

Thiết kế giao diện, thiết kế chi tiết và xây dựng kiến trúc

Week 8 -12

Hai V. Pham

Tham khảo Slides PGS Nguyễn Văn Ba - Tác giả cuốn sách
phân tích thiết Hệ thống hướng đối tượng

1. Giới thiệu

- ▶ **Phân tích** là để trả lời câu hỏi “what” và chỉ tập trung đáp ứng các **yêu cầu chức năng** đối với hệ thống.
 - ▶ **Thiết kế** là để trả lời câu hỏi “how”, tập trung đáp ứng các **yêu cầu phi chức năng** (tiện ích, hiệu năng, tương thích với phần cứng...) đối với hệ thống. Thiết kế sẽ chỉnh sửa lại mô hình phân tích và sẽ đưa thêm vào nhiều lớp mới.

2

2. Làm nguyên mẫu giao diện người dùng

- 2.1. Mục đích
 - 2.2. Mô tả các giao diện của hệ thống
 - 2.3. Làm nguyên mẫu

3

2.1. Mục đích

- ▶ Dựa vào các bộ tạo lập GUI (Graphical user interface builder) ta thành lập sớm và nhanh một nguyên mẫu (prototype) giao diện người dùng, có tính thăm dò, nhằm vào các mục đích sau:
 - Tạo ra một môi trường làm việc cụ thể, dễ tiếp xúc, dễ làm thử, làm cho người dùng trở nên yên tâm hơn, và năng động hơn trong việc đóng góp cho việc phát triển hệ thống.
 - Qua quá trình dùng thử, ta thu thập được nhiều ý kiến phản hồi có ích từ phía người dùng.
 - Sớm phát hiện được các yêu cầu hay chức năng bị bỏ sót, sớm nhìn thấy các điểm yếu, chỗ khó khăn nhất của hệ thống.

4

2.2. Mô tả các giao diện của hệ thống

- ▶ Như đã biết, cứ mỗi cặp **đối tác – ca sử dụng** liên quan, có ít nhất một lớp biên để chuyển đổi các thông tin vào-ra. Thể hiện của lớp biên chính là giao diện mà bây giờ ta cần phải mô tả.
- ▶ Muốn mô tả chúng, ta dõi theo từng bước trong kịch bản của mỗi ca sử dụng, xét nội dung của tương tác giữa đối tác và hệ thống, các thông tin vào và ra, các hành động được yêu cầu để xác định các phần tử của giao diện.

5

2.2. Mô tả các giao diện của hệ thống

- Bản mô tả một giao diện thường mở đầu với các điểm sau:
- ▶ Tên của giao diện;
 - ▶ Diễn tả ngắn nội dung giao diện với độ 2 – 10 dòng văn tự;
 - ▶ Mức độ phức tạp của giao diện (phức tạp/chuẩn/dơn giản);
 - ▶ Ghi chú thêm, nếu có.
 - ▶ Tiếp đó là các mô tả chi tiết, với những đặc điểm kỹ thuật khác biệt, tùy thuộc vào 4 loại giao diện sau:
 - Các giao diện đối thoại;
 - Các thông tin xuất (thư, báo cáo v.v...);
 - Các giao diện dữ liệu từ/dến các hệ thống ngoài;
 - Các giao diện chức năng đến các hệ thống ngoài.

6

2.3. Làm nguyên mẫu

- ▶ Làm nguyên mẫu nên bắt đầu càng sớm càng tốt. Chẳng hạn có thể bắt đầu làm nguyên mẫu ngay sau khi đưa ra các ca sử dụng.
- ▶ Ngày nay có nhiều bộ tạo lập giao diện người dùng (GUI builders) cho phép làm các nguyên mẫu giao diện mà không tốn mây công sức.
- ▶ Bước đầu thì các trường là rỗng hoặc cho giá trị giả. Các nút và các phần tử đối thoại khác có thể chưa có hiệu ứng rõ rệt và cần giải thích "miệng".
- ▶ Qua nhiều vòng lặp, giao diện trở nên sinh động hơn và dần đi tới phương án cuối. Như vậy, người dùng có thể làm việc thử với các nguyên mẫu.

7

2.3. Làm nguyên mẫu

- ▶ Nguyên mẫu chỉ có ý nghĩa thăm dò => nên làm nhanh và không cầu toàn. Chưa nên chú ý nhiều về trình bày, về mỹ thuật mà cần chú ý nội dung (các trường, các frame) và luồng dẫn dắt từ phần tử giao diện này sang phần tử giao diện khác.
- ▶ Quá trình phát triển nguyên mẫu làm đồng thời với quá trình phân tích và thiết kế, hỗ trợ cho phân tích và thiết kế.
- ▶ Chú ý, nguyên mẫu và giao diện nói chung chỉ là mặt ngoài của hệ thống, chưa phản ánh hết tầm sâu của hệ thống. Vậy, cần nói rõ với người dùng là không nên ảo tưởng ở nguyên mẫu. Thực ra làm nguyên mẫu chỉ là một sự hỗ trợ tốt, chứ không thể là sự thay thế cho các bước phân tích và thiết kế hệ thống một cách nghiêm túc được.

8

3. Thiết kế hệ thống

- 3.1. Mục đích
- 3.2. Phân rã HT thành các HT con
- 3.3. Mô tả các thành phần vật lý của HT
- 3.4. Bố trí các thành phần khả thi vào các nút phần cứng

9

3.1. Mục đích

- ▶ Thiết kế hệ thống chính là **thiết kế kiến trúc tổng thể** của nó.
- ▶ Các thành phần tạo nên kiến trúc là gì phụ thuộc vào từng cách nhìn đối với hệ thống. Trong phần này, ta tiếp cận kiến trúc theo 3 góc nhìn (theo hệ con, theo thành phần phần mềm, theo các đơn vị phần cứng):
 - Phân rã hệ thống thành các hệ con (các gói).
 - Mô tả các thành phần vật lý của hệ thống.
 - Bố trí các thành phần khả thi vào các nút phần cứng.

10

3.2. Phân rã HT thành các HT con

- ▶ Khái niệm về hệ con
 - Các lớp là những thực thể cấu trúc rất nhỏ so với một HT thực. Bởi vậy, khi sáp các lớp trong hệ thống đã lên tới hàng chục, ta nên gom các lớp liên quan với nhau thành từng nhóm gọi là các hệ con.
 - **Hệ con** (subsystem) là một sự gom nhóm **lôgic** các lớp có **sự gắn kết** bên trong mạnh và **sự tương liên** bên ngoài yếu.
 - Thuật ngữ hệ con được nhiều tác giả dùng, và G. Booch lại gọi là phạm trù (category), thực ra không phải là thuật ngữ chuẩn của UML. Trái lại, UML dùng thuật ngữ **gói** (package), cho nên ta cũng sẽ biểu diễn hệ con dưới dạng gói, mang theo khuôn dập <<subsystem>>.

11

3.2. Phân rã HT thành các HT con

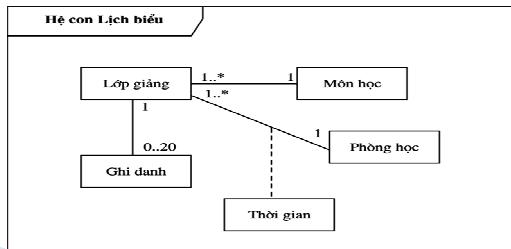
- Nội dung của một hệ con (gồm các lớp và các mối liên quan giữa chúng) được UML 2.0 diễn tả trong một khung (frame), với một tựa đề viết trong một hình chữ nhật cắt góc theo khuôn dạng:

[<loại>] Tên [<tham số>]

12

3.2. Phân rã HT thành các HT con

- Số các lớp trong một hệ con không nên ít quá hay nhiều quá (thường thì có khoảng mươi lớp là vừa).



13

3.2. Phân rã HT thành các HT con

Sự **gắn kết** cao của các lớp trong cùng một hệ con thể hiện:

- **Về mục đích:** Chúng phải cung cấp các dịch vụ có cùng bản chất cho người dùng. Như vậy chúng phải thuộc vào cùng một lĩnh vực và đề cập một số thuật ngữ chung (chẳng hạn hệ con giao diện đề cập các thuật ngữ như: cửa sổ, thực đơn, nút nhấn,...).
 - **Về xu thế phát triển:** Người ta tách các lớp bền vững với các lớp có nhiều khả năng thay đổi. Đặc biệt, thường tách các lớp nghiệp vụ với các lớp ứng dụng và xếp chúng vào các hệ con khác nhau.
 - **Về ứng dụng các công nghệ:** Để tận dụng các dịch vụ công nghệ có sẵn, như các thư viện chương trình (lớp/thành phần), các GUI, các hệ quản trị cơ sở dữ liệu v.v..., ta thường tách các hệ con giao tiếp, hệ con quản trị dữ liệu ra khỏi phần lõi (ứng dụng và nghiệp vụ) của HT.

14

3.2. Phân rã HT thành các HT con

- Sự **tương liên** giữa các hệ con thể hiện ở mối liên quan phụ thuộc giữa chúng. Mà sự phụ thuộc giữa hai hệ con phản ánh các mối liên quan tĩnh (thừa kế, liên kết...) và các mối liên quan động (trao đổi thông tin) giữa các lớp thuộc hai hệ con đó. Sự phụ thuộc giữa các hệ con phải càng đơn giản, lồng lêo thì càng tốt. Để đảm bảo tính tương liên yêu này, khi thành lập hệ con, áp dụng các quy tắc sau:
 - Các lớp thuộc vào cùng một phâ hệ thừa kế nên được xếp vào cùng một hệ con.
 - Các lớp có mối liên quan kết nhập và hợp thành với nhau thường được xếp vào cùng một hệ con.
 - Các lớp cộng tác với nhau nhiều, trao đổi thông tin nhiều, thể hiện qua các biến đỗ tương tác, thì nên đặt chung vào một hệ con.
 - Nên tránh sự phụ thuộc vòng quanh giữa các lớp.

15

3.2. Phân rã HT thành các HT con

► Kiến trúc phân tầng

- Một hệ con thường được định nghĩa bởi các dịch vụ mà nó cung cấp. Mối liên quan giữa một hệ con với phần còn lại của hệ thống có thể là ngang hàng hay là khách hàng/dịch vụ.
- Trong mối **liên quan ngang hàng** (peer-to-peer) thì mỗi bên đều có thể truy cập các dịch vụ của bên kia. Bấy giờ sự giao tiếp không nhất thiết là ở dạng câu hỏi và trả lời liên sau đó, mà có thể là một sự giao tiếp loạn quanh, rất dễ dẫn tới những sai lỗi đáng tiếc về thiết kế.
- Còn mối **liên quan khách hàng/dịch vụ** (client/server) thì đơn giản hơn: bên khách hàng gọi bên dịch vụ và bên dịch vụ thực hiện một dịch vụ theo yêu cầu và trả kết quả cho bên khách hàng. Bên khách hàng thì phải biết giao diện của bên dịch vụ, song bên dịch vụ thì không cần biết giao diện của bên khách hàng.

16

3.2. Phân rã HT thành các HT con

► Từ hai hình thức giao tiếp đó mà ta có hai cách để chia hệ thống thành các hệ con:

- Tổ chức hệ thống thành các **tầng** theo chiều ngang, với mối quan hệ khách hàng/dịch vụ luôn luôn hướng từ tầng trên xuống (các) tầng dưới. Một thí dụ của hệ thống phân thành tầng là hệ thống tạo cửa sổ trong giao diện người dùng của máy tính.
- Tổ chức hệ thống thành **lát** theo chiều đứng, với quan hệ ngang hàng giữa các lát, tuy nhiên các lát là khá độc lập hoặc tương liên yếu với nhau. Một thí dụ của hệ thống phân lát là một hệ điều hành, thường gồm các hệ con như là các hệ quản lý tệp, điều khiển thiết bị, quản lý sự kiện và ngắt...

17

3.2. Phân rã HT thành các HT con

- Rõ ràng là tổ chức phân tầng là đáng được ưu tiên hơn, vì nó mang lại nhiều ưu thế trong thiết kế, trong cài đặt cũng như trong sử dụng lại.
- Song đối với các hệ thống lớn thì ta thường phải phối hợp cả hai cách tổ chức phân tầng và phân lát, chẳng hạn phân hệ thống thành tầng, nhưng trong mỗi tầng thì lại phân thành lát.

18

3.2. Phân rã HT thành các HT con

- Khi thực hiện phân tầng, thì số tầng là tùy thuộc sự phức tạp của hệ thống:
 - Trong một hệ đơn giản, thì số tầng có thể chỉ là hai (2-tiers). Bây giờ tầng khách hàng thì quản lý giao diện người dùng và các quá trình khai thác, còn tầng dịch vụ thì xử lý việc cất giữ các dữ liệu.
 - Trong một hệ phức tạp hơn, thì người ta tách tầng trên thành tầng giao diện – ứng dụng, và ở dưới nó là tầng nghiệp vụ (hay lĩnh vực), bền vững hơn và có nhiều khả năng sử dụng lại hơn. Vậy đó là một kiến trúc khách hàng/dịch vụ ba tầng (3-tiers).

19

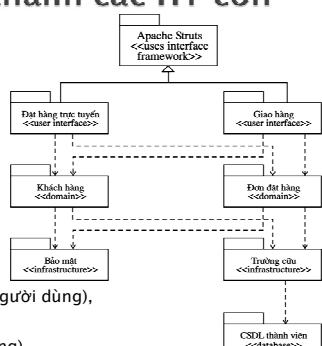
3.2. Phân rã HT thành các HT con

- Cuối cùng thì trong các hệ lớn, số tầng còn có thể nhiều hơn (*n-tiers*), mà điển hình là kiến trúc năm tầng, với các tầng kế từ trên xuống là:
 - **Tầng trình bày:** Chuyển các dữ liệu cho người dùng và biến đổi các hành động của người dùng thành các sự kiện vào của hệ thống.
 - **Tầng ứng dụng:** bao gồm các đối tượng điều khiển và dẫn dắt các quy luật của ứng dụng.
 - **Tầng nghiệp vụ:** bao gồm các đối tượng nghiệp vụ (hay lĩnh vực), cung cấp cái đặt các quy tắc quản lý chúng.
 - **Tầng truy cập dữ liệu:** phục hồi các đối tượng nghiệp vụ từ các phương tiện lưu trữ.
 - **Tầng lưu trữ dữ liệu:** bảo đảm sự lưu giữ lâu dài các dữ liệu.

20

3.2. Phân rã HT thành các HT con

- Một thí dụ về kiến trúc khách hàng/dịch vụ năm tầng cho nhu bên, trong đó mỗi gói (hệ con) đều có mang khuôn dập thích hợp, như là:
 - <<user interface>> (khuôn khổ giao diện người dùng),
 - <<user interface>> (giao diện người dùng),
 - <<domain>> (lĩnh vực),
 - <<infrastructure>> (cơ sở hạ tầng),
 - <<database>> (cơ sở dữ liệu).



21

3.3. Mô tả các thành phần vật lý của HT

► Thành phần và biểu đồ thành phần

- Nếu như biểu đồ gói (hệ con) mà ta nói ở phần trên phản ánh cho góc nhìn về cấu trúc **lôgic** của hệ thống (ở mức cao so với biểu đồ lớp), thì biểu đồ thành phần, với các đơn nguyên trong đó là các thành phần, lại cho ta một cách nhìn về cấu trúc **vật lý** của hệ thống.
- Chữ "vật lý" ở đây được hiểu theo nghĩa là sự mô tả hướng tới các sản phẩm phần mềm, là kết quả của sự cài đặt và thực sự tồn tại, chứ không phải là các sản phẩm lôgic, kết quả của quá trình phân tích. Tuy nhiên ở đây ta cũng chưa đề cập tới phần cứng, mặc dù tính vật lý của nó cũng là đương nhiên.

22

3.3. Mô tả các thành phần vật lý của HT

- UML định nghĩa **thành phần** (component) là một bộ phận vật lý và thay thế được của hệ thống, thích ứng và cung cấp sự thực hiện cho một tập các giao diện.
- Nói đơn giản hơn, thì thành phần là một cài đặt của một tập hợp các phần tử lôgic, như các lớp hay các hợp tác.

23

3.3. Mô tả các thành phần vật lý của HT

► Có ba loại thành phần:

- Các **thành phần triển khai** (deployment components): Đó là các thành phần cần và đủ để tạo nên một hệ thống khả thi, như là các thư viện động (DLL) và các mã khả thi (executable). Định nghĩa thành phần của UML là đủ rộng để bao hàm các mô hình đối tượng kinh điển, như là COM+, CORBA, và Enterprise Java Beans, cũng như các mô hình đối tượng khác biệt như là các trang Web động, các bảng cơ sở dữ liệu, và các mã khả thi sử dụng những cơ chế truyền thông riêng.

24

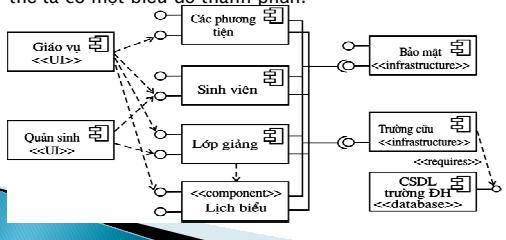
3.3. Mô tả các thành phần vật lý của HT

- Các **thành phần sản phẩm làm việc** (work product components): Đó là các thành phần có từ quá trình phát triển hệ thống, bao gồm các tệp mã nguồn, các tệp dữ liệu, từ đó mà ta đã tạo lập ra các thành phần triển khai. Các thành phần này không trực tiếp tham gia vào hệ thống thực thi, nhưng không có chúng thì không tạo được hệ thống thực thi.
 - Các **thành phần thực hiện** (execution components): Đó là các thành phần được tạo nên như là một kết quả của một hệ thực hiện, chẳng hạn một đối tượng COM+, được cá thể hóa từ một DLL.

25

3.3. Mô tả các thành phần vật lý của HT

- ▶ Để tổ chức các thành phần lại với nhau, ta có hai cách:
 - Gom các thành phần vào các gói, nghĩa là đưa chúng vào các hệ con;
 - Thiết lập các mối liên quan phụ thuộc giữa chúng, và như thế ta có một biểu đồ thành phần.



26

3.3. Mô tả các thành phần vật lý của HT

- ▶ Các mục đích mô hình hóa của biểu đồ thành phần
 - Như đã trình bày, có nhiều loại thành phần (ít nhất có ba loại chính là thành phần triển khai, thành phần sản phẩm làm việc và thành phần thực hiện), vậy biểu đồ thành phần lập ra phải có mục đích mô tả loại thành phần nào.

27

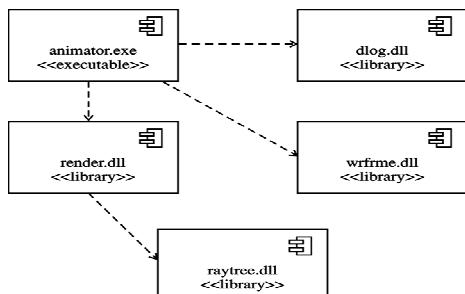
3.3. Mô tả các thành phần vật lý của HT

- **Mô hình hoá các thành phần khả thi và thư viện:** Có thể nói mục đích chính của việc sử dụng biểu đồ thành phần là để mô hình hoá các thành phần triển khai, tạo nên cái cài đặt của hệ thống.
 - Nếu ta muốn cài đặt một hệ thống chỉ gồm đúng một tệp khả thi (.EXE), thì ta chẳng cần dùng tới thành phần.
 - Ngược lại nếu hệ thống gồm nhiều tệp khả thi và liên kết với các thư viện đối tượng thì ta cần dùng biểu đồ thành phần để giúp ta hiểu thị, đặc tả, thành lập và tư liệu hoá các quyết định của chúng ta đối với hệ thống vật lý.

28

3.3. Mô tả các thành phần vật lý của HT

- *Thí dụ:* Một biểu đồ các thành phần triển khai



29

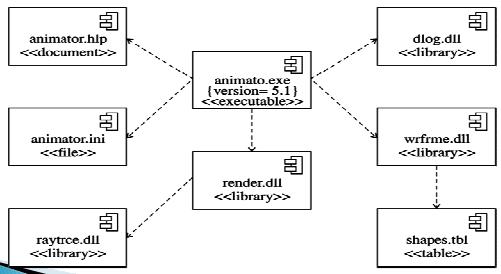
3.3. Mô tả các thành phần vật lý của HT

- Mô hình hóa các bảng, các tệp và các tài liệu: Bên cạnh các tệp khả thi và thư viện tạo nên phần chạy được của hệ thống, thì còn nhiều thành phần bổ trợ khác, gọi là các thành phần phụ trợ, cần cho việc cài đặt hệ thống. Chẳng hạn để cài đặt, ta vẫn cần các tệp dữ liệu, các tư liệu trợ giúp, các scripts, các tệp log, các tệp khởi tạo, các tệp xếp chỗ hay gỡ bỏ. Mô hình hóa các thành phần này cũng là một phần quan trọng để diễn tả hình trang của hệ thống.

30

3.3. Mô tả các thành phần vật lý của HT

- Thí dụ: Từ biểu đồ trên, thêm vào các bảng, các tệp và các tư liệu



31

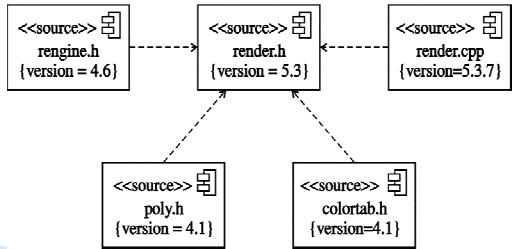
3.3. Mô tả các thành phần vật lý của HT

- Mô hình hóa mã nguồn: Mô hình hóa mã nguồn cũng là một mục đích của việc tạo lập biểu đồ thành phần. Các tệp nguồn dùng để chứa các chi tiết về các lớp, các giao diện, các hợp tác và các phần tử lôgic khác; chúng tạo nên một bước trung gian để tạo lập các thành phần vật lý, nhị phân (nhờ một công cụ nào đó). Các công cụ (chẳng hạn chương trình biên dịch) thường đòi hỏi các tệp nguồn phải được tổ chức theo một quy cách nhất định nào đó (thông thường là một hay hai tệp cho một lớp). Các mối liên quan phụ thuộc giữa các tệp này phản ánh một trật tự biên dịch.

32

3.3. Mô tả các thành phần vật lý của HT

- Thí dụ: Mô hình hóa các tệp nguồn dùng để xây dựng thư viện render.dll từ các thí dụ trước



33

3.4. Bố trí các thành phần khai thí vào các nút phần cứng

- Để bố trí các thành phần phần mềm lên các phần cứng, ta dùng các biểu đồ triển khai.
 - Biểu đồ triển khai** (deployment diagram) là một biểu đồ diễn tả sự bố trí các executable artifacts trên underlying platform. Nó gồm các nút và các kết nối giữa các nút đó.
 - Một **nút** (node) là một phần tử vật lý tồn tại vào lúc chạy và biểu diễn cho một tài nguyên tính toán (computational resource), nói chung thì phải có ít nhất một chỗ nhớ, và thông thường thì có thêm khả năng xử lý. Một nút được biểu diễn bởi một hình hộp, có mang tên.
 - Nếu tên này không gạch dưới, thì nút thể hiện một lớp các tài nguyên,
 - Nếu tên được gạch dưới, thì nút thể hiện một cá thể tài nguyên.

34

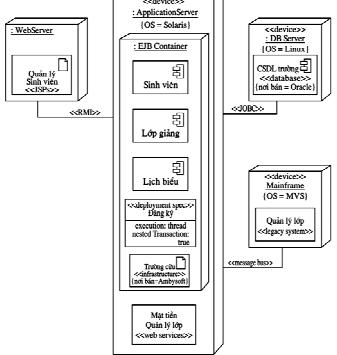
3.4. Bố trí các thành phần khả thi vào các nút phần cứng

- ▶ Như vậy có hai mức diễn tả của biểu đồ triển khai:
mức lớp (tương tự một biểu đồ lớp) và mức cá thể
(tương tự một biểu đồ đối tượng).
 - ▶ Theo định nghĩa như trên, nút có thể là:
 - Một thiết bị (nút cứng), thường mang khuôn dập <<device>> (hoặc cụ thể hơn là processor, console, kiosk, printer,...);
 - Một môi trường thực hiện (nút mềm), thường mang khuôn dập <<execution env>>, như là EJB Container hay là J2EEServer.
 - ▶ Hình sau đây cho một biểu đồ triển khai (ở mức cá thể) biểu diễn cho hình trạng vật lý của hệ thống thông tin về trường đại học.

35

3.4. Bố trí các thành phần khả thi vào các nút phần cứng

- ▶ Nút WebServer chưa có khuôn dập, đó là vì người phát triển hệ thống chưa có quyết định: nút đó có thể đơn giản là một loại phần mềm
nào đó (chẳng hạn một Browser), hoặc là một thiết bị vật lý.
 - ▶ Các nút có thể chứa các nút khác hoặc các sản vật phần mềm. Chẳng hạn nút ApplicationServer chứa nút EJBContainer (một nút mềm), và nút này lại chứa ba thành phần phần mềm, một đặc tâ bối trí, và một sản vật phần mềm.



36

3.4. Bố trí các thành phần khai thí vào các nút phần cứng

- ▶ Các **kết nối** (connections) là các mối liên quan giao tiếp giữa các cặp nút, thể hiện về mặt vật lý bằng một đường truyền (như là một kết nối Ethernet, một đường truyền tuần tự hay một bus dùng chung).
 - ▶ Mỗi kết nối hỗ trợ cho một hay nhiều giao thức truyền thông, mà ta cần chỉ rõ bằng các khuôn dập thích hợp. Bảng sau cho một số khuôn dập thường dùng cho các kết nối, cùng với ý nghĩa của chúng.

37

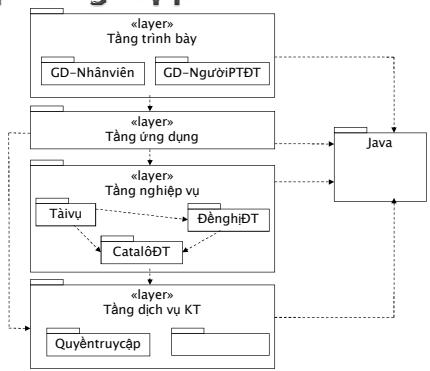
3.4. Bố trí các thành phần khai thi vào các nút phần cứng

Khuôn dập	Ý nghĩa
Asynchronous	Một kết nối không đồng bộ,
HTTP	HyperText Transport Protocol, một giao thức Internet
JDBC	Java Database Connectivity, một Java API để truy cập CSDL
ODBC	Open Database Connectivity, một MicroSoft API để truy cập CSDL
RMI	Remote Method Invocation, một giao thức liên lạc của Java
RPC	Remote Procedure Call
Synchronous	Một kết nối đồng bộ, trong đó bên gửi chờ trả lời từ bên nhận
Web services	Liên lạc qua các giao thức web services như là SOAP và UDDI

38

4. Bài tập tổng hợp

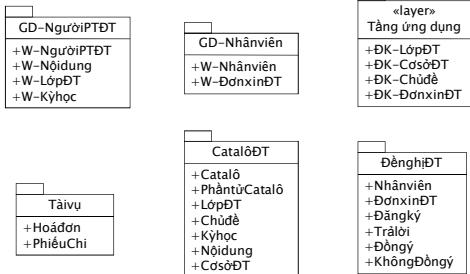
Đề xuất một kiến trúc phân tầng cho HT YCĐT



39

4. Bài tập tổng hợp

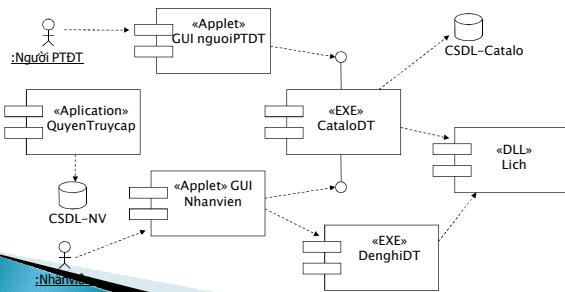
Trong kiến trúc phân tầng trên, thì gói java chứa các lớp cơ sở dùng chung cho mọi tầng, còn các gói con thì chứa các lớp như sau:



40

4. Bài tập tổng hợp

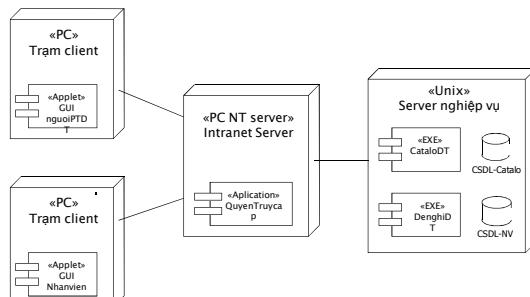
Câu hỏi 27: Hãy đề xuất một BĐ thành phần cho hai vòng lặp đầu của HT. Chỉ cần đưa ra các thành phần không lặp lẩm, đủ để biểu diễn cho một hợp tác giữa nhiều lớp. Đừng quên ngôn ngữ đích là Java.



41

4. Bài tập tổng hợp

Câu hỏi 28: Hãy đề xuất một BD bố trí cho hai vòng lặp đầu của HT.



42

Thiết kế chi tiết

43

Nội dung

1. Mục đích của thiết kế chi tiết
2. Quy trình thiết kế chi tiết
3. Thiết kế chi tiết các tầng
4. Bài tập tổng hợp

44

1. Mục đích thiết kế chi tiết

- ▶ Trong phân tích, ta tập trung nghiên cứu cấu trúc lôgic của các thông tin, cần thiết cho việc xây dựng một giải pháp nghiệp vụ. Nói cách khác thì phân tích luôn luôn được triển khai theo **nhân quan ứng dụng**, và chưa tính tới các điều kiện về công nghệ.

45

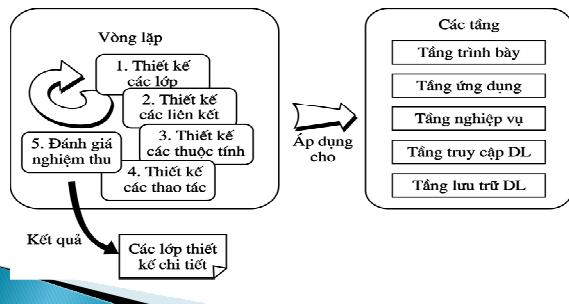
1. Mục đích thiết kế chi tiết

- Việc thiết kế chi tiết tiếp nhận ở đầu vào các mô hình đã được thiết lập từ các bước phân tích và thiết kế hệ thống trước đây, bao gồm các mô hình về cấu trúc (các biểu đồ lớp, thành phần, hệ con, kiến trúc phân tầng và nút) và các mô hình động thái (các biểu đồ ca sử dụng, tương tác, máy trạng thái và giao diện người dùng).
- Kết quả ở đầu ra của thiết kế chi tiết phải là một **mô hình sẵn sàng cho lập trình**, nghĩa là nó phải bao gồm mọi quyết định về cài đặt, thích ứng với ngôn ngữ lập trình và môi trường cài đặt đã được chọn lựa (hay sẵn có). Mô hình này lại có thể vẫn được diễn tả thông qua các biểu đồ của UML (kết hợp với các tư liệu văn tự).

46

2. Quy trình thiết kế chi tiết

- Bước thiết kế chi tiết được tiến hành theo một **quy trình lặp** được diễn tả trên hình sau:



47

2.1. Thiết kế các lớp

Thiết kế lớp đưa ra nhiều sự thay đổi đối với các lớp phân tích. Đó có thể là:

a) **Phân bổ lại hay giải phóng bớt trách nhiệm cho các lớp phân tích**

- Khi phân tích, thì các lớp được đưa ra theo nhu cầu, mà chưa tính đến hiệu năng hay sự thích ứng với điều kiện kỹ thuật. Vì vậy trong thiết kế cần có sự phân bổ lại trách nhiệm

48

b) Thêm các lớp mới để cài đặt các cấu trúc dữ liệu

- ▶ Khi phân tích, thì nhiều cấu trúc dữ liệu được xem là mặc định. Trong thiết kế, thì các cấu trúc dữ liệu đó phải được xem xét việc cài đặt.
- ▶ Thường thì nhiều cấu trúc dữ liệu là có sẵn trong thư viện của ngôn ngữ lập trình, như là mảng, danh sách, hàng đợi, ngăn xếp, tập hợp, túi (bag), từ điển (dictionary), cây...

49

c) Thêm các lớp mới để cài đặt các khái niệm phân tích

- ▶ Có nhiều khái niệm dùng trong phân tích, nhưng không có trong các ngôn ngữ lập trình, cần phải tìm cách để cài đặt chúng bằng các lớp thiết kế.

50

d) Thêm các lớp mới vì mục đích tối ưu hoá

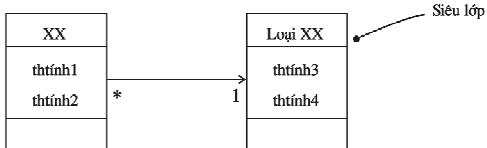
Nhiều biện pháp tối ưu hoá được thực hiện bằng cách đưa thêm các lớp mới

- ▶ Các **siêu lớp**: Ta gọi siêu lớp (metaclasse) là một lớp mà đối tượng của nó lại là lớp.

51

d) Thêm các lớp mới vì mục đích tối ưu hoá (tt)

- Kết nối lớp XX với lớp mới LoạiXX bằng một liên kết nhiều-1, thường là một chiều, hướng tới LoạiXX. Lớp LoạiXX là một siêu lớp, vì mỗi đối tượng của nó (một loại) lại là một tập hợp các đối tượng.
 - Ưu điểm của giải pháp trên chính là đã tránh được sự lặp lại nhiều lần giá trị của các thuộc tính của siêu lớp trên mỗi đối tượng của lớp XX.



52

- **Các lớp dẫn xuất:** Nếu ta muốn loại bỏ sự tính toán lại, để tăng hiệu năng, thì ta có thể lập ra các đối tượng/lớp mới để lưu giữ các thuộc tính (dữ liệu) dẫn xuất. Tuy nhiên cần nhớ rằng phải cập nhật các thuộc tính dẫn xuất mỗi khi các thuộc tính cơ sở thay đổi.

53

- Triggers: Khi một thuộc tính cơ sở có nhiều thuộc tính dẫn xuất từ nó được cập nhật, thì một trigger (một thủ tục tự khởi động khi một sự kiện định trước xảy ra) sẽ được kích hoạt và báo cho các đối tượng chứa các thuộc tính dẫn xuất biết là thuộc tính cơ sở đã thay đổi giá trị. Sau đó thì chính đối tượng chứa thuộc tính dẫn xuất có trách nhiệm cập nhật thuộc tính dẫn xuất của mình.

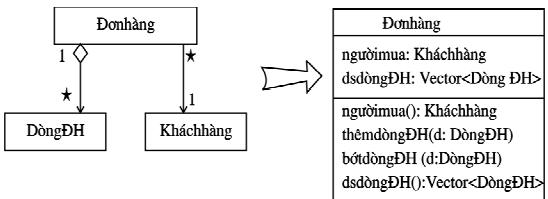
54

- ▶ **Các lớp cha:** Khi có cùng một thao tác hay thuộc tính được phát hiện trên nhiều lớp khác nhau, thì có thể đưa nó vào một lớp cha để các lớp nới trên sử dụng nó theo cách thừa kế.

55

2.2. Thiết kế các liên kết

- ▶ Khái niệm liên kết không có sẵn trong các ngôn ngữ lập trình. Ta phải biến đổi nó thành thuộc tính hay bằng các thuộc tính, tùy theo cơ sở của nó. Cùng với các thuộc tính đưa thêm đó, lại phải có thêm các thao tác để sử dụng và quản lý các thuộc tính này.



56

- ▶ Nếu liên kết là một chiều (như trên hình vừa rồi), thì thuộc tính chỉ đưa vào một phía (phía gốc của sự lưu hành). Nhưng nếu liên kết là hai chiều, thì thuộc tính cài đặt nó phải đưa vào cả hai phía (nhưng cũng có thể chỉ đưa vào một phía và thực hiện tìm kiếm ở phía kia).
- ▶ Khi đưa thuộc tính thể hiện liên kết vào cả hai phía, thì lại phải bảo đảm tính đồng bộ. Chẳng hạn, với liên kết hai chiều giữa lớp Sinhvien và lớp TrườngDH, thì ta phải bảo đảm rằng khi một sinh viên biết trường mà mình đang học, thì trường này phải có sinh viên đó trong danh sách sinh viên của trường. Tính đồng bộ đó phải được bảo đảm bởi các thao tác quản lý thuộc tính thể hiện liên kết.

57

- ▶ Các liên kết lại thường kèm theo nhiều ràng buộc.
Chẳng hạn cơ sở (quy định số lượng tối đa và tối thiểu), hợp thành (ràng buộc sự tồn tại đồng thời của toàn thể và bộ phận), hạn định (cho phép thu hẹp cơ sở) hoặc là các tính chất như là {OR}, {AND}, {subset}... Các ràng buộc này phải được bảo đảm bởi các thao tác sử dụng và quản lý các thuộc tính thể hiện liên kết.
 - ▶ Chẳng hạn, trong khi ta cài đặt liên kết và kết nháp không có gì là khác biệt, thì khi cài đặt hợp thành, lại phải chú ý là mỗi lần huỷ bỏ cái toàn thể, thì phải huỷ bỏ cái bộ phận.
 - ▶ Hạn định thì được cài đặt bằng cách thay một vectơ bằng một bảng băm (Hashtable) mà khoá là có kiểu của hạn định đó.

58

- ▶ Khi một liên kết có các thuộc tính, nhưng không có thao tác, thì liên kết đó có thể cài đặt như sau:
 - Nếu liên kết là một-một, thì các thuộc tính của liên kết có thể đưa vào một trong hai lớp tham gia liên kết;
 - Nếu liên kết là nhiều-một, thì các thuộc tính của liên kết có thể đưa vào lớp ở đầu nhiều.
 - Nếu liên kết là nhiều-nhiều, thì tốt nhất là lập riêng một lớp liên kết chứa các thuộc tính của liên kết.
 - ▶ Với trường hợp liên kết hai ngôi có cả thuộc tính và thao tác, cũng như với trường hợp liên kết nhiều ngôi, thì ta phải lập thành lớp liên kết. Một lớp liên kết là một tập hợp các bộ đôi (hay bộ n) các đối tượng thuộc các lớp tham gia liên kết.

59

2.3. Thiết kế các thuộc tính

- Việc thiết kế các thuộc tính chủ yếu là định nghĩa các kiểu cho các thuộc tính đã được đưa ra trong phân tích.
 - Mặc dù phần lớn các thuộc tính là thuộc các kiểu cơ sở có sẵn trong ngôn ngữ lập trình, song vẫn có các thuộc tính là tương ứng với các cấu trúc dữ liệu cần được làm rõ.
 - Các cấu trúc dữ liệu này được diễn tả như là một lớp, song để phân biệt chúng với các lớp bình thường, ta đưa thêm khuôn dập <<struct>>. Với kiểu liệt kê, thì ta dùng khuôn dập <<enum>>.
 - Trong các cấu trúc dữ liệu này, thì các thuộc tính đều là công cộng, vì vậy ít khi cần có thêm các thao tác liên quan.

60

► Thiết kế các thuộc tính còn là xác định tầm nhìn và cách truy cập vào chúng.

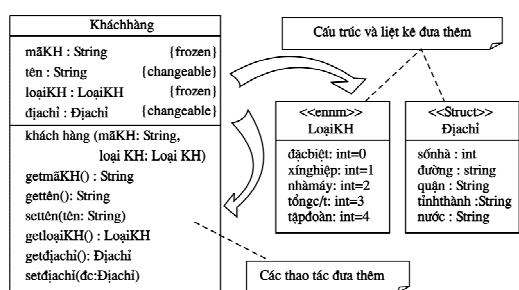
- Mặc định, thì các thuộc tính là riêng tư.
- Các tính chất UML gắn cho thuộc tính như là {frozen}, {changeable} hay {readOnly} cần được thực hiện ngầm bởi các thao tác truy cập:
 - Các thuộc tính {changeable} cần có các thao tác truy cập set<ten thuộc tính> và get<ten thuộc tính>.
 - Tính chất {frozen} đòi hỏi phải khởi tạo thuộc tính trong cấu tử (constructor) của lớp. Điều này thường được thực hiện thông qua một tham số khởi tạo.
 - Với thuộc tính {readOnly} thì chỉ có thao tác get mà không có thao tác set

61

► Cuối cùng, thì thiết kế các thuộc tính còn phải chỉ rõ các phương thức dùng để cập nhật các thuộc tính dẫn xuất. Các kỹ thuật vận dụng cho mục đích này đã được đề cập trong mục thiết kế lớp.

62

► *Thí dụ:* Thiết kế các thuộc tính



63

2.4. Thiết kế các thao tác

- Thiết kế các thao tác là bước cuối cùng trong vòng lặp thiết kế chi tiết. Có lẽ đây là công việc thu hút nhiều sức lực và thời gian nhất. Nội dung thiết kế các thao tác là đưa ra một hình ảnh khá chi tiết cho các phương thức trong đề án.
 - Đề thiết kế và tư liệu hoá các phương thức, UML đã cung cấp sẵn cho ta các biểu đồ mô hình hoá động thái, như là:
 - Một biểu đồ hoạt động, cho phép mô tả một phương thức mà trong đó có ít các lớp khác can thiệp vào. Biểu đồ này là rất hữu hiệu khi mô tả một giải thuật có nhiều giai đoạn và có nhiều phép chọn lựa tinh huống.
 - Một biểu đồ tương tác, cho phép trình bày một phương thức mà trong đó có nhiều lớp tham gia, nhưng lại ít sự chọn lựa tinh huống.

64

3. Thiết kế chi tiết các tầng

- ▶ 3.1. Thiết kế tầng trình bày
 - ▶ 3.2. Thiết kế tầng ứng dụng
 - ▶ 3.3. Thiết kế tầng nghiệp vụ
 - ▶ 3.4. Thiết kế việc lưu trữ các dữ liệu

65

3.1. Thiết kế tầng trình bày

- ▶ Vòng lặp của quy trình thiết kế chi tiết được áp dụng cho từng tầng trong kiến trúc khách hàng/dịch vụ (tầng trình bày, tầng ứng dụng, tầng nghiệp vụ, tầng truy cập dữ liệu và tầng lưu trữ dữ liệu).
 - ▶ Tầng trình bày giới hạn ở phần trông thấy được của ứng dụng: đó là phần thực hiện giao diện giữa người và máy tính. Đây cũng là phần dễ thay đổi nhất của hệ thống, bởi vì kỹ thuật làm giao diện tiền bộ rất nhanh, mà người dùng thì sau một thời gian khai thác hệ thống lại thường muốn có những đổi mới trong hình thức giao tiếp.

66

- ▶ Hình thức giao diện người dùng phổ biến nhất hiện nay là giao tiếp qua các cửa sổ (loại Windows). Trong hình thức này thì người dùng đối diện với ba loại khái niệm chính:
- Các cửa sổ và nội dung bên trong đó, mà người dùng có thể trông thấy, dịch chuyển, thay đổi kích cỡ... Đó là phần nhìn thấy hay là phần **cấu trúc tĩnh** của giao diện.
 - Các hành động mà người dùng có thể kích hoạt tạo nên sự thay đổi trạng thái của giao diện. Đó là phần **hành vi** của giao diện.
 - Các luồng thông tin mà hệ thống đưa ra cho người dùng hay người dùng đưa vào hệ thống thông qua các danh sách chọn lựa, các trường biên tập... Đó là phần trao đổi thông tin với ứng dụng, tức là phần **chức năng** của giao diện.

67

- ▶ Thiết kế và tư liệu hoá tầng trình bày trở thành việc xem xét ba sắc thái nói trên của nó, tức là sự hiển thị, hành vi và chức năng.
- ▶ Sự trợ giúp dễ dàng và thuận lợi của các công cụ GUI là rất dễ bị lạm dụng, dẫn tới sự vi phạm nghiêm trọng nguyên tắc cổ kết cao và tương liên yếu của các tầng. Bởi vì dễ có xu hướng là từ giao diện người dùng ta cho gọi trực tiếp tới các đối tượng thực thể để lấy thông tin, và điều đó sẽ tạo ra một sự tương liên chằng chịt, khó khống chế giữa tầng trình bày với các tầng dưới nó.
- ▶ Để tránh nguy cơ này, trong thiết kế tầng trình bày, ta nên áp dụng **khuôn khổ MVC**, với định hướng là tạo ra một sự phân công tách biệt giữa ba phần như sau:

68

- ▶ M là **Model** (mô hình), đó là lớp thực thể (chẳng hạn Khách hàng, Đơn hàng) thuộc tầng nghiệp vụ. Đây là nguồn gốc của thông tin được hiển thị trên giao diện người dùng. Tuy nhiên giao diện không truy cập trực tiếp vào đối tượng thực thể, mà tầng ứng dụng sẽ gom thông tin từ các đối tượng thực thể vào các "**tài liệu**" (document), và giao diện người dùng truy cập vào tài liệu để lấy tin. Thường thì có sự tương ứng 1-1 giữa khung nhìn và tài liệu, song một tài liệu có thể tương ứng với một hay nhiều đối tượng thực thể.

69

- V là **View** (khung nhìn), đó là phần hiển thị của giao diện mà người dùng trông thấy. Khung nhìn phải quản lý các vật thể đồ họa (cấu trúc tĩnh) của mình, và có thể truy cập vào các tài liệu để lấy thông tin hiển thị hay để cập nhật thông tin.
- C là **Controller** (bộ điều khiển), đó là phần đảm trách việc quản lý động thái (tức là hành vi) của giao diện, thực hiện sự chuyển đổi trạng thái của giao diện theo các sự kiện kích hoạt từ phía người dùng. Có sự tương ứng 1-1 giữa View và Controller. Thông thường thì View + Controller tạo nên tầng trình bày.

70

3.2. Thiết kế tầng ứng dụng

- Tầng ứng dụng bao gồm trước hết các lớp điều khiển (đã được chỉ ra trong phân tích, trên mô hình cấu trúc tĩnh của hệ thống), và như thế nó có nhiệm vụ dẫn dắt các quá trình của ứng dụng, cũng như thực hiện các quy tắc cản bảo đảm của ứng dụng.

71

- Theo James Martin, thì trong một hệ thống có thể có ba loại quy tắc, hay còn gọi là **luật** (rules):
- Các **luật toàn vẹn** phát biểu rằng điều gì đó phải luôn đúng. Chẳng hạn một thuộc tính phải có giá trị nguyên từ 1 đến 5.
 - Các **luật dẫn xuất** phát biểu rằng một giá trị (hay một tập các giá trị) được tính như thế nào. Chẳng hạn $\text{thànhtiền} = \text{đơngiá} \times \text{sốlượng}$.
 - Các **luật hành vi** mô tả sắc thái động của hành vi, như là các điều kiện nào phải đúng khi bước vào thực hiện một hành động. Chẳng hạn khi cửa số lò vi sóng mở ra thì đèn phải bật sáng.

72

- Các luật như trên, thường gọi là các luật nghiệp vụ, đã được nghiên cứu trong giai đoạn phân tích và đã được ánh xạ vào các lớp điều kiện. Tuy nhiên còn có các luật hay đúng ra là các yêu cầu có tính chất kỹ thuật, thì nay trong thiết kế chi tiết ta mới phải ý chú đến. Đó là:
 - Sự **đồng bộ hoá** của nhiều cửa sổ trên các dữ liệu chung.
 - Sự **tối ưu hoá** việc nạp tài các dữ liệu tại các trạm người dùng.
 - Sự **kiểm soát** chặt chẽ quá trình tương tác với người dùng.
 - Để áp dụng các yêu cầu trên, thì việc thiết kế chi tiết đối với tầng ứng dụng được thực hiện dựa trên sự định nghĩa các "tài liệu", biểu diễn cho hình ảnh bộ nhớ khi trao đổi dữ liệu với các cửa sổ. Theo định nghĩa, thì có một tài liệu cho mỗi cửa sổ. Trên cơ sở của khuôn khổ cài đặt này, ta sẽ lần lượt xem xét ở sau đây ba yêu cầu về kỹ thuật vừa nói trên.

73

a) Đồng bộ hóa các cửa sổ

- ▶ Đồng bộ hóa các cửa sổ chính là việc làm cho các cửa sổ khác nhau là không mâu thuẫn với nhau khi chúng trình bày một số dữ liệu có chung nguồn gốc. Chẳng hạn cùng một số thông tin địa lý, nhưng được trình bày trên 2 cửa sổ: cửa sổ văn tự và cửa sổ bản đồ.
 - ▶ Để đảm bảo tính đồng bộ của các thông tin trình bày, ta phải thiết lập các mối liên quan kết nhập giữa các tài liệu tương ứng với các cửa sổ (ở tầng ứng dụng) để chỉ rõ sự chia sẻ thông tin giữa chúng.

74

b) Tối ưu hoá việc tàng trữ thông tin từ các thực thể

- b) Tuy ưu hóa việc tải nạp thông tin từ các thực thể

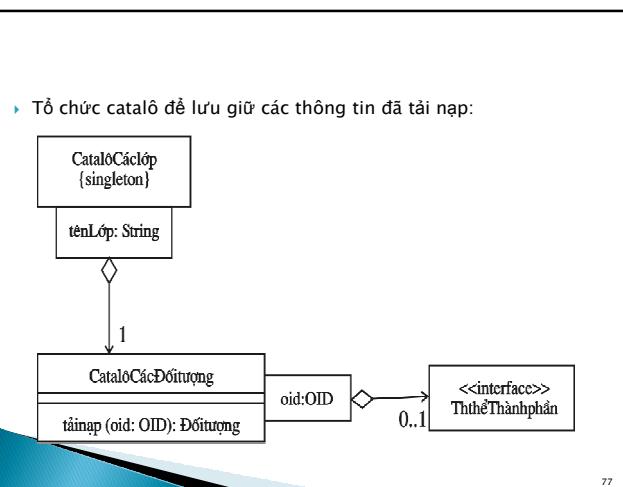
 - Trong một hệ phân tán, thì các đối tượng thực thể thường cư trú một cách phân tán, làm cho việc truy cập vào chúng trở nên tốn kém. Để tối ưu hoá việc tải nạp thông tin từ các thực thể tới tàng ứng dụng (cho tài liệu), ta phải tìm cách giảm thiểu số lần truy cập. Muốn thế, ta phải ghi nhớ các thông tin đã tải nạp, để tránh việc truy cập lặp đi lặp lại trên cùng một thực thể.

75

b) Tối ưu hoá việc tải nạp thông tin từ các thực thể (tt)

- Người ta dùng một **catalô tham chiếu** các đối tượng thực thể theo OID (Object identifier) của chúng.
 - Để tối ưu hóa, catalô được tổ chức thành catalô con, sắp xếp theo tên của lớp, để từ đó tìm đến các đối tượng theo OID.
 - Giao diện "Thambiền" biểu diễn cho các thực thể đến từ các thành phần phân tán. Khi tải nạp mới một thực thể, thì một đối tượng của giao diện Thambiền được tạo lập để ghi nhận các thông tin đã tải nạp.
 - Nhờ đó những lần tải nạp sau sẽ tìm lại được các thông tin này (trong catalô) mà không phải truy cập trực tiếp vào thực tế nữa.

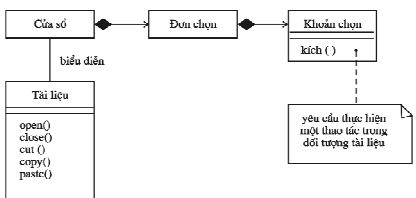
76



77

c) Tô chức sự kiêm soát chặt chẽ quá trình tương tác với người dùng

- Khi làm việc với một cửa sổ, người dùng có thể phát ra một lệnh (command) bằng cách kích vào một nút hay kích vào một khán trên một đơn chọn. Lệnh đó có mục đích gởi một thao tác nào đó trong đối tượng tài liệu, ứng với cửa sổ đó:



78

- ▶ Tuy nhiên lệnh phát ra từ cửa sổ không thể truyền lập tức và trực tiếp tới tài liệu (nơi nhận lệnh), vì nhiều lý do:
 - Lệnh có thể cần bị chặn lại để kiểm tra quyền sử dụng;
 - Lệnh phải lưu giữ lại, để đưa vào hàng đợi, đến lượt mới được thực hiện;
 - Dây các lệnh thực hiện liên tiếp cần phải quản lý và ghi nhớ để khi cần có thể trả lại (undo), hay phục hồi (redo).
 - Có khi lệnh còn phải được tham số hoá để thích ứng theo các hoàn cảnh cụ thể.
- ▶ Tất cả các yêu cầu về quản lý lệnh đó không thể giao cho cửa sổ phụ trách được, vì các cửa sổ được thiết kế theo các bộ dụng cụ (toolkit), phải giữ tính độc lập với ứng dụng. Vậy chính ứng dụng phải tổ chức lấy việc quản lý các lệnh này một cách chặt chẽ và sát với yêu cầu của mình.

79

3.3. Thiết kế tầng nghiệp vụ

- ▶ Tầng nghiệp vụ chính là nơi trú ngụ của các lớp thực thể phản ánh lĩnh vực ứng dụng. Đây là tầng ổn định nhất và có khả năng tái sử dụng cao.
- ▶ Bước đầu, thì mỗi lớp thực thể bao gồm các thuộc tính và thao tác như đã phát hiện ở giai đoạn trước. Tuy nhiên sau đó thì ta phải đưa thêm:
 - Các thao tác get (lấy) và set (đặt) cho mỗi thuộc tính, vì các thuộc tính đều được khai báo là riêng tư;
 - Các dịch vụ mà mỗi lớp thực thể phải cung ứng, nhằm đáp ứng các yêu cầu tìm kiếm thông tin theo những sắc thái nào đó của hệ thống;
 - Các thao tác đáp lại các yêu cầu cập nhật, bằng cách chuyển các tác dụng cập nhật xuống tầng dưới (tầng lưu trữ dữ liệu).

80

3.4. Thiết kế việc lưu trữ các dữ liệu

- ▶ Có những đối tượng cần phải lưu trữ lại một cách lâu dài (trên bộ nhớ ngoài), chứ không thể dễ dàng bị xóa bỏ cùng với sự kết thúc chương trình. Gọi đó là các đối tượng trường cửu (persistent). Thường thì đó là các đối tượng thực thể, và ít khi là các đối tượng biên (giao diện) hay đối tượng điều khiển.

81

a) Chọn lựa cách lưu trữ dữ liệu

- ▶ Việc mô hình hoá hệ thống bằng các lớp và đối tượng tạo nhiều thuận lợi cho việc lưu trữ dữ liệu. Tuy nhiên ngày nay có ba cách lưu trữ có thể dùng:
- ▶ Các **hệ thống tệp** (tập tin): Đó là phương tiện lưu trữ thô sơ nhất. Lưu trữ dữ liệu bằng tệp hầu như không gây nên tổn kém gì. Tuy nhiên nó chỉ cho phép đọc và viết các đối tượng, mà không có khả năng đặt ra các câu hỏi tìm kiếm dữ liệu phức tạp. Vì vậy nó ít được dùng với các hệ thống có tầm quan trọng đáng kể.

82

- ▶ Các **cơ sở dữ liệu quan hệ** (RDBMS): Tinh tế hơn trong quản lý dữ liệu và tìm kiếm thông tin với các câu hỏi phức tạp. Tồn tại nhiều hệ quản trị CSDL thích ứng theo các yêu cầu khác nhau về khối lượng, sự phân tán và hệ điều hành. Cho nên đây là cách lưu trữ dữ liệu được dùng phổ cập hơn cả.
- ▶ Các **cơ sở dữ liệu hướng đối tượng** (OODBMS): Cho phép lưu trữ và quản lý các đối tượng một cách trực tiếp. Nhờ đó mà có thể nói khâu thiết kế cho việc lưu trữ dữ liệu hầu như chẳng còn việc gì phải làm. Tuy nhiên các hệ quản trị CSDL đối tượng còn chưa thực sự hoàn chỉnh và chưa chiếm được thị phần lớn trên thực tế.
- ▶ Dưới đây ta đề cập việc thiết kế CSDL với sự chọn lựa một hệ quản trị CSDL quan hệ.

83

b) Ánh xạ các lớp sang bảng

- ▶ Mỗi lớp trường cữu tương ứng với một bảng trong mô hình quan hệ:
 - mỗi thuộc tính của lớp tương ứng với một cột của bảng,
 - mỗi cá thể (đối tượng) của lớp tương ứng với một dòng (một bộ-n) của bảng, trong đó OID của đối tượng đóng vai trò là khoá chính (primary key) trong bảng.
- ▶ Có một số thuộc tính của lớp có thể có kiểu phức tạp (các cấu trúc dữ liệu) và không tương ứng được với các kiểu của SQL. Vậy phải có sự biến đổi các kiểu phức tạp đó:
 - Hoặc là bảng nhiều cột, mỗi cột tương ứng với một trường trong cấu trúc dữ liệu.
 - Hoặc là bảng một bảng riêng biệt, liên hệ với bảng chính bằng một khoá ngoài, cho phép kết nối các đối tượng với các giá trị của thuộc tính phức tạp.

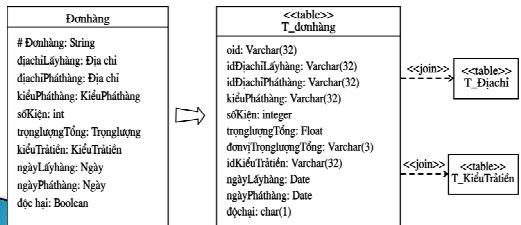
84

▶ Sự tương đương giữa lớp và bảng

Mô hình đối tượng	Mô hình quan hệ
Lớp	Bảng
Thuộc tính có kiểu đơn	Cột
Thuộc tính có kiểu phức tạp	Các cột hay khoá ngoài
Đối tượng	Bộ-n
OID	Khoá chính
Liên kết và Kết nối	Khoá ngoài hay bảng
Thừa kế	Khoá chính đồng nhất trên nhiều bảng

85

- ▶ *Thí dụ:* ánh xạ lớp Đơn hàng sang bảng. Kết quả của sự ánh xạ là một biểu đồ lớp (trong UML), diễn tả cấu trúc lưu trữ, trong đó mỗi lớp đều mang khuôn dập <<table>>, và tên lớp được gắn thêm tiền tố T_. Một công cụ trợ giúp (Rational Rose chẳng hạn) sẽ biến đổi biểu đồ này thành ngôn ngữ định nghĩa dữ liệu (DDL), cho phép tạo lập các bảng trong CSDL.



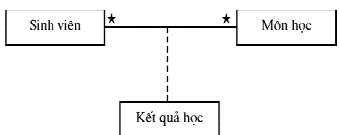
86

- ▶ Ánh xạ từ lớp sang bảng lại không nên hiểu máy móc là ánh xạ 1-1, vì ánh xạ 1-1 thường dẫn tới các hiện tượng không tốt như sau:

- *Quá nhiều bảng và quá nhiều kết nối phải thực hiện:* Thông thường thì biểu đồ lớp có thể chứa rất nhiều lớp, nếu cứ ánh xạ 1-1 thành các bảng, thì ta sẽ có rất nhiều bảng. Mà khi đã có nhiều bảng, thì để tìm kiếm thông tin trong CSDL, ta lại phải áp dụng nhiều phép kết nối (join), ảnh hưởng năng nề tới tốc độ truy cập. Vì vậy nên cố gắng gom các bảng thường đi liền với nhau trong tìm kiếm thành các bảng lớn.

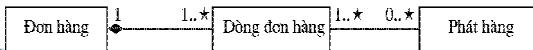
87

- ☐ **Thiếu các bảng:** Như ta đã thấy trong mục thiết kế các liên kết, thì nếu giữa hai lớp có một liên kết nhiều-nhiều mà liên kết đó lại có thuộc tính, thì phải định nghĩa một lớp liên kết chứa thuộc tính đó. Chẳng hạn giữa Sinh viên và Môn học có một liên kết nhiều-nhiều ("Sinh viên hoàn thành môn học"); liên kết này lại có thuộc tính (như là năm học, kết quả thi), do đó trong mô hình đối tượng, phải thêm một lớp liên kết là lớp Kết quả học. Chuyên sang mô hình quan hệ, thì ngoài hai bảng Sinh viên và Môn học, ta cũng có bảng Kết quả học.



88

- Tuy nhiên trong trường hợp một liên kết nhiều-nhiều giữa hai lớp mà không có thuộc tính, thì lớp liên kết là không cần có trong biểu đồ lớp. Chẳng hạn giữa **DòngĐơn hàng** và **Pháthàng** có một liên kết nhiều-nhiều, tuy nhiên liên kết này không có thuộc tính nào, cho nên không cần có lớp liên kết ở đây (bởi vì liên kết nhiều-nhiều sẽ được cài đặt trực tiếp bằng các thuộc tính bội, được phép dùng trong các ngôn ngữ lập trình hướng đối tượng). Tuy nhiên chuyển qua mô hình quan hệ, thì nhất thiết lại phải có một bảng đặt tương ứng giữa các đối tượng của hai bên để biểu diễn cho liên kết nhiều-nhiều này. Do đó ánh xạ 1-1 làm mất quan hệ này.



89

- Mất khả năng thừa kế: Mô hình quan hệ không sẵn sàng hỗ trợ cho sự thừa kế. Vì vậy một ánh xạ đơn giản các lớp sang các bảng, mà không có các xử lý đặc biệt (xem ở dưới) thì không còn giữ được mối liên quan khái quát hoá và sự thừa kế.
 - Suy giảm hiệu năng: Một ánh xạ 1-1 thường chuyển biểu đồ lớp thành mô hình quan hệ ở dạng chuẩn 3. Tuy nhiên có nhiều ứng dụng hướng tới một số báo cáo nhất định, đòi hỏi phải hạ chuẩn đối với dữ liệu (chẳng hạn phải lặp lại cùng một dữ liệu ở nhiều bảng, hay sáp nhập một số bảng) nhằm tạo khả năng truy cập thuận lợi nhất cho những mục đích định sẵn của ứng dụng đó.

90

c) ánh xạ các liên kết

- ▶ Trong phần trên, ta đã đề cập việc thiết kế chi tiết các liên kết, vốn không có sẵn trong các ngôn ngữ lập trình hướng đối tượng.
- ▶ *Với liên kết một-một:* Lập một bảng cho mỗi lớp (bảng A, B). Khoá chính của mỗi bảng cũng là khoá ngoài đối với bảng kia. Thực ra lập thành hai bảng là chỉ thực sự cần thiết khi liên kết là một ánh xạ vào (1 đến 0..1), còn khi liên kết là một đối một, thì cách tốt nhất là chỉ lập một bảng.

91

- *Với liên kết một-nhiều:* Lập một bảng cho mỗi lớp (bảng A, B). Khoá chính của bảng A (đầu Một) là khoá ngoài trong bảng B (đầu Nhiều).
- *Với liên kết nhiều-nhiều:* Lập một bảng cho mỗi lớp (bảng A, B). Lập thêm một bảng kết nối, hay còn gọi là bảng giao (bảng C). Khoá chính của mỗi bảng A, B được định nghĩa là khoá ngoài trong bảng C. Khoá chính của C có thể là một cột riêng song cũng có thể là khoá bởi hợp thành từ hai khoá ngoài.

92

- *Với kết nhập và hợp thành:* Trong mô hình quan hệ thì kết nhập và hợp thành cũng chỉ được mô hình hoá như một liên kết bình thường. Hợp thành khi có cơ số là 1-1, thì nên cài đặt thành một bảng duy nhất. Nếu hợp thành được cài đặt thành hai bảng riêng biệt, thì khả năng loại bỏ lan truyền phải được xem xét và cài đặt trong CSDL vật lý. Còn kết nhập, thì cứ nên để thành hai bảng, vì các đối tượng bị kết nhập vẫn có thể tồn tại một cách độc lập.
- *Với các liên kết đệ quy:* Đó là loại liên kết giữa một lớp với chính nó, cho ta các kết nối giữa các cặp đối tượng của lớp. Liên kết đệ quy được cài đặt bằng cách thêm vào một cột, mà giá trị là các giá trị khoá chính của lớp đó, xem như là một khoá ngoài.

93

d) ánh xạ mối liên quan khái quát hóa

- ▶ Nếu hầu hết các ngôn ngữ lập trình hướng đối tượng hỗ trợ cho mối liên quan này giữa các lớp (sự thừa kế), thì mô hình quan hệ không hỗ trợ cho nó. Có nhiều cách để chuyển đổi mối liên quan này vào mô hình quan hệ:
 - *Lập một bảng cho mỗi lớp* (lớp trên cũng như lớp dưới). Mỗi liên quan giữa các bảng được giải quyết theo hai cách:
 - Dùng mộtOID chung cho mọi bảng trong cùng một phânhệ thừa kế. Như vậy để truy cập dữ liệu của một đối tượng phải thực hiện phép kết nối (join) giữa lớp trên và lớp dưới (cha và một con).
 - Lập một khung nhìn SQL (view) cho mỗi cặp lớp trên/lớp dưới. Như vậy số bảng tăng thêm, và thực chất cũng phải dùng phép kết nối khi truy cập dữ liệu.
- ▶ Cả hai cách làm này đều cho phép tăng thêm các lớp con trong tương lai, mà không làm xáo trộn các lớp cũ.

94

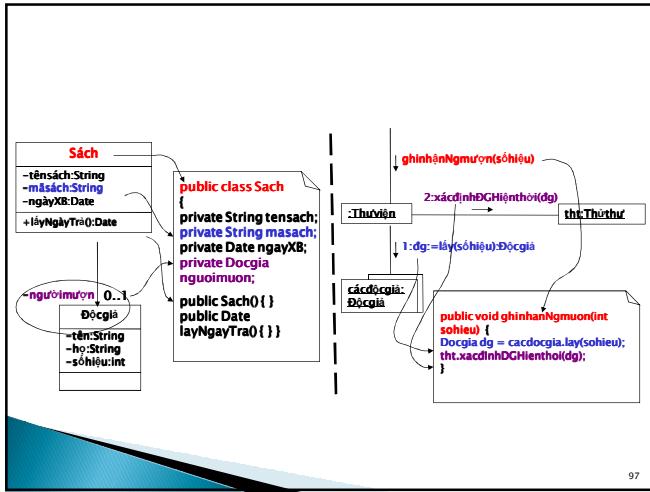
- ▶ *Chỉ lập một bảng* (ứng với lớp trên), mọi thuộc tính ở các lớp dưới đều dồn về bảng trên. Như vậy số bảng là tối thiểu (=1), nhưng số cột tăng thêm, và với mỗi cá thể, có những cột là không dùng tới. Vậy chỉ thích hợp khi số các thuộc tính trong các lớp dưới là ít. Không thuận lợi cho việc thêm mới các lớp dưới.
- ▶ *Lập một bảng cho mỗi lớp dưới*, lập lại các thuộc tính của lớp trên vào bảng của mỗi lớp dưới (hà chuẩn). Như vậy số bảng rút bớt (không có bảng cho lớp cha), và việc tăng thêm lớp con trong tương lai không gây ánh hưởng, nhưng việc điều chỉnh lớp cha buộc phải điều chỉnh lại các bảng. Thích hợp khi số các thuộc tính trong lớp cha là ít.

95

4. Bài tập tổng hợp

- ▶ Sản sinh sườn chương trình (trong Java) từ các BĐ lớp:
 - Một lớp UML trở thành một lớp Java
 - Các thuộc tính UML trở thành biến cá thể Java
 - Các thao tác UML trở thành các phương thức Java

96



97

► **Câu hỏi 29:** Từ BD lớp ở câu hỏi 24, hãy đề xuất sườn chương trình Java cho lớp ĐơnxinDT.

entity
ĐơnxinDT

- ngàygui:Date
- ngayxinDT:Date

+nối(phantut:PhantuCatalo)
+gán(ngayxinDT:Date)
+tuchoi()
+chaphan()
+chonky(k:Kyhoc)
+ketthuckY()
+huy()
+huyKy(k:Kyhoc)
-gửiTuchoi(lýdo:String)
-gửiĐongy()
-gửiĐơn()

Chú ý: các liên kết đã được cài đặt bằng thuộc tính riêng tư

entity
ĐơnxinDT

```

package DenghiDT;
import java.util.*;
import Catalo.Kyhoc;
import Catalo.PhantuCatalo;
public class DonxinDT {
    private Date ngaygui;
    private Date ngayxinDT;
    private Nhanvien nguoinixin;
    private Dangky dangky;
    private PhantuCatalo phantuCatalo;
    private Tra loi tralo;
    public DonxinDT(){}
    public void noi(PhantuCatalo phantu){}
    public void gan(Date ngayxinDT){}
    public void tuchoi(){}
}

```

98

► **Câu hỏi 30:** Từ BD lớp ở câu hỏi 20, hãy đề xuất sườn chương trình Java cho lớp LopDT.

entity
Kyhoc
PhantuCatalo
Chuode
CosoDT
Noidung

LopDT

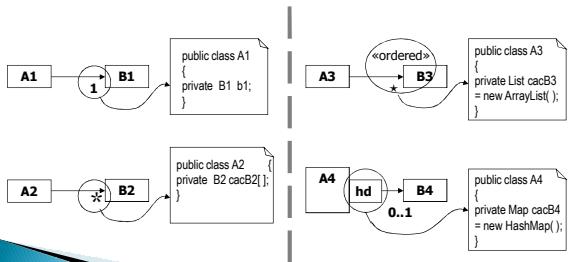
- tên:String
- thoigian:short
- giain:

+taolopNoidung(nguoinhoc:String, muc tieu:String, congcu:String, kehoach:String)
+taolopKy(ngayBaithau:Date, dia diemString)
+ghép(chuode.Chuode.cosoDT.CosoDT)
+diéuchinhNoidung(nguoinhoc:String, muc tieu:String, congcu:String, kehoach:String)
+diéuchinhKy(ngayBaithau:Date, dia diemString)
+diéuchinh(chuode.Chuode.cosoDT.CosoDT)
+huyKy(k:Kyhoc)

Đáng chú ý ở đây là việc cài đặt mối liên quan **khái quát hoá** và các liên kết với **cố số “*”**, đặc biệt là khi có thêm **hạn định hoặc «ordered»**.

99

- Nguyên tắc cài đặt liên kết “*” là dùng một thuộc tính tập thể các tham chiếu đối tượng thay vì một tham chiếu đối tượng đơn. Kiểu của thuộc tính tập thể thường dùng là ArrayList (trước đây là Vector) và HashMap (trước đây là HashTable).

10
0

Với lớp LớpDT ta dùng:

- Một ArrayList cho liên kết có thứ tự với lớp Kýhọc.
- Một HashMap cho liên kết với lớp Chủđề (dùng tên chủ đề làm hạn định) thay vì một bảng đơn.

10
1

```

package CataloDT;
import java.util.*;
public class LopDT extends PhantuCatalo {
    private String ten;
    private short thoigian;
    private int gia;
    private List cacKy = new ArrayList();
    private Map cacChude = new HashMap();
    private Noidung noidung;
    private CosoDT cosoDT;
    public LopDT() {}
    public void taolapNoidung(String nguoihoc, String muc tieu, String congcu, String kehoach) {}
    public void taolapKyhoc(Date ngayBatdau, String diadiem) {}
    public void ghep(Chude chude, Kyhoc kyhoc) {}
}
  
```

```

public void dieuchinhNoidung(
    String nguoihoc, String muc tieu,
    String congcu, String kehoach) {}
public void dieuchinhKyhoc(
    Date ngayBatdau, String diadiem) {}
public void dieuchinh(Chude chude,
    Kyhoc kyhoc) {}
public void huyKy(k:Kyhoc) {}
public String layTen() {return ten;}
public void ganTen(String t) {ten=t;}
public short layThoigian() {
    return thoigian;
}
public void ganThoigian(short t)
{thoigian=t;}
public int layGia() {return gia;}
public void ganGia(int g) {gia=g;}
  
```

10
2