch: String

#sốTiền: float

#sốDư: float

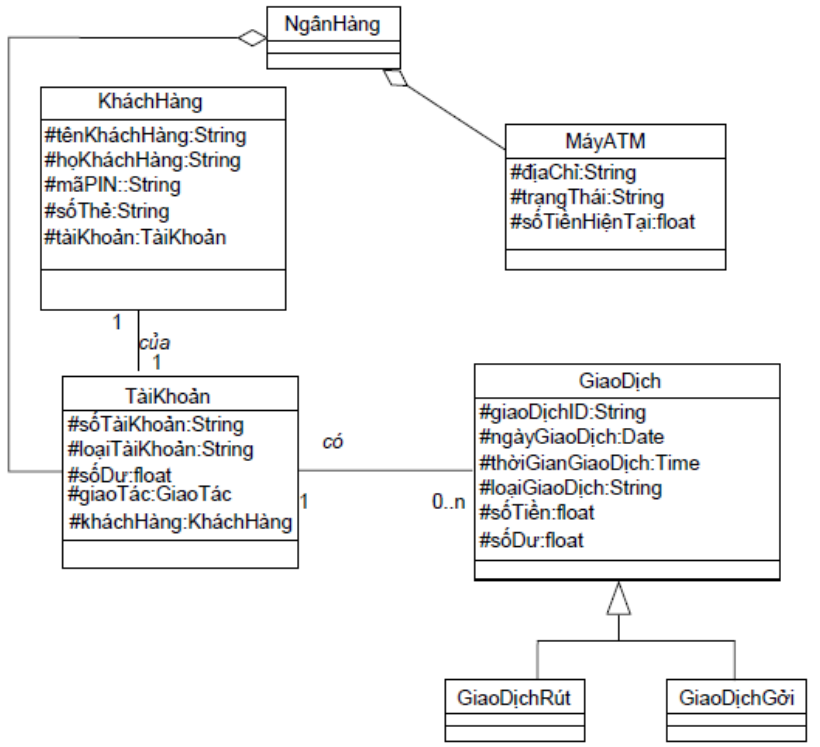
Lớp GiaoTác và lớp TàiKhoản có một mối kết hợp. Khi tính chế thuộc tính cho lớp TàiKhoản, chúng ta đã thêm vào thuộc tính #giaoTác dùng để cài đặt mối kết hợp này. Vậy khi tính chế thuộc tính cho lớp GiaoTác chúng ta cũng có thể thêm vào một thuộc tính cũng để cài đặt cho mối kết hợp này nếu chúng ta có nhu cầu biết thông tin về tài khoản từ một giao tác. Tuy nhiên, với bài toán của chúng ta đây không có nhu cầu đó, vậy nên chúng ta không thêm vào lớp GiaoTác thuộc tính tham chiếu tới lớp TàiKhoản.

Lớp MáyATM

#địaChỉ: String

#trạngThái: String

#sốTiềnHiệnTại:float



Sơ đồ lớp của hệ thống ATM sau khi đã tính chế thuộc tính

Tinh chế hành vi (method)

Mục tiêu chính của hoạt động này là mô tả thuật toán cho các hành vi đã được xác định ở giai đoạn phân tích. Việc mô tả thuật toán cũng có thể được thực hiện trên nhiều cách, hoặc bằng văn bản (mã giả) hoặc bằng sơ đồ. Việc sử dụng sơ đồ (trong UML có thể dùng sơ đồ hoạt động, tuần tự,…) cho phép chúng ta dễ dàng hơn trong việc chuyển đổi chúng sang một ngôn ngữ lập trình một cách thủ công hoặc tự động (thông qua một CASE Tool).

Trong giai đoạn này chúng cũng nên giảm tối đa mức độ phức tạp các kết nối thông điệp giữa các lớp số lượng thông điệp được gởi và nhận bởi một đối tượng. Mục tiêu nhằm gia tăng tính đoàn kết (cohension) trong số các đối tượng và các thành phần phần mềm để cải tiến tính liên kết (coupling), bởi vì chúng ta phải giữ một số lượng tối thiểu các thông tin cần thiết chuyển đổi giữa các thành phần. Áp dụng các tiên đề và hệ luận thiết kế chúng ta có các hướng dẫn sau:

- Một tập lớn các lớp đơn giản sẽ tốt hơn một tập nhỏ các lớp phức tạp
- Tạo một lớp tổng quát cho các lớp mà chúng ta tìm thấy có một số nội dung giốn nhau. Mục tiêu là tăng tối ta việc tái sử dụng
- Luôn tập trung vào mục tiêu của lớp khi định nghĩa nhằm tránh việc thiết kế lạc đề hoặc mở rộng vượt khỏi phạm vi ý nghĩa của lớp: điều này thường xảy ra khi chúng ta tạo ra các thay đổi trên lớp đang tồn tại khi bài toán có sự thay đổi và càng ngày tạo ra các lớp với nội dung phức tạp không còn đúng với mục tiêu ban đầu của lớp.

Một lớp có thể có những loại hành vi sau:

- Constructor: methode tạo thể hiện (đối tượng) của lớp
- *Destructor*: methode huỷ thể hiện của lớp
- *Conversion*: method chuyển đổi một đơn vị đo lường này sang một đơn vị đo lường
- khác
- *Copy*: method sao chép nội dung của một thể hiện sang một thể hiện khác
- *Attribute set*: method gán giá trị cho một hoặc nhiều thuộc tính
- *Attribute get*: method trả về giá trị của một hoặc nhiều thuộc tính
- *I/O method*: method cung cấp tới hoặc nhần dữ liệu từ một thiết bị
- *Domain specific*: method xác định tới ứng ụng

Hiển thị hành vi

Theo UML một hành vi của lớp được trình bày theo cú pháp:

<phạm vi> <tên> <(danh sách tham số)> : <kiểu trả về>

Trong đó,

<phạm vi> được qui định giống như của thuộc tính

<danh sách tham số>: mỗi tham số cách nhau bởi dấu phẩy và có cú pháp như sau:

<tên tham số>: <kiểu dữ liệu> = <giá trị mặc định>.

<kiểu trả về> là một đặc tả độc lập ngôn ngữ về cài đặt giá trị trả về của một hành vi. Nếu mục này bị bỏ qua thi hành vi không trả về giá trị

Ví du:

+get_Tên(): String

+get_SốTàiKhoản(vtàiKhoản : TàiKhoản): String

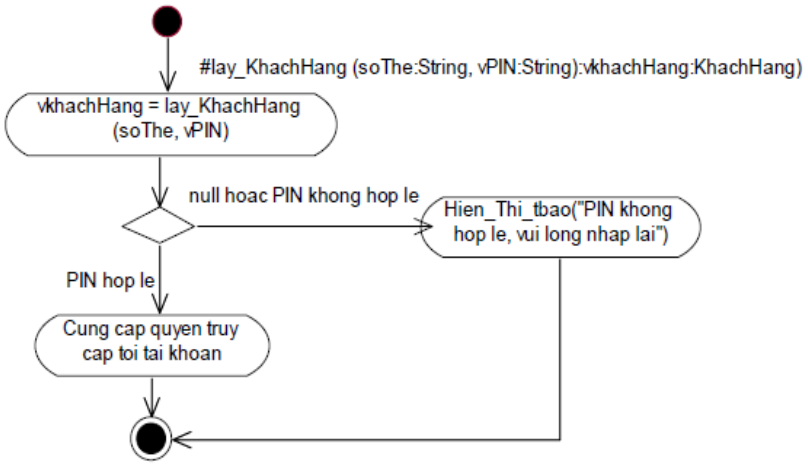
Thiết kế hành vi cho hệ thống ATM

Tại thời điểm này, chúng ta đã xác định các đối tượng hình thành tầng nghiệp vụ của hệ thống, cũng như là các dịch vụ mà các đối tượng này cung cấp. Công việc còn lại là thiết kế hành vi, giao diện người dùng và xử lý truy cập cơ sở dữ liệu. Với hệ thống ATM, chúng ta đã xác định các toán tử ở giai đoạn phân tích bao gồm:

Lớp KháchHàng

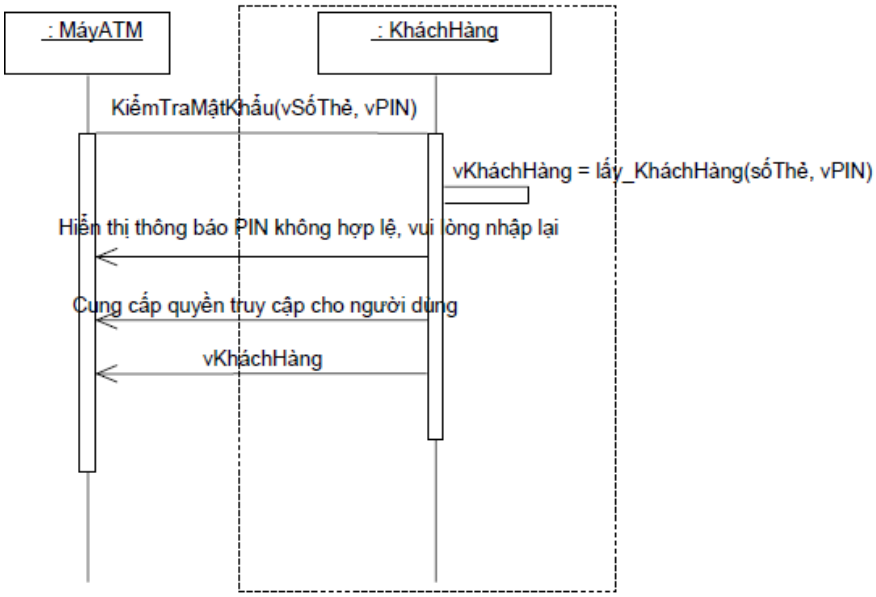
KháchHàng::+kiểmTraMậtKhẩu(sốThẻ:String, vPIN:String): vkháchHàng: KháchHàng

Thiết kế thuật giải cho hành vi dùng sơ đồ hoạt động



Hành vi *kiểmTraMậtKhẩu()* trước hết sẽ thi hành tạo một đối tượng khách hàng và thực hiện lấy thông tin về khách hàng dựa trên dựa trên một số thẻ và mã PIN. Tại đây, chúng ta lại nhận thấy rằng cần phải có một hành vi khác để thực hiện điều này đó là lấy_KháchHàng() và có phạm vi nội bộ dạng protected (#). Hành vi này sẽ lấy tham số đầu vào là số thẻ và mã PIN, kết quả trả về là một đối tượng khách hàng tìm thấy hoặc là "null" nếu ngược lại. Nếu giá trị trả về là "null", sẽ gởi một thông điệp tới hệ thống để thực hiện thông báo "PIN không hợp lệ, vui lòng nhập lại", ngược lại, sẽ gán quyền truy cập cho người dùng.

Thiết kế thuật giải dùng sơ đồ tuần tự



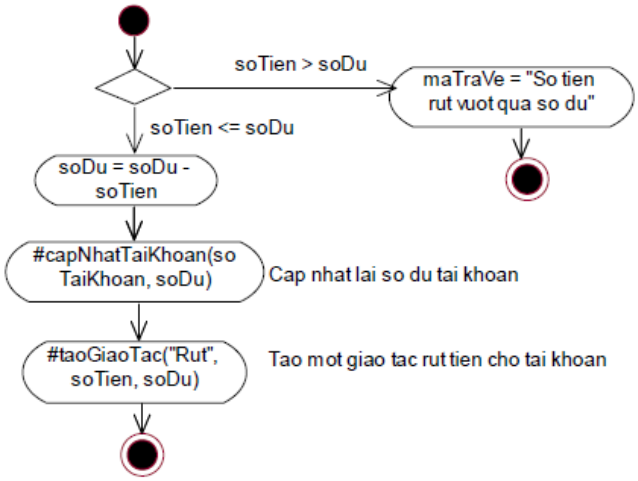
Với đồ trên chúng ta chỉ quan tâm đến các lớp (nghiệp vụ) sẽ cung cấp các dịch vụ gì để thực hiện *kiểmTraMậtKhẩu()*. Chúng ta chưa quan tâm đến các đối tượng ở các tầng khác như là: truy cập cơ sở dữ liệu hoặc giao diện hiển thị. Ví dụ như hành vi lấy_KháchHàng(sốThẻ, PIN) sẽ phải truy cập cơ sở dữ liệu để thực hiện, cũng như đối tượng: MáyATM chỉ là giả lập giao diện giữa khách hàng và hệ thống, chúng ta chưa xác định đối tượng giao diện cụ thể nào (form, trang web,…) sẽ thực hiện. Phần này sẽ được đề cập chi tiết trong những chương sau.

Khi thực hiện một việc gửi tiền, số tiền được gửi được chuyển đến một đối tượng tài khoản và được dùng như là một đối số cho hành vi *gửiTiền()*. Tài khoản này điều chỉnh số dư hiện hành

của nó bằng cách cộng thêm với số tiền gửi. Sau đó, tài khoản này sẽ lưu thông tin lần gửi bằng cách tạo ra một đối tượng giao tác.

Một lần nữa chúng ta lại khảm phá ra những hành vi khác: *cậpNhậtTàiKhoản()*, hành vì này là cục bộ (#) cho phép cập nhật dữ liệu tài khoản. *tạoGiaoTác()*, cũng là hành vi cục bộ cho phép tạo một giao tác tương ứng với tài khoản này.

TàiKhoản::+rútTiền(sốTiền:float): mãTrảVề:String

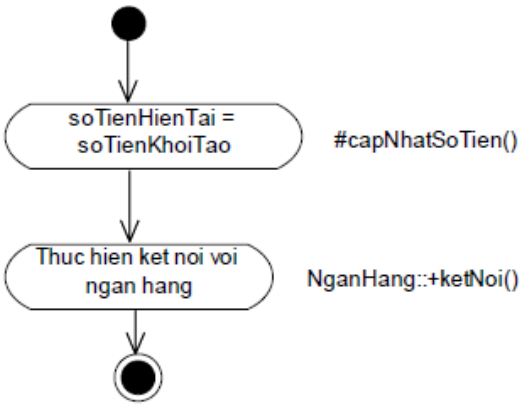


Một số tiền mà khách hàng muốn rút sẽ được chuyển đến một đối tượng tài khoản như là một tham số đầu vào. Tài khoản này sẽ kiểm tra số dư hiện hành của nó so với số tiền này. Nếu vẫn lớn hơn hoặc bằng số tiền rút thì tài khoản sẽ cập nhật lại số dư và tạo một giao tác rút tiền, ngược lại thông báo lỗi với mãTrảVề "Số tiền rút vượt quá số dư".

Một lần nữa chúng ta thấy hành vi *rútTiền* lại sử dụng *cậpNhậtTàiKhoản* và *tạoGiaoTác* đã đề cập đến trong khi thiết kế hành vi gửiTiền.

Lớp MáyATM

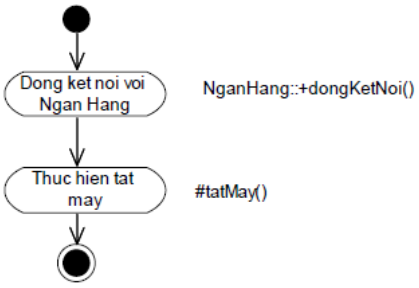
MáyATM::+khởiĐộngMáy(sốTiềnKhởiTạo:float)



Sau khi bật máy ATM, một số tiền khởi tạo được nhập từ nhân viên vận hành sẽ được chuyển đến đối tượng máyATM như là một tham số đầu vào. Đối tượng máyATM sẽ cập nhật lại số

tiền ban đầu cho máy và sau đó thực hiện việc kết nối tới ngân hàng nhằm thực hiện việc liên kết truy cập cơ sở dữ liệu. Quá trình này chúng ta lại có nhu cầu phát sinh thêm hành vi cục bộ *#cậpNhậtSốTiền* và hành vi toàn cục *NgânHàng::+kếtNối()*.

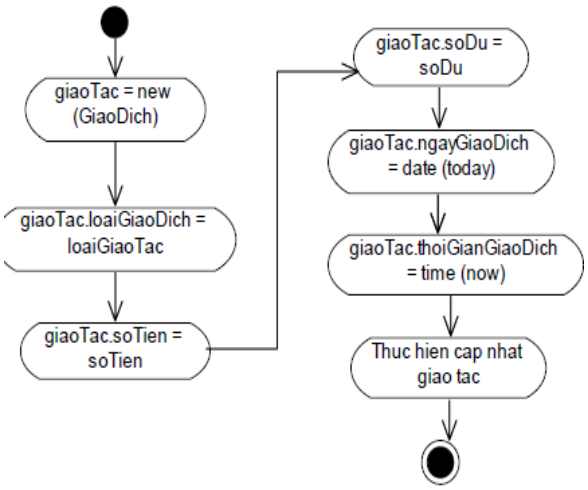
MáyATM::+đóngMáy()

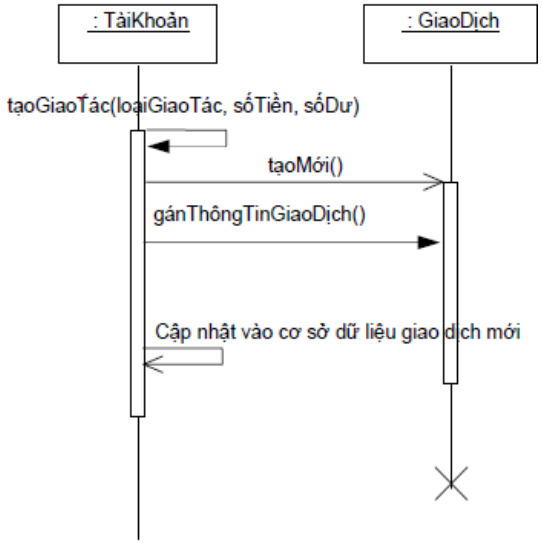


Đối tượng máyATM thực hiện đóng máy bằng cách gọi thực hiện việc đóng kết nối với ngân hàng và gọi thực hiện tắt máy. Quá trình này phát sinh thêm hai hành vi: *+đóngKếtNối()*do đối tượng NgânHàng đảm nhận và *#tắtMáy()* là hành vi cục bộ của đối tượng MáyATM.

Như vậy, chúng ta đã thiết kế xong các hành vi đã được xác định ở giai đoạn phân tích quá trình thiết kế này lại phát sinh các hành vi: #lấy_KháchHàng(), #cậpNhậtTàiKhoản(), #tạoGiaoTác(), +kếtNối(),+đóngKếtNối(), #cậpNhậtSốTiền(), #tắtMáy(). Chúng ta lặp lại quá trình thiết kế cho những hành vi này. Chú ý rằng các hành vi liên quan đến việc truy cập dữ liệu hoặc xử lý giao diện sẽ được thiết kế ở những giai đoạn tiếp theo trong những chương sau. Các hành vi đó là: #lấy_KháchHàng(), #cậpNhậtTàiKhoản(),…. Sau đây chúng ta tiếp tục thiết kế thuật giải cho các hành vi *tạoGiaoTác(), +kếtNối(), +đóngKếtNối(), #cậpNhậtSốTiền():*

TàiKhoản::#tạoGiaoTác(loạiGiaoTác:String, sốTiền:float, soDu:float)





Một đối tượng tài khoản tạo một giao tác liên quan đến nó bằng cách truyền các tham số: loại giao tác (gửi, rút), số tiền, và số dư sau khi thực hiện giao tác. Đối tượng tài khoản sẽ tạo mới một đối tượng giao tác ứng với thuộc tính giaoTác của nó và gán các thông tin về giao tác đó, sau đó, lưu lại giao tác này vào cơ sở dữ liệu. Quá trình này lại phát sinh mới hai hành vi: hành vi +gánThôngTinGiaoDịch() của đối tượng giao dịch có phạm vi toàn cục; hành vi cập nhật vào cơ sở dữ liệu mà chúng ta sẽ thiết kế chi tiết trong các chương sau. Tạm thời ở đây là vấn đề nội bộ của đối tượng tài khoản.

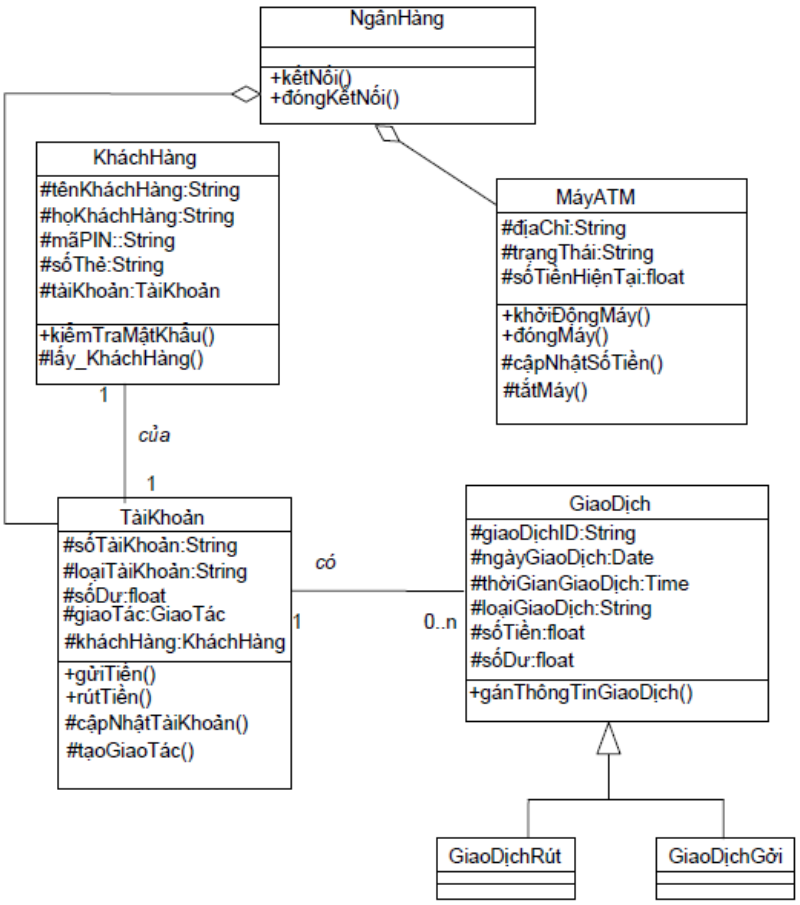
GiaoDịch:: +gánThôngTinGiaoDịch(loạiGD:String, sốTiền:float, sốDư:float, ngàyGD:Date, giờGD:Time

Các hành vi *+kếtNối(), +đóngKếtNối(), +cậpNhậtSốTiền()* thì đơn giản do đó chúng ta chỉ mô tả khai báo nó:

NgânHàng::+kếtNối()

NgânHàng::+đóngKếtNối()

MáyATM::#cậpNhậtSốTiền(sốTiền:float)



Sơ đồ lớp của hệ thống máy ATM với kết quả tinh chế đầu tiên về thuộc tính và hành vi

Tổng kết

Bước đầu tiên trong tiến trình thiết kế hướng đối tượng là việc áp dụng các tiên đề và hệ luận để thiết kế các lớp, thuộc tính, hành vi, mối kết hợp, cấu trúc, phạm vi; quá trình này là một quá trình lặp và tinh chế. Trong giai đoạn phân tích, chỉ cần tên thuộc tính là đủ. Tuy nhiên, trong giai đoạn thiết kế, các thông tin chi tiết sẽ phải được thêm vào mô hình (đặc biệt là các định nghĩa của thuộc tính và hành vi).

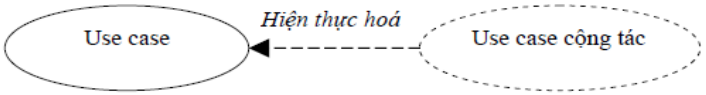
Thiếu đi một nghi thức thiết kế tốt có thể phát sinh các kẽ hở về tính bao bọc (encapsulation). Vấn đề này xảy ra khi khi chi tiết về cài đặt bên trong của một lớp được phơi bày ra ở giao diện. Càng nhiều thông tin chi tiết bên trong của lớp bị phơi bày, sự uyển chuyển trong các thay đổi hệ thống sau này càng giảm. Bởi vì nó làm giảm đi tính độc lập của đối tượng trong hệ thống. Do đó, chúng ta sử dụng khai báo phạm vi cục bộ (private) hoặc bảo vệ (protected) để xác định việc cài đặt đối tượng; sử dụng khai báo phạm vi toàn cục (public) để xác định chức năng của đối tượng.

Thiết kế hướng đối tượng là một quá trình lặp. Do đó, chúng ta đừng ngại việc thay đổi tới một lớp, mỗi sự thay đổi hãy tin rằng sẽ là một sự cải tiến mô hình hiện tại. Một điều ghi nhớ rằng các vấn đề cần được giải quyết càng sớm càng tốt để khỏi phải trả giá lớn trong các giai đoạn sau.

2. Thiết kế Use case

Mô hình use case chúng ta đạt được ở giai đoạn phân tích mô tả tính năng hệ thống từ phía người sử dụng. Các chức năng này được xem như là sự biểu diễn các yêu cầu chức năng mà hệ

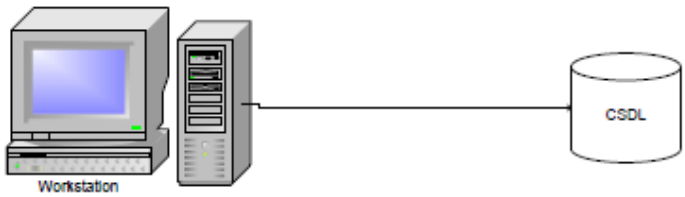
thống đáp ứng. Trong giai đoạn thiết kế, các chức năng này phải được mô tả ở gốc độ để cài đặt. Như vậy đối với một người thiết kế viên, chúng ta phải đặt câu hỏi là làm sao để biểu diễn sơ đồ use case này đủ chi tiết và trên một ngôn ngữ để có thể cài đặt được. Một cách tiếp cận là tách biệt diễn đạt chức năng thành hai phần: phần mô tả chức năng nhìn từ bên ngoài (từ phía người dùng: hệ thống có những chức năng gì cung cấp cho người sử dụng) và phần mô tả chức chức nhìn từ bên trong (từ phía người phát triển: làm sao để hiện thực hoá các chức năng để cài đặt nó).



Việc hiện thực hoá được đảm nhận bởi một các use case cộng tác và do đó thiết kế một use case chính tập trung vào thiết kế use case cộng tác. Nội dung thiết kế use case cộng tác bao gồm: xác định thêm các đối tượng hệ thống phần mềm cùng cộng tác trong việc thực hiện use case và sự tương tác giữa chúng. Trong tiếp cận kiến trúc ba tầng (tree-layer), các đối tượng hệ thống phần mềm là các đối tượng ở tầng giao diện và tầng truy cập dữ liệu. Sau đó, xác định sự tương tác giữa các đối tượng trong use case nhằm phát hiện các method cho các đối tượng của hai tầng này cũng như hoàn thiện việc tinh chế method cho các lớp. Kết quả của giai đoạn này là một sơ đồ lớp đầy đủ để chuẩn bị cho việc cài đặt hệ thống phần mềm.

Kiến trúc 3 tầng (three - layer)

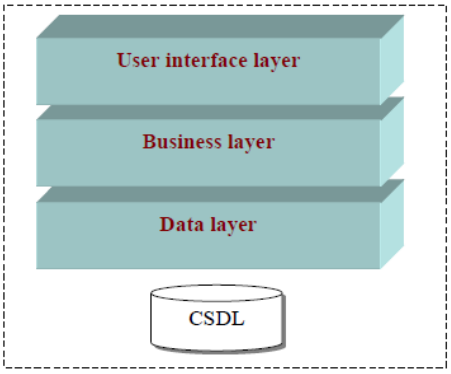
Hầu hết các hệ thống được phát triển sử dụng các công cụ CASE ngày nay hoặc trong các môi trường phát triển ứng dụng client – server có xu hướng xây dựng một kiến trúc hai tầng (two – layer): giao diện (interface) và dữ liệu (data). Trong một hệ thống hai tầng, các màn hình giao diện người dùng liên kết để truy cập dữ liệu thông qua các đoạn chương trình được cài trực tiếp trên các giao diện. Ví dụ, trong một chương trình viết trên Visual Basic trong một form giao diện, một thủ tục xử lý biến cố trong button "Update" của form này có tên bUpdate_Click() có thể thực hiện luôn việc truy cập và cập nhật CSDL trực tiếp, như vậy thủ tục cài đặt luôn các ngữ nghĩa về tác nghiệp (business). Việc thiết kế theo mô hình này tạo ra một sự phụ thuộc rất lớn giữa giao diện và CSDL và do đó, rất khó để cải tiến, bảo trì và tái ử dụng.



Kiến trúc hai lớp: giao diện và dữ liệu

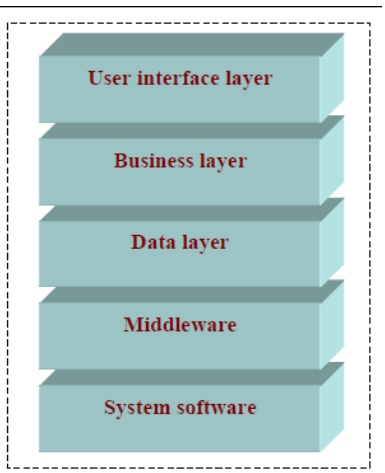
Một cách tiếp cận khác về kiến trúc tốt hơn chính là tạo ra sự độc lập giữa giao diện và người sử dụng bằng cách cô lập các chức năng của giao diện với các chức năng tác nghiệ (business), và cô lập các chức năng tác nghiệp với các chi tiết về truy cập CSDL đó là cách tiếp cận ba tầng (three-layer). Từ cách tiếp cận này cho phép chúng ta tạo ra được các đối tượng đại diện các đối tượng hữu hình trong thực tế nhưng hoàn toàn độc lập với cách thức mà các đối tượng này trình bày tới người dùng hoặc là với cách mà dữ liệu của nó được lưu trữ vật lý trong CSDL. Do đó, ba

tầng trong cách tiếp cận này là: tầng giao diện người dùng (user interface layer), tầng tác nghiệp (business layer), và tầng truy cập dữ liệu (data layer).



Sơ đồ biểu diễn tiếp cận ba tầng

Một tiếp cận khác đầy đủ hơn của một kiến trúc hệ thống có thể được trình bày như sơ đồ sau:



Một sơ đồ khác của cách tiếp cận ba tầng (nhiều tầng)

Trong đó,

Tầng Middleware: chứa các thành phần xây dựng giao diện (ví dụ: thành phần dạng ActiveX), thành phần giao diện tới các hệ quản trị CSDL (ví dụ: ODBC, JDBC driver), các dịch vụ hệ điều hành độc lập với platform, các thành phần nhúng OLE (ví dụ: các công cụ soạn thảo sơ đồ nhúng, các bảng tính nhúng,…).

Tầng System software: chứa các thành phần về hệ điều hành, CSDL, giao diện tới các phần cứng (ví dụ: các driver phần cứng cụ thể), v.v…

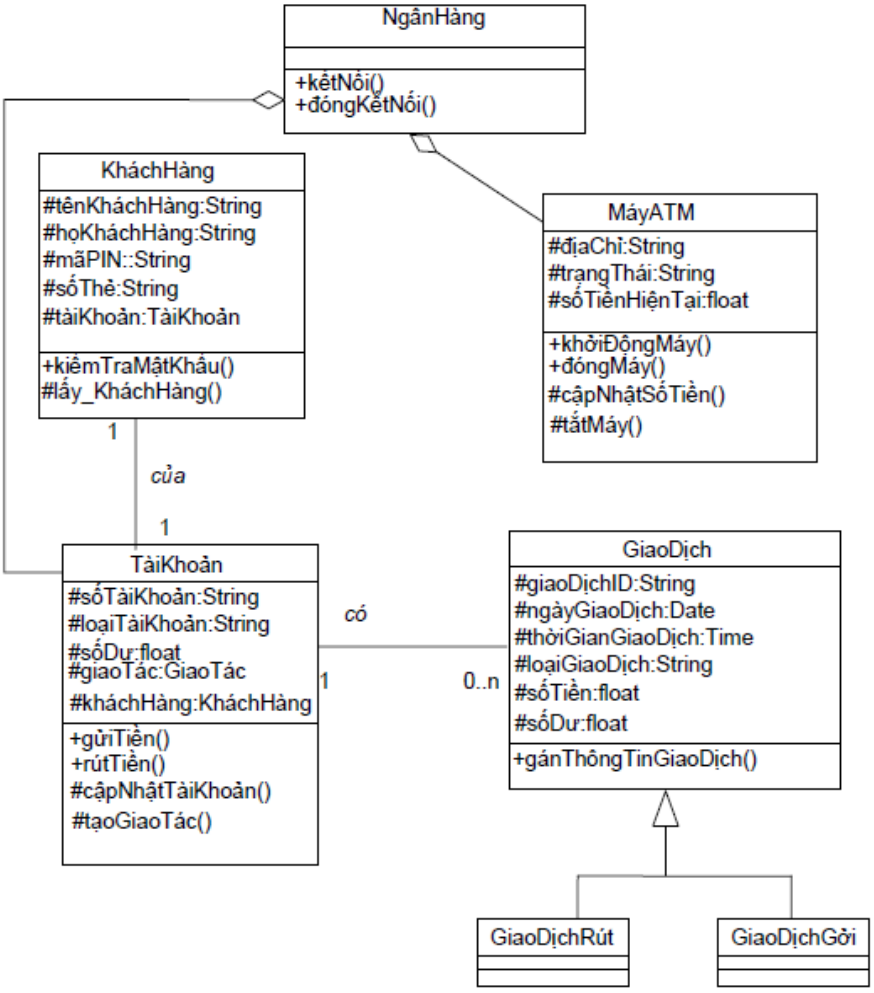
Xác định lớp ở tầng dịch vụ tác nghiệp (business layer)

Tầng này chứa đựng tất cả đối tượng mô tả tác thành phần nghiệp hệ thống (bao gồm cả dữ liệu và hành vi). Nó diễn đạt các đối tượng tồn tại trong thực tế vào trong hệ thống cần quản lý. Ví dụ, đơn đặt hàng, khách hàng, hoá đơn, nhà cung cấp,… hầu hết các phương pháp luận phân tích thiết kế đều đưa ra phương pháp xác định đối tượng này trong giai đoạn phân tích. Tuy nhiên, khi xác định các đối tượng ở lớp này chúng ta phải luôn nhớ hai điều sau:

- Các đối tượng ở tầng tác nghiệp không nên quan tâm đến cách thức nó được hiển thị và bởi ai. Các đối tượng này được thiết kế để độc lập với bất kỳ một giao diện cụ thể, và vì vậy cách thức chi tiết để hiển thị một đối tượng nên tồn tại trong tầng giao diện thay vì trong tầng tác nghiệp.
- Các đối tượng ở tầng tác nghiệp cũng không nên quan tâm đến nguồn gốc của nó hình thành. Có nghĩa là các đối tượng này sẽ độc lập về dữ liệu của nó được lấy từ truy cập CSDL hay là từ truy xuất tập tin.

2.1 Xác định các class tầng tác nghiệp

Các class ở tầng tác nghiệp đã được xác định trong giai đoạn phân tích (xem chương 3). Trong phần này chúng tả mô tả lại theo từng use case để cho phép chúng ta có một cách nhìn về những gì mà các đối tượng ở tầng tác nghiệp kế hợp với nhau trong hoạt động đáp ứng yêu cầu sử dụng được mô tả thông qua use case. Chú ý rằng một lớp trong tầng này đều có thể tham gia xử lý trong nhiều use case khác nhau.



Các kết quả thiết kế lớp trong mục 1 đã cho chúng ta một sơ đồ thiết kế về tầng nghiệp vụ này.

2.2 Xác định lớp ở tầng truy cập dữ liệu (data layer)

Tầng này chứa các đối tượng nhằm mục đích cung cấp các dịch vụ về dữ liệu cho tầng tác nghiệp. Nói chung, tất cả các nhu cầu truy cập CSDL hoặc tập tin để thao tác trên dữ liệu được lưu trữ của hệ thống đều phải thông qua tầng này. Do đó, các đối tượng tầng này phải truy cập vật lý CSDL ở các vị trí (CSDL quan hệ, file, internet,…) và xử lý nó. Hai nhiệm vụ chính của tầng này là:

- Chuyển dịch các yêu cầu: chuyển dịch tất cả các yêu cầu liên quan đến dữ liệu từ tầng tác nghiệp đến một phương thức truy cập dữ liệu thích hợp (dạng SQL, truy xuất file,…)
- Chuyển dịch kết quả: chuyển dịch tất cả dữ liệu truy cập được tới các đối tượng tác nghiệp thích hợp.

Xác định các đối tượng lưu trữ và persistence

Một chương trình sẽ tạo ra một số lượng dữ liệu trong quá trình thực thi. Mỗi dữ liệu sẽ có một thời gian sống (lifetime) khác nhau. Dựa vào thời gian sống này chúng ta có thể phân thành những loại dữ liệu sau:

- Là kết quả tạm thời để đánh giá một biểu thức
- Các biến trong quá trình thực thi một thủ tục (các tham số và biến trong phạm vi cục bộ)
- Các biến toàn cục và các biết cấp phát một cách tự động
- Dữ liệu tồn tại giữa các lần thực thi một chương trình
- Dữ liệu tồn tại giữa các phiên bản của một chương trình
- Dữ liệu tồn tại lâu hơn chương trình

Ba loại dữ liệu đầu tiên đều gọi là dữ liệu tạm thời (transient data). Là dữ liệu có thời gian sống phụ thuộc vào thời gian sống của tiến trình sử dụng nó. Khi tiến trình kết thúc thì dữ liệu này bị giải phóng. Ngược lại, ba loại dữ liệu cuối gọi là dữ liệu persistent (persistent data). Các dữ liệu này tồn tại lâu hơn tiến trình sử dụng nó và có thể độc lập với tiến trình này. Các ngôn ngữ lập trình cung cấp sự hỗ trợ hoàn hảo cho các loại dữ liệu tạm thời. Các loại dữ liệu persistent sẽ được quản lý bởi hệ quản trị cơ sở dữ liệu hoặc hệ thống file lưu trữ.

Đối với các đối tượng cũng được áp dụng một cách tương tự. Các đối tượng cũng có một thời gian sống. Chúng được tạo ra một cách tường minh và có thể tồn tại trong một khoảng thời gian ngắn (thời gian của một ứng dụng, của một phiên,…). Tuy nhiên, cũng có đối tượng tồn tài lâu hơn thời gian của một ứng dụng, để làm điều này thì đối tượng phải được lưu trữ ra tập tin hoặc cơ sở dữ liệu. Việc lưu trữ này sẽ cung cấp cho đối tượng thời gian sống lâu hơn quá trình của tiến trình. Đặc tính này người ta gọi là persistent.

Trong hệ thống ATM, các lớp persistent gồm: KháchHàng, TàiKhoản, GiaoTác, GiaoTácGửi, GiaoTácRút. Các lớp như MáyATM, NgânHàng được xác định không phải là lớp persistent bởi vì chúng ta không có nhu cầu lưu trữ thông tin của các đối tượng các lớp này.

Chuyển đổi đối tượng sang mô hình quan hệ

Trong cơ sở dữ liệu quan hệ, một lược đồ được hình thành bởi các bảng (table) gồm các cột và dòng. Trong mô hình đối tượng, tương ứng tới một bảng là một lớp (hoặc nhiều lớp). Các thành phần tương ứng như sau:

- Một cột ứng với một thuộc tính (persistent) lớp
- Một dòng của bảng ứng với một đối tượng (thể hiện của một lớp)
- Một stored procedure có thể tương ứng với một method.

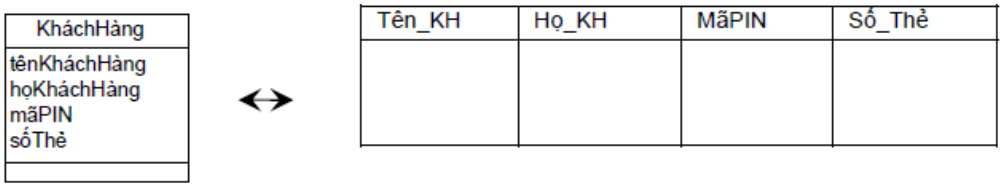
Như vậy, việc chuyển đổi sơ đồ lớp sang lược đồ quan hệ gồm có những công việc sau:

- Chuyển đổi lớp – bảng
- Chuyển đổi mối liên kết
  - Chuyển đổi liên kết kết hợp
  - Chuyển đổi liên kết kế thừa

Chuyển đổi lớp – bảng

Đa số các trường hợp thì việc chuyển đổi này là một – một. Một lớp sẽ chuyển thành một bảng cụ thể như sau:

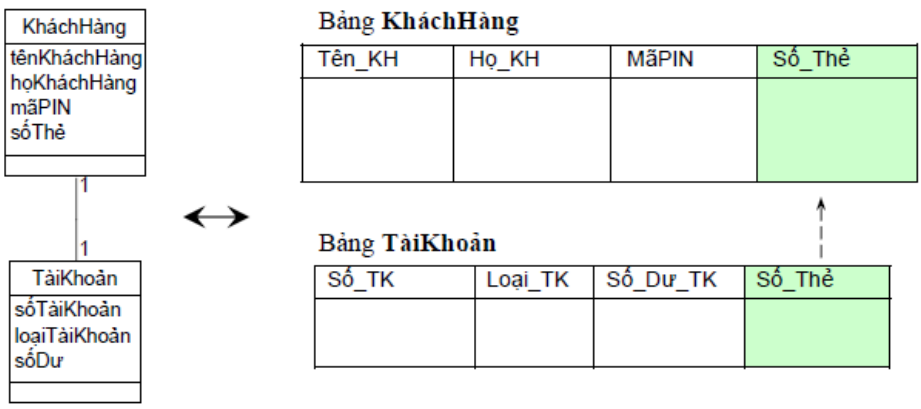
- Một lớp → một bảng
- Một thuộc tính (persistent) → một cột: chỉ có các thuộc tính có nhu cầu lưu trữ và được đòi hỏi bởi ứng dụng sẽ được chuyển thành cột của bảng.
- Một đối tượng (thể hiện) → một dòng



Chuyển đổi mối liên kết

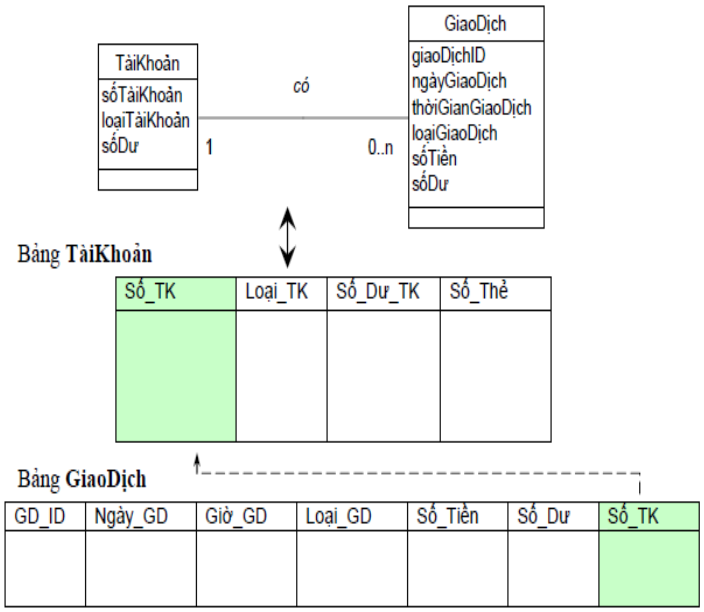
Chuyển đổi liên kết kết hợp

Trường hợp 1



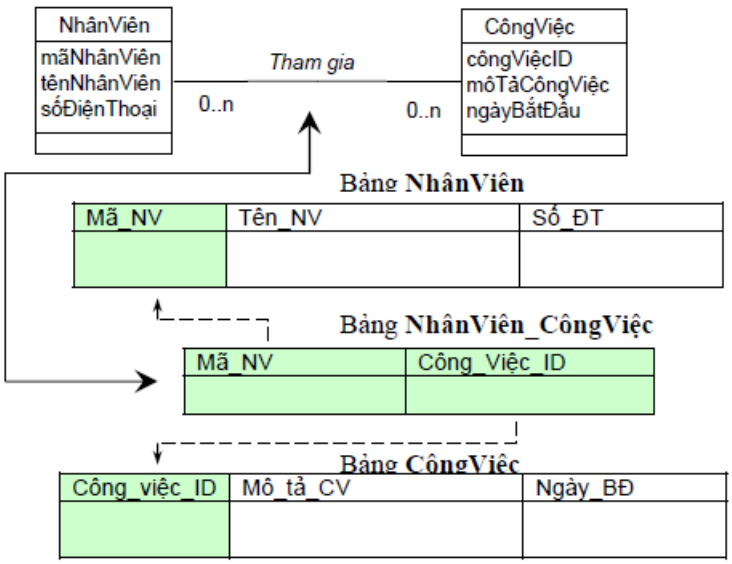
Trong chuyển đổi mối kết hợp dạng 1 – 1, chúng ta có thể lấy cột khoá chính trong một bảng chuyển qua bảng khác làm khoá ngoại. Trong mối kết hợp 1 – 1 trên (giữa lớp KháchHàng và TàiKhoản), chúng ta có thể lấy khoá của bảng KháchHàng (Số_Thẻ) đưa vào bảng TàiKhoản là khoá ngoại. Hoặc ngược lại, chúng ta có thể lấy khoá của bảng TàiKhoản (Số_TK) đưa vào bảng KháchHàng làm khoá ngoại. Hoặc thực hiện cả hai trường hợp.

Trường hợp 2



Trường hợp 3

Trong chuyển đổi mối kết hợp dạng 1 – n, chúng ta lấy cột khoá chính của bảng ứng với lớp phía 1 trong mối kết hợp đưa vào bảng ứng với lớp phía n làm khoá ngoại. Trong ví dụ trên, bảng GiaoDịch sẽ lấy thuộc tính Số_TK trong bảng TàiKhoản làm khoá ngoại.

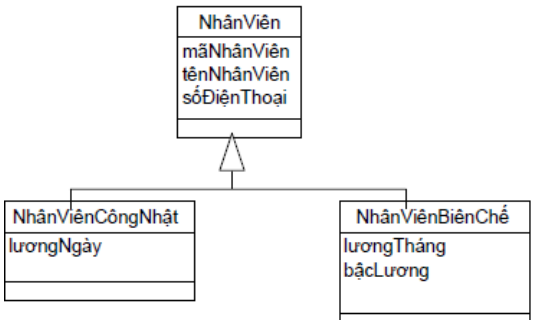


Trong chuyển đổi mối kết hợp n – n, chúng ta tạo ra một bảng cho mối kết hợp đó bằng cách lấy các khoá chính của các bảng đưa vào bảng mới này như là các khoá ngoại. Trong ví dụ trên, mối

kết hợp Tham gia giữa lớp NhânViên và lớp CôngViệc sẽ được mô tả bởi một bảng NhânViên_CôngViệc bao gồm các cột Mã_NV và Công_Việc_ID lần lượt là khoá chính của bảng NhânViên và bảng CôngViệc.

Chuyển đổi liên kết kế thừa

Trong lược đồ quan hệ không có khái niệm kế thừa mà chúng ta thường dùng liên kết khoá chính – khoá ngoại để diễn đạt điều này. Sau đây là các trường hợp mà chúng ta có thể quyết định chọn một:



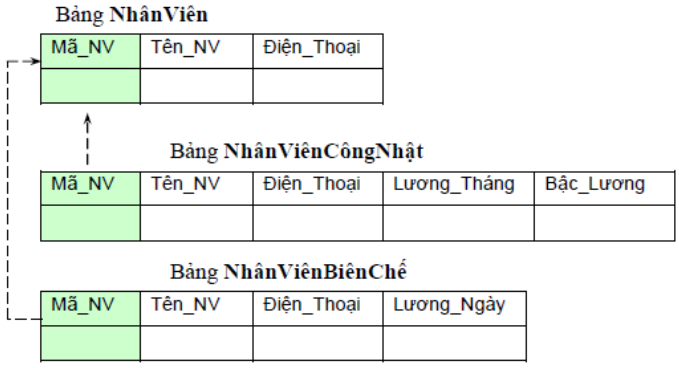
Trường hợp 1

Bảng NhânViên

Mã_NV	Tên_NV	Điện_Thoại	Lương_Ngày	Lương_Tháng	Bậc_Lương	***Loại_NV***

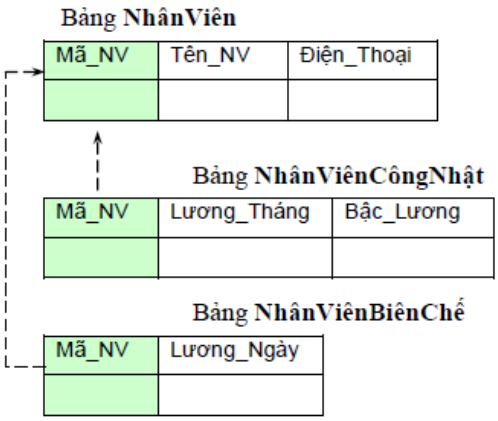
Chỉ sử dụng một bảng lưu trữ tất cả các loại nhân viên. Do đó, các thuộc tính của bảng được hình thành từ các thuộc tính của lớp NhânViên, NhânViênCôngNhật và Nhân ViênBiênChế. Ngoài ra chúng ta cũng đưa vào thêm một thuộc tính nhằm phân loại dòng này thuộc đối tượng của lớp nào.

Trường hợp 2



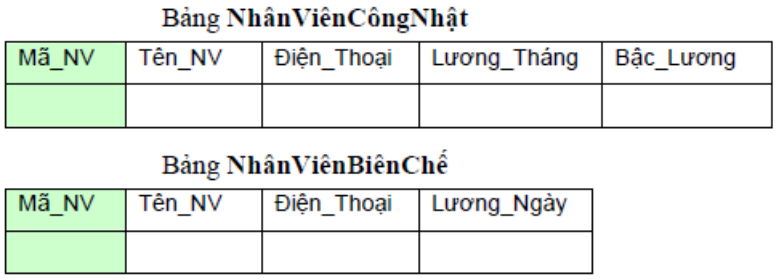
Sử dụng ba bảng tương ứng cho ba lớp. Tuy nhiên, nhằm mô tả sự thừa kế trong các bảng NhânViênCôngNhật và NhânViênBiênChế chúng ta thêm vào tất cả các thuộc tính của bảng nhân viên. Các thể hiện tương ứng của các nhân viên công nhật hay biên chế sẽ chỉ lưu trong bảng tương ứng.

Trường hợp 3



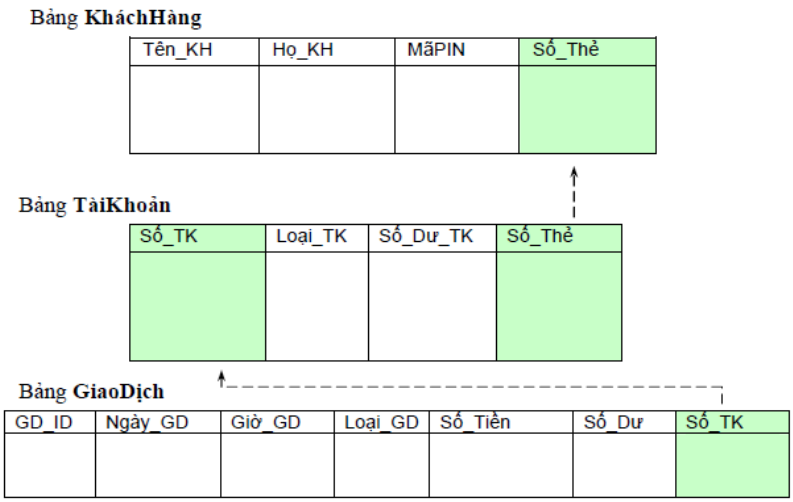
Giống như trường hợp 2. Ở hai bảng NhânViênCôngNhật và NhânViênBiênChế chỉ lưu khoá ngoại tham chiếu đến bảng nhân viên để tham khảo thông tin thừa kế.

Trường hợp 4



Chỉ dùng hai bảng NhânViênCôngNhật và NhânViênBiênChế. Tuy nhiên, tất cả các thuộc tính của lớp NhânViên sẽ được đưa vào hai bảng này nhằm mô tả sự thừa kế. Nếu chúng ta muốn truy xuất thông tin về lớp NhânViên thì chúng ta có thể tạo một khung nhìn (view) hộicủa hai bảng này.

Sơ đồ cài đặt các đối tượng persistent của hệ thống ATM dùng mô hình quan hệ như sau:



Trong lược đồ trên của máy ATM, chúng ta áp dụng trường hợp 1 cho cấu trúc các lớp GiaoDịch, GiaoDịchGửi và GiaoDịchRút.

Xác định lớp ở tầng truy cập dữ liệu

Mục tiêu chính của việc tạo ra một tầng truy cập dữ liệu chính là để tạo ra các lớp có nhiệm vụ truy cập tới các vị trí mà dữ liệu thực sự được lưu trữ nhằm giúp cho tầng nghiệp vụ không quan tâm đến vị trí cũng như hình thức lưu trữ của dữ liệu (dạng tập tin, cơ sở dữ liệu quan hệ, cơ sơ dữ liệu đối tượng, internet, DCOM,…). Các đối tượng ở tầng này phải có trách nhiệm cung cấp một liên kết giữa nghiệp vụ (cách nhìn theo đối tượng) và dữ liệu lưu trữ. Tiến trình xác định các lớp ở tầng này gồm các bước sau:

- Với mỗi lớp persistent ở tầng nghiệp vụ, tạo một lớp tương ứng ở tầng truy cập dữ liệu. Ví dụ, nếu chúng ta có ba lớp ở tầng nghiệp vụ là Class1, Class2, Class3 thì chúng ta tạo ra ba lớp tương ứng ở tầng truy cập dữ liệu là: ClassDB1, ClassDB2, ClassDB3.
- Xác định mối kết hợp: tương tự như xác định mối kết hợp giữa các lớp ở tầng nghiệp vụ
- Đơn giản hoá các lớp và mối kết hợp: mục tiêu chính là để loại các lớp và cấu trúc dư thừa hoặc không cần thiết. Thông thường, chúng ta kết hợp nhiều lớp thành một và đơn giản hoá cấu trúc lớp cha – lớp con.
    - Các lớp dư thừa: nếu chúng ta có hai lớp trở lên cùng cung cấp các dịch vụ tương tự nhau, chúng ta giữ lại một và loại đi một.
    - Các method: xem lại các lớp chỉ có một hoặc hai method xem có thể có thể kết hợp với các lớp khác? Thông thường, chúng ta chỉ quan tâm đến các method có nhu cầu truy cập đến dữ liệu lưu trữ. Các method đó là: đọc dữ liệu từ dữ liệu lưu trữ của đối tượng, xoá dữ liệu của đối tượng khỏi dữ liệu lưu trữ và cập nhật các thay đổi của đối tượng xuống dữ liệu lưu trữ.
- Tạo mối kết hợp giữa lớp tầng truy cập dữ liệu và lớp của nó tương ứng ở tầng nghiệp vụ là mối kết hợp dạng thành phần (aggregation). Tạo thuộc tính tham chiếu cho các lớp của tầng nghiệp vụ tham chiếu đến lớp ở tầng truy cập dữ liệu dựa trên mối liên kết vừa xác định.
- Lặp lại tiến trình này

Xác định method các lớp tầng truy cập dữ liệu

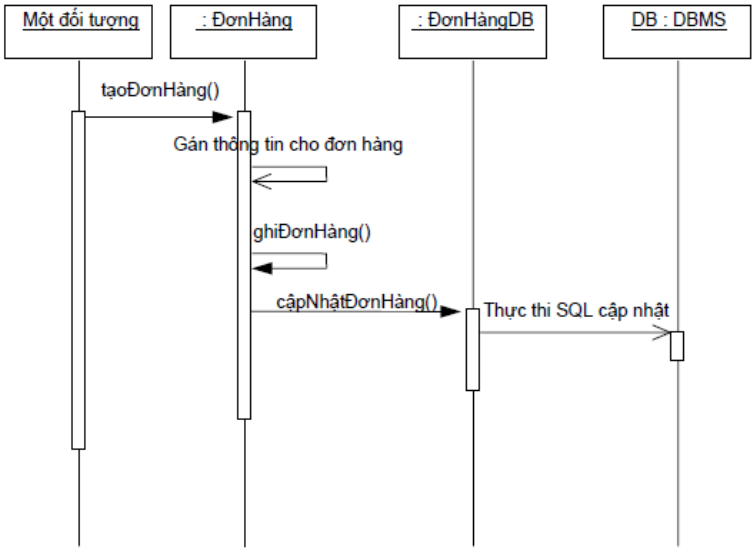
Trong thiết kế hướng đối tượng nhằm đảm bảo tính bao bọc trong các cài đặt chi tiết, chúng ta mong muốn các đối tượng persistent được quan sát giống như một đối tượng tạm thời (transient) trong hệ thống. Nghĩa là chúng ta phải tạo được một cách nhìn trong suốt cho các đối tượng trong hệ thống mà không phân biệt và xử lý khác nhau giữa đối tượng persistent và bất kỳ đối tượng nào khác. Tuy nhiên, đây cũng là một vấn đề của hệ thống bởi vì dữ liệu và trạng thái của các đối tượng persistent được quản lý tách biệt khỏi chương trình ứng dụng. Do đó, sự nhất quán giữa đối tượng trong ứng dụng và trạng thái của nó trong cơ sở dữ liệu phải luôn luôn được đặt ra trong thiết kế hệ thống hướng đối tượng. Để đảm bảo điều này thì có nhiều mặt một ứng dụng phải kiểm soát:

- Đọc và lưu các đối tượng persistent
- Xoá các đối tượng persistent
- Quản lý giao tác trên các đối tượng persistent

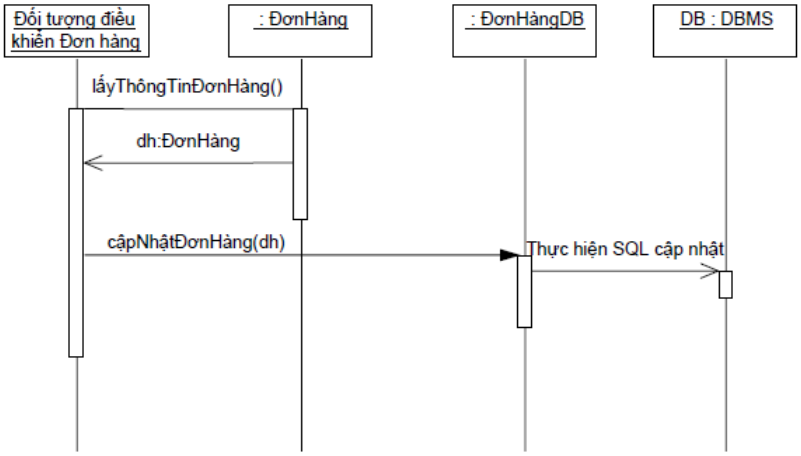
- Kiểm soát cơ chế khoá và truy cập đồng hành

Ghi đối tượng persistent

Có hai trường hợp cần xem xét: thời điểm ban đầu khi đối tượng được tao ra và phải ghi vào cơ sở dữ liệu; các thời điểm tiếp theo khi chương trình cập nhật trạng thái của đối tượng và thay đổi này cũng được ghi vào cơ sở dữ liệu.



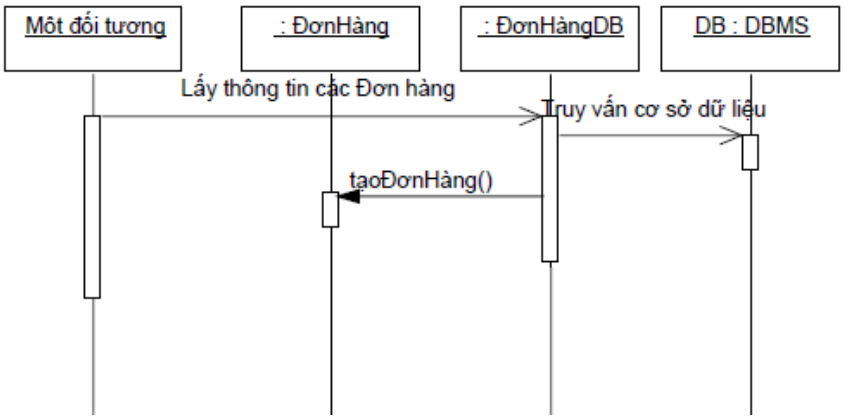
Sơ đồ tuần tự trên mô tả một trong những giải pháp cập nhật đối tượng persistent. Khi một đối tượng đơn hàng được tạo ra trong ứng dụng, đối tượng này ngay lập tức được ghi vào cơ sở dữ liệu bởi lớp ĐơnHàngDB tầng truy cập dữ liệu. Tương tự cho hoạt động cập nhật, khi một đối tượng trong ứng dụng thay đổi trạng thái của đối tượng đơn hàng. Ngay lập tức sự thay đổi này được cập nhật vào cơ sở dữ liệu bởi ĐơnHàngDB.



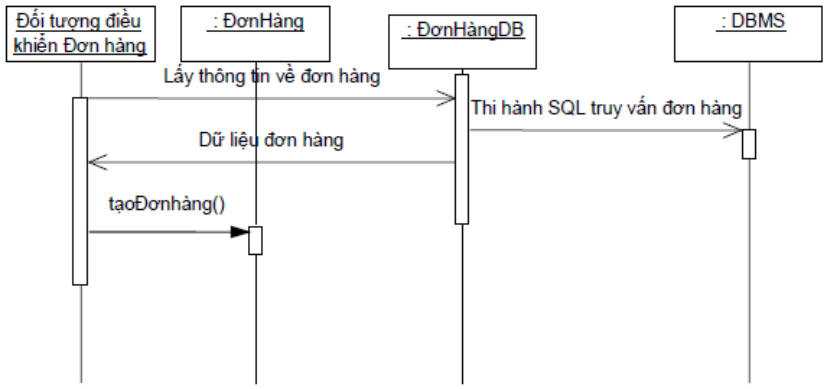
Một giải pháp khác cho cập nhật đối tượng persistent. Chúng ta xây dựng một đối tượng điều khiển và khi nào thông tin đối tượng persistent được cập nhật sẽ được quản lý bởi đối tượng này. Method capNhatĐơnHàng() sẽ được thiết kế thông minh để nhận ra một đơn hàng là tạo mới hay cập nhật. Trong giải pháp này, việc cập nhật được quản lý một cách tường minh và hệ thống chấp nhận các khoảng thời điểm thiếu sự nhất quán giữa đối tượng và dữ liệu lưu trữ về đối tượng.

Đọc đối tượng persistent

Việc lấy thông tin về các đối tượng persistent trong sơ sở dữ liệu cần thiết trước khi một ứng dụng gởi các thông điệp tới đối tượng. Vấn đề là khi gửi thông điệp cho một đối tượng thì phải đảm bảo đối tượng đó đã tồn tại trong bộ nhớ. Do vậy, chúng ta cần thiết kế một method truy vấn cơ sở dữ liệu để nhận đúng thông tin về đối tượng. Thông thường các đối tượng trong hệ thống liên kết với nhau qua mối kết hợp và vì vậy chúng ta đọc thông tin đầu tiên cho đối tượng gốc, rồi sau đó chúng ta sẽ đọc tiếp theo các đối tượng còn lại dựa theo mối kết hợp với đối tượng gốc.

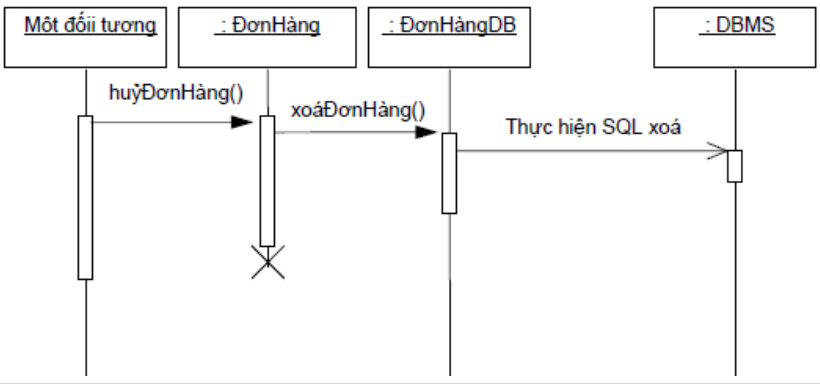


Một giải pháp khác là dùng một đối tượng điều khiển quản lý việc độc và tạo đối tượng đơn hàng được minh hoạ theo sơ đồ sau:

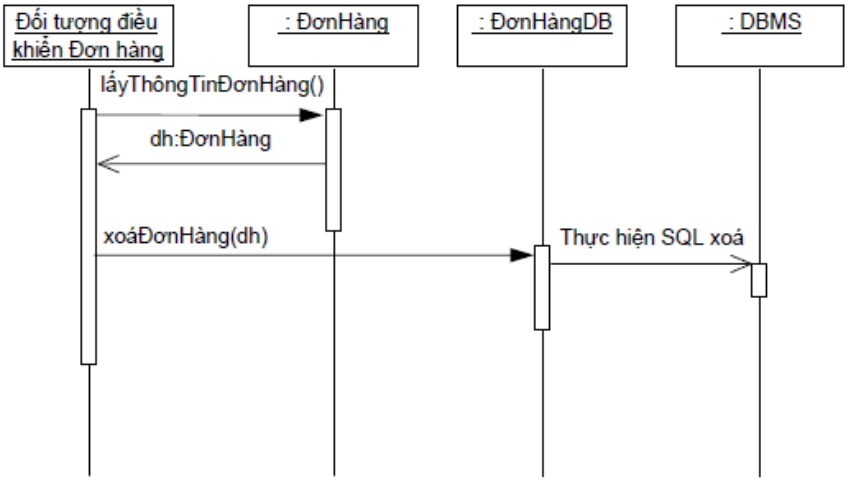


Xoá đối tượng persistent

Không giống như các đối tượng tạm thời (transtient) trong hệ thống, việc kết thúc sự hiện diện của đối tượng trong ứng dụng sẽ kết thúc vòng đời của đối tượng đó. Ngược lại, đối với đối tượng persistent, việc kết thúc vòng đời của nó đòi hỏi phải huỷ bỏ dữ liệu của nó khỏi nơi lưu trữ (hoặc ít nhất đánh dấu nó không còn hoạt động).



Một giải pháp khác dùng một đối tượng điều khiển

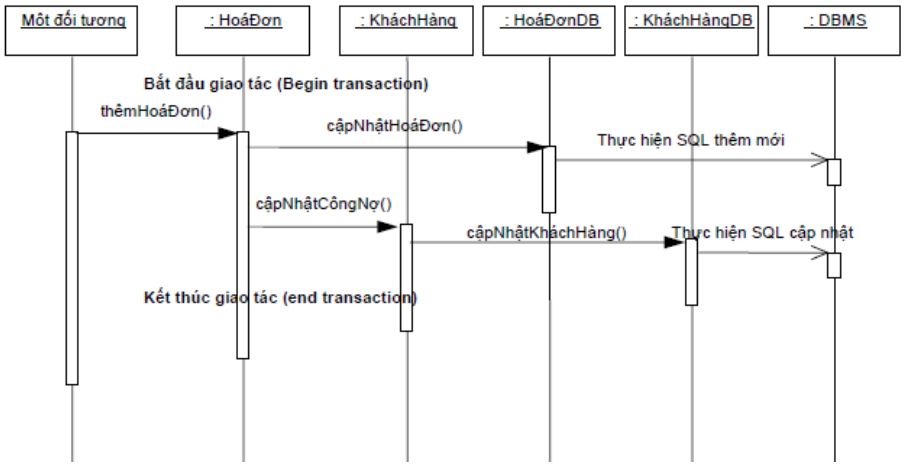


Quản lý giao tác (transaction)

Một giao tác xác định một tập các phép toán cập nhật trong cơ sở dữ liệu thành một đơn vị nhằm đáp ứng cho một yêu cầu cập nhật nghiệp vụ mà trong đó, hệ thống đòi hỏi phải cập nhật nhiều nguồn dữ liệu của nhiều đối tượng persistent khác nhau. Tất cả các phép toán cập nhật một giao tác hoặc thành công (thì giao tác thành công) hoặc không một phép toán nào thành công (thì giao tác không thành công).

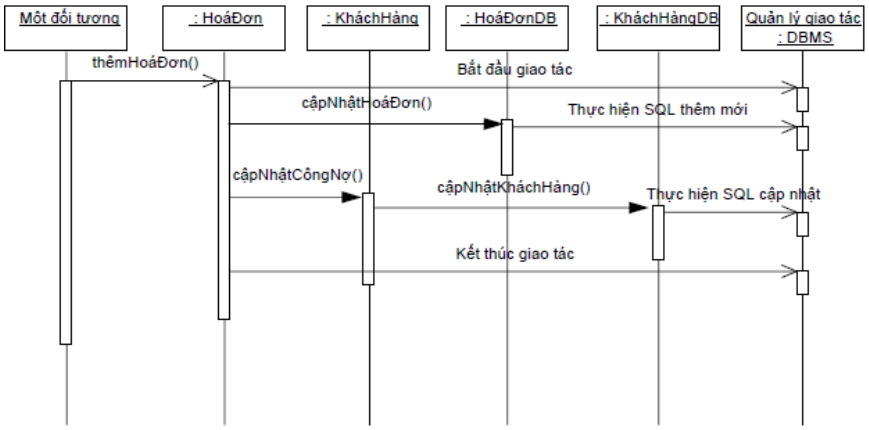
Có hai cách mô hình hoá một giao tác như sau:

- Dùng văn bản để chỉ ra bắt đầu và kết thúc một giao tác



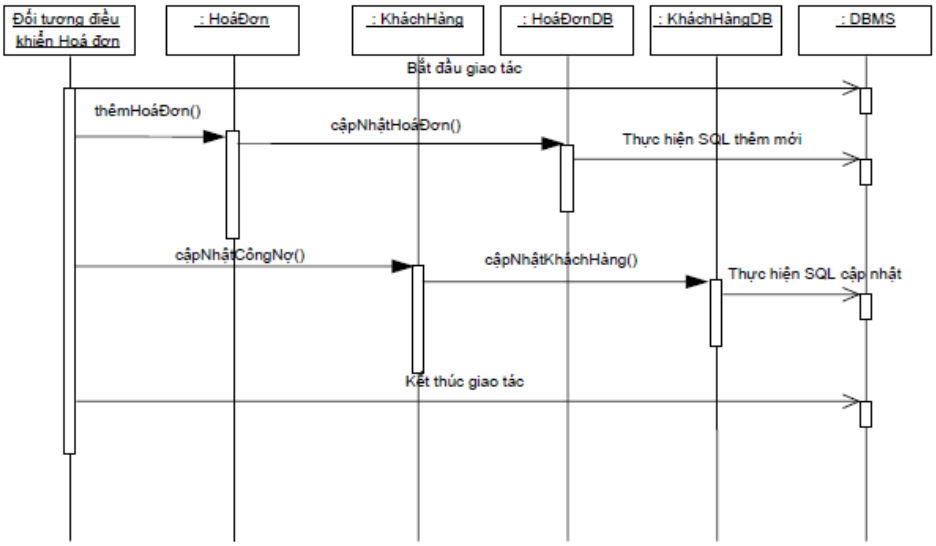
Sơ đồ trên minh hoạ việc quản lý một giao tác thêm một hoá đơn thanh toán. Giao tác này bao gồm hai phép toán cập nhật. Một là thêm mới một hoá đơn và hai là cập nhật lại thông tin công nợ của khách hàng. Giao tác kết thúc thành công nếu cả hai phép toán cập nhật đều thành công. Nếu một trong hai phép toán thực hiện không thành công thì giao tác sẽ không thành công và lúc này tất cả hai phép toán trên xem như chưa được thực hiện.

- Dùng đối tượng quản lý giao tác



Chúng ta tạo một đối tượng Quản lý giao tác, tuỳ thuộc vào ngôn nhữ lập trình mà đối tượng này có kiểu khác nhau. Đối với một số ngôn ngữ thì nó là một đối tượng kiểu kết nối (connection) hoặc đối tượng kiểu thành phần hệ quản trị cơ sở dữ liệu trong ngôn ngữ lập trình đó. Sơ đồ minh hoạ trên chọn đối tượng quản lý giao tác là một đối tượng kiểu thành phần hệ quản trị cơ sở dữ liệu.

Một giải pháp dùng đối tượng điều khiển để quản lý giao tác được minh hoạ như sơ đồ dưới đây:



Trong sơ đồ, đối tượng điều khiển sẽ quản lý việc bắt đầu và kết thúc giao tác cũng như nội dung của một giao tác bằng việc quản lý các lệnh cập nhật của giao tác. Giải pháp này so với giải pháp trên thì co ưu điểm sau: làm cho đối tượng hoá đơn độc lập với đối tượng khách hàng. Thay vì làm cho đối tượng hoá đơn phụ thuộc vào đối tượng khách hàng qua việc gọi cậpNhậtCôngNợ() ở giải pháp trên. Giải pháp này chuyển sự phụ thuộc của hoá đơn và khách hàng vào đối tượng điều khiển và giữa đối tượng hoá đơn và khách hàng lúc này hoàn toàn độc lập.

Xác định lớp truy cập dữ liệu cho hệ thống ATM

Từ các lớp persistent của hệ thống ATM được xác định là: KháchHàng, TàiKhoản, GiaoDịch, GiaoDịchRút, GiaoDịchGửi chúng ta tạo ra các lớp truy cập dữ liệu tương ứng:

KháchHàngDB

TàiKhoảnDB

GiaoDịchDB (GiaoDịch, GiaoDịchRút, GiaoDịchGửi)

Các method được xác định:

KháchHàngDB::+đọcKháchHàng()

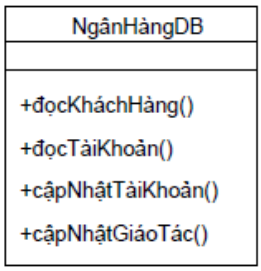
KháchHàng::+đọcTàiKhoản()

TàiKhoảnDB::+cậpNhậtTàiKhoản()

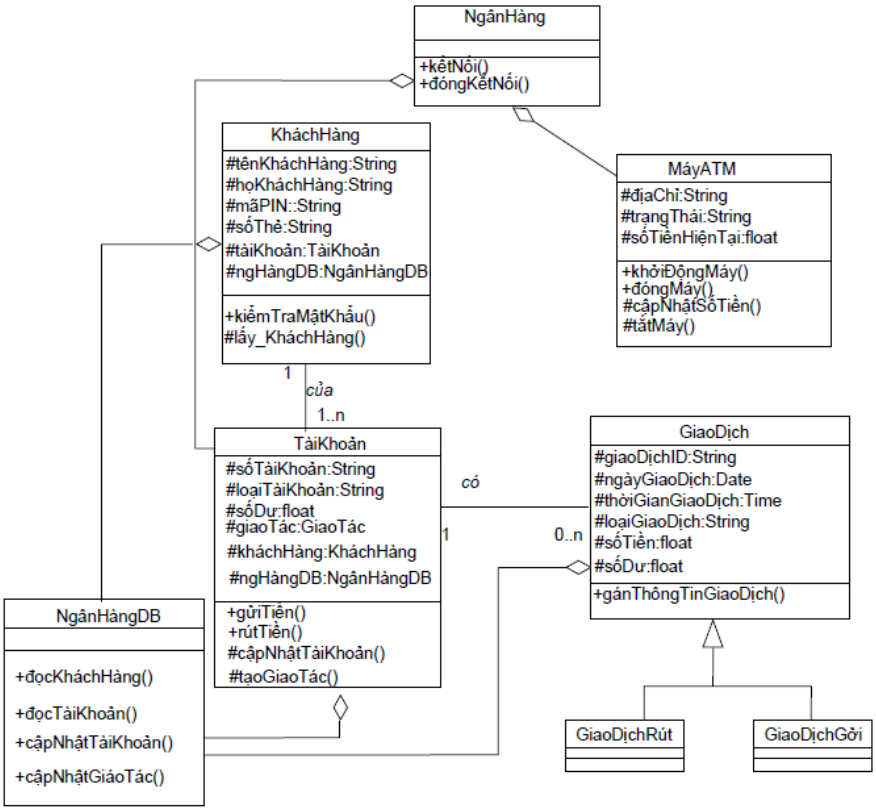
GiaoDịchDB::+cậpNhậtGiaoDịch()

Chúng ta gán mặc định cho các nghi thức ở tầng này là toàn cục (public), vì đa số đều được truy cập bởi những lớp khác, rồi trong quá trình thiết kế chi tiết cho từng method chúng ta sẽ xác định lại nghi thức này cho phù hợp nhằm đảm bảo tính bao bọc.

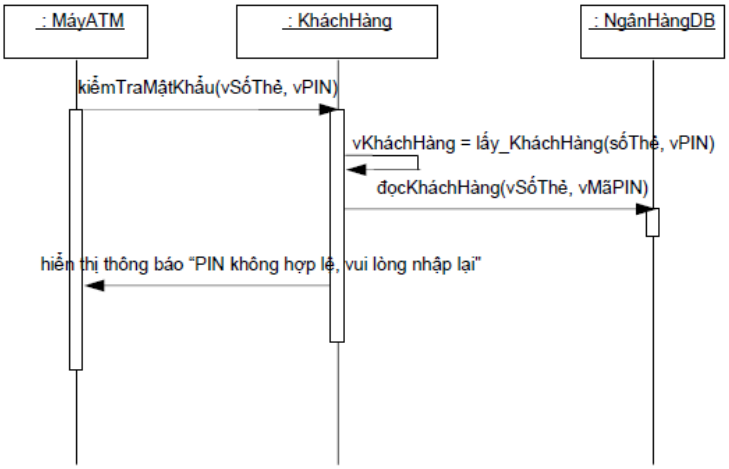
Bởi vì số lượng các method cho mỗi lớp này là ít (<= 2), cho nên chúng ta kết hợp tất cả lại thanh một lớp chung và gọi là NgânHàngDB.



Sau đó chúng ta tạo liên kết aggregation từ lớp NgânHàngDB tới lần lượt các lớp: KháchHàng, TàiKhoản và GiaoDịch. Ứng với mỗi lớp vừa mới tạo liên kết tới lớp NgânHàngDB chúng ta thêm vào một thuộc tính tham chiếu đến lớp này.



Các method của lớp NgânHàngDB lần lượt sẽ được thiết kế chi tiết như sau: *NgânHàngDB::+đọcKháchHàng (vSốThẻ:String, vMãPIN:String): KháchHàng*



Tinh chế chi tiết cho use case "Đăng Nhập" liên quan đến truy cập cơ sở dữ liệu theo sơ đồ trên, chúng ta đưa thêm vào một đối tượng NgânHàngDB và việc đọc dữ liệu thực sự về khách hàng sẽ do đối tượng này đả nhận qua method đọcKháchHàng. Sau đây chúng ta minh hoạ đoạn mã tương ứng:

KháchHàng::#lấy_KháchHàng(sốThẻ:String, mãPIN:String): KháchHàng

vNgânHàngDB: NgânHàngDB

return vNgânHàngDB.đọcKháchHàng (sốThẻ, mãPIN)

Ở đây chúng ta cần tạo ra một thể hiện của lớp truy cập NgânHàngDB và rồi gởi thông điệp tới đối tượng này để lấy thông tin về đối tượng khách hàng. Method đọcKháchHàng của NgạnHàngDB sẽ thực sự đọc dữ liệu từ cơ sở dữ liệu. Có thể minh hoạ method này truy cập cơ sở dữ liệu quan hệ bằng SQL như sau:

NgânHàngDB::+đọcKháchHàng (vSốThẻ:String, vMãPIN:String): KháchHàng

-- kết nối cơ dữ liệu ở đây

return

*SELECT * FROM KháchHàng*

WHERE Số_Thẻ = vSốThẻ AND Mã_PIN = vMãPIN

Giả sử rằng, câu truy vấn sẽ trả về dữ liệu khách hàng vào một đối tượng khách hàng trả và về cho method này. Tuỳ thuộc vào ngôn ngữ cài đặt cụ thể được chọn mà chúng ta thay đổi cho phù hợp, đoạn mã lệnh trên đây chỉ minh hoạ ý tưởng và cách viết độc lập ngôn ngữ.

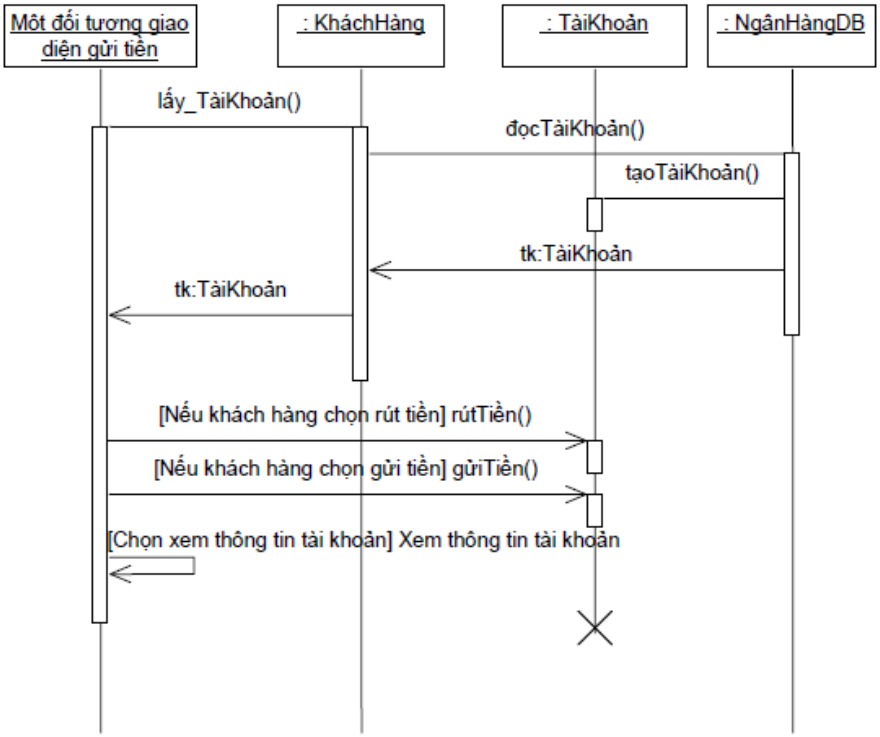
NgânHàngDB::+đọcTàiKhoản(sốThẻ:String): TàiKhoản

Đây là một method đọc dữ liệu từ cơ sở dữ liệu để tạo một đối tượng persistent. Do đó, húng ta có thể áp dụng một trong hai mẫu sơ đồ đọc đối tượng persistent ở phần trên.

Các use case liên quan đến method đọcTàiKhoản để tạo một đối tượng tài khoản phục vụ cho hoạt động của use case bao gồm: Giao dịch, Gửi tiền, Rút tiền, Truy vấn thông tin tài khoản.

Sau đây chúng ta minh hoạ sơ đồ tuần tự thiết kế chi tiết các use case này qua sự tượng tác giữa tầng nghiệp vụ và tầng truy cập dữ liệu, đặc biệt là tương tác nhằm tạo đối tượng tài khoản.

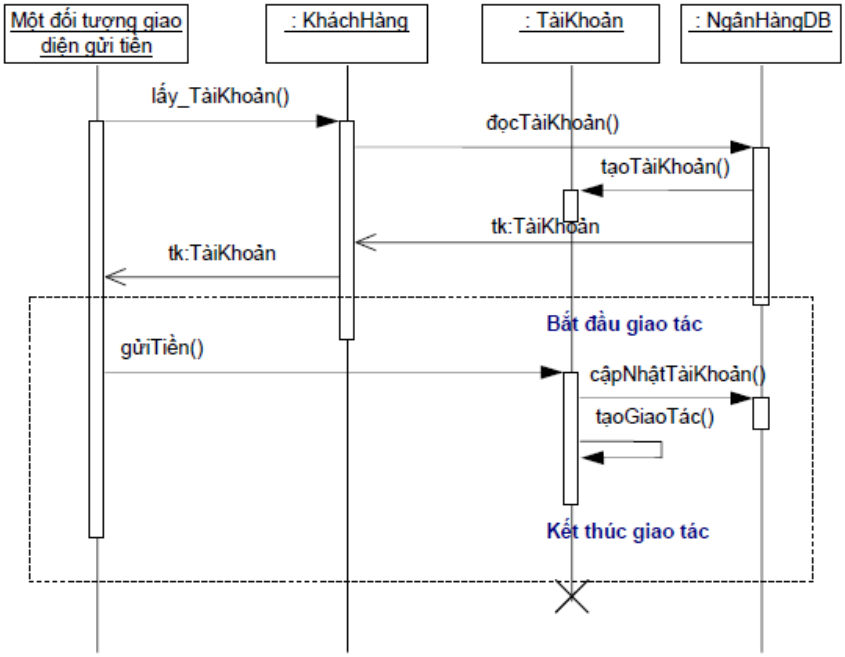
Use case Giao dịch



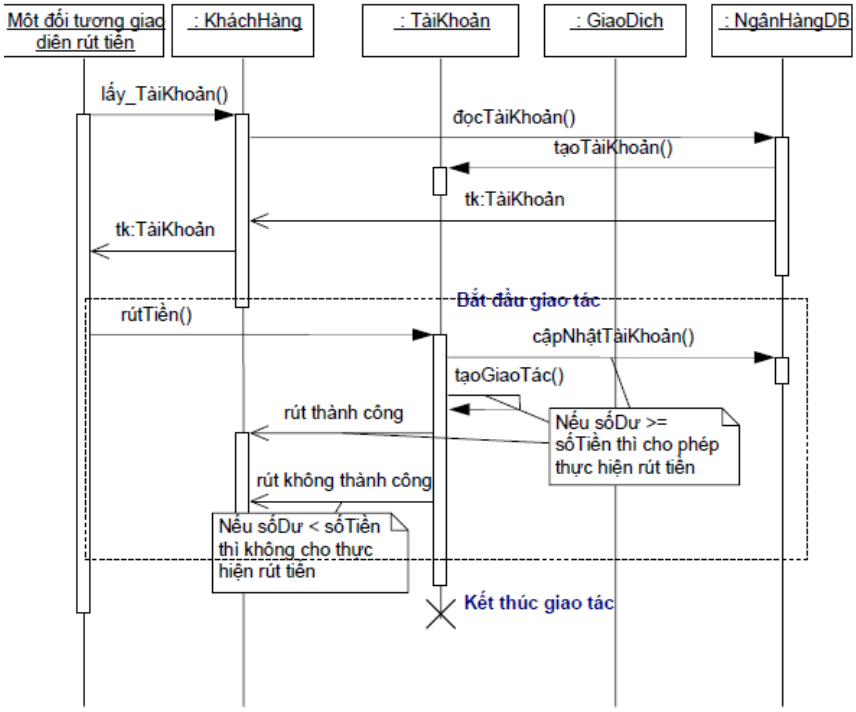
Sơ đồ tuần tự về use case Giao dịch được thiết kế: đầu tiên chúng ta giả lập một đối tượng tầng giao diện sẽ tương tác tới một đối tượng khách hàng (đã được tạo ra khi đăng nhập thành công) để tham khảo đến tài khoản của khách hàng bằng cách gởi thông điệp đến đối tượng truy cập dữ liệu là NgânHàngDB để thực hiện việc đọc dữ liệu từ cơ sở dữ liệu và tạo ra đối tượng tài khoản. Sau đó, đối tượng này sẽ được trả về cho đối tượng giao diện như một đối tượng để tham chiếu. Đối tượng giao diện sẽ được tinh chế trong phần sau của chương này.

Tuỳ theo loại giao dịch mà khách hàng chọn thì use cae này sẽ gọi thực hiện các use case mở rộng tương ứng. Các use case Rút tiền và Gửi tiền sẽ thực hiện dựa trên đối tượng tài khoản vừa đọc đọc được; use case Truy vấn thông tin tài khoản sẽ hiển thị thông tin về tài khoản vừa đọc được, do đó, nó được thực hiện bởi một đối tượng giao diện.

Use case Gửi tiền



Use case Rút tiền



Việc thực hiện gửi tiền hoặc rút tiền sẽ do đối tượng tài khoản đảm nhận, đối tượng này được tạo ra do thừa hưởng từ use case Giao dịch. Sau đó, được thiết kế bằng một giao tác gồm hai gửiTiền() hoạt động: cập nhật lại số dư tài khoản và tạo một giao tác mới. Việc tạo một giao tác mới sẽ được thực hiện bởi method tạoGiaoTác() sẽ được thiết kế trong phần tiếp theo sau.

Sau đây là đoạn mã minh hoạ cho các method:

KháchHàng::-lấy_TàiKhoản(sốThẻ:String): TàiKhoản

vNgânHàngDB: NgânHàngDB

return vNgânHàngDB.đọaTàiKhoản(sốThẻ)

NgânHàngDB::+đọcTàiKhoản(vSốThẻ:String): TàiKhoản

-- kết nối cơ sở dữ liệu ở đây

Return

SELECT * FROM TàiKhoản

WHERE Số_Thẻ = vSốThẻ

TàiKhoản::+gửiTiền(sốTiền:float)

vNgânHàngDB: NgânHàngDB

NgânHàngDB.cậpNhậtTàiKhoản (sốTàiKhoản, sốDư + sốTiền)

tạoGiaoTác("gửi", sốTiền, sốDư + sốTiền)

TàiKhoản::+rútTiền(sốTiền:float): String

vNgânHàngDB: NgânHàngDB

if sốDư < sốTiền

return "Số tiền rút vượt quá số dư tài khoản"

else

NgânHàngDB.cậpNhậtTàiKhoản (sốTàiKhoản, sốDư + sốTiền)

tạoGiaoTác("gửi", sốTiền, sốDư + sốTiền)

return ""

endif

NgânHàngDB::+cậpNhậtTàiKhoản(vSốTàiKhoản:String, vSốDư:float)

Việc sử dụng method này được minh hoạ theo sơ đồ của use case Gửi tiền và Rút tiền. Sau đây là đoạn mã minh hoạ:

NgânHàngDB::+cậpNhậtTàiKhoản(vSốTàiKhoản:String, vSốDư:float)

UPDATE TàiKhoản

SET sốDư = vSốDư

WHERE Số_TK = vSốTàiKhoản

NgânHàngDB::+cậpNhậtGiaoDịch()

Để thiết kế chi tiết method này chúng ta tiếp tục với các use case Gửi tiền và Rút tiền qua việc mô tả chi tiết method tạoGiaoTác() của đối tượng tài khoản qua sự tương tác giữa tầng nghiệp vụ và tầng truy cập dữ liệu.

TàiKhoản::#tạoGiaoTác(loạiGD:String, sốTiền:float, sốDư: float)

vNgânHàngDB: NgânHàngDB

vGiaoDịch = tạoMới(GiaoDịch)

vGiaoDịch.gánThôngTin(Date(today), Time(now), loạiGD, sốTiền, sốDư,

sốTàiKhoản)

vNgânHàngDB.cậpNhậtGiaoDịch(sốTàiKhoản, loạiGD, sốTiền, sốDư)

NgânHàngDB::+cậpNhậtGiaoDịch(vSốTàiKhoản:String, vLoạiGD:String, vSốDư:float)

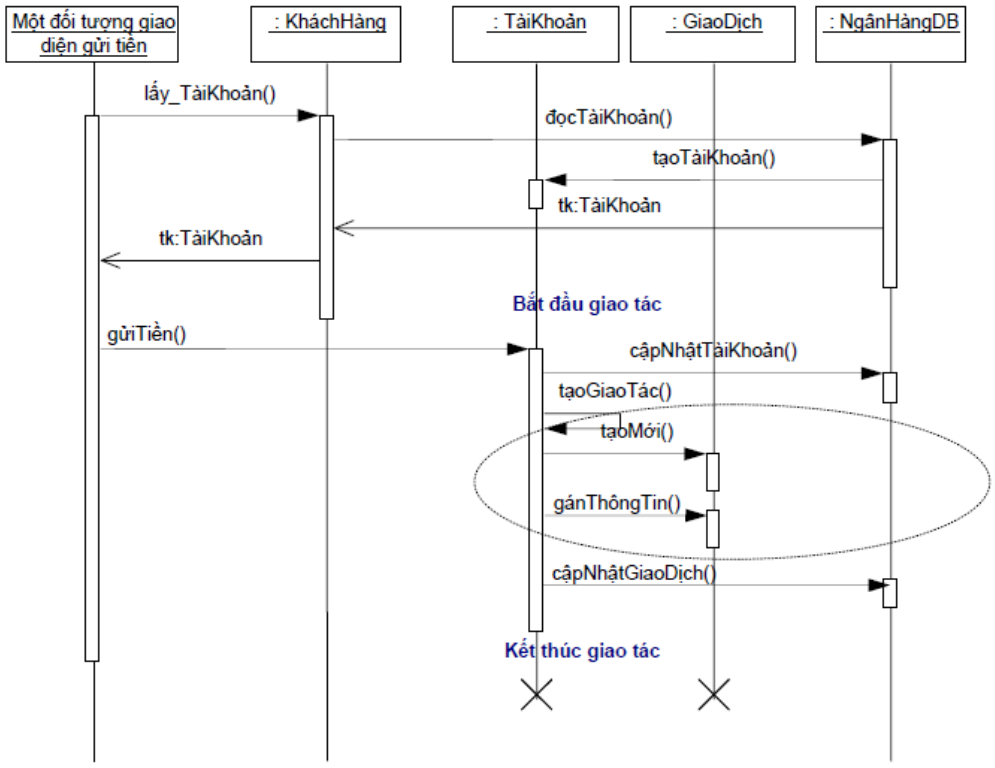
--kết nối cơ sở dữ liệu ở đây

vGD_ID: String

vGD_ID = Tạo_GD_ID()

INSERT INTO GiaoDịch (GD_ID, Ngày_GD, Giờ_GD, Loại_GD, Số_Tiền, Số_Dư,

Số_TK) VALUES (vGD_ID, Date(today), Time(now), vLoạiGD, vSốTiền, vSốDư, vSốTàiKhoản)



2.3 Xác định lớp tầng giao diện người dùng (user interface layer)

Bao gồm các đối tượng hệ thống mà người dùng sẽ tương tác và các đối tượng được dùng để quản lý hoặc điều khiển giao diện. Các class ở tầng này đảm nhận hai công việc chính:

- *Trả lời tương tác người dùng:* các đối tượng mức này phải được thiết kế để chuyển dịch những hành động của người dùng tới một tình huống xử lý phù hợp. Có thể là mở hoặc đóng một giao diện khác, hoặc gởi một thông điệp xuống tầng tác nghiệp hoặc khởi động một vài tiến trình ở tầng tác nghiệp
- Hiển thị các đối tượng tác nghiệp: tầng này phải trình bày một hình ảnh tốt nhất các đối tượng tác nghiệp tới người dùng trong một giao diện. Điều này có nghĩa là một hình ảnh trực quan thao tác được có thể bao gồm: các textbox, listbox, commbobox, page, form, menu, toolbar, ….

Tiến trình thiết kế tầng giao diện

Một tiến trình thiết kế tầng giao diện bao gồm 4 bước sau:

1. Xác định các đối tượng ở tầng giao diện: đây là một hoạt động diễn ra luôn cả trong giai đoạn phân tích hệ thống. Mục tiêu chính là để xác định các lớp tương tác với các tác nhân con người thông qua việc phân tích các use case trong giai đoạn phân tích.

   Như đã mô tả trong các chương trước, mỗi use case bao gồm các tác nhân và các công việc mà tác nhân muốn hệ thống thực hiện. Do đó, nó mô tả một bức tranh đầy đủ về các yêu cầu giao diện của hệ thống. Trong giai đoạn này chúng ta cũng cần thiết tập trung vào cách thức cài đặt giao diện. Các sơ đồ tuần tự và hợp tác của use case cũng giúp chúng ta xác định chính xác yêu cầu về giao diện.

2. Xác định các lớp tầng giao diện

a. Xác định các đối tượng tầng giao diện: xây dựng sơ đồ lớp mô tả các đối tượng giao diệ

b. Tạo bản mẫu giao diện (prototype): sau khi xác định mô hình thiết kế, chúng ta chuẩn bị cho một bản mẫu của giao diện. Thông thường các bản mẫu này đã được xác định trong những giai đoạn trước, nếu vậy thì chúng ta hãy cố gắng tích hợp với các đối tượng đã được xác định nhằm thống nhất các lớp giao diện trong sơ đồ lớp và các giao diện thiết kế.

c. Xác định hành vi và thuộc tính cho các lớp giao diện: phân tích lại hành động của người dùng trong use case và tinh chế lại sơ đồ tuần tự để phát hiện các method cũng như xem xét lại các mối quan hệ của các đối tượng để xác định các thuộc tính (chủ yếu là các thuộc tính tham chiếu) cho lớp tầng giao diện.

3. Thử nghiệm giao diện.

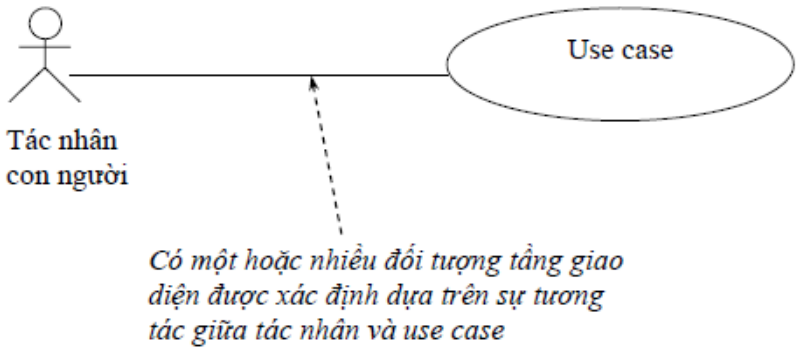
4. Tinh chế và lặp lại các bước trên

Xác định các lớp tầng giao diện qua việc phân tích use case

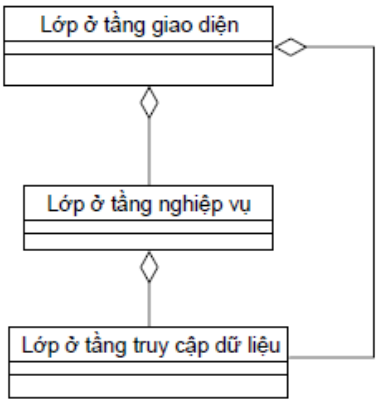
Trong một use case, đối tượng tầng giao diện xử lý tất cả trao đổi với tác nhân. Thực sự, các đối tượng này hoạt động như một vùng đệm giữa người dùng và phần còn lại của hệ thống là các đối tượng nghiệp vụ. Một đối tượng giao diện có thể tham gia vào nhiều use case. Bắt đầu từ use case, chúng ta xác định được các mục tiêu và nhiệm vụ của người dùng. Những người dùng khác nhau sẽ có những nhu cầu khác nhau trên giao diện, ví dụ, các người dùng chuyên nghiệp thì cần một giao diện có tính hiệu quả trong khi các người dùng bình thường thì cần có giao diện dễ sử

dụng. Do đó, với các đối tượng người dùng khác nhau mà các giao diện được thiết kế sẽ khác nhau về mục tiêu, trách nhiệm, cách vận hành và hình thức trình bày. Việc xác định các lớp tầng giao diện gồm hai bước sau:

- Với mỗi lớp (tầng nghiệp vụ), nếu lớp đó có tương tác với một tác nhân con người trong một use case, chúng ta thức hiện như sau:
  - Xác định các đối tượng giao diện cho lớp đó, các trách nhiệm cũng như các yêu cầu của các đối tượng này. Việc này được thực hiện bằng cách phân tích lại sơ đồ tuần tự hoặc hợp tác của use case.



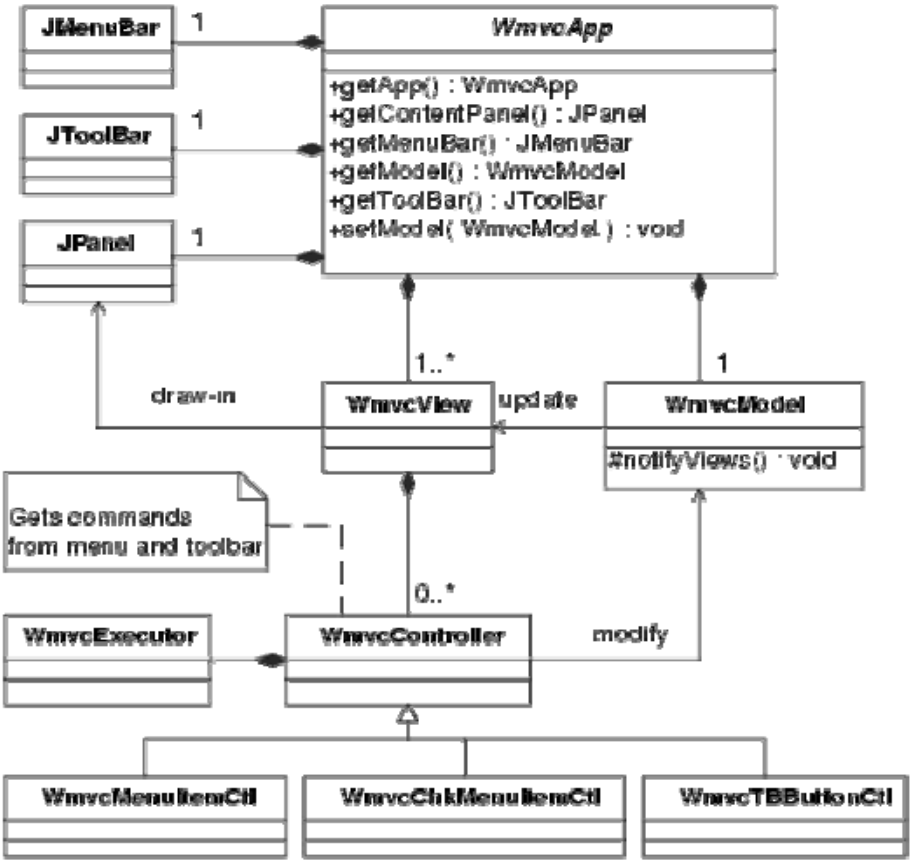
  - Xác định sự liên kết giữa các đối tượng giao diện: các lớp giao diện cũng giống như các lớp khác, đều có mối quan hệ với các lớp ở tầng nghiệp vụ cùng tham gia trong một use case với nó. Các mối liên kết này cũng được xác định tương tự như của các lớp trong tầng nghiệp vụ. Sự liên kết này thường theo sơ đồ dưới đây.



- Lặp lại các bước trên và tinh chế

Ưu điểm của của việc sử dụng use case để xác định các đối tượng ở tầng giao diện là nó tập trung vào người dùng và đưa người dùng vào như một phần của kế hoạch thiết kế nhằm tìm ra một giao diện tốt nhất cho người dùng. Khi các đối tượng này đã được xác định, chúng ta phải xác định các thành phần cơ bản hoặc các đối tượng được dùng trong các công việc cũng như là các hành vi và các đặc điểm tạo ra sự khác biệt của mỗi loại đối tượng, bao gồm luôn cả mối quan hệ giữa các đối tượng và giữa đối tượng và người dùng.

Tuy nhiên, trong thiết kế giao diện khi chúng ta đã xác định môi trường phát triển thì chúng ta cũng nên tìm hiểu các khung mẫu (framework) cũng như các thư viện mà môi trường đó hỗ trợ để phát huy việc tái sử dụng trong thiết kế giao diện và tận dụng tối đa các hỗ trợ từ môi trường.



Một dạng khung mẫu MVC (Model – View - Controller) được hỗ trợ trong môi trường Java áp dụng cho các thiết kế giao diện ứng dụng với Java

Xây dựng bản mẫu (prototype) cho giao diện

Việc xây dựng bản mẫu giao diện thường được thiết kế trong giai đoạn xác định yêu cầu và phân tích hệ thống. Công việc này được thực hiện sử dụng một công cụ gọi là CASE Tool hoặc các công cụ phát triển. Bao gồm ba bước sau:

- Tạo các đối tượng giao diện (như là các button, các vùng nhập liệu,…)
- Liên kết và gán các hành vi hoặc hành động thích hợp tới các đối tượng giao diện này và các sự kiện của nó.
- Thử nghiệm và lặp lại bước một

Thiết kế tầng giao diện cho hệ thống ATM

Xác định các đối tượng ở tầng giao diện, các yêu cầu và trách nhiệm của nó

Đối với mỗi lớp ở tầng nghiệp vụ: NgânHàng, MáyATM, KháchHàng, TàiKhoản, GiaoDịch, GiaoDịchGửi, GiaoDịchRút chúng ta xác định các lớp giao diện của nó bằng việc quan sát các sơ đồ tuần tự và hợp tác của các use case: Đăng nhập, Đăng nhập không hợp lệ, Giao dịch,

Rút tiền, Gửi tiền, Truy vấn thông tin tài khoản, Khởi động hệ thống, Đóng hệ thống. Chúng ta xác định được các lớp tầng giao diện sau đây:

KháchHàngGD: biểu diễn giao diện tương tác giữa khách hàng và use case Đăng nhập, Đăng nhập không hợp lệ

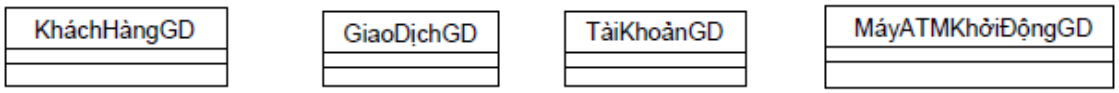
GiaoDịchRútGD: biểu diễn tương tác giữa khách hàng và use case Rút tiền

GiaoDịchGửiGD: biểu diễn tương tác giữa khách hàng và use case Gửi tiền

Tài khoảnGD: biểu diễn tương tác giữa khách hàng và use case Truy vấn thông tin tài khoản

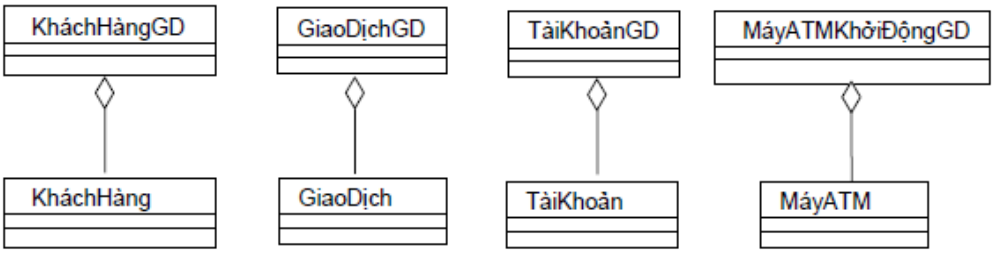
MáyATMKhởiĐộngGD: biểu diễn tương tác giữa nhân viên vận hành và use case Khởi động hệ thống.

Các đối tượng giao diện GiaoDịchGửiGD, GiaoDịchRútGD có một hình thức trình bày chung của giao dịch và chỉ thay đổi loại giao dịch. Do đó, chúng ta gom lại thành một lớp và gọi là: GiaoDịchGD.



Xác định sự liên kết các lớp của tầng giao diện với các lớp của tầng nghiệp vụ

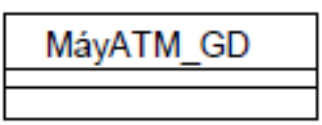
Mối kết hợp được xác lập là mối kết hợp dạng aggregation như sau: với mỗi lớp giao diện chúng ta tạo liên kết aggregation tới lớp tương ứng trong tầng nghiệp vụ. Mối liên kết này cho biết trong các thể hiện của lớp tầng giao diện sẽ sử dụng đối tượng ở tầng nghiệp vụ như là một thành phần để thực hiện gởi thông điệp.



Xác định các đối tượng giao diện điều khiển như là: toolbar, menu, form điều khiển,....

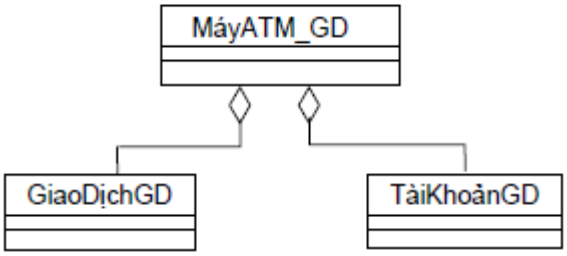
Ngoài các lớp được xác định dựa vào use case, chúng ta xác định thêm các đối tượng giao diện có nhiệm vụ điều khiển chính, giao diện chính, thực đơn, tool bar,…

Với hệ thống ATM chúng ta có thêm một đối tượng giao diện điều khiển chính hoạt động giao diện của máy ATM gọi là MáyATM_GD.



Vì đối tượng của lớp MáyATM_GD sẽ gởi thông điệp đến tất cả các đối tượng giao diện rút, gửi và truy vấn thông tin nhằm điều khiển các giao diện này. Do đó, trước khi gởi thông điệp thì đối

tượng lớp MáyATM_GD sẽ phải tạo ra các đối tượng kia như là một thành phần của nó. Chính vì vậy mà chúng ta sẽ xác lập mối kết hợp aggregation từ lớp MáyATM_GD đến các lớp: GiaoDichGD, TaiKhoanGD



Thiết kế giao diện mẫu (prototype)

Xác định các method

Mục tiêu của các đối tượng giao diện là cho phép người dùng thao tác với hệ thống nhằm phục vụ cho công việc nghiệp vụ như là: nhấn một nút, nhập vào một ký tự,…. Các thao tác này sẽ được chuyển tới các đối tượng tầng nghiệp vụ để xử lý. Khi hoàn thành xử lý, giao diện sẽ được cập nhật lại thông tin nhằm hiển thị các thông tin mới hoặc mở một giao diện mới,…. Xác định hành vi cho đối tượng giao diện bằng cách xác định các sự kiện mà của hệ thống phải đáp ứng tương ứng tới các thao tác người dùng trên giao diện và các hành động khi sự kiện xảy ra. Một hành động được xem như một hành vi và được hình thành bởi sự kết hợp một đối tượng với một thông điệp gởi tới đối tượng đó. Xem xét lại từng đối tượng giao diện theo từng use use case của hệ thống ATM chúng ta lần lượt xác định các method cho các lớp giao diện.

KháchHàngGD - Use case Đăng nhập

Khi khách hàng đưa thẻ ATM vào máy các sự kiện và hành động:

- Đưa thẻ vào máy -> hiển thị giao diện đăng nhập (KháchHàngGD)
- Khách hàng chọn đồng ý -> kiểm tra mật khẩu (KháchHàng)

  -> hiển thị giao diện điều khiển chính (MáyATM_GD)

  -> thông báo nếu đăng nhập không thành công

  (KháchHàngGD)

  -> đóng giao diện đăng nhập (KháchHàngGD)
- Khách hàng chọn huỷ bỏ -> đóng giao diện đăng nhập (KháchHàngGD)

Chúng ta xác định được các method:

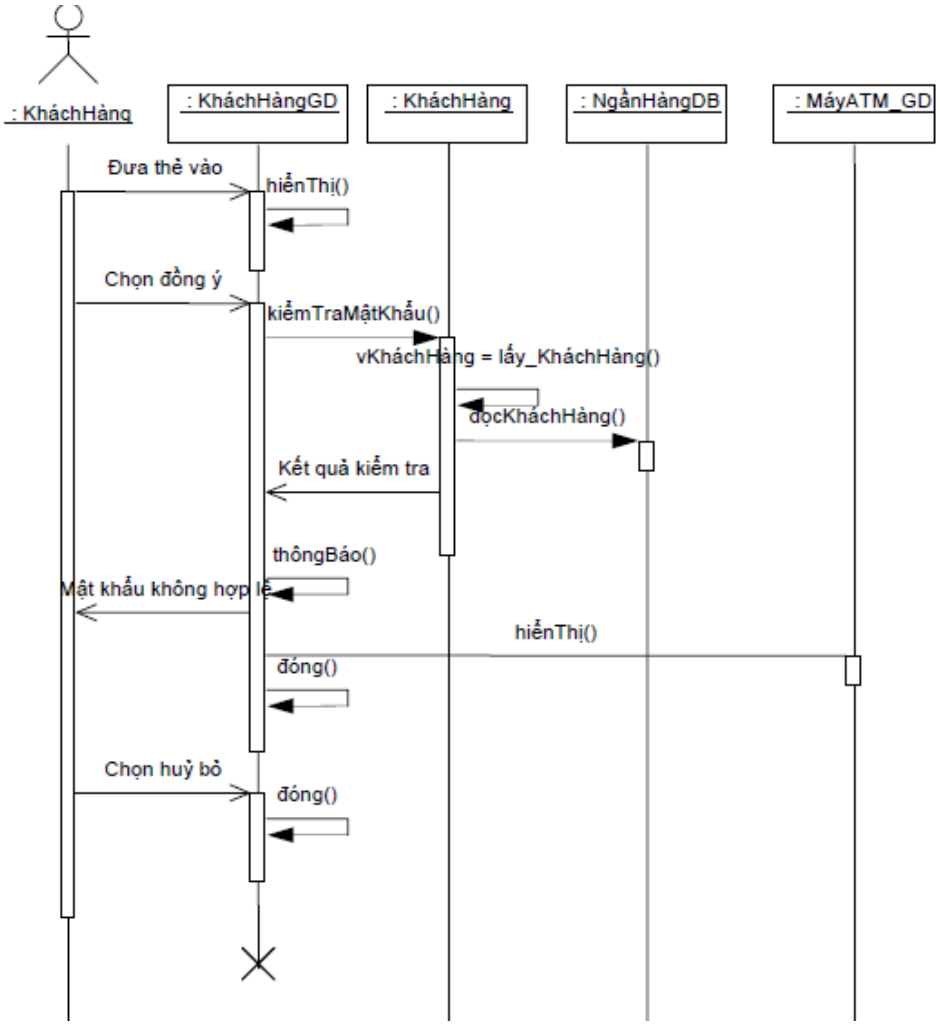
KháchHàngGD::+hiểnThị()

KháchHàngGD::-thôngBáo(thôngBáo:String)

KháchHàngGD::+đóng()

MáyATM_GD::+hiểnThị()

Sơ đồ tuần tự tinh chế tầng giao diện cho use case Đăng nhập

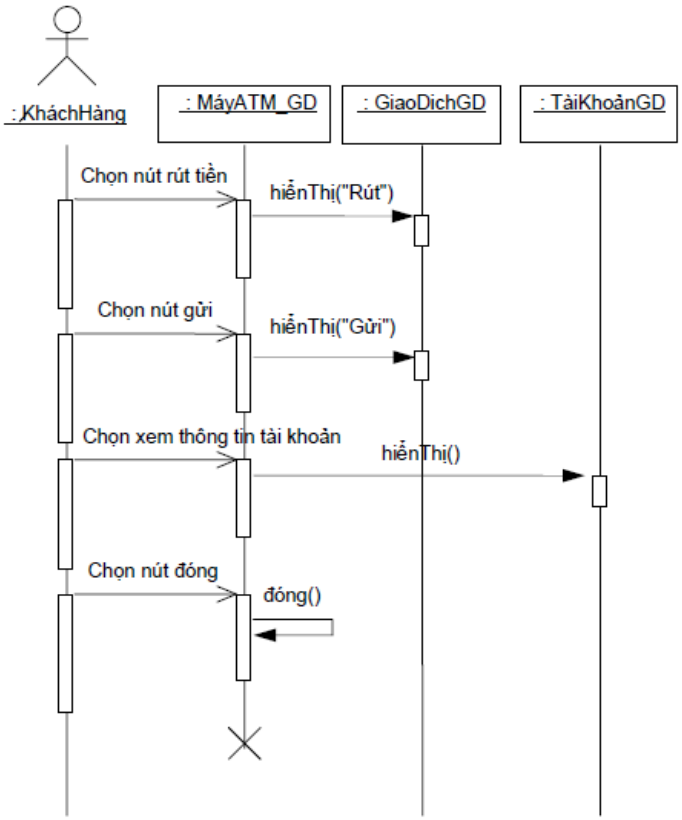


MáyATM_GD

Khi giao diện chính của máy được hiển thị các tương tác của khách hàng làm phát sinh các sự

kiện và các hành động:

- Chọn nút rút tiền -> hiển thị giao diện rút tiền (GiaoDịchGD)
- Chọn nút gửi tiền -> hiển thị giao diện gửi tiền (GiaoDịchGD)
- Chọn nút xem tài khoản -> hiển thị giao diện xem thông tin tài khoản (TàiKhoảnGD)
- Chọn nút thoát -> đóng giao diện chính (MáyATM_GD)



Chúng ta xác định được các method

GiaoDịchGD::+hiểnThị(loạiGD:String)

TàiKhoản::+hiểnThị()

MáyATM_GD::+đóng()

GiaoDịchGD - Use case Rút tiền

Khi khách hàng chọn dịch vụ rút tiền từ giao diện chính các sự kiện và hành động:

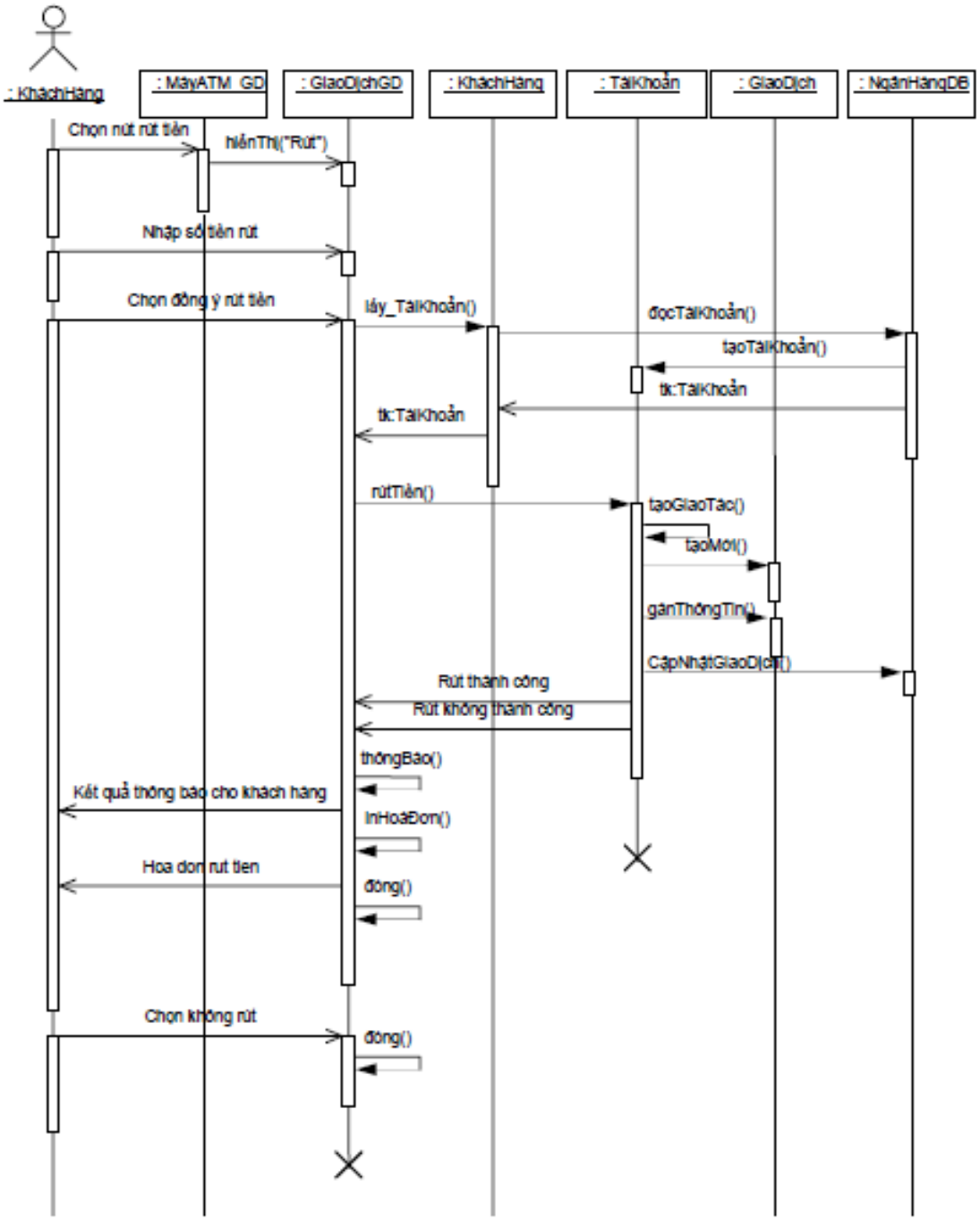
- Chọn rút tiền -> hiển thị giao diện rút tiền (GiaoDịchGD)
- Khách hàng chọn rút tiền -> thực hiện rút tiền (TàiKhoản)
  - ➔ thông báo kết quả (GiaoDịchGD)
  - ➔ in hoá đơn rút (GiaoDịchGD)
  - ➔ đóng giao diện rút tiền (GiaoDịchGD)
- Khách hàng chọn đóng -> đóng giao diện rút tiền (GiaoDịchGD)

Chúng ta xác định được các method:

GiaoDịchGD::-thôngBáo(thôngBáo:String)

GiaoDịchGD::-inHoáĐơn()

GiaoDịchGD::+đóng()

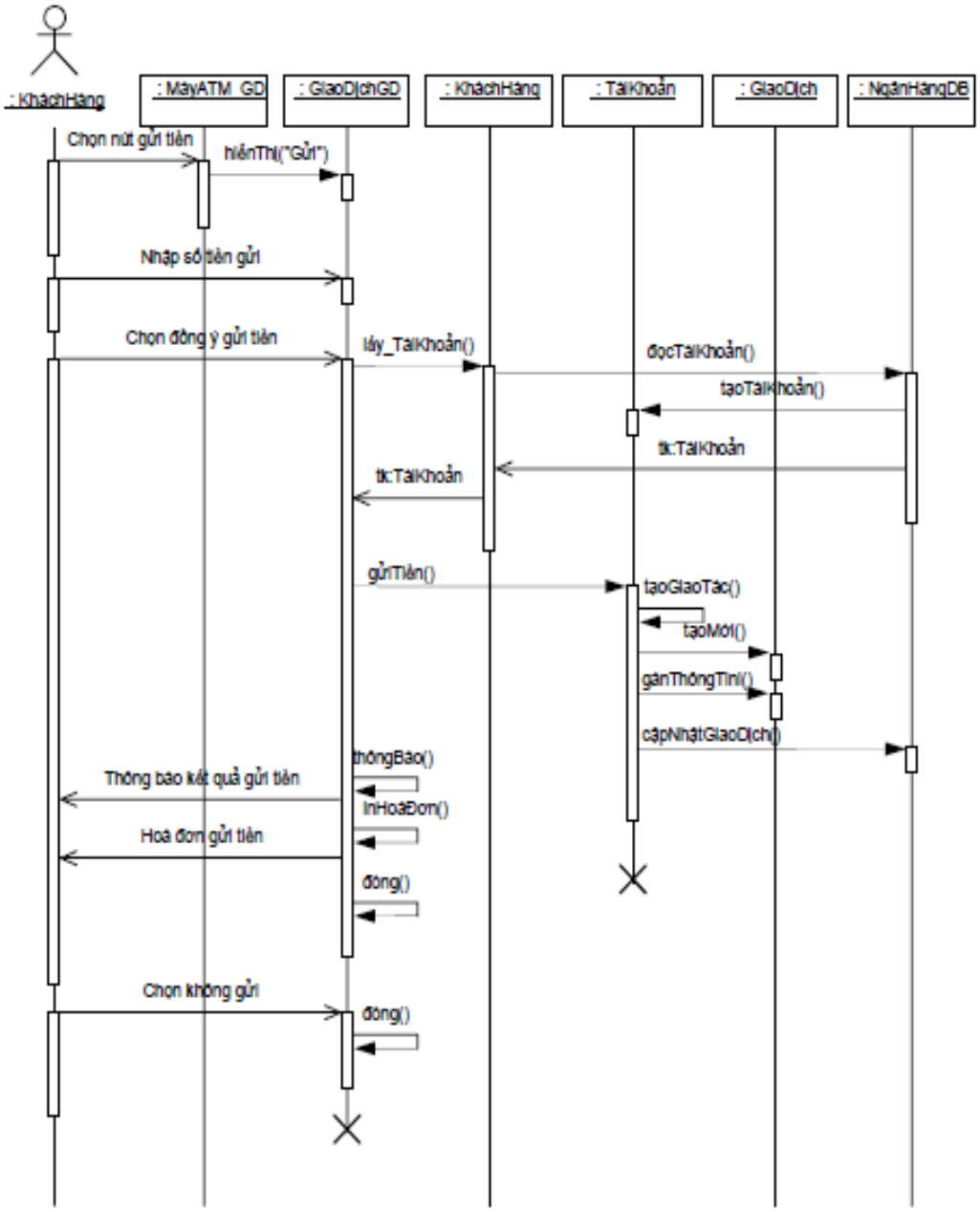


GiaoDịchGD - Use case Gửi tiền

Khi khách hàng chọn dịch vụ gửi tiền từ giao diện chính các sự kiện và hành động:

- Chọn gửi tiền -> hiển thị giao diện gửi tiền (GiaoDịchGD)
- Khách hàng chọn gửi tiền -> thực hiện gửi tiền (TàiKhoản)
  - ➔ thông báo kết quả (GiaoDịchGD)
  - ➔ in hoá đơn gửi (GiaoDịchGD)
  - ➔ đóng giao diện gửi tiền (GiaoDịchGD)
- Khách hàng chọn đóng -> đóng giao diện gửi tiền (GiaoDịchGD)

Các method xác định trong use case này giống như của use case Rút tiền



Trong sơ đồ tuần tự trên cho rút tiền và gưi tiền, chúng ta quan sát ba đối tượng: Kháchhàng (tác nhân), và hai đối tượng giao diện MáyATM_GD, GiaoDịchGD. Bắt đầu các dòng từ tác nhân khách hàng mô tả các thao tác khách hàng trên giao diện và trả lời của hệ thống từ giao diện. Từ các dòng này, các đối tượng giao diện sẽ tương tác với các đối tượng ở tầng nghiệp vụ bằng các thông điệp nhằm thực hiện tác vụ của hệ thống.

TàiKhoảnGD - Use case Truy vấn thông tin tài khoản3

Khi khách hàng chọn truy vấn thông tin tài khoản từ giao diện chính các sự kiện và hành động:

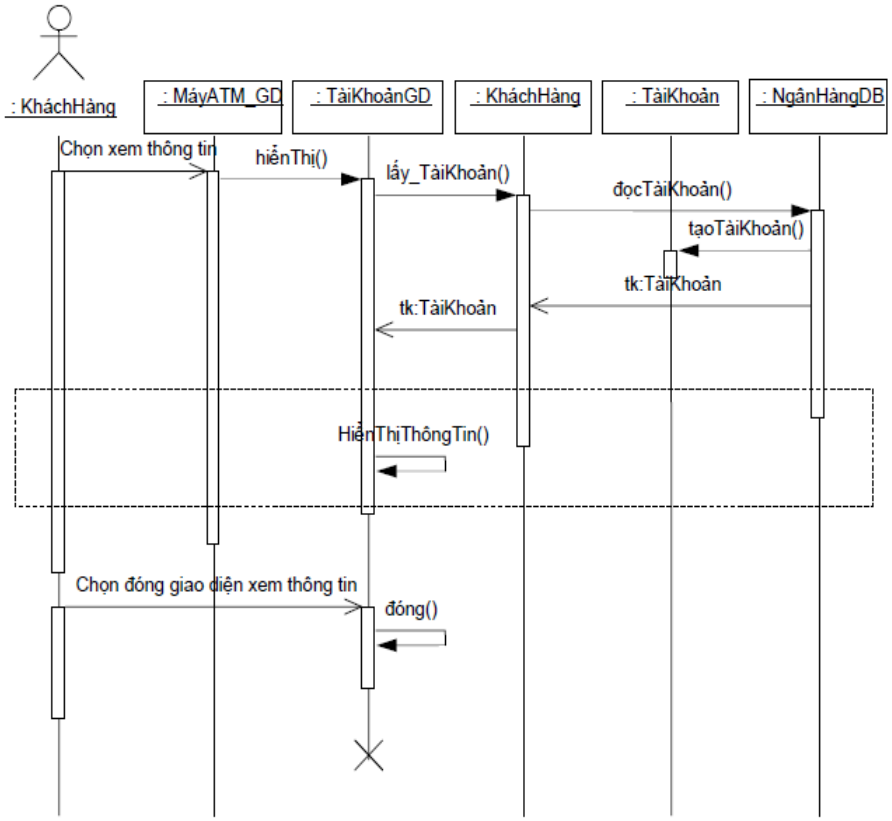
- Chọn xem thông tin tài khoản -> hiển thị giao diện truy vấn (TàiKhoảnGD)
  - ➔ đọc thông tin tài khoản (KháchHàng)
  - ➔ hiển thị thông tin tài khoản (TàiKhoảnGD)
- Khách hàng chọn đóng -> đóng giao diện truy vấn (TàiKhoảnGD)

Chúng ta xác định được các method:

TàiKhoảnGD::+hiểnThị()

TàiKhoảnGD::-hiểnThịThôngTin(tk:TaiKhoan)

TàiKhoảnGD::+đóng()



MáyATMKhởiĐộngGD - Use case Khởi động hệ thống

Khi máy được bật công tắc khởi động các sự kiện và hành động:

- Khởi động máy hoàn thành -> hiện thị giao diện khởi động máy (MáyATMKhởiĐộngGD)
- Nhân viên chọn đóng -> cập nhật số tiền cho hiện hành cho máy (MáyATM)

  ➔ thực hiện kết nối tới mạng ngân hàng (NgânHàng)

  ➔ đóng giao diện khởi động (MáyATMKhởiĐộngGD)

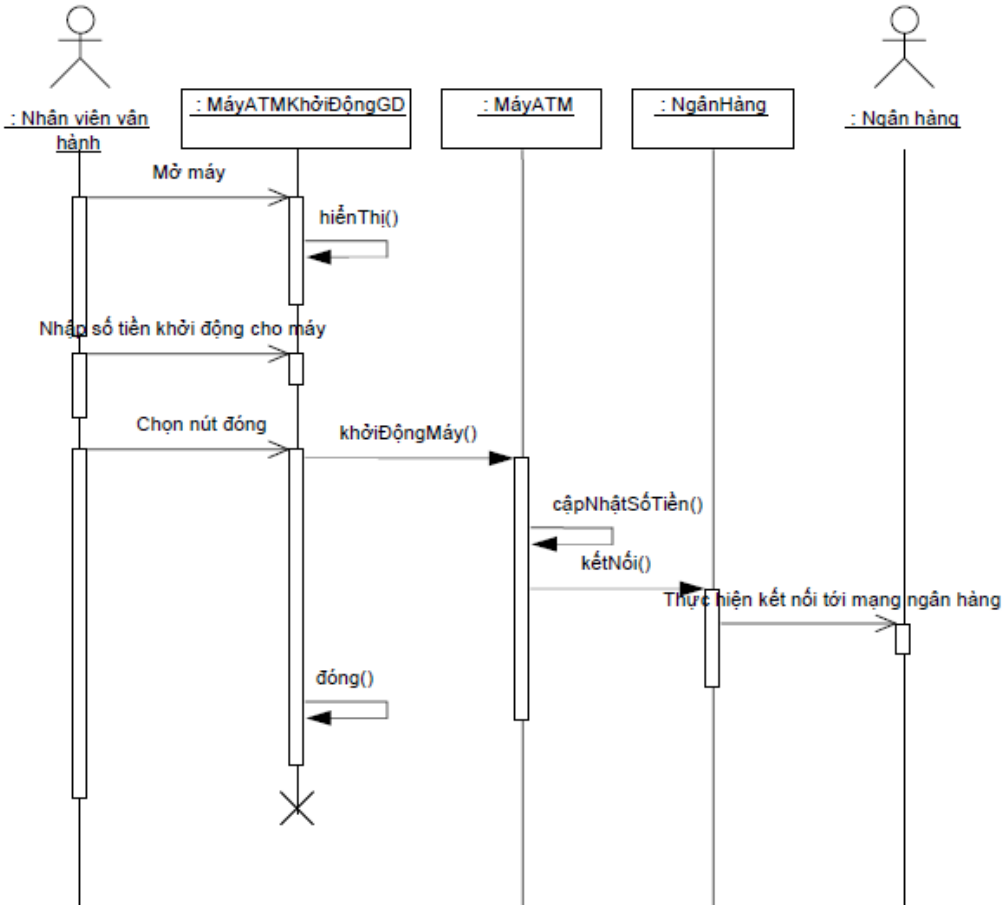
Chúng ta xác định được các method:

MáyATMKhởiĐộngGD::+hiểnThị()

MáyATM::+cậpNhậtSốTiền(sốTiền:float)

NgânHàng::+kếtNối()

MáyATMKhởiĐộngGD::+đóng()



Use case Đóng máy

Khi nhân viên bật công tắc tắt máy, các sự kiện và hành động:

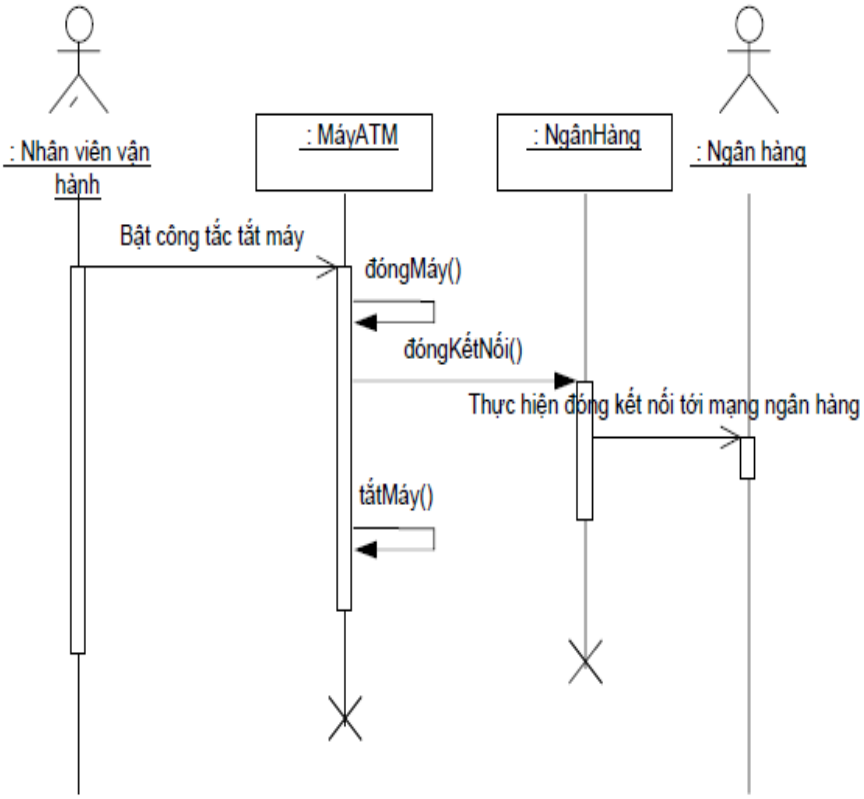
- Trước khi tắt máy -> thực hiện đóng kết nối tới mạng ngân hàng (NgânHàng)

  ➔ tắt máy (MáyATM)

Chúng ta xác định được các method

NgânHàng::+đóngKếtNối()

MáyATM::-tắtMáy()

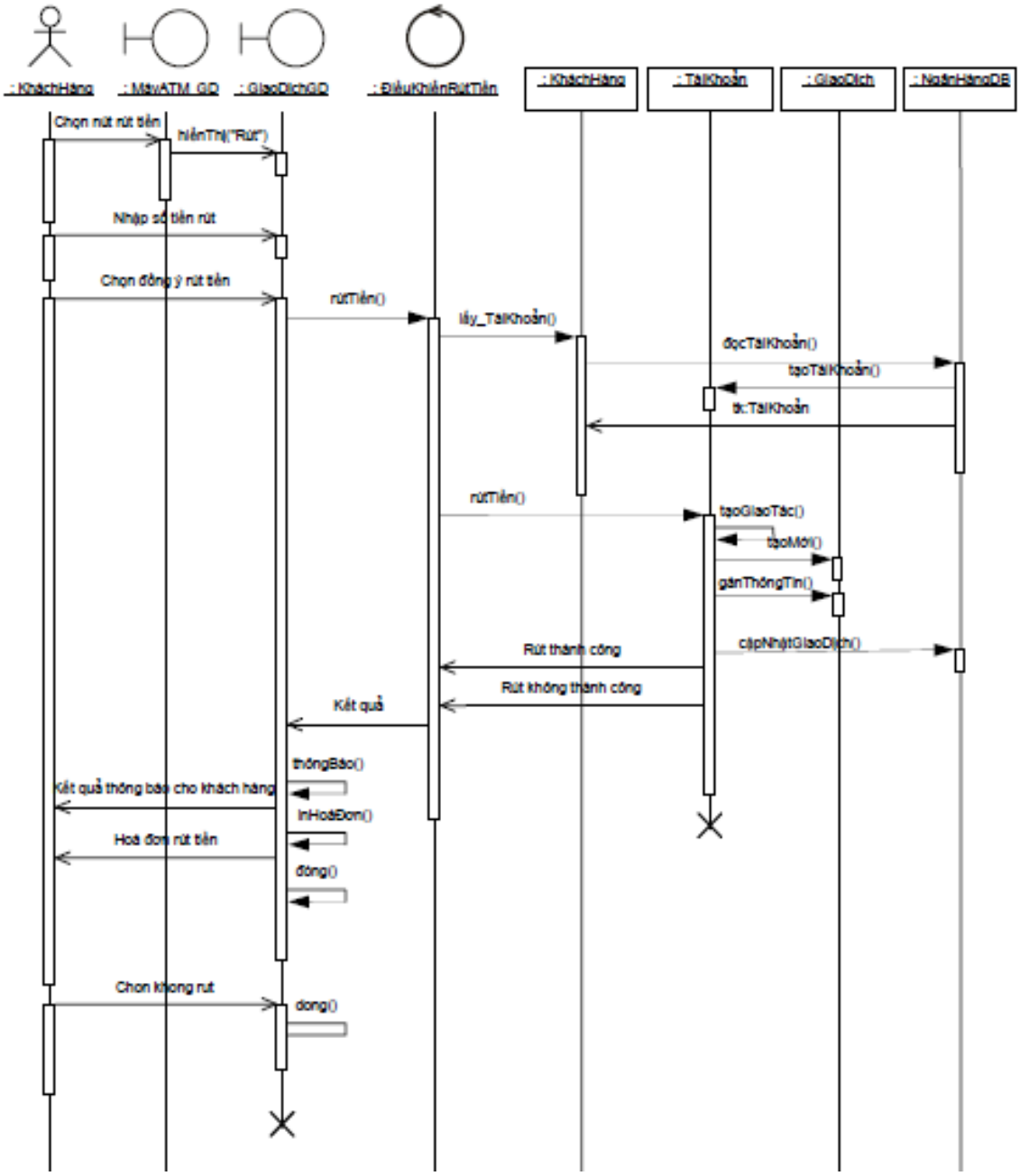


Một giải pháp khác nhằm quản lý việc điều khiển các lớp nghiệp vụ để đáp ứng cho các sự kiện ở tầng giao diện là dùng đối tượng điều khiển. Các thông điệp từ tầng giao diện sẽ được tiếp nhận bởi đối tượng điều khiển và đối tượng này sẽ điều khiển các hoạt động của các đối tượng nghiệp vụ nhằm thực thi cho thông điệp đó. Như vậy đối tượng giao diện cũng có thể hiểu như là một đối tượng bao bọc tầng nghiệp vụ từ các đối tượng tầng giao diện.

Về quan niệm tổng quát, một đối tượng điều khiển có thể đảm nhận nhiều use case hoặc một use case có thể có nhiều đối tượng điều khiển. Tuy nhiên, không phải tất cả use case đều phải có đối tượng điều khiển, bởi vì ý nghĩa của đối tượng điều khiển là tạo ra sự phối hợp, do đó, trong trường hợp use case chỉ có một lớp thì ý nghĩa phối hợp không còn.

Trong UML lớp điều khiển là một stereotype và có ký hiệu như sau:

Sơ đồ tuần tự cho use case Rút tiền có đối tượng điều khiển:

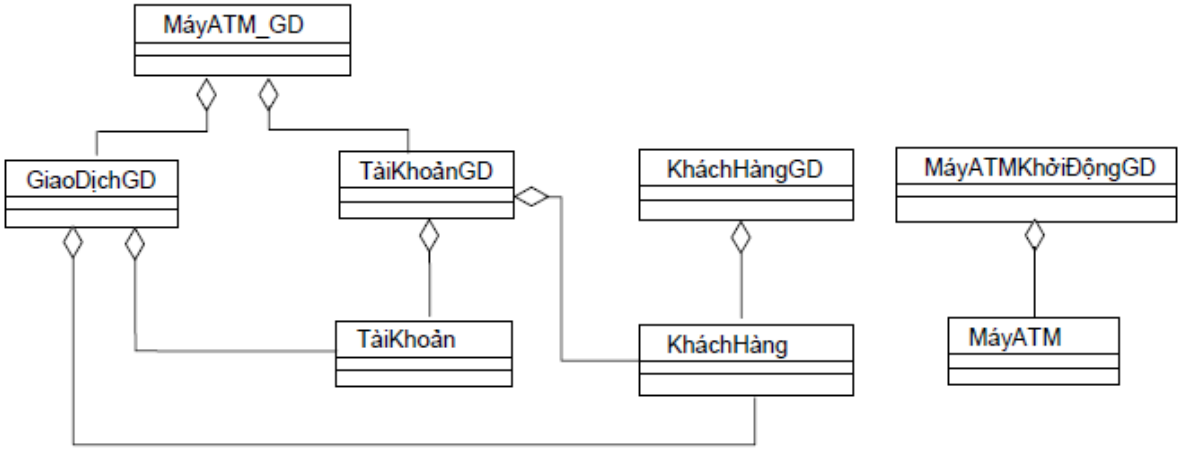


Các lớp tầng giao diện được có thể được chọn một kiểu stereotype phù hợp: boundary, form, interface, page, webpage,… Ví dụ, trong sơ đồ use case Rút tiền trên chúng ta chọn stereotype cho các lớp giao diện là boundary. Quan sát sơ đồ trên chúng ta thấy các đối tượng giao diện sẽ tượng tác (rút tiền) với các đối tượng nghiệp vụ qua một đối tượng điều khiển, và đối tượng này sẽ điều phối các hoạt động của các đối tượng nghiệp vụ (KháchHàng, TàiKhoản, GiaoDịch) để thực hiện việc rút tiền thay vì trong các sơ đồ trước đó, việc điều phối này do đối tượng giao diên đảm nhận.

Xác định thuộc tính các lớp tầng giao diện

Các thuộc tính được xác định cho các lớp tầng giao diện chủ yếu là các thuộc tính mô tả tham chiếu. Một lần nữa, dựa vào các sơ đồ tuần tự chúng ta tinh chế lại mối kết hợp giữa các lớp ở tầng giao diện và các lớp tầng nghiệp vụ:

\



Các thuộc tính tham chiếu của các lớp lần lượt là:

Lớp MáyATM_GD

-giaoDịchGD:GiaoDịchGD

-tàiKhoảnGD:TàiKhoảnGD

Lớp KháchHàngGD

-khách Hàng:KháchHàng

Lớp GiaoDịchGD

-tàiKhoản:TàiKhoản

-khách Hàng:KháchHàng

Lớp TàiKhoảnGD

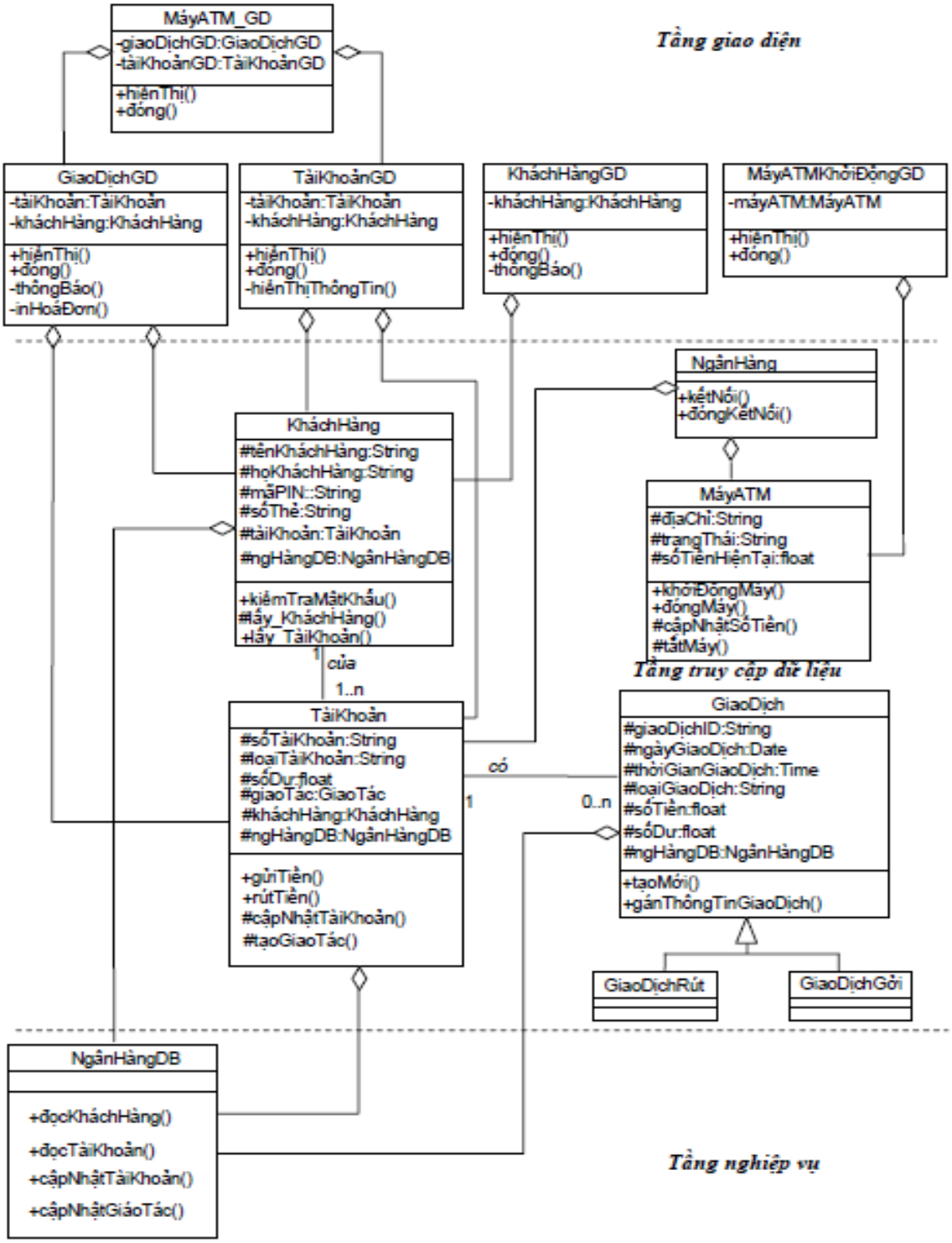
-tàiKhoản:TàiKhoản

-khách Hàng:KháchHàng

Lớp MáyATMKhởiĐộngGD

-máyATM:MáyATM

Sơ đồ lớp đầy đủ ba tầng của hệ thống ATM:



TỔNG KẾT

Như vậy phần này đã cung cấp cho các bạn cái nhìn tổng quan về thiết kế hệ thống (thiết kế lớp và use case):

- Cung cấp cơ bản về kiến trúc ba tầng (tree-layer)
- Xác định các đối tượng ở tầng nghiệp vụ
- Xác định các đối tượng ở tầng truy cập dữ liệu và thiết kế method
- Xác định các đối tượng ở tầng giao diện và thiết kế method