**UML 2.0**

**Mục lục**

[CHƯƠNG 1: USE CASE VIEW 4](#_Toc514535918)

[1. Tổng quan 4](#_Toc514535919)

[2. Actor 4](#_Toc514535920)

[3. Use case 5](#_Toc514535921)

[CHƯƠNG 2: STATE MACHINE VIEW 10](#_Toc514535922)

[1. Tổng quát 10](#_Toc514535923)

[2. State Machine 11](#_Toc514535924)

[3. Sự kiện 11](#_Toc514535925)

[4. Trạng thái 14](#_Toc514535926)

[5. Chuyển đổi 15](#_Toc514535927)

[6. Trạng thái hỗn hợp 22](#_Toc514535928)

[CHƯƠNG 3: STATIC VIEW 28](#_Toc514535929)

[1. Tổng quan 28](#_Toc514535930)

[2. Trình phân loại 29](#_Toc514535931)

[3. Relationship 34](#_Toc514535932)

[4. Association 36](#_Toc514535933)

[5. Aggregation and composition. 40](#_Toc514535934)

[6. Liên kết. 41](#_Toc514535935)

[7. Bidirectionality. 41](#_Toc514535936)

[8. Generalization 42](#_Toc514535937)

[9. Inheritance 44](#_Toc514535938)

[10. Multiple inheritance 45](#_Toc514535939)

[11. Single and multiple classification 46](#_Toc514535940)

[12. Static and dynamic classification 47](#_Toc514535941)

[13. Realization 47](#_Toc514535942)

[14. Dependency 49](#_Toc514535943)

[15. Contraint 53](#_Toc514535944)

[16. Instance 55](#_Toc514535945)

[17. Sơ đồ đối tượng 56](#_Toc514535946)

[CHƯƠNG 4: DESIGN VIEW 58](#_Toc514535947)

[1. Tổng quan 58](#_Toc514535948)

[2. Trình phân loại có cấu trúc 59](#_Toc514535949)

[3. Cộng tác 61](#_Toc514535950)

[4. Pattern 62](#_Toc514535951)

[5. Component 63](#_Toc514535952)

[CHƯƠNG 5: ACTIVITY VIEW 66](#_Toc514535953)

[1. Activity 66](#_Toc514535954)

[2. Phân vùng 67](#_Toc514535955)

[3. Object flows 68](#_Toc514535956)

[4. Activities và các chế độ xem khác 69](#_Toc514535957)

[5. Action 69](#_Toc514535958)

[CHƯƠNG 6: INTERACTION VIEW 72](#_Toc514535959)

[1. Interaction 72](#_Toc514535960)

[2. Sequence Diagram 73](#_Toc514535961)

[3. Execution Specification (đặc tả thực thi) 74](#_Toc514535962)

[4. Structured control constructs ( Cấu trúc điều khiển có cấu trúc ) 76](#_Toc514535963)

[5. Communication Diagrams 77](#_Toc514535964)

[CHƯƠNG 7: DEPLOYMENT VIEW 79](#_Toc514535965)

[1. Tổng quan 79](#_Toc514535966)

[2. Node 79](#_Toc514535967)

[3. Artifact 79](#_Toc514535968)

[CHƯƠNG 8: MODEL MANAGEMENT VIEW 81](#_Toc514535969)

[1. Tổng quan 81](#_Toc514535970)

[2. Package 81](#_Toc514535971)

[3. Phụ thuộc vào gói 83](#_Toc514535972)

[4. Hiển thị 84](#_Toc514535973)

[5. Import 85](#_Toc514535974)

[6. Model 86](#_Toc514535975)

# **CHƯƠNG 1: USE CASE VIEW**

## **1. Tổng quan**

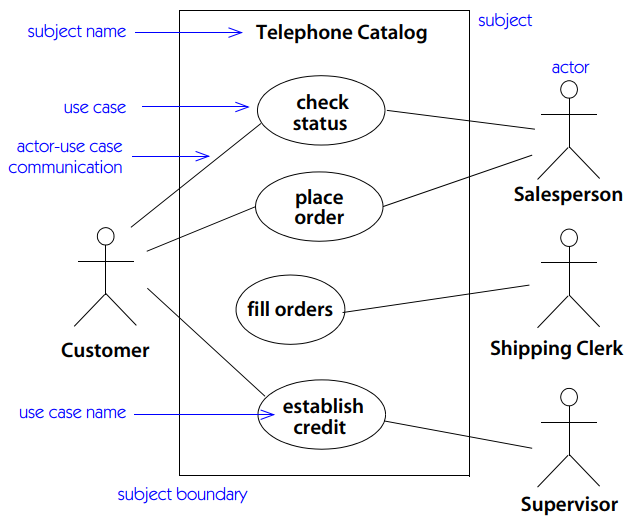
use case view ghi lại hành vi của **hệ thống**, **hệ thống con**, **lớp** hoặc **thành phần** khi nó xuất hiện, cho người dùng bên ngoài. Nó phân vùng chức năng hệ thống thành các giao dịch có ý nghĩa với các **actors** — những người dùng lý tưởng của một hệ thống. Các phần chức năng tương tác được gọi là **use cases**. Một use case mô tả sự tương tác với actor như một chuỗi các **thông điệp** giữa hệ thống với một hoặc nhiều actor. Các actor chính bao gồm con người, cũng như các hệ thống và quy trình máy tính khác. **Hình 6-1** cho thấy một use case sử dụng cho một ứng dụng bán hàng danh mục điện thoại. Mô hình đã được đơn giản hóa để làm ví dụ.

## **2. Actor**

Một **actor** là một sự lý tưởng hóa vai trò của một người bên ngoài, quá trình, hoặc một điều tương tác với một hệ thống, hệ thống con hoặc lớp. Một actor mô tả các tương tác mà một lớp người dùng bên ngoài có thể có với hệ thống. Vào thời gian chạy, một người dùng vật lý có thể bị ràng buộc với nhiều tác nhân trong hệ thống. Những người dùng khác nhau có thể bị ràng buộc vào cùng một actor và do đó đại diện cho nhiều trường hợp do cùng một actor định nghĩa. Ví dụ: một người có thể là khách hàng và nhân viên thu ngân của cửa hàng vào những thời điểm khác nhau.

Mỗi tác nhân tham gia vào một hoặc nhiều use case. Nó tương tác với use case (và do đó hệ thống hoặc lớp sẽ chứa use case) bằng cách trao đổi thông điệp. Việc thực hiện thêm vào một actor không ảnh hưởng đến use case; một actor có thể được biểu thị đặc trưng đầy đủ bởi một tập các thuộc tính xác định trạng thái của nó.

Các actor có thể được định nghĩa trong các hệ thống phân cấp **tổng quát**, trong đó mô tả tóm lược actor được chia sẻ và được tăng bởi một hoặc nhiều mô tả actor cụ thể. Một actor có thể là một con người, một hệ thống máy tính, hoặc một số quá trình đang thực thi. Một actor được vẽ như một người cây nhỏ với tên bên dưới nó.



## **3. Use case**

Một **use case** là một đơn vị liên kết của chức năng có thể nhìn thấy được từ bên ngoài được cung cấp bởi một trình phân loại (được gọi là **chủ đề**) và được biểu diễn bằng chuỗi các thông điệp được trao đổi bởi chủ thể và một hoặc nhiều **actors** của đơn vị hệ thống. Mục đích của use case là định nghĩa một phần của hành vi liên kết mà không tiết lộ cấu trúc bên trong của đối tượng. Định nghĩa của use case bao gồm tất cả hành vi mà nó yêu cầu - các chuỗi chính, các biến thể khác nhau về hành vi bình thường, và tất cả các điều kiện ngoại lệ có thể xảy ra với hành vi đó, cùng với đáp ứng mong muốn. Từ quan điểm của người dùng, đây có thể là những tình huống bất thường. Từ quan điểm của hệ thống, chúng là các biến thể bổ sung phải được mô tả và xử lý.

Trong mô hình, việc thực hiện từng trường hợp sử dụng độc lập với các trường hợp khác, mặc dù việc triển khai các use case có thể tạo ra các phụ thuộc ngầm định giữa các use case do các đối tượng được chia sẻ. Mỗi use case đại diện cho một phần chức năng trực giao mà thực thi có thể được trộn lẫn với việc thực hiện các use case khác.

Động lực của một use case có thể được xác định bằng các tương tác UML, được hiển thị dưới dạng biểu đồ biểu đồ **trạng thái**, biểu đồ **trình tự**, biểu đồ **nối tiếp** hoặc mô tả văn bản không chính thức. Khi các use case được thực hiện, chúng được nhận biết bằng cách **cộng tác** giữa các lớp trong hệ thống. Một lớp có thể tham gia vào nhiều sự cộng tác và trong nhiều use case.

Ở cấp hệ thống, các use case đại diện cho hành vi bên ngoài của đối tượng như việc hiển thị cho người dùng bên ngoài. Một use case giống như một thao tác, một thao tác có thể được yêu cầu bởi người dùng bên ngoài. Tuy nhiên, không giống như một thao tác, một use case có thể tiếp tục nhận đầu vào từ các tác nhân của nó trong quá trình thực hiện nó. Trường hợp sử dụng có thể được áp dụng cho toàn bộ hệ thống và cũng có thể được áp dụng nội bộ cho các đơn vị nhỏ hơn của một hệ thống, chẳng hạn như hệ thống con và các lớp riêng lẻ. Một use case nội bộ đại diện cho hành vi mà một hệ thống phụ trình bày cho phần còn lại của hệ thống. Ví dụ, một use case cho một lớp đại diện cho một đoạn hàm thống nhất mà một lớp cung cấp cho các lớp khác đóng vai trò nhất định trong hệ thống. Một lớp có thể có nhiều trường hợp sử dụng.

Một use case là một mô tả logic của một lát chức năng. Nó không biểu hiện một cấu trúc trong việc thực hiện một hệ thống. Thay vào đó, mỗi use case phải được ánh xạ lên các lớp thực hiện một hệ thống. Hành vi của use case là ánh xạ lên các quá trình chuyển đổi và hoạt động của các lớp. Bởi vì một lớp có thể đóng nhiều vai trò trong việc thực hiện một hệ thống, do đó nó có thể nhận ra các phần của nhiều trường hợp sử dụng. Một phần của nhiệm vụ thiết kế là tìm các lớp triển khai kết hợp chặt chẽ các vai trò thích hợp để thực hiện tất cả các use case, mà không đưa ra các phiền phức không cần thiết. Việc triển khai use case có thể được mô hình hóa như một tập hợp của một hoặc nhiều **cộng tác**. Sự cộng tác là **việc thực hiện** một use case.

Một use case có thể tham gia vào một số mối quan hệ, ngoài việc liên kết với các actor (**Bảng 6-1**).

Bảng 6-1: Các loại quan hệ use case

|  |  |  |
| --- | --- | --- |
| Mối quan hệ | Chức năng | Ký hiệu |
| Association (kết hợp) | Đường dẫn giao tiếp giữa một actor và một use case mà nó tham gia. |  |
| Extend (mở rộng) | Việc chèn hành vi bổ sung vào một use case cơ bản mà không biết về nó. |  |
| Include (bao gồm) | Chèn hành vi bổ sung vào use case cơ sở mô tả rõ ràng việc chèn. |  |
| Use case generalization (tổng quát) | Mối quan hệ giữa use case tổng quát và nhiều use case cụ thể được kế thừa và thêm các tính năng cho use case đó. |  |

Một use case được vẽ như một hình elip với tên của nó bên trong hoặc bên dưới nó. Nó được kết nối bởi các đường liền mạch để các actor giao tiếp với nó.

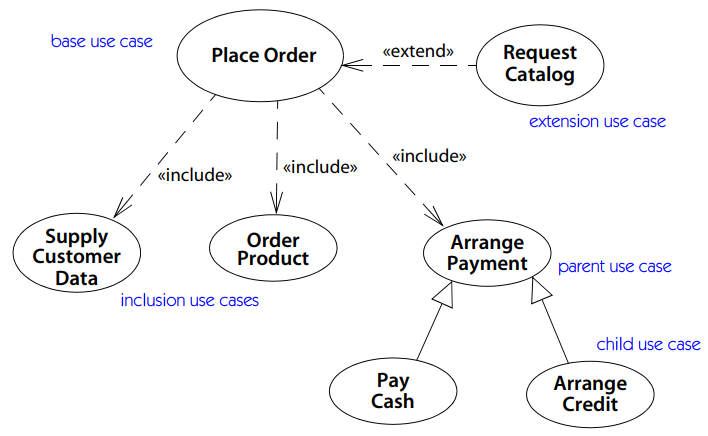
Mô tả về một use case lớn có thể được đưa vào các use case đơn giản khác. Điều này tương tự như cách mô tả của một lớp có thể được định nghĩa gia tăng từ mô tả của một siêu lớp. Một use case có thể kết hợp hành vi của các use case khác như các mảnh của hành vi riêng của nó. Điều này được gọi là mối quan hệ bao gồm. Use case được bao gồm không phải là trường hợp đặc biệt của use case ban đầu và không thể được thay thế cho nó.

Một use case cũng có thể được định nghĩa là một phần mở rộng gia tăng cho một use case cơ bản khác. Đây được gọi là mối quan hệ mở rộng. Có thể có một số phần mở rộng của cùng một use case cơ sở mà tất cả có thể được áp dụng cùng nhau. Các mở rộng cho một use case cơ sở thêm vào ngữ nghĩa của nó; nó là use case cơ bản được khởi tạo, không phải là use case mở rộng.

Các mối quan hệ bao gồm và mở rộng được vẽ như mũi tên đứt quãng với từ khóa «include» hoặc «extend».

Một use case cũng có thể được chuyên biệt hóa thành một hoặc nhiều use case con. Đây là use case tổng quát. Bất kỳ use case con nào cũng có thể được sử dụng trong trường hợp mà use case cha được mong đợi.

Việc sử dụng use case tổng quát hóa được rút ra giống như bất kỳ khái quát hóa nào, một đường thẳng ở use con và một đầu mũi tên hình tam giác lớn ở đầu cha. Hình 6-2 cho thấy các mối quan hệ use case sử dụng trong ứng dụng danh mục bán hàng.



# **CHƯƠNG 2: STATE MACHINE VIEW**

## **1. Tổng quát**

State machine view mô tả hành vi động của các đối tượng theo thời gian bằng cách mô hình hóa vòng đời của các đối tượng của mỗi lớp. Mỗi đối tượng được coi là một thực thể riêng biệt giao tiếp với phần còn lại của thế giới bằng cách phát hiện các sự kiện và trả lời chúng. Sự kiện thể hiện các loại thay đổi mà một đối tượng có thể phát hiện — việc nhận các cuộc gọi hoặc tín hiệu rõ ràng từ một đối tượng này đến một đối tượng khác, một sự thay đổi về các giá trị nhất định hoặc thời gian trôi qua. Bất cứ điều gì có thể ảnh hưởng đến một đối tượng có thể được mô tả như một sự kiện. Các diễn biến trong thế giới thực được mô hình hóa dưới dạng tín hiệu từ thế giới bên ngoài vào hệ thống.

Trạng thái là tập hợp các giá trị của đối tượng cho một lớp đã cho có cùng tính chất phản hồi với các sự kiện xảy ra. Nói cách khác, tất cả các đối tượng có cùng trạng thái phản ứng theo cùng một cách chung cho một sự kiện, vì vậy tất cả các đối tượng trong một trạng thái đã được thực thi cùng một hiệu ứng — một hành động hoặc hoạt động - khi chúng nhận được cùng một sự kiện. Tuy nhiên, các đối tượng ở các trạng thái khác nhau có thể phản ứng khác với cùng một sự kiện, bằng cách thực hiện các hiệu ứng khác nhau. Một máy trạng thái mô tả một số lượng nhỏ các trạng thái mà một đối tượng có thể giữ. Đối với mỗi trạng thái, máy trạng thái xác định ảnh hưởng của việc nhận từng loại sự kiện như một kết quả và thay đổi trạng thái mới. Ví dụ, một máy rút tiền tự động phản ứng với nút hủy một cách khi nó đang xử lý giao dịch và một cách khác khi nó không hoạt động.

Các máy trạng thái mô tả hành vi của các lớp, nhưng chúng cũng mô tả hành vi động của các use case, các cộng tác và các phương thức. Đối với một trong các đối tượng này, một trạng thái đại diện cho một bước trong việc thực thi nó. Chúng ta nói chủ yếu về các lớp và các đối tượng trong việc mô tả các máy trạng thái, nhưng chúng có thể được áp dụng cho các phần tử khác một cách đơn giản.

## **2. State Machine**

Một máy trạng thái là một đồ thị của các trạng thái và chuyển tiếp. Thông thường một máy trạng thái được gắn vào một lớp và mô tả đáp ứng của một thể hiện của lớp đối với các sự kiện mà nó nhận được. Máy trạng thái cũng có thể được gắn liền với hành vi, use case và cộng tác để mô tả việc thực thi của chúng.

Một máy trạng thái là một mô hình của tất cả các lịch sử tồn tại có thể có của một đối tượng của một lớp. Đối tượng được kiểm tra trong sự cô lập. Bất kỳ ảnh hưởng bên ngoài từ phần còn lại của thế giới được tóm tắt như là một sự kiện. Khi đối tượng phát hiện một sự kiện, nó phản hồi theo cách phụ thuộc vào trạng thái hiện tại của nó. Phản ứng có thể bao gồm việc thực thi hiệu ứng và thay đổi trạng thái mới. Các máy trạng thái có thể được cấu trúc để chia sẻ các quá trình chuyển đổi và chúng có thể mô hình đồng thời.

Một máy trạng thái là một tầm nhìn cục bộ của một đối tượng, một tầm nhìn tách nó ra khỏi phần còn lại của mọi vật và kiểm tra hành vi của nó trong sự cô lập. Đây là chế độ giảm tầm nhìn của một hệ thống. Đây là một cách tốt để xác định hành vi một cách chính xác, nhưng thường thì đó không phải là cách hay để hiểu hoạt động tổng thể của một hệ thống. Để có ý tưởng tốt hơn về hiệu ứng toàn hệ thống của hành vi, các chế độ xem tương tác thường hữu ích hơn. Các máy trạng thái rất hữu ích cho việc hiểu các cơ chế điều khiển, tuy nhiên, chẳng hạn như các giao diện người dùng và các bộ điều khiển thiết bị.

## **3. Sự kiện**

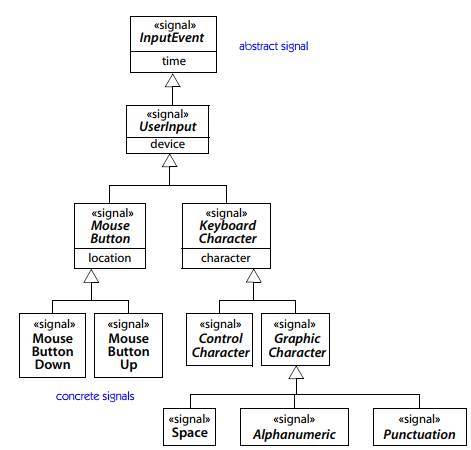
Một sự kiện là một loại xuất hiện đáng để chú ý trong thời gian và không gian. Nó xảy ra tại một thời điểm; nó không có thời gian. Mô hình hóa một thứ gì đó nếu sự kiện xảy ra có hậu quả. Một biến cố cụ thể của một sự kiện được gọi là một sự kiện. Sự kiện có thể có các tham số mô tả từng sự kiện riêng lẻ, giống như các lớp có các thuộc tính mô tả từng đối tượng. Sự kiện có thể được chia thành nhiều loại rõ ràng và tiềm ẩn: sự kiện tín hiệu, sự kiện gọi, sự kiện thay đổi và sự kiện thời gian. Bảng 7-1 là danh sách các loại sự kiện và mô tả sự kiện của chúng.

Bảng 7-1: Các loại sự kiện

|  |  |  |
| --- | --- | --- |
| Các loại sự kiện | Mô tả | Cú pháp |
| Call even (Gọi sự kiện) | Nhận một yêu cầu gọi đồng bộ rõ ràng bởi một đối tượng. |  |
| Change even (Thay đổi sự kiện) | Thay đổi giá trị của biểu thức Boolean. |  |
| Singal even (Tín hiệu sự kiện) | Biên nhận một thông tin rõ ràng, được đặt tên, không đồng bộ giữa các đối tượng. |  |
| Time even (Thời gian sự kiện) | Sự xuất hiện của một thời gian tuyệt đối hoặc một khoảng thời gian tương đối. |  |

Tín hiệu. Một tín hiệu được phân loại thiết kế rõ ràng như một phương tiện giao tiếp giữa hai đối tượng; việc tiếp nhận tín hiệu là một sự kiện cho đối tượng tiếp nhận. Đối tượng gửi, gửi một cách rõ ràng và khởi tạo một tín hiệu và gửi nó tới một hoặc một tập các đối tượng tường minh. Tín hiệu thể hiện giao tiếp một chiều không đồng bộ, loại cơ bản nhất. Người gửi không đợi người nhận xử lý tín hiệu mà vẫn tiếp tục với công việc của mình một cách độc lập. Để mô hình giao tiếp hai chiều, có thể sử dụng nhiều tín hiệu, ít nhất một tín hiệu trong mỗi hướng. Người gửi và người nhận có thể là cùng một đối tượng.

Tín hiệu có thể được khai báo trong sơ đồ lớp làm bộ phân loại, sử dụng từ khóa «signal»; các tham số của tín hiệu được khai báo là các thuộc tính. Khi phân loại, tín hiệu có thể có mối quan hệ tổng quát. Tín hiệu có thể là con của các tín hiệu khác; chúng kế thừa các thuộc tính của cha chúng, và chúng kích hoạt các hiệu ứng chuyển tiếp chứa kiểu tín hiệu gốc (Hình 7-1).



Gọi sự kiện. Một sự kiện được gọi là việc tiếp nhận cuộc gọi của một hoạt động bởi một đối tượng. Lớp nhận sẽ chọn một hoạt động sẽ được thực hiện như một phương thức hay trình kích hoạt sự kiện gọi trong máy trạng thái (hoặc có thể cả hai). Các tham số của phép toán là các tham số của sự kiện gọi. Khi đối tượng tiếp nhận xử lý sự kiện cuộc gọi bằng cách thực hiện chuyển tiếp được kích hoạt bởi sự kiện hoặc không thực hiện bất kỳ chuyển tiếp nào, kiểm soát và trả về đối tượng gọi. Tuy nhiên, không giống như một phương thức, quá trình chuyển đổi máy trạng thái được kích hoạt bởi một sự kiện gọi có thể tiếp tục thực hiện riêng của nó song song với người gọi.

Thay đổi sự kiện. Một sự kiện thay đổi là sự thỏa mãn của một biểu thức Boolean phụ thuộc vào các giá trị thuộc tính được chỉ định. Đây là một cách khai báo để đợi cho đến khi điều kiện được thỏa mãn, nhưng nó phải được sử dụng cẩn thận, vì nó thể hiện tính toán liên tục và có khả năng không cố định (hành động ở khoảng cách, vì giá trị hoặc giá trị được kiểm tra có thể ở xa). Điều này thể hiển vừa tốt vừa xấu. Nó là tốt bởi vì nó tập trung mô hình vào sự phụ thuộc thực sự — một hiệu ứng xảy ra khi một điều kiện nhất định được thỏa mãn - thay vì trên cơ chế kiểm tra điều kiện. Nó là tệ bởi vì nó che khuất mối quan hệ nhân quả giữa hành động thay đổi giá trị cơ bản và hiệu ứng cuối cùng. Chi phí thử nghiệm một sự kiện thay đổi có khả năng lớn, bởi vì về nguyên tắc nó là liên tục. Tuy nhiên, trong thực tế, có tối ưu hóa tránh tính toán không cần thiết. Thay đổi sự kiện chỉ nên được sử dụng khi một hình thức giao tiếp rõ ràng hơn là không tự nhiên.

Lưu ý sự khác biệt giữa điều kiện bảo vệ và sự kiện thay đổi. Điều kiện bảo vệ được đánh giá một lần khi sự kiện kích hoạt trong quá trình chuyển đổi xảy ra và người nhận xử lý sự kiện. Nếu sai, quá trình chuyển đổi sẽ không cháy và điều kiện không được đánh giá lại. Một sự kiện thay đổi được đánh giá liên tục cho đến khi nó trở thành sự thật, tại thời điểm mà quá trình chuyển đổi kích hoạt

Sự kiện thời gian. Các sự kiện thời gian biểu thị thời gian trôi qua. Một sự kiện thời gian có thể được xác định trong chế độ tuyệt đối (thời gian trong ngày) hoặc chế độ tương đối (thời gian trôi qua kể từ một sự kiện nhất định). Trong một mô hình cấp cao, các sự kiện thời gian có thể được coi là các sự kiện từ vũ trụ; trong một mô hình triển khai, chúng được gây ra bởi các tín hiệu từ một số đối tượng cụ thể, hoặc là hệ điều hành hoặc một đối tượng trong ứng dụng.

## **4. Trạng thái**

Một máy trạng thái định nghĩa một số lượng nhỏ các trạng thái được đặt tên. Một trạng thái có thể được mô tả theo ba cách bổ sung: như một tập hợp các giá trị đối tượng tương tự về chất lượng trong một số khía cạnh; như một khoảng thời gian trong đó một đối tượng chờ một số sự kiện hoặc sự kiện xảy ra; hoặc là một khoảng thời gian trong đó một đối tượng thực hiện một số hoạt động liên tục. Một trạng thái có thể có một cái tên, mặc dù thường nó là vô danh và được mô tả đơn giản bằng các hiệu ứng và các mối quan hệ của nó. Trong các hiệu ứng, các trạng thái là các đơn vị điều khiển mà từ đó các máy trạng thái được xây dựng.

Trong một máy trạng thái, một tập hợp các trạng thái được kết nối bằng các hiệu ứng chuyển tiếp. Một quá trình chuyển đổi kết nối hai trạng thái (hoặc nhiều hơn, nếu có một ngã ba hoặc tham gia kiểm soát). Một quá trình chuyển đổi được xử lý bởi trạng thái mà nó bỏ lại. Khi một đối tượng ở trong trạng thái, nó nhạy cảm với sự kiện kích hoạt khi chuyển tiếp rời khỏi trạng thái.

Trạng thái được hiển thị dưới dạng hình chữ nhật với các góc tròn (Hình 7-2)

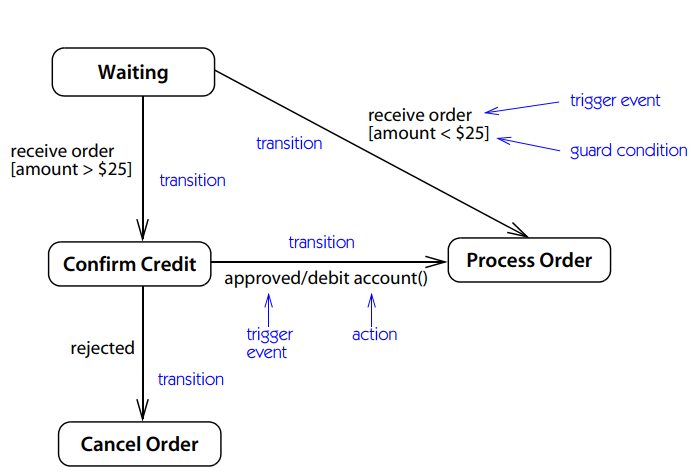


## **5. Chuyển đổi**

Một quá trình chuyển đổi để lại một trạng thái xác định đáp ứng của một đối tượng trong trạng thái đến sự xuất hiện của một sự kiện. Nói chung, quá trình chuyển đổi có kích hoạt sự kiện, điều kiện bảo vệ, hiệu ứng và trạng thái đích.

Bảng 7-2 cho thấy các loại chuyển tiếp và các hiệu ứng tiềm ẩn được gọi bởi các hiệu ứng chuyển tiếp.

|  |  |  |
| --- | --- | --- |
| Các loại chuyển đổi | Mô tả | Cú pháp |
| Entry transition | Mô tả một hoạt động nhập được thực thi khi một trạng thái nhập. |  |
| Exit transition | Mô tả một hoạt động thoát được thực hiện khi một trạng thái thoát |  |
| External transition | Phản hồi cho sự kiện mà gây ra thay đổi trạng thái hoặc tự chuyển đổi, cùng với hiệu ứng được chỉ định. Nó cũng có thể gây ra việc thực hiện các hoạt động xuất cảnh và/hoặc nhập cảnh cho các trạng thái bị thoát hoặc được nhập vào. |  |
| Internal transition | Phản hồi cho một sự kiện gây ra việc thực hiện một hiệu ứng nhưng không gây ra thay đổi trạng thái hoặc thực hiện các hoạt động xuất hoặc nhập. |  |



Chuyển đổi bên ngoài. Chuyển đổi bên ngoài là một quá trình chuyển đổi thay đổi trạng thái hoạt động. Đây là loại chuyển tiếp phổ biến nhất. Nó được rút ra như một mũi tên từ trạng thái nguồn đến trạng thái đích, với các thuộc tính khác được hiển thị như một chuỗi văn bản được gắn vào mũi tên (Hình 7-3).

Sự kiện kích hoạt. Trình kích hoạt chỉ định sự kiện cho phép chuyển đổi. Sự kiện có thể có các tham số, có sẵn cho một hiệu ứng được chỉ định như một phần của quá trình chuyển đổi. Nếu tín hiệu có tín hiệu con, việc nhận bất kỳ tín hiệu con nào của tín hiệu cho phép chuyển đổi. Ví dụ, nếu một quá trình chuyển đổi có tín hiệu MouseButton làm trigger (xem Hình 7-1), thì việc nhận MouseButtonDown sẽ kích hoạt quá trình chuyển đổi.

Một sự kiện không phải là một thứ liên tục; nó xảy ra tại một thời điểm. Khi một đối tượng nhận được một sự kiện xảy ra, sự kiện xảy ra được đặt vào một nhóm sự kiện cho đối tượng. Một đối tượng xử lý một sự kiện xảy ra cùng một lúc. Khi đối tượng là dộc lập, sự kiện xảy ra sẽ bị xóa khỏi nhóm. Quá trình chuyển đổi phải kích hoạt tại thời gian đối tượng xử lý sự kiện; sự kiện xảy ra không được "ghi nhớ" cho đến sau đó (ngoại trừ trường hợp đặc biệt của các sự kiện trì hoãn, vẫn còn trong nhóm sự kiện cho đến khi chúng kích hoạt quá trình chuyển đổi hoặc cho đến khi đối tượng đi vào trạng thái không còn trì hoãn). Nếu hai sự kiện xảy ra đồng thời, chúng được xử lý cùng một lúc. Sự kiện xảy ra không kích hoạt bất kỳ chuyển đổi nào đơn giản bị bỏ qua và bị mất. Đây không phải là một lỗi. Việc bỏ qua các sự kiện không mong muốn dễ dàng hơn nhiều so với cố gắng chỉ định tất cả chúng.

Điều kiện bảo vệ. Một quá trình chuyển đổi có thể có một điều kiện bảo vệ, đó là một biểu thức Boolean. Nó có thể tham chiếu các thuộc tính của đối tượng sở hữu máy trạng thái, cũng như các tham số của sự kiện kích hoạt. Điều kiện bảo vệ được đánh giá khi xảy ra sự kiện kích hoạt. Nếu biểu thức đánh giá là đúng, thì quá trình chuyển đổi kích hoạt— tức là, các hiệu ứng của nó xảy ra. Nếu biểu thức đánh giá là sai, thì quá trình chuyển đổi sẽ không kích hoạt. Điều kiện bảo vệ chỉ được đánh giá một lần, tại thời điểm xảy ra sự kiện kích hoạt. Nếu điều kiện là sai và sau đó trở thành sự thật, sẽ quá muộn để kích hoạt quá trình chuyển đổi.

Sự kiện tương tự có thể là kích hoạt cho nhiều chuyển đổi để lại duy nhất một trạng thái. Mỗi quá trình chuyển đổi với cùng một sự kiện phải có điều kiện bảo vệ khác nhau. Nếu sự kiện xảy ra, quá trình chuyển đổi được kích hoạt bởi sự kiện có thể kích hoạt nếu điều kiện của nó là đúng. Thông thường, tập hợp các điều kiện bảo vệ bao gồm tất cả các khả năng để sự xuất hiện của sự kiện được đảm bảo để kích hoạt một số chuyển đổi. Nếu tất cả các khả năng không được bảo hiểm và không có chuyển đổi được kích hoạt, sau đó một sự kiện chỉ đơn giản là bỏ qua. Chỉ có một quá trình chuyển đổi có thể kích hoạt (trong vòng một luồng điều khiển) để đáp ứng với một sự kiện xảy ra. Nếu một sự kiện cho phép nhiều quá trình chuyển đổi, chỉ một sự kiện chuyển tiếp. Một quá trình chuyển đổi trên một trạng thái lồng nhau được ưu tiên hơn một quá trình chuyển đổi trên một trong các trạng thái bao quanh của nó. Nếu hai quá trình chuyển đổi xung đột được bật cùng một lúc, một trong số chúng sẽ vô hiệu hóa. Sự lựa chọn có thể là ngẫu nhiên hoặc nó có thể phụ thuộc vào chi tiết thực hiện, nhưng người lập mô hình không nên dựa vào kết quả có thể dự đoán được.

Nếu một trạng thái trực giao đang hoạt động, mỗi vùng trong nó đang hoạt động, có nghĩa là nhiều trạng thái (ít nhất một trạng thái trong mỗi vùng) có thể hoạt động đồng thời. Nếu nhiều trạng thái đang hoạt động, một sự kiện xảy ra duy nhất có thể kích hoạt chuyển tiếp trong mỗi vùng trực giao. Các quá trình chuyển đổi đồng thời được thực hiện đồng thời và không tương tác, ngoại trừ gián tiếp có thể do các hiệu ứng trên các đối tượng dùng chung.

Chuyển đổi hoàn thành. Một quá trình chuyển đổi thiếu một sự kiện kích hoạt rõ ràng được kích hoạt bằng cách hoàn thành hoạt động ở trạng thái mà nó rời đi (đây là một quá trình chuyển đổi hoàn tất). Một quá trình chuyển đổi hoàn thành có thể có điều kiện bảo vệ, được đánh giá tại thời điểm hoạt động trong trạng thái hoàn thành (và không phải sau đó). Quá trình chuyển đổi hoàn thành được ưu tiên hơn các sự kiện bình thường và không đợi bước chạy bình thường chưa hoàn thành.

Hiệu ứng. Khi một đám cháy chuyển tiếp, hiệu ứng của nó (nếu có) được thực thi. Một hiệu ứng có thể là một hành động hoặc một hoạt động. Một hành động là một tính toán nguyên thủy, chẳng hạn như một câu lệnh gán hoặc tính toán số học đơn giản. Các hành động khác bao gồm gửi tín hiệu đến một đối tượng khác, gọi một phép toán, tạo hoặc hủy một đối tượng, và nhận và thiết lập các giá trị thuộc tính. Một hiệu ứng cũng có thể là một hoạt động — đó là danh sách các hành động hoặc hoạt động đơn giản hơn. Một hành động hoặc hoạt động không thể bị chấm dứt hoặc bị ảnh hưởng bởi các hiệu ứng đồng thời. (Hồ sơ nâng cao có thể thêm hành động chấm dứt hoặc gián đoạn các hoạt động khác.) Khái niệm, một hoạt động được xử lý tại

Hiệu ứng. Khi một đám cháy chuyển tiếp, hiệu ứng của nó (nếu có) được thực thi. Một hiệu ứng có thể là một hành động hoặc một hoạt động. Một hành động là một tính toán nguyên thủy, chẳng hạn như một câu lệnh gán hoặc tính toán số học đơn giản. Các hành động khác bao gồm gửi tín hiệu đến một đối tượng khác, gọi một phép toán, tạo hoặc hủy một đối tượng, và nhận và thiết lập các giá trị thuộc tính. Một hiệu ứng cũng có thể là một hoạt động — đó là danh sách các hành động hoặc hoạt động đơn giản hơn. Một hành động hoặc hoạt động không thể bị chấm dứt hoặc bị ảnh hưởng bởi các hiệu ứng đồng thời. (Hồ sơ nâng cao có thể thêm hành động chấm dứt hoặc gián đoạn các hoạt động khác.) Khái niệm, một hoạt động được xử lý tại một thời điểm; do đó, một sự kiện thứ hai không thể được xử lý trong khi thực hiện một hiệu ứng.

Hệ thống tổng thể có thể thực hiện nhiều hoạt động đồng thời. Một hệ thống triển khai có thể xử lý các ngắt phần cứng và chia sẻ thời gian giữa một số hành động. Một hoạt động không liên tục trong chuỗi điều khiển riêng của nó. Sau khi bắt đầu, nó phải được hoàn thành và nó không được tương tác với các hiệu ứng hoạt động đồng thời khác. Điều này được gọi là ngữ nghĩa run-to-completion. Nhưng không nên sử dụng hiệu ứngnhư một cơ chế giao dịch lâu dài. Thời gian của nó nên ngắn gọn so với thời gian phản hồi cần thiết cho các sự kiện bên ngoài. Nếu không, hệ thống có thể không thể để đáp ứng kịp thời.

Một hiệu ứng có thể sử dụng các tham số của sự kiện kích hoạt và các thuộc tính của đối tượng sở hữu như là một phần của biểu thức của nó.

Thay đổi trạng thái. Khi thực hiện hiệu ứng đã hoàn thành, trạng thái đích của chuyển đổi sẽ trở thành hoạt động. Điều này có thể kích hoạt một hoạt động thoát hoặc một hoạt động nhập (hoặc một số nếu máy trạng thái đi qua một số trạng thái lồng nhau từ nguồn tới trạng thái đích).

Trạng thái lồng nhau. Các trạng thái có thể được lồng trong các trạng thái hỗn hợp khác (xem mục sau). Một quá trình chuyển đổi để lại trạng thái bên ngoài được áp dụng cho tất cả các trạng thái được lồng trong nó. Quá trình chuyển đổi đủ điều kiện kích hoạt bất kỳ khi nào trạng thái lồng nhau đang hoạt động. Nếu quá trình chuyển đổi cháy, trạng thái mục tiêu của quá trình chuyển đổi trở nên hoạt động. Trạng thái hỗn hợp rất hữu ích cho việc thể hiện ngoại lệ và điều kiện lỗi, bởi vì quá trình chuyển đổi trên chúng áp dụng cho tất cả các trạng thái lồng nhau mà không cần mỗi trạng thái lồng nhau để xử lý ngoại lệ một cách rõ ràng.

Các hoạt động xuất nhập. Một quá trình chuyển đổi qua một hoặc nhiều cấp độ lồng nhau có thể thoát và vào trạng thái. Trạng thái có thể chỉ định các hoạt động được thực hiện bất cứ khi nào trạng thái được nhập hoặc thoát. Nhập trạng thái đích thực hiện một hoạt động nhập vào được gắn với trạng thái. Nếu quá trình chuyển đổi rời khỏi trạng thái ban đầu, thì hoạt động thoát của nó được thực thi trước khi hiệu ứng chuyển tiếp và hoạt động nhập vào trạng thái mới.

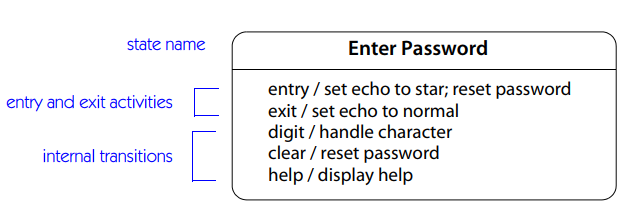
Các hoạt động nhập thường được sử dụng để thực hiện thiết lập cần thiết trong một trạng thái. Bởi vì một hoạt động nhập không thể tránh được, bất kỳ hành động nào xảy ra bên trong trạng thái đều có thể cho rằng thiết lập đã xảy ra, bất kể trạng thái được nhập vào như thế nào. Tương tự, hoạt động thoát là một hoạt động xảy ra bất cứ khi nào trạng thái được thoát, một cơ hội để thực hiện làm sạch. Nó đặc biệt hữu ích khi có các chuyển tiếp mức cao đại diện cho các điều kiện lỗi hủy bỏ các trạng thái lồng nhau. Hoạt động thoát có thể dọn dẹp các trường hợp như vậy để trạng thái của đối tượng vẫn nhất quán. Các hoạt động xuất nhập về nguyên tắc có thể gắn liền với các quá trình chuyển tiếp đến và đi, nhưng xác nhận chúng là các hiệu ứng đặc biệt của trạng thái cho phép trạng thái được xác định độc lập với các quá trình chuyển đổi và do đó được đóng gói.

Chuyển đổi nội bộ. Chuyển đổi nội bộ có trạng thái nguồn nhưng không có trạng thái đích. Các quy tắc kích hoạt cho quá trình chuyển đổi nội bộ cũng giống như quy tắc chuyển đổi thay đổi trạng thái. Chuyển đổi nội bộ không có trạng thái đích, vì vậy trạng thái hoạt động không thay đổi do kết quả của quá trình kích hoạt. Nếu quá trình chuyển đổi nội bộ có hiệu lực, hiệu ứng sẽ được thực hiện, nhưng không có sự thay đổi trạng thái nào xảy ra, và do đó không có hoạt động thoát hoặc mục nhập nào được thực thi. Chuyển đổi nội bộ hữu ích cho việc tạo mô hình các tình huống gián đoạn không thay đổi trạng thái (chẳng hạn như tính số lần xuất hiện của một sự kiện hoặc đưa ra trợ giúp trên màn hình).

Các hoạt động nhập và xuất sử dụng ký hiệu tương tự như chuyển tiếp nội bộ, ngoại trừ việc sử dụng các từ dành riêng và thoát ra thay cho tên trình kích hoạt sự kiện, mặc dù các hiệu ứng này được kích hoạt bởi các chuyển tiếp bên ngoài vào hoặc ra khỏi trạng thái.

Tự chuyển đổi gọi các hoạt động xuất cảnh và nhập cảnh vào trạng thái của nó (khái niệm, nó thoát ra và sau đó nhập lại vào trạng thái); do đó, nó không tương đương với một quá trình chuyển đổi nội bộ. Hình 7-4 cho thấy các hoạt động nhập cảnh và thoát cũng như các chuyển đổi nội bộ.

Hình 7-4. Chuyển tiếp nội bộ và các hành động nhập và thoát



## **6. Trạng thái hỗn hợp**

Một trạng thái đơn giản không có cấu trúc con, chỉ là một tập hợp các chuyển tiếp và các hoạt động vào và ra có thể. Trạng thái hỗn hợp là trạng thái được phân tách thành các vùng, mỗi vùng chứa một hoặc nhiều trạng thái trực tiếp. Bảng 7-3 liệt kê các loại trạng thái.

Sự phân tách của một trạng thái phi chính thống thành các trạng thái trực tiếp tương tự như sự chuyên hóa của một lớp. Một trạng thái bên ngoài bị phân tách thành nhiều trạng thái bên trong, mỗi trạng thái kế thừa các chuyển tiếp của trạng thái bên ngoài. Chỉ có một trạng thái trực tiếp cho mỗi trạng thái phi chính thống có thể hoạt động cùng một lúc. Trạng thái bên ngoài đại diện cho tình trạng ở trong bất kỳ trạng thái bên trong nào.

Chuyển tiếp vào hoặc ra khỏi trạng thái hỗn hợp gọi bất kỳ hoạt động nhập hoặc thoát khỏi hoạt động của trạng thái. Nếu có một số trạng thái hỗn hợp, quá trình chuyển đổi giữa nhiều cấp có thể gọi nhiều hoạt động nhập (ngoài cùng bên ngoài) hoặc một số hoạt động thoát (trong cùng đầu tiên). Nếu có ảnh hưởng đến bản thân quá trình chuyển đổi, hiệu ứng sẽ được thực thi sau bất kỳ hoạt động thoát nào và trước khi bất kỳ hoạt động nhập nào được thực thi

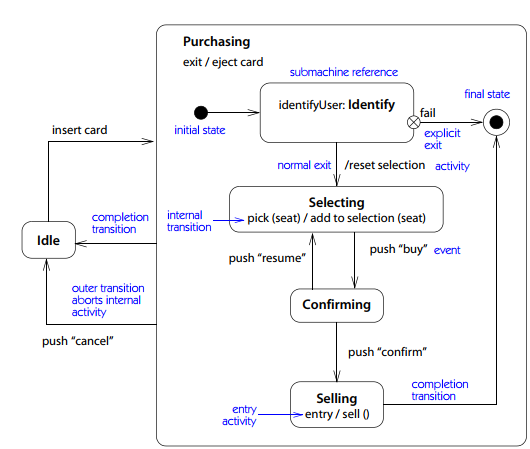
|  |  |  |
| --- | --- | --- |
| Các loại trạng thái | Mô tả | Ký hiệu |
| Simple state | Trạng thái không có cấu trúc con. |  |
| Orthogonal state | Một trạng thái được chia thành hai hoặc nhiều vùng. Một trạm trạng thái trực tiếp từ mỗi vùng đồng thời hoạt động khi trạng thái hỗn hợp đang hoạt động. |  |
| Nonorthogonal  state | Trạng thái hỗn hợp chứa một hoặc nhiều trạng thái trực tiếp, chính xác một trong số đó hoạt động cùng một lúc khi trạng thái hỗn hợp đang hoạt động |  |
| Initial state | Một trạng thái giả cho biết trạng thái bắt đầu khi trạng thái bao quanh được gọi. |  |
| Final state | Trạng thái đặc biệt có kích hoạt cho biết trạng thái kèm theo đã hoàn thành hoạt động. |  |
| Terminate | Trạng thái đặc biệt có kích hoạt chấm dứt thực hiện đối tượng sở hữu máy trạng thái. |  |
| Junction | Trạng thái giả tạo chuỗi các phân đoạn chuyển đổi thành một quá trình chuyển đổi hoàn thành chạy đến sự chuyển đổi hoàn thành |  |
| Choice | Trạng thái giả thực hiện chia nhánh động trong một quá trình chuyển đổi chạy hoàn thành. |  |
| History state | Trạng thái giả có kích hoạt khôi phục trạng thái hoạt động trước đó trong trạng thái hỗn hợp |  |
| Submachine state | Một trạng thái tham chiếu đến định nghĩa máy trạng thái, thay thế khái niệm trạng thái con của trạng thái. |  |
| Entry point | Trạng thái giả hiển thị bên ngoài trong máy trạng thái xác định trạng thái bên trong làm mục tiêu. |  |
| Exit point | Trạng thái giả hiển thị bên ngoài trong máy trạng thái xác định trạng thái nội bộ dưới dạng nguồn |  |

Bảng 7-3: Các loại trạng thái

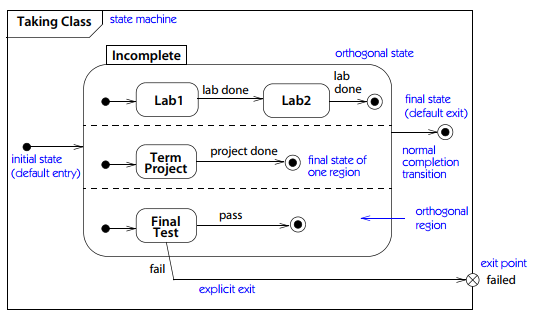
Mỗi vùng của trạng thái hỗn hợp có thể có trạng thái ban đầu. Chuyển đổi sang ranh giới trạng thái hỗn hợp là ngầm chuyển tiếp sang trạng thái ban đầu. Một đối tượng mới bắt đầu ở trạng thái ban đầu của trạng thái ngoài cùng của nó. Tương tự, trạng thái hỗn hợp có thể có trạng thái cuối cùng. Việc chuyển đổi sang trạng thái cuối cùng sẽ kích hoạt quá trình chuyển đổi hoàn thành trên trạng thái hỗn hợp. Nếu một đối tượng đến trạng thái cuối cùng của trạng thái ngoài cùng nhất, nó sẽ bị phá hủy. Các trạng thái ban đầu, các trạng thái cuối cùng, các hoạt động nhập cảnh và các hoạt động thoát cho phép định nghĩa trạng thái được đóng gói độc lập với các chuyển tiếp đến và đi từ nó.

Hình 7-5 cho thấy sự phân tách tuần tự của một trạng thái, bao gồm một trạng thái ban đầu. Đây là ví dụ cho điều khiển cho một máy bán vé. Sự phân tách của một trạng thái trực giao thành các vùng trực giao biểu diễn tính toán độc lập. Khi một trạng thái trực giao được nhập vào, số lượng các chuỗi điều khiển tăng lên như một trạng thái trực tiếp trong mỗi vùng trực giao trở thành hoạt động. Khi trạng thái trực giao được thoát, số lượng chuỗi điều khiển giảm. Thông thường, sự tương qian được thực hiện bởi một đối tượng riêng biệt cho mỗitrạgn thái, nhưng các trạng thái trực giao cũng có thể biểu diễn sự đồng thời hợp lý trong một đối tượng duy nhất. Hình 7-6 cho thấy sự phân tách đồng thời của việc tham gia một lớp đại học.

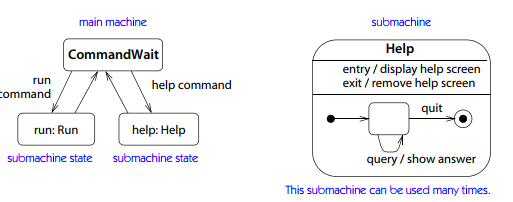
Nó thường thuận tiện để tái sử dụng một mảnh của một máy trạng thái trong các máy nhà nước khác. Một máy trạng thái có thể được đặt tên và được tham chiếu từ trạng thái của một hoặc nhiều máy khác. Máy trạng thái đích là một máy con, và trạng thái tham chiếu nó được gọi là trạng thái submachine. Nó ngụ ý sự thay thế (khái niệm) của một bản sao của máy trạng thái được tham chiếu tại nơi tham chiếu, một loại chương trình con máy trạng thái. Một máy con có thể định nghĩa các điểm vào được đặt tên và các điểm thoát kết nối với các trạng thái bên trong. Chuyển tiếp sang trạng thái tiểu liên có thể sử dụng các điểm kết nối tham chiếu các điểm vào và thoát này, ẩn đi cấu trúc bên trong của máy con từ các máy khách bên ngoài. Thay vì một máy phụ, một trạng thái có thể chứa một hoạt động - đó là, một tính toán hoặc hoạt động liên tục cần có thời gian để hoàn thành và có thể bị gián đoạn bởi các sự kiện. Hình 7-7 cho thấy một tham chiếu máy phụ.



Một quá trình chuyển đổi sang trạng thái máy phụ sẽ làm kích hoạt trạng thái ban đầu của máy con mục tiêu. Để nhập một máy con ở các trạng thái khác, một điểm vào có thể được tham chiếu. Một điểm vào xác định một trạng thái trong máy phụ mà không lộ các nội dung của máy phụ.



Hình 7-6. Máy trạng thái với trạng thái hỗn hợp trực giao



# **CHƯƠNG 3: STATIC VIEW**

## **1. Tổng quan**

Static view là nền tảng của UML. Các phần tử của static view của một mô hình là các khái niệm có ý nghĩa trong một ứng dụng, bao gồm các khái niệm thế giới thực, các khái niệm trừu tượng, các khái niệm thực hiện, các khái niệm máy tính — tất cả các loại khái niệm được tìm thấy trong các hệ thống. Ví dụ, một hệ thống vé cho một nhà hát có các khái niệm như vé, đặt chỗ, kế hoạch thuê bao, nhiệm vụ chuyển nhượng chỗ ngồi, các trang web tương tác để đặt hàng và dữ liệu lưu trữ để dự phòng.

Static view nắm bắt cấu trúc đối tượng. Một hệ thống hướng đối tượng hợp nhất cấu trúc dữ liệu và các tính năng hành vi thành một cấu trúc đối tượng duy nhất. Static view bao gồm tất cả các mối quan tâm về cấu trúc dữ liệu truyền thống, cũng như tổ chức các hoạt động trên dữ liệu. Cả dữ liệu và hoạt động được lượng hóa thành các lớp. Trong phối cảnh hướng đối tượng, dữ liệu và hành vi có liên quan chặt chẽ. Ví dụ, một đối tượng Ticket mang dữ liệu, chẳng hạn như giá, ngày hiệu suất và số chỗ ngồi, cũng như các hoạt động trên đó, chẳng hạn như tự đặt trước hoặc tính giá của nó với mức giảm giá đặc biệt.

Static view mô tả các khai báo hành vi, chẳng hạn như các hoạt động, như các phần tử mô hình riêng biệt, nhưng nó không chứa các chi tiết về hành vi động của chúng. Nó đối xử với họ như những thứ được đặt tên, sở hữu bởi các lớp học, và được gọi. Việc thực thi động của chúng được mô tả bởi các khung nhìn khác mô tả chi tiết nội bộ của các thuốc nhuộm của chúng. Các static view khác này bao gồm chế độ xem tương tác và chế độ xem máy trạng thái. Chế độ xem động yêu cầu chế độ xem tĩnh để mô tả những thứ tương tác động - bạn không thể nói điều gì đó tương tác mà không nói điều gì đang tương tác. Static view là nền tảng mà trên đó các khung nhìn khác được xây dựng.

Các phần tử chính trong static view là các trình phân loại và mối quan hệ của chúng. Trình phân loại là phần tử mô hình mô tả những thứ có chứa giá trị. Có một số loại phân loại, bao gồm các lớp, giao diện và các kiểu dữ liệu. Những thứ hành vi, chẳng hạn như trường hợp sử dụng và tín hiệu, cũng được coi là bộ phân loại. Các mục đích triển khai thực hiện phía sau một số trình phân loại, chẳng hạn như các thành phần, các hợp tác và các nút.

Các mô hình lớn phải được tổ chức thành các đơn vị nhỏ hơn để hiểu và sử dụng lại con người. Một gói là một đơn vị tổ chức có mục đích chung để sở hữu và quản lý nội dung của một mô hình. Mỗi phần tử được sở hữu bởi một số gói. Mô hình là một tập các gói mô tả một khung nhìn hoàn chỉnh của một hệ thống và có thể được sử dụng nhiều hay ít độc lập với các mô hình khác; nó chỉ định một gói gốc gián tiếp sở hữu các gói mô tả hệ thống.

Một đối tượng là một đơn vị rời rạc mà từ đó người lập mô hình hiểu và xây dựng một hệ thống. Nó là một thể hiện của một lớp - đó là, một cá nhân với danh tính có cấu trúc và hành vi được mô tả bởi lớp. Một đối tượng có một phần trạng thái có thể nhận dạng được với hành vi được xác định rõ ràng có thể được gọi.

Mối quan hệ giữa các trình phân loại là liên kết, khái quát hóa và các loại phụ thuộc khác nhau, bao gồm việc thực hiện và sử dụng.

## **2. Trình phân loại**

Một trình phân loại mô hình hóa một khái niệm rời rạc mô tả mọi thứ (các đối tượng) có iden- tity, state, behavior, các mối quan hệ và một cấu trúc bên trong tùy chọn. Các loại phân loại bao gồm lớp, giao diện và kiểu dữ liệu. Các loại phân loại khác là các khái niệm hành vi, những thứ trong môi trường hoặc cấu trúc triển khai. Các phân loại này bao gồm trường hợp sử dụng, diễn viên, cộng tác, thành phần và nút, cũng như các loại hành vi khác nhau. Bảng 4-1 liệt kê các loại bộ tách và chức năng của chúng. Phân loại thuật ngữ metamodel bao gồm tất cả các điều khoản này, nhưng vì lớp là thuật ngữ quen thuộc nhất, chúng ta sẽ thảo luận nó trước và định nghĩa các khái niệm khác bằng sự khác biệt từ nó.

**Table 4-1:** *Kinds of Classifier**s*

|  |  |  |
| --- | --- | --- |
| *Classifier* | *Function* | *Notation* |
| [actor](#_bookmark528) | Một người dùng bên ngoài hệ thống |  |
| [artifact](#_bookmark565) | Một phần thông tin hệ thống vật lý | «artifact»  Name |
| [class](#_bookmark690) | Một khái niệm từ hệ thống mô hình hóa | Name |

**Table 4-1:** *Kinds of Classifiers (continued)*

|  |  |  |
| --- | --- | --- |
| *Classifier* | *Function* | *Notation* |
| [collaboration](#_bookmark734) | Một mối quan hệ theo ngữ cảnh giữa các đối tượng đang đóng vai trò | Name |
| [component](#_bookmark787) | Một phần mô-đun của một hệ thống có giao diện được xác định rõ ràng | Name |
| [enumeration](#_bookmark1076) | Kiểu dữ liệu với các giá trị được xác định trước bằng chữ | «enumeration»  Name |
| [primitive type](#_bookmark1731) | Bộ mô tả một tập hợp các giá trị nguyên thủy thiếu bản sắc | Name |
| [interface](#_bookmark1336) | A named set of operations that characterize behavior | «interface»  Name |
| [node](#_bookmark1537) | Một tập hợp các hoạt động có tên mô tả hành vi |  |
| [role](#_bookmark1876) | Phần nội bộ trong ngữ cảnh của một bộ cộng tác hoặc phân loại có cấu trúc | role:Name |
| [signal](#_bookmark1937) | Giao tiếp không đồng bộ giữa các đối tượng | «signal»  Name |
| [structured](#_bookmark2061) [classifier](#_bookmark2061) | Trình phân loại có cấu trúc bên trong |  |
| [use case](#_bookmark2226) | Một đặc tả của hành vi của một thực thể trong sự tương tác của nó với các tác nhân bên ngoài |  |

Class. Một lớp đại diện cho một khái niệm rời rạc trong ứng dụng được mô hình hóa đại diện cho những thứ của một loại cụ thể - một vật thể (chẳng hạn như một chiếc máy bay), một thứ kinh doanh (như một đơn đặt hàng), một thứ hợp lý (chẳng hạn như lịch phát sóng), điều ứng dụng (chẳng hạn như nút hủy), một điều máy tính (chẳng hạn như bảng băm), hoặc một điều hành vi (chẳng hạn như một nhiệm vụ). Một lớp là bộ mô tả cho một tập hợp các đối tượng có cấu trúc, hành vi và các mối quan hệ tương tự. Tất cả các thuộc tính và các phép toán được gắn vào các lớp hoặc các bộ phân loại khác. Các lớp là các phân đoạn xung quanh các hệ thống hướng đối tượng được tổ chức.

Một đối tượng là một thực thể rời rạc với nhận dạng, trạng thái và hành vi bất khả xâm phạm. Đối tượng là các phần riêng lẻ trong một hệ thống thời gian chạy; các lớp là các khái niệm riêng lẻ để hiểu và mô tả vô số các đối tượng riêng lẻ.

Một lớp định nghĩa một tập các đối tượng có trạng thái và hành vi. Nhà nước được mô tả bởi các thuộc tính và các hiệp hội. Các thuộc tính thường được sử dụng cho các giá trị dữ liệu thuần túy không có nhận dạng, chẳng hạn như số và chuỗi và các liên kết được sử dụng cho các kết nối giữa các đối tượng có nhận dạng. Các phần riêng lẻ của hành vi bất khả xâm phạm được mô tả bởi các hoạt động; một phương pháp là việc thực hiện một hoạt động. Lịch sử lâu dài của một đối tượng được mô tả bởi một máy trạng thái gắn liền với một lớp. Ký hiệu cho một lớp là một hình chữ nhật có các ngăn cho tên của lớp, các phép tính và các phép toán, như trong Hình 4-1.

Một tập hợp các lớp có thể sử dụng mối quan hệ khái quát hóa và cơ chế thừa kế được xây dựng trên nó để chia sẻ các phần chung của mô tả trạng thái và hành vi. Tổng quát liên quan đến các lớp cụ thể hơn (các lớp con) cho các lớp tổng quát hơn (các lớp siêu lớp) có chứa các thuộc tính chung cho một số lớp con. Một lớp học có thể có không hoặc nhiều phụ huynh (superclasses) và số không hoặc nhiều trẻ em (subclasses). Một lớp kế thừa các mô tả về trạng thái và hành vi từ cha mẹ và các tác nhân khác, và nó định nghĩa các mô tả trạng thái và hành vi mà con cái của nó và các hậu duệ khác kế thừa.

Một lớp có một tên duy nhất trong vùng chứa của nó, thường là một gói nhưng đôi khi là một lớp khác. Lớp có khả năng hiển thị đối với vùng chứa của nó; khả năng hiển thị chỉ định cách nó có thể được sử dụng bởi các lớp khác bên ngoài vùng chứa.

|  |
| --- |
| Subscription |
| series: String priceCategory: Category number: Integer |
| cost (): Money  reserve (series: String, level: SeatLevel) cancel () |

class name

attributes

operations

**Interface**. interface là mô tả về hành vi của các đối tượng mà không cần thực hiện hoặc trạng thái của chúng. Một hoặc nhiều lớp hoặc thành phần có thể nhận ra một interface và mỗi lớp thực hiện các hoạt động được tìm thấy trong interface.

**Data types**. Một kiểu nguyên thủy là mô tả về các giá trị nguyên thuỷ thiếu bản sắc (sự tồn tại độc lập và khả năng tác dụng phụ). Các loại nguyên thủy bao gồm số và chuỗi. Các kiểu nguyên thủy được truyền theo giá trị và là các đặc tính bất biến. Kiểu nguyên thủy không có thuộc tính nhưng có thể có các phép toán. Các thao tác không sửa đổi các giá trị dữ liệu, nhưng chúng có thể trả về các giá trị dữ liệu dưới dạng kết quả.

Mô hình người dùng cũng có thể khai báo các kiểu liệt kê. Một kiểu liệt kê khai báo một tập hợp các số đếm liệt kê có thể được sử dụng làm giá trị.

***Levels of meaning***. Các lớp có thể tồn tại ở nhiều mức ý nghĩa trong một mô hình, bao gồm các mức phân tích, thiết kế và triển khai. Khi mô tả các khái niệm trong thế giới thực, điều quan trọng là nắm bắt trạng thái, các mối quan hệ và hành vi của thế giới thực. Nhưng các khái niệm thực hiện, chẳng hạn như ẩn thông tin, hiệu quả, khả năng hiển thị và các phương thức, không phải là các khái niệm thực tế có liên quan. Chúng là những thiết kế có liên quan. Nhiều tính chất tiềm năng của một lớp đơn giản là không liên quan ở cấp độ phân tích. Lớp phân tích cấp đại diện cho một khái niệm logic trong miền ứng dụng hoặc trong chính ứng dụng. Mô hình phân tích phải là một đại diện tối thiểu của hệ thống được mô hình hoá, đủ để nắm bắt được logic thiết yếu của hệ thống mà không gặp phải các vấn đề về hiệu năng hoặc xây dựng.

Khi đại diện cho một thiết kế cấp cao, các khái niệm như nội địa hóa của các lớp cụ thể, hiệu quả điều hướng giữa các đối tượng, phân tách hành vi bên ngoài và thực hiện nội bộ, và đặc tả của các hoạt động chính xác có liên quan đến một lớp. Một lớp thiết kế cấp đại diện cho quyết định đóng gói thông tin trạng thái và các hoạt động trên nó thành một đơn vị rời rạc. Nó nắm bắt các quyết định thiết kế quan trọng, nội địa hoá thông tin và chức năng cho các đối tượng. Các lớp cấp thiết kế chứa cả nội dung trong thế giới thực và nội dung hệ thống máy tính.

Cuối cùng, khi biểu diễn mã ngôn ngữ lập trình, biểu mẫu của một lớp khớp chặt chẽ với ngôn ngữ lập trình đã chọn và một số khả năng của một lớp chung có thể được tha thứ nếu chúng không thực hiện trực tiếp ngôn ngữ. Một lớp cấp thực hiện ánh xạ trực tiếp vào mã ngôn ngữ lập trình.

Cùng một hệ thống có thể được mô hình hoá từ nhiều quan điểm, chẳng hạn như một mô hình logic thu thập thông tin trong thế giới thực và một mô hình thiết kế nắm bắt các quyết định đại diện nội bộ. Một lớp định hướng thực hiện có thể nhận ra một lớp logic từ một mô hình khác. Một lớp logic nắm bắt các thuộc tính và các mối quan hệ của các thông tin trong thế giới thực. Một lớp thực hiện đại diện cho sự giải mã của một lớp như được tìm thấy trong một ngôn ngữ lập trình cụ thể. Nó nắm bắt hình thức chính xác của một lớp, khi cần thiết bởi ngôn ngữ. Tuy nhiên, trong nhiều trường hợp, thông tin phân tích, thiết kế và triển khai có thể được lồng vào một lớp duy nhất.

Relationships among classifiers are association, generalization, and various kinds of dependency, including realization and usage (see Table 4-2)

## **3. Relationship**

Mối quan hệ giữa các bộ phân loại là sự kết hợp, khái quát hóa và các loại phụ thuộc khác nhau, bao gồm việc thực hiện và sử dụng (xem Bảng 4-2).

**Table 4-2:** *Kinds of Relationship**s*

|  |  |  |
| --- | --- | --- |
| *Relationship* | *Function* | *Notation* |
| [association](#_bookmark576) | Mô tả kết nối giữa các phiên bản của các lớp |  |
| [dependency](#_bookmark967) | Mối quan hệ giữa hai yếu tố mô hình |  |
| [generalization](#_bookmark1191) | Mối quan hệ giữa mô tả cụ thể hơn và tổng quát hơn, được sử dụng cho các khai báo kiểu thừa kế và đa hình |  |
| [realization](#_bookmark1798) | Mối quan hệ giữa đặc điểm kỹ thuật và triển khai của nó |  |
| [usage](#_bookmark2222) | Một tình huống trong đó một phần tử yêu cầu một phần tử khác cho hoạt động chính xác của nó | «kind» |

Mối quan hệ association mô tả các kết nối ngữ nghĩa giữa các đối tượng riêng lẻ của các lớp đã cho. Các hiệp hội cung cấp các kết nối mà các đối tượng của các lớp khác nhau có thể tương tác. Các mối quan hệ còn lại liên quan đến các mô tả của các trình phân loại, không phải là các cá thể của chúng.  
Mối quan hệ generalization liên quan đến các mô tả chung về các phân loại cha (các lớp cha) cho các bộ phân loại con chuyên biệt hơn (các lớp con). Sự tổng quát hóa mô tả các bộ phân loại ra khỏi các phần khai báo gia tăng, mỗi phần bổ sung cho mô tả được kế thừa từ tổ tiên của nó. Cơ chế thừa kế xây dựng mô tả đầy đủ các bộ phân loại từ các mô tả gia tăng sử dụng các mối quan hệ tổng quát hóa. Tổng quát và thừa kế cho phép các trình phân loại khác nhau chia sẻ các thuộc tính, hoạt động và các mối quan hệ mà chúng có chung, không lặp lại.  
Mối quan hệ realization liên quan đến một đặc điểm kỹ thuật để thực hiện. Giao diện là một đặc tả của hành vi mà không cần thực hiện; một lớp bao gồm cấu trúc thực hiện. Một hoặc nhiều lớp có thể nhận ra một giao diện và mỗi lớp thực hiện các hoạt động được tìm thấy trong giao diện.

Mối quan hệ dependency liên quan đến các lớp có hành vi hoặc triển khai ảnh hưởng đến các lớp khác. Có một số loại phụ thuộc ngoài việc thực hiện, bao gồm phụ thuộc vết (kết nối lỏng lẻo giữa các phần tử trong các mô hình khác nhau), sàng lọc (ánh xạ giữa hai mức ý nghĩa), cách sử dụng (yêu cầu cho sự hiện diện của phần tử khác) trong một mô hình duy nhất) và ràng buộc (gán giá trị cho các tham số mẫu). Mức độ phụ thuộc sử dụng thường được sử dụng để biểu diễn các mối quan hệ triển khai, chẳng hạn như các mối quan hệ cấp mã. Phụ thuộc đặc biệt hữu ích khi được tóm tắt trên các đơn vị tổ chức mô hình, chẳng hạn như các gói, trên đó nó cho thấy cấu trúc kiến trúc của một hệ thống. Ví dụ, các ràng buộc về biên dịch có thể được hiển thị bởi các phụ thuộc.

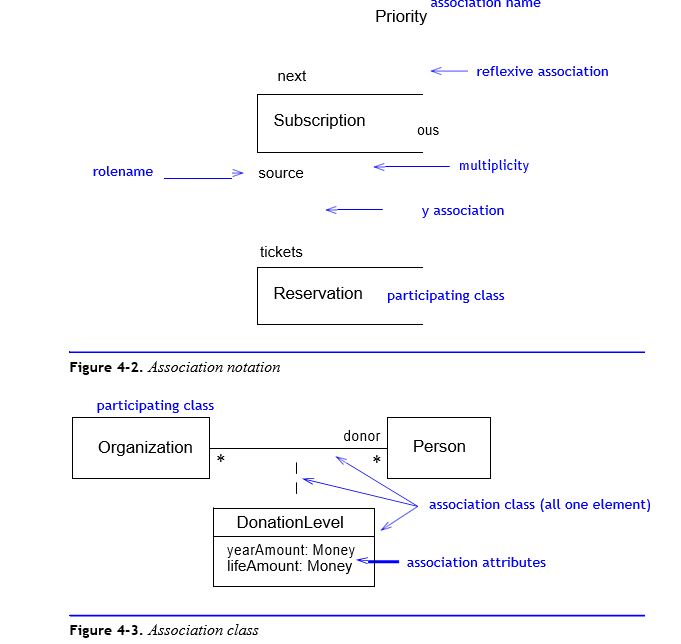
## **4. Association**

Một Association mô tả các kết nối rời rạc giữa các đối tượng hoặc các cá thể khác trong một hệ thống. Một Association liên quan đến một danh sách theo thứ tự (tuple) của hai hoặc nhiều phân loại, với sự lặp lại cho phép. Loại liên kết phổ biến nhất là sự phân phối nhị phân giữa một cặp phân loại. Một thể hiện của một Association là một liên kết. Một liên kết bao gồm một tuple (một danh sách thứ tự) của các đối tượng, mỗi đối tượng được rút ra từ lớp tương ứng của nó. Một liên kết nhị phân bao gồm một cặp đối tượng.

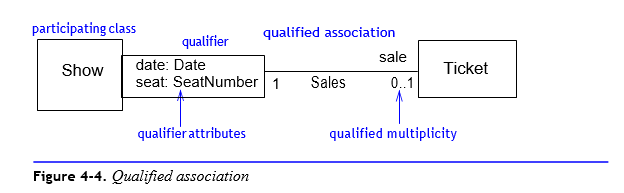
Các Association mang thông tin về các mối quan hệ giữa các đối tượng trong một hệ thống. Khi hệ thống thực thi, các liên kết giữa các đối tượng được tạo và hủy. Các hiệp hội là "keo" gắn kết một hệ thống với nhau. Không có hiệp hội, không có gì ngoài các lớp riêng biệt không hoạt động cùng nhau.  
Một đối tượng có thể được liên kết với chính nó nếu cùng một lớp xuất hiện nhiều lần trong một liên kết. Nếu cùng một lớp xuất hiện hai lần trong một Association, hai trường hợp không phải là cùng một đối tượng, và thường thì chúng không phải là.

Mỗi kết nối của một liên kết đến một lớp được gọi là một kết thúc kết hợp. Hầu hết thông tin về một Association được gắn vào một trong các kết thúc của nó. Association kết thúc có thể có tên (rolenames) và khả năng hiển thị. Thuộc tính quan trọng nhất mà chúng có là bội số - có bao nhiêu trường hợp của một lớp có thể liên quan đến một trường hợp của lớp khác. Đa dạng là hữu ích nhất cho các hiệp hội nhị phân vì định nghĩa của nó cho các liên kết n-ary phức tạp. Nếu giới hạn trên bội số lớn hơn một, thứ tự và tính duy nhất của các giá trị liên quan có thể được xác định trên đầu kết hợp.

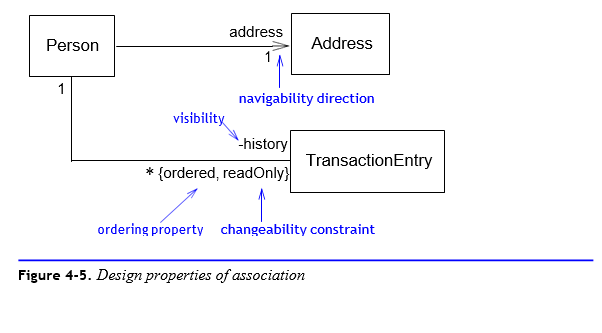
Ký hiệu cho một liên kết nhị phân là một dòng hoặc đường dẫn kết nối các lớp tham gia. Tên hiệp hội được đặt dọc theo dòng với rolename và đa dạng ở mỗi đầu, như trong Hình 4-2.



Một Association cũng có thể có các thuộc tính của riêng nó, trong trường hợp đó nó vừa là một phân ly và một lớp - một lớp kết hợp (xem Hình 4-3). Nếu giá trị của một thuộc tính asso- ciation là duy nhất trong một tập hợp các đối tượng liên quan, thì nó là một bộ định tính và liên kết là một liên kết đủ điều kiện (xem Hình 4-4). Vòng loại là một giá trị chọn một đối tượng duy nhất từ tập hợp các đối tượng liên quan trên một liên kết. Bảng tra cứu và mảng có thể được mô hình hóa dưới dạng các liên kết đủ điều kiện. Vòng loại quan trọng cho việc lập mô hình tên và mã nhận dạng. Các vòng loại cũng mô hình hóa các chỉ mục trong một mô hình thiết kế.

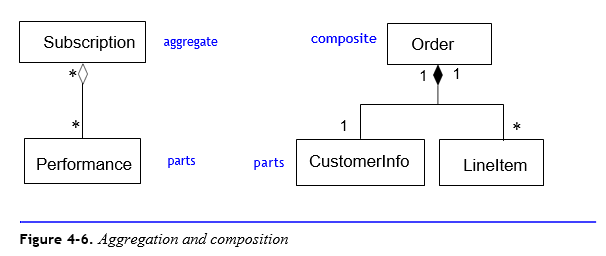


Trong quá trình phân tích, các association đại diện cho các mối quan hệ hợp lý giữa các đối tượng. Không có nhu cầu lớn để áp đặt hướng hoặc quan tâm về cách thực hiện chúng. Cần tránh các liên kết thừa vì chúng không thêm thông tin logic. Trong quá trình thiết kế, các hiệp hội nắm bắt các quyết định thiết kế về cấu trúc dữ liệu, cũng như phân chia trách nhiệm giữa các lớp học. Định hướng của các association là quan trọng, và các association dự phòng có thể được bao gồm cho hiệu quả truy cập đối tượng, cũng như để bản địa hóa thông tin trong một lớp cụ thể. Tuy nhiên, ở giai đoạn này của mô hình hóa, các association không nên được so sánh với con trỏ C ++. Một liên kết điều hướng ở giai đoạn thiết kế đại diện cho thông tin trạng thái có sẵn cho một lớp, nhưng nó có thể được ánh xạ thành mã ngôn ngữ lập trình theo nhiều cách khác nhau. Việc thực hiện có thể là một con trỏ, một lớp container được nhúng trong một lớp, hoặc thậm chí một đối tượng bảng hoàn toàn riêng biệt. Các loại tài sản thiết kế khác bao gồm khả năng hiển thị và khả năng thay đổi của các liên kết. Hình 4-5 cho thấy một số mối quan hệ thiết kế của các association.



## **5. Aggregation and composition.**

Một tập hợp là một association đại diện cho một mối quan hệ toàn bộ. Nó được thể hiện bằng một trang sức bằng kim cương rỗng ở cuối con đường gắn liền với lớp tổng hợp. Thành phần là một hình thức liên kết mạnh mẽ hơn trong đó tổng hợp có trách nhiệm duy nhất để quản lý các bộ phận của nó, chẳng hạn như phân bổ và phân bổ. Nó được thể hiện bằng một trang trí đầy kim cương trên đầu hỗn hợp. Một đối tượng có thể thuộc nhiều nhất một thành phần. Có một liên kết riêng biệt giữa mỗi lớp đại diện cho một phần và lớp đại diện cho toàn bộ, nhưng để thuận tiện, các đường dẫn gắn liền với toàn bộ có thể được nối với nhau để toàn bộ các liên kết được vẽ như một cây. Hình 4-6 cho thấy một tổng hợp và một hỗn hợp.



## **6. Liên kết**.

Một thể hiện của một association là một liên kết. Một liên kết là một danh sách có thứ tự (tuple) của các tham chiếu, mỗi trong số đó phải là một thể hiện của lớp tương ứng trong liên kết hoặc một thể hiện của một hậu duệ của lớp đó. Các liên kết trong một hệ thống cấu thành một phần của trạng thái hệ thống. Các liên kết không tồn tại độc lập với các đối tượng; họ lấy danh tính của họ từ các đối tượng họ liên quan (trong điều kiện cơ sở dữ liệu, danh sách các đối tượng là chìa khóa cho liên kết). Trong trường hợp của túi, có thể có nhiều liên kết tương ứng với một tuple của các đối tượng, tuy nhiên. Về mặt khái niệm, một hiệp hội khác với các lớp mà nó liên quan. Trong thực tế, các hiệp hội thường được thực hiện bằng cách sử dụng các điểm trong các lớp tham gia, nhưng chúng có thể được thực hiện như các đối tượng container tách biệt với các lớp mà chúng kết nối.

## **7. Bidirectionality.**

Các kết thúc khác nhau của một liên kết có thể phân biệt được, ngay cả khi hai trong số chúng liên quan đến cùng một lớp. Điều này đơn giản có nghĩa là các đối tượng khác nhau của cùng một lớp có thể liên quan. Bởi vì các đầu có thể phân biệt được, kết hợp không đối xứng (trừ trường hợp đặc biệt); các đầu không thể thay đổi được. Trong bài diễn văn chính thức, đây chỉ là ý nghĩa thông thường; đối tượng và đối tượng của động từ không thể hoán đổi cho nhau. Một hiệp hội đôi khi được cho là hai chiều. Điều này có nghĩa là các mối quan hệ hợp lý hoạt động theo cả hai cách. Tuyên bố này thường xuyên hiểu lầm, ngay cả bởi một số nhà phương pháp luận. Nó không có nghĩa là mỗi lớp "biết" lớp khác, hoặc trong một thực thi, có thể truy cập mỗi lớp từ lớp kia. Nó đơn giản có nghĩa là bất kỳ mối quan hệ logic nào có nghịch đảo, có hay không nghịch đảo là dễ tính toán. Để khẳng định khả năng đi qua một hiệp hội theo một hướng nhưng không phải là một quyết định thiết kế khác, các hiệp hội có thể được đánh dấu bằng khả năng điều hướng.

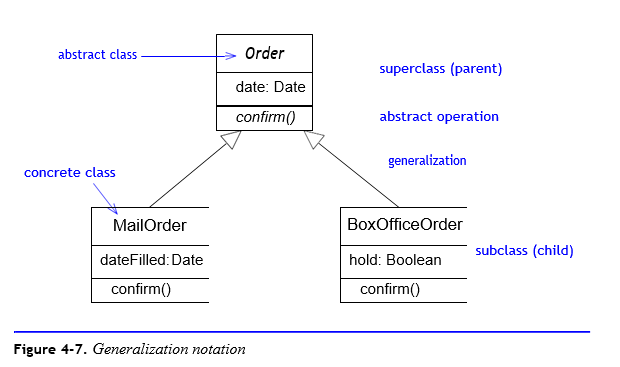
Tại sao mô hình cơ bản lại quan hệ, thay vì mô hình con trỏ phổ biến trong các ngôn ngữ lập trình? Lý do là một mô hình cố gắng nắm bắt ý định đằng sau việc thực hiện. Nếu một mối quan hệ giữa hai lớp được mô hình hóa như một cặp con trỏ, thì các con trỏ vẫn có liên quan. Cách tiếp cận của hiệp hội đã chứng minh rằng các mối quan hệ có ý nghĩa trong cả hai hướng, bất kể chúng được thực hiện như thế nào. Thật đơn giản để chuyển đổi một liên kết thành một cặp con trỏ để thực hiện, nhưng rất khó để nhận ra rằng hai con trỏ là nghịch đảo của nhau trừ khi thực tế này là một phần của mô hình.

## **8. Generalization**

Mối quan hệ Generalization là một mối quan hệ phân loại giữa một mô tả chi tiết hơn và một mô tả cụ thể hơn được xây dựng trên nó và mở rộng nó. Mô tả cụ thể hơn hoàn toàn phù hợp với mô tả chung hơn (nó có tất cả các thuộc tính, thành viên và mối quan hệ của nó) và có thể chứa thông tin bổ sung. Ví dụ, thế chấp là một loại khoản vay cụ thể hơn. Một thế chấp giữ các đặc điểm cơ bản của một khoản vay nhưng thêm các đặc điểm bổ sung, chẳng hạn như một ngôi nhà để bảo đảm cho khoản vay. Mô tả tổng quát hơn được gọi là phụ huynh; một phần tử trong đóng cửa chuyển tiếp là một tổ tiên. Mô tả cụ thể hơn được gọi là đứa trẻ; một yếu tố trong việc đóng cửa chuyển tiếp là một hậu duệ. Trong bài kiểm tra, Loan là lớp cha và Mortgage là lớp con. Tổng quát được sử dụng cho các bộ phân loại (các lớp, các giao diện, các kiểu dữ liệu, các ca sử dụng, các tác nhân, các tín hiệu, vv). Đối với các lớp, các thuật ngữ superclass và subclass được sử dụng cho cha mẹ và con.

Một khái quát hóa được rút ra như một mũi tên từ con đến cha mẹ, với một hình tam giác rỗng lớn ở đầu kết nối với cha (Hình 4-7). Một số mối quan hệ tổng quát có thể được rút ra như một cây với một nhánh đầu mũi tên thành nhiều dòng cho trẻ em.

**Mục đích tổng quát**. Tổng quát hóa có hai mục đích. Đầu tiên là định nghĩa các điều kiện mà theo đó một thể hiện của một lớp (hoặc phần tử khác) có thể được sử dụng khi một biến (như tham số hoặc biến thủ tục) được khai báo là các giá trị giữ của một lớp đã cho. Đây được gọi là nguyên tắc thay thế (từ Barbara Liskov). Quy tắc là một thể hiện của hậu duệ có thể được sử dụng bất cứ nơi nào người khai báo được khai báo. Ví dụ, nếu một biến được khai báo để giữ các khoản vay, thì một đối tượng thế chấp là một giá trị pháp lý.



Generalization cho phép các phép toán đa hình có nghĩa là các phép toán có phương thức (phương thức) được xác định bởi lớp đối tượng mà chúng được áp dụng thay vì được trình bày một cách rõ ràng bởi người gọi. Điều này làm việc vì một lớp cha có thể có nhiều trẻ em có thể, mỗi lớp thực hiện biến thể riêng của một hoạt động, được định nghĩa trên toàn bộ các lớp. Ví dụ, lãi suất com- puting sẽ làm việc khác nhau cho một khoản vay thế chấp và một khoản vay ô tô, nhưng mỗi người trong số họ là một biến thể về lãi suất tính toán trên lớp Loan cho vay. Một vari- thể được khai báo để giữ lớp cha, và sau đó một đối tượng của bất kỳ lớp con nào có thể được sử dụng, bất kỳ trong số đó có các hoạt động riêng của nó. Điều này đặc biệt hữu ích vì các lớp mới có thể được thêm vào sau, mà không cần sửa đổi các cuộc gọi đa dạng hiện có. Ví dụ: một loại khoản vay mới có thể được thêm vào sau và mã hiện có sử dụng hoạt động tính lãi sẽ vẫn hoạt động. Một hoạt động đa hình có thể được khai báo mà không cần thực hiện trong một lớp cha với mục đích là một thực thi phải được cung cấp bởi mỗi lớp con cháu. Một hoạt động không đầy đủ như vậy là trừu tượng (được hiển thị bằng cách in nghiêng tên của nó).

Mục đích tổng quát khác là cho phép mô tả gia tăng của một phần tử bằng cách chia sẻ các mô tả về tổ tiên của nó. Điều này được gọi là thừa kế. Thừa kế là cơ chế mà theo đó một mô tả về các đối tượng của một lớp được phân tách ra khỏi các đoạn khai báo từ lớp và tổ tiên của nó. Thừa kế cho phép các phần được chia sẻ của mô tả được khai báo một lần và được chia sẻ bởi nhiều lớp, thay vì được lặp lại trong mỗi lớp sử dụng nó. Chia sẻ này làm giảm kích thước của một mô hình. Quan trọng hơn, nó làm giảm số lượng thay đổi phải được thực hiện trên một bản cập nhật cho mô hình và làm giảm cơ hội vô tình vô tình. Thừa kế hoạt động theo cách tương tự cho các loại phần tử khác, chẳng hạn như trạng thái, tín hiệu và các trường hợp sử dụng.

## **9. Inheritance**

Mỗi loại phần tử có thể khái quát có một tập hợp các thuộc tính có thể kế thừa. Đối với bất kỳ phần tử mô hình nào, chúng bao gồm các ràng buộc. Đối với phân loại, chúng cũng bao gồm các tính năng (thuộc tính, hoạt động và tiếp nhận tín hiệu) và tham gia vào các liên kết. Một đứa trẻ kế thừa tất cả các thuộc tính có thể kế thừa của tất cả tổ tiên của nó. Tập hợp các thuộc tính hoàn chỉnh của nó là tập hợp các thuộc tính kế thừa cùng với các thuộc tính mà nó giải quyết trực tiếp.

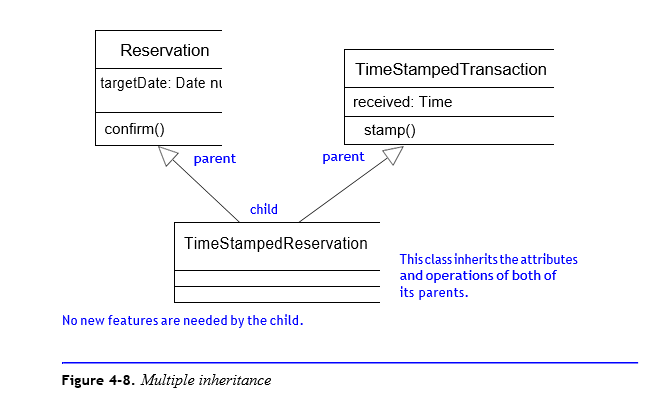
Đối với một trình phân loại, không có thuộc tính nào có cùng chữ ký có thể được khai báo nhiều lần (trực tiếp hoặc được kế thừa). Nếu không, có xung đột và mô hình bị hình thành. Nói cách khác, một thuộc tính được khai báo trong một tổ tiên có thể không được redeclared trong một hậu duệ. Một hoạt động có thể được khai báo trong nhiều hơn một trình phân loại, cung cấp các đặc tả là nhất quán (cùng các tham số, các ràng buộc và ý nghĩa). Khai báo bổ sung chỉ đơn giản là thừa. Một phương thức có thể được khai báo bởi nhiều lớp trong một hệ thống phân cấp. Một phương pháp gắn liền với một hậu duệ thay thế và thay thế (ghi đè) một phương thức có cùng chữ ký được khai báo trong bất kỳ tổ tiên nào. Nếu hai hoặc nhiều bản sao khác biệt của một phương thức vẫn được thừa kế bởi một lớp (thông qua nhiều thừa kế từ các lớp khác nhau), thì chúng xung đột và mô hình bị hình thành. (Một số ngôn ngữ lập trình cho phép một trong các phương thức được chọn một cách rõ ràng. Chúng ta thấy nó đơn giản và an toàn hơn để xác định lại phương thức trong lớp con.) Các ràng buộc trên một phần tử là sự kết hợp của các ràng buộc trên chính phần tử đó và tất cả các tổ tiên của nó; nếu bất kỳ trong số đó không nhất quán, thì mô hình bị bệnh.

Trong một lớp cụ thể, mỗi thao tác được thừa kế hoặc khai báo phải có một phương thức de- fined, hoặc trực tiếp hoặc bằng cách thừa kế từ một tổ tiên.

Trong một số trường hợp, một định nghĩa lại của một định nghĩa được thừa kế có thể được de- clared trong một lớp con. Định nghĩa lại có thể thay đổi tên hoặc một số thuộc tính của một đối tượng địa lý, nhưng nó có thể gây nhầm lẫn và nên được sử dụng một cách tiết kiệm.

## **10. Multiple inheritance**

Nếu một trình phân loại có nhiều hơn một lớp cha, nó sẽ kế thừa từ mỗi phần tử (Hình 4-8). Các tính năng của nó (các thuộc tính, các hoạt động và các tín hiệu) là sự kết hợp của các thuộc tính của cha mẹ nó. Nếu cùng một lớp xuất hiện như một tổ tiên bởi nhiều hơn một đường dẫn, nó vẫn chỉ thể hiện một bản sao của mỗi thành viên của nó. Nếu một đối tượng có cùng chữ ký được khai báo bởi hai lớp không thừa hưởng nó từ một tổ tiên chung (khai báo độc lập), thì xung đột khai báo và mô hình bị hình thành. UML không cung cấp quy tắc giải quyết mâu thuẫn cho tình huống này vì trải nghiệm đã cho thấy rằng nhà thiết kế nên giải quyết rõ ràng. Một số ngôn ngữ, chẳng hạn như Eiffel, xung đột giấy phép được lập trình viên giải quyết một cách rõ ràng, an toàn hơn nhiều so với các quy tắc giải quyết xung đột tiềm ẩn, thường dẫn đến những bất ngờ cho nhà phát triển.



## **11. Single and multiple classification**

Nhiều ngôn ngữ hướng đối tượng giả định phân loại đơn - một đối tượng chỉ có một lớp trực tiếp. Không có sự cần thiết hợp lý rằng một đối tượng có một lớp duy nhất. Chúng ta thường nhìn vào các vật thể trong thế giới thực từ nhiều góc đồng thời. Trong công thức chung của UML, một đối tượng có thể có một hoặc nhiều lớp trực tiếp — nhiều phân loại. Các đối tượng cư xử như thể nó thuộc về một lớp tiềm ẩn đó là một đứa trẻ của mỗi lớp trực tiếp - có hiệu quả, nhiều thừa kế mà không cần phải thực sự khai báo lớp mới.

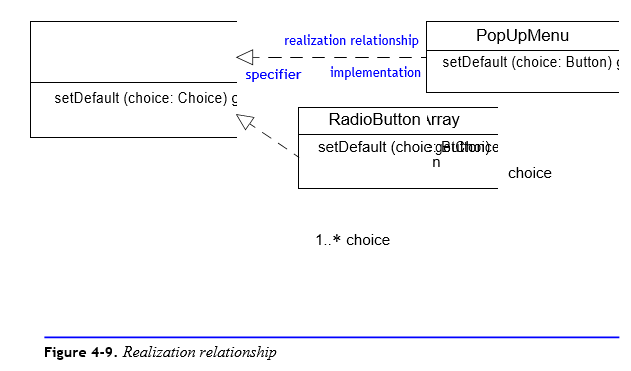
## **12. Static and dynamic classification**

Trong công thức đơn giản nhất, một đối tượng có thể không thay đổi lớp của nó sau khi nó được tạo ra (phân loại tĩnh). Một lần nữa, không có sự cần thiết hợp lý cho sự hạn chế này. Nó chủ yếu nhằm làm cho việc thực hiện các ngôn ngữ lập trình hướng đối tượng dễ dàng hơn. Trong công thức tổng quát hơn, một đối tượng có thể thay đổi trực tiếp lớp động của nó (phân loại động). Khi làm như vậy, nó có thể bị mất hoặc đạt được sự cống hiến hoặc hiệp hội. Nếu nó mất chúng, thông tin trong chúng sẽ bị mất và không thể phục hồi sau này, ngay cả khi nó thay đổi trở lại lớp gốc. Nếu nó đạt được các thuộc tính hoặc các liên kết, thì chúng phải được khởi tạo tại thời điểm thay đổi, theo cách tương tự như việc khởi tạo một đối tượng mới.

Khi nhiều phân loại được kết hợp với phân loại động, một đối tượng có thể thêm và loại bỏ các lớp để phân loại của nó trong suốt cuộc đời của nó. Các lớp động đôi khi được gọi là các vai trò hoặc các kiểu. Một mô hình mô hình phổ biến là yêu cầu

## **13. Realization**

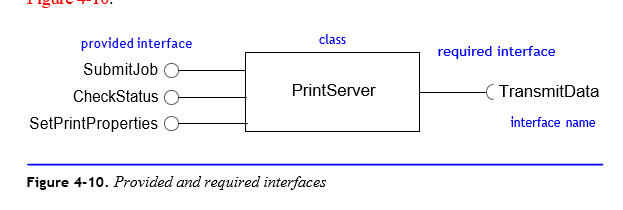
Mối quan hệ Realization (Hình 4-9) kết nối một phần tử mô hình, chẳng hạn như một lớp, với một phần tử mô hình khác, chẳng hạn như một giao diện, cung cấp đặc tính hành vi của nó chứ không phải cấu trúc hoặc thực hiện của nó. Khách hàng phải hỗ trợ (bằng kế thừa hoặc bằng cách khai báo trực tiếp) ít nhất là tất cả các hoạt động mà nhà cung cấp có. Mặc dù việc thực hiện có nghĩa là được sử dụng với các phần tử đặc tả, chẳng hạn như các giao diện, nó cũng có thể được sử dụng với một phần tử thực hiện cụ thể để chỉ ra rằng đặc tả của nó (nhưng không thực hiện) phải được hỗ trợ. Điều này có thể được sử dụng để hiển thị mối quan hệ của một phiên bản tối ưu hóa của một lớp với một phiên bản đơn giản nhưng không hiệu quả, ví dụ.



Cả generalization và realization liên quan đến một mô tả tổng quát hơn cho các phiên bản hoàn chỉnh hơn của nó. Việc khái quát hóa liên quan đến hai phần tử ở cùng một mức ngữ nghĩa (ví dụ như cùng một mức trừu tượng), thường trong cùng một mô hình; việc thực hiện liên quan đến hai phần tử ở các mức ngữ nghĩa khác nhau (một phân tích lớp và một lớp thiết kế, ví dụ, hoặc một giao diện và một lớp), thường được tìm thấy trong các mô hình khác nhau. Có thể có hai hoặc nhiều phân cấp toàn bộ lớp ở các giai đoạn phát triển khác nhau có các yếu tố liên quan đến việc thực hiện. Hai biểu đồ không cần phải có cùng một dạng vì các lớp thực hiện có thể có các phụ thuộc thực hiện không liên quan đến các lớp xác định.

Việc thực hiện được hiển thị dưới dạng mũi tên đứt quãng với đầu mũi tên rỗng khép kín (Hình 4-9) trên lớp tổng quát hơn. Nó tương tự như biểu tượng khái quát hóa với một đường đứt nét, để chỉ ra rằng nó tương tự như một loại thừa kế.

Giao diện được thực hiện bởi một lớp được gọi là một giao diện được cung cấp, bởi vì lớp này cung cấp các dịch vụ của giao diện cho người gọi bên ngoài. Mối quan hệ này có thể được hiển thị bằng cách gắn một vòng tròn nhỏ vào hình chữ nhật của lớp bằng một dòng; vòng tròn được gắn nhãn với tên giao diện. Một giao diện mà một lớp sử dụng để thực hiện hành vi bên trong của nó được gọi là một giao diện được yêu cầu. Mối quan hệ này có thể được hiển thị bằng cách liên kết một hình bán nguyệt nhỏ với hình chữ nhật của lớp bằng một dòng. Ký pháp này không được sử dụng để khai báo các giao diện, mà là để hiển thị mối quan hệ của chúng với các lớp. Xem hình 4-10.



## **14. Dependency**

Một Dependency chỉ ra một mối quan hệ ngữ nghĩa giữa hai hoặc nhiều hơn các mô hình. Nó liên quan đến các yếu tố mô hình và không đòi hỏi một tập hợp các ý nghĩa cho ý nghĩa của nó. Nó chỉ ra một tình huống trong đó thay đổi đối với yếu tố nhà cung cấp có thể yêu cầu thay đổi hoặc cho thấy sự thay đổi về ý nghĩa của thân chủ khách hàng trong sự phụ thuộc.

Các mối quan hệ liên kết và khái quát hóa là những phụ thuộc bởi điều này, nhưng chúng có ngữ nghĩa cụ thể với những hậu quả quan trọng. Do đó, họ có tên riêng và ngữ nghĩa chi tiết. Chúng tôi thường sử dụng sự từ chối cho tất cả các mối quan hệ khác không phù hợp với các danh mục sắc nét hơn. Bảng 4-3 liệt kê các loại phụ thuộc áp dụng cho chế độ xem tĩnh.

Sự phụ thuộc theo dõi là một kết nối khái niệm giữa các phần tử trong các mô hình khác nhau, thường là các mô hình ở các giai đoạn phát triển khác nhau. Nó thiếu các ngữ nghĩa chi tiết. Nó thường được sử dụng để theo dõi các yêu cầu hệ thống trên các mô hình và theo dõi các thay đổi được thực hiện cho các mô hình có thể ảnh hưởng đến các mô hình khác.

**Table 4-3:** *Kinds of Dependencie**s*

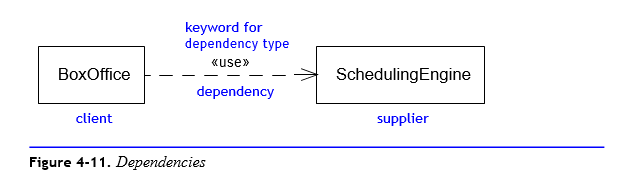
|  |  |  |
| --- | --- | --- |
| *Dependency* | *Function* | *Keyword* |
| [access](#_bookmark452) | A private import of the contents of another package | access |
| [binding](#_bookmark633) | Assignment of values to the parameters of a template to generate a new model element | bind |
| [call](#_bookmark662) | Statement that a method of one class calls an operation of another class | call |
| [creation](#_bookmark911) | Statement that one class creates instances of another class | create |
| [derivation](#_bookmark984) | Statement that one instance can be com- puted from another instance | derive |
| [instantiation](#_bookmark1308) | Statement that a method of one class creates instances of another class | instantiate |
| [permission](#_bookmark1681) | Permission for an element to use the con- tents of another element | permit |
| [realization](#_bookmark1798) | Mapping between a specification and an implementation of it | realize |
| [refinement](#_bookmark1840) | Statement that a mapping exists between elements at two different semantic levels | refine |
| [send](#_bookmark1909) | Relationship between the sender of a signal and the receiver of the signal | send |
| [substitution](#_bookmark2092) | Statement that the source class supports the interfaces and contracts of the target class and may be substituted for it | substitute |
| [trace dependency](#_bookmark2178) | Statement that some connection exists between elements in different models, but less precise than a mapping | trace |
| [usage](#_bookmark2222) | Statement that one element requires the presence of another element for its correct functioning (includes call, creation, instan- tiation, send, and potentially others) | use |

[Refinement](#_bookmark1840) là một mối quan hệ giữa hai phiên bản của một khái niệm ở các giai đoạn phát triển khác nhau hoặc ở các mức trừu tượng khác nhau, được biểu thị bằng hai yếu tố mô hình riêng biệt. Hai yếu tố mô hình không có nghĩa là cùng tồn tại trong mô hình chi tiết cuối cùng. Một trong số họ thường là một phiên bản ít hoàn thành của người khác. Trong nguyên tắc, có một ánh xạ từ phần tử mô hình ít thành phẩm đến phần tử mô hình đã hoàn thành hơn. Điều này không có nghĩa là bản dịch là tự động. Thông thường, phần tử chi tiết hơn chứa các quyết định thiết kế đã được thực hiện bởi người ký, các quyết định có thể được thực hiện theo nhiều cách. Về nguyên tắc, các thay đổi đối với một mô hình có thể được xác thực với mô hình kia, với các độ lệch được gắn cờ. Trong thực tế, các công cụ không thể làm tất cả điều này ngày hôm nay, mặc dù một số ánh xạ đơn giản hơn có thể được thực thi. Do đó, sàng lọc chủ yếu là một lời nhắc nhở cho người lập mô hình rằng nhiều mô hình có liên quan theo cách có thể dự đoán được.

Một [derivation](#_bookmark984) dependency chỉ ra rằng một phần tử có thể được tính từ một phần tử khác (nhưng phần tử dẫn xuất có thể được bao gồm một cách rõ ràng trong hệ thống để tránh việc tính toán lại chi phí). Bắt nguồn, thực hiện, sàng lọc và theo dõi là các phụ thuộc trừu tượng - chúng liên quan đến hai phiên bản của cùng một điều cơ bản.

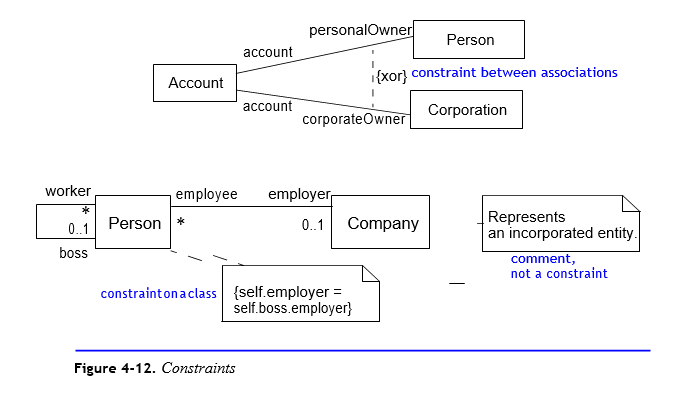
Một phụ thuộc usage là một tuyên bố rằng hành vi hoặc việc thực hiện một phần tử ảnh hưởng đến hành vi hoặc việc thực hiện của một phần tử khác. Thông thường, điều này xuất phát từ các mối quan tâm thực hiện, chẳng hạn như các yêu cầu của trình biên dịch để định nghĩa một lớp là cần thiết để biên dịch một lớp khác. Hầu hết các phụ thuộc sử dụng có thể được lấy từ mã và không cần khai báo rõ ràng, trừ khi chúng là một phần của kiểu thiết kế từ trên xuống để hạn chế tổ chức của hệ thống (ví dụ, bằng cách sử dụng các thành phần và thư viện được định nghĩa trước) ). Loại phụ thuộc sử dụng cụ thể có thể được chỉ định, nhưng điều này thường bị bỏ qua vì mục đích của mối quan hệ là làm nổi bật sự phụ thuộc. Các chi tiết chính xác thường có thể thu được từ mã thực hiện. Khuôn mẫu sử dụng bao gồm gọi và instantiation. Sự phụ thuộc của cuộc gọi chỉ ra rằng một phương thức trên một lớp gọi một phép toán trên một lớp khác; instantiation chỉ ra rằng một phương thức trên một lớp tạo ra một thể hiện của một lớp khác.  
Một số loại phụ thuộc thêm các phần tử vào một không gian tên. Tính năng nhập khẩu thêm các tên trong nội dung không gian tên đích vào không gian tên nhập khẩu. Phụ thuộc truy cập cũng thêm các tên vào một không gian tên, nhưng các tên được thêm vào không hiển thị bên ngoài vùng tên mà chúng được thêm vào.

Một ràng buộc là gán các giá trị cho các tham số của một khuôn mẫu. Nó là một mối quan hệ có cấu trúc cao với ngữ nghĩa chính xác thu được bằng cách thay thế các đối số cho các tham số trong một bản sao của mẫu.  
Các phụ thuộc sử dụng và ràng buộc liên quan đến ngữ nghĩa mạnh mẽ giữa các phần tử ở cùng một mức ngữ nghĩa. Chúng phải kết nối các phần tử trong cùng một mức mô hình (cả phân tích hoặc cả hai thiết kế và ở cùng một mức trừu tượng). Dấu vết và tinh tế phụ thuộc vaguer và có thể kết nối các yếu tố từ các mô hình khác nhau hoặc mức độ trừu tượng.



## **15. Contraint**

UML cung cấp một tập hợp các khái niệm và các mối quan hệ cho các hệ thống mô hình hóa như là các đồ thị của các phần tử mô hình hóa. Tuy nhiên, một số điều được thể hiện tốt hơn về ngôn ngữ - nghĩa là sử dụng sức mạnh của một ngôn ngữ văn bản. Ràng buộc là một biểu thức Boolean được biểu diễn như một chuỗi được diễn giải bằng một ngôn ngữ được chỉ định. Ngôn ngữ tự nhiên, đặt ký hiệu lý thuyết, các ngôn ngữ hạn chế hoặc các ngôn ngữ lập trình khác nhau có thể được sử dụng để thể hiện các ràng buộc. UML bao gồm định nghĩa của một ngôn ngữ ràng buộc, được gọi là OCL, đó là thuận tiện cho việc thể hiện các ràng buộc UML và dự kiến sẽ được hỗ trợ rộng rãi. Xem mục nhập cho OCL và cuốn sách [Warmer-99] để biết thêm thông tin về OCL.



Các ràng buộc có thể được sử dụng để chỉ ra các mối quan hệ khác nhau, chẳng hạn như các hạn chế trên các đường dẫn của các kết hợp. Đặc biệt, các ràng buộc có thể được sử dụng để xác định các thuộc tính tồn tại (có tồn tại một X sao cho điều kiện C là đúng) và các thuộc tính phổ quát (cho tất cả y trong Y, điều kiện D phải đúng).

Một số ràng buộc chuẩn được định nghĩa trước là các phần tử tiêu chuẩn UML, bao gồm các liên kết trong một độc quyền hoặc mối quan hệ và các ràng buộc khác nhau về các mối quan hệ của các lớp con trong khái quát hóa.

Ràng buộc được hiển thị dưới dạng biểu thức văn bản trong dấu ngoặc ôm. Nó có thể được viết bằng ngôn ngữ tự nhiên hoặc ngôn ngữ tự nhiên. Chuỗi văn bản có thể được đặt trong một ghi chú hoặc được gắn vào mũi tên phụ thuộc.

Các ràng buộc được xác định trước có thể được biểu diễn trong OCL. Ví dụ, mô hình xor trong Hình 4-12 có thể được viết trong OCL là:  
context Tài khoản inv:personalOwner -> size> 0 xor companyOwner -> size> 0

## **16. Instance**

Instance là một thực thể chạy với bản sắc, có nghĩa là, một cái gì đó có thể được phân biệt với các thực thể chạy khác. Nó có một giá trị tại bất kỳ thời điểm nào. Theo thời gian, giá trị có thể thay đổi theo các hoạt động trên nó.

Một mục đích của một mô hình là mô tả các trạng thái có thể của một hệ thống và hành vi của chúng. Một mô hình là một tuyên bố về tiềm năng, của các bộ sưu tập có thể có của các đối tượng có thể tồn tại và lịch sử hành vi có thể có mà các đối tượng có thể không hợp nhất. Khung nhìn tĩnh định nghĩa và hạn chế các cấu hình có thể có của các giá trị mà một hệ thống thực thi có thể giả định. Chế độ xem động xác định các cách thức mà hệ thống thực thi có thể truyền từ cấu hình này sang cấu hình khác. Cùng với nhau, chế độ xem tĩnh và các chế độ xem động khác nhau dựa trên cơ sở đó xác định cấu trúc và cấu trúc của hệ thống.

Một cấu hình tĩnh cụ thể của một hệ thống tại một thời điểm được gọi là ảnh chụp nhanh. Ảnh chụp nhanh bao gồm các đối tượng và các phiên bản, giá trị và liên kết khác. Một đối tượng là một thể hiện của một lớp. Mỗi đối tượng là một cá thể trực tiếp của lớp mô tả hoàn toàn nó và một cá thể gián tiếp của tổ tiên của lớp đó. (Nếu nhiều phép nhân bản được phép, thì một đối tượng có thể là cá thể trực tiếp của nhiều lớp.) Tương tự, mỗi liên kết là một cá thể của một liên kết và mỗi giá trị là một kiểu dữ liệu.

Một đối tượng có một giá trị dữ liệu cho mỗi thuộc tính trong lớp của nó. Giá trị của mỗi lần tri ân phải nhất quán với kiểu dữ liệu của thuộc tính. Nếu thuộc tính có tùy chọn hoặc bội số, thì thuộc tính có thể giữ 0 hoặc nhiều giá trị. Một liên kết bao gồm một bộ các giá trị, mỗi giá trị trong số đó là một tham chiếu đến một đối tượng của một lớp đã cho (hoặc một trong các phần tử con của nó). Các đối tượng và liên kết phải tuân theo bất kỳ sự ràng buộc nào về các lớp hoặc các kết hợp mà chúng là các cá thể (bao gồm cả ràng buộc rõ ràng và các ràng buộc dựng sẵn, chẳng hạn như tính đa dạng).

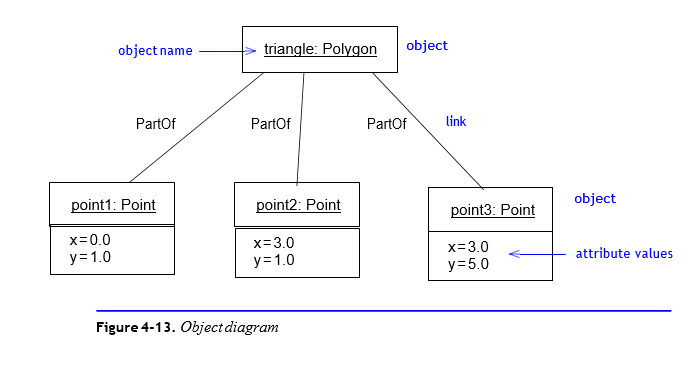
Trạng thái của một hệ thống là một cá thể hệ thống hợp lệ nếu mỗi cá thể trong nó là một cá thể của một phần tử trong một mô hình hệ thống tốt và nếu tất cả các ràng buộc được mô hình thỏa mãn bởi các cá thể.  
Khung nhìn tĩnh định nghĩa tập các đối tượng, các giá trị và các liên kết có thể tồn tại trong một ảnh chụp nhanh. Về nguyên tắc, bất kỳ sự kết hợp nào của các đối tượng và liên kết nhất quán với chế độ xem tĩnh là một cấu hình có thể có của mô hình. Điều này không có nghĩa là mọi ảnh chụp có thể có hoặc sẽ xảy ra. Một số ảnh chụp nhanh có thể hợp pháp theo phương thức tĩnh nhưng có thể không truy cập động theo các chế độ xem động trong hệ thống.

Các phần hành vi của UML mô tả các chuỗi ảnh chụp hợp lệ có thể xảy ra do kết quả của cả các hiệu ứng hành vi bên ngoài lẫn bên trong. Chế độ xem động xác định cách hệ thống di chuyển từ ảnh chụp nhanh này sang ảnh chụp khác.

## **17. Sơ đồ đối tượng**

Sơ đồ chụp nhanh là hình ảnh của một hệ thống tại một thời điểm. Bởi vì nó chứa hình ảnh của các đối tượng, nó được gọi là một sơ đồ đối tượng. Nó có thể hữu ích như một ví dụ của hệ thống, ví dụ, để minh họa các cấu trúc dữ liệu phức tạp hoặc để hiển thị hành vi thông qua một chuỗi các ảnh chụp nhanh theo thời gian (Hình 4-13). Một đối tượng không được giới hạn đối tượng cụ thể. Nó cũng có thể bao gồm các thông số giá trị của các đối tượng trong đó một số giá trị có thể không được chỉ định không đầy đủ, ví dụ, bằng cách chỉ ra một phạm vi giá trị thay vì một giá trị cụ thể.

Ảnh chụp nhanh là ví dụ về hệ thống, không phải là định nghĩa của hệ thống. Định nghĩa về cấu trúc hệ thống và hành vi là mục tiêu của mô hình hóa và thiết kế. Ví dụ có thể giúp làm rõ ý nghĩa đối với con người, nhưng chúng không phải là định nghĩa.



# **CHƯƠNG 4: DESIGN VIEW**

## **1. Tổng quan**

Phần lớn mô hình hệ thống được thiết kế để hiển thị các khía cạnh logic và thiết kế của hệ thống độc lập với bao bì cuối cùng của nó trong môi trường thực hiện. Tuy nhiên, các khía cạnh quan trọng là quan trọng đối với cả mục đích sử dụng lại và mục đích hiệu quả. Khung nhìn thiết kế cho thấy các quyết định về sự phân hủy của một hệ thống thành các đơn vị mô-đun với các ranh giới đóng gói và các giao diện bên ngoài. Mặc dù các phần tử trong khung nhìn thiết kế trừu tượng hơn mã cuối cùng, chúng đòi hỏi kiến ​​thức về việc thực hiện các giao dịch mà cuối cùng sẽ được tái hiện trong mã.  
Các hệ thống phức tạp đòi hỏi nhiều mức cấu trúc. Trong quá trình mô hình hóa sớm, một lớp được định nghĩa bởi các thuộc tính bên ngoài của nó.

Trong mô hình thiết kế, dấu hiệu bên trong của một lớp cao cấp có thể được mở rộng thành các phần cấu thành. Một trình phân loại có cấu trúc là một trình phân loại với các phần nội bộ được kết nối trong ngữ cảnh của trình phân loại. Một bộ phân loại có cấu trúc có thể có một ranh giới lỏng lẻo hoặc một ràng buộc chặt chẽ, nơi tất cả các thông tin liên lạc xảy ra trên các điểm tương tác được xác định rõ ràng được gọi là các cổng. Các loại bộ phận bên trong có thể là các bộ phân loại cấu trúc; do đó sự phân hủy của hệ thống có thể kéo dài nhiều cấp độ.

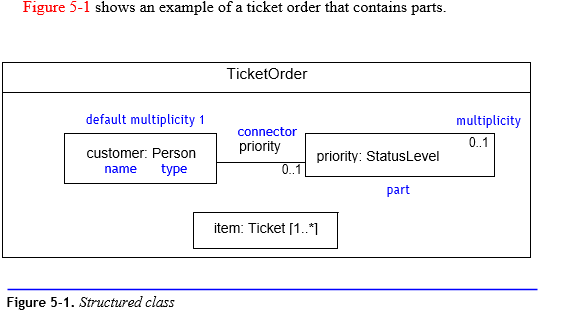
Trong một thiết kế, các đối tượng độc lập thường làm việc cùng nhau để thực hiện các hoạt động và các hành vi khác. Trong một khoảng thời gian giới hạn, các đối tượng có liên quan đến sự tham gia của họ trong một bối cảnh được chia sẻ. Cộng tác là mô tả về một nhóm đối tượng có mối quan hệ tạm thời trong ngữ cảnh thực hiện hành vi. Cộng tác là một mối quan hệ khái niệm, không phải là một đối tượng cụ thể, mặc dù có thể có sự cản trở trong việc thực hiện liên quan đến nó. Các kết nối giữa các đối tượng có thể bao gồm các loại mối quan hệ tức thời khác nhau, chẳng hạn như tham số, biến và quan hệ có nguồn gốc, cũng như các liên kết thông thường.

Khung nhìn thiết kế cho thấy việc tổ chức hợp lý các phần có thể tái sử dụng của hệ thống thành các đơn vị thay thế, được gọi là các thành phần. Một thành phần có một bộ các giao diện bên ngoài và một thực thi nội bộ ẩn. Các thành phần tương tác thông qua giao diện để tránh sự phụ thuộc vào các thành phần cụ thể khác. Trong quá trình thực hiện, bất kỳ thành phần nào hỗ trợ giao diện có thể được thay thế cho nó, cho phép các phần khác nhau của một hệ thống được phát triển mà không phụ thuộc vào việc thực hiện bên trong.

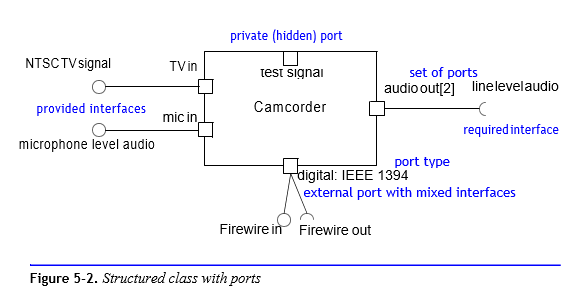
## **2. Trình phân loại có cấu trúc**

Một trình phân loại có cấu trúc là một trình phân loại có cấu trúc bên trong. Nó chứa một tập hợp các bộ phận được kết nối bằng các đầu nối. Một thể hiện của một lớp có cấu trúc chứa một đối tượng hoặc một tập hợp các đối tượng tương ứng với từng phần. Một phần có một loại và một bội số trong vùng chứa của nó. Một đối tượng là một phần chỉ có thể thuộc về một đối tượng có cấu trúc. Tất cả các đối tượng trong một đối tượng có cấu trúc duy nhất đều liên quan ngầm định với các đối tượng của chúng trong cùng một đối tượng. Mối quan hệ tiềm ẩn này có thể được khai thác trong việc thực hiện hành vi trên lớp có cấu trúc.

Trình kết nối là mối quan hệ theo ngữ cảnh giữa hai phần trong một trình tạo cấu trúc. Nó định nghĩa một mối quan hệ giữa các đối tượng phục vụ như là một phần của cùng một đối tượng có cấu trúc. Một kết nối giữa hai phần của một lớp có cấu trúc khác với một liên kết giữa hai lớp được liên kết theo thành phần cho cùng một lớp. Trong hiệp hội, không có yêu cầu rằng hiệp hội kết nối hai đối tượng được chứa trong cùng một đối tượng hỗn hợp. Mỗi liên kết với một lớp sin-gle là độc lập, trong khi tất cả các kết nối trong một lớp có cấu trúc duy nhất chia sẻ một ngữ cảnh duy nhất tại thời gian chạy. Các kết nối có thể được thực hiện bởi các liên kết thông thường hoặc bởi các mối quan hệ thoáng qua, chẳng hạn như các tham số thủ tục, các biến, các giá trị toàn cục hoặc các cơ chế khác.



Các bộ phân loại có cấu trúc có thể được đóng gói chặt chẽ bằng cách buộc tất cả các tương tác giữa môi trường bên ngoài và các phần bên trong đi qua các cổng. Cổng là điểm tương tác với giao diện được xác định rõ. Cổng có thể được kết nối với các bộ phận bên trong hoặc các cổng trên các bộ phận bên trong, hoặc nó có thể được kết nối trực tiếp với hành vi chính của bộ phân loại cấu trúc. Các tin nhắn nhận được bởi một cổng sẽ tự động được chuyển tiếp đến phần hoặc hành vi (hoặc ngược lại trên đầu ra). Mỗi cổng có một bộ giao diện được cung cấp và các giao diện bắt buộc xác định các tương tác bên ngoài của nó. Giao diện được cung cấp chỉ định các dịch vụ mà một thông báo tới cổng có thể yêu cầu. Giao diện bắt buộc chỉ định các dịch vụ mà một thông báo từ cổng có thể yêu cầu từ môi trường bên ngoài. Các kết nối bên ngoài đến một trình phân loại en- capsulated chỉ có thể đi đến các cổng. Kết nối bên ngoài là hợp pháp nếu các khớp trong khớp với nhau. Trong trường hợp các yêu cầu đầu vào, giao diện được cung cấp phải hỗ trợ ít nhất các dịch vụ được yêu cầu bởi kết nối bên ngoài. Trong trường hợp yêu cầu đầu ra, giao diện bắt buộc phải yêu cầu không nhiều hơn các dịch vụ được cung cấp bởi kết nối bên ngoài.  
Hình 5-2 cho thấy các cổng và giao diện cho một máy quay.



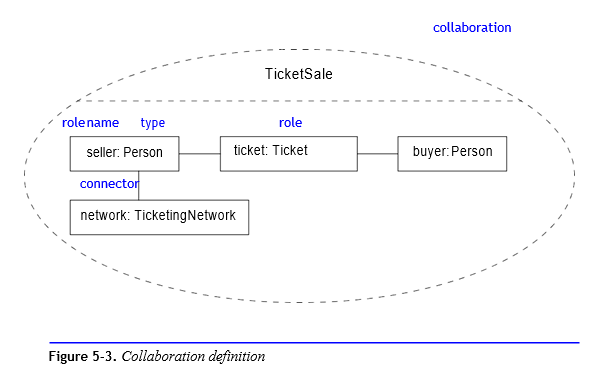
## **3. Cộng tác**

Cộng tác là một mô tả về một tập hợp các đối tượng tương tác để thực hiện một số hành vi trong một ngữ cảnh. Nó mô tả một xã hội của các đối tượng hợp tác được lắp ráp để thực hiện một số mục đích. Cộng tác chứa các vai trò được lấp đầy bởi các đối tượng trong thời gian chạy. Một vai trò đại diện cho một mô tả về các đối tượng có thể tham gia vào việc thực hiện cộng tác. Trình kết nối đại diện cho mô tả về các mối liên kết giữa các vai trò của cộng tác. Mối quan hệ giữa các vai trò và kết nối bên trong một sự cộng tác chỉ có ý nghĩa trong ngữ cảnh đó. Vai trò và các trình kết nối có thể có các kiểu (các trình phân loại và các liên kết) xác định đối tượng nào có thể bị ràng buộc với chúng. Các kiểu kết hợp là tùy chọn, vì các mối quan hệ trong một sự cộng tác có thể tạm thời và được triển khai bằng các cơ chế khác, chẳng hạn như các tham số. Các mối quan hệ được mô tả bởi một sự hợp tác chỉ áp dụng cho các đối tượng liên kết với các vai trò trong một cá thể cộng tác cụ thể; chúng không áp dụng cho các phân loại và các liên kết cơ bản ngoài việc cộng tác.

Chế độ xem tĩnh mô tả các thuộc tính vốn có của một lớp. Ví dụ, một vé có một chương trình và một bộ liên kết với nó. Mối quan hệ này áp dụng cho tất cả các trường hợp của lớp. Sự cộng tác mô tả các thuộc tính mà một thể hiện của một lớp có khi nó đóng một vai trò cụ thể trong một sự cộng tác. Ví dụ, một vé trong một hợp tác bán vé có một người bán, cái gì đó không liên quan đến một vé nói chung nhưng là một phần thiết yếu của sự hợp tác bán vé.

Một đối tượng trong một hệ thống có thể tham gia vào nhiều hơn một sự hợp tác. Các collabo trong đó nó xuất hiện không cần phải liên quan trực tiếp, mặc dù việc thực thi của chúng được kết nối (có thể là tình cờ) thông qua đối tượng được chia sẻ. Ví dụ: một người có thể là người mua trong một cộng tác và người bán trong một cộng tác khác. Hơi ít thường xuyên hơn, một đối tượng có thể đóng nhiều vai trò trong cùng một cộng tác.

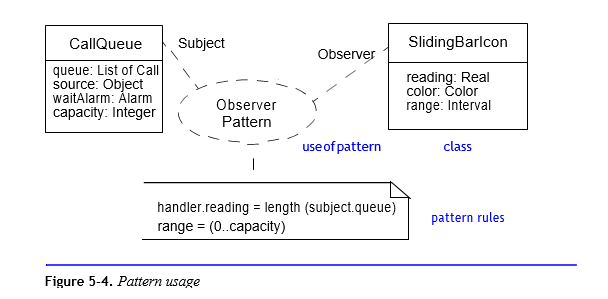
Cộng tác là một loại phân loại có cấu trúc. Hình 5-3 cho thấy sự hợp tác để bán vé.



## **4. Pattern**

Pattern là một sự hợp tác được tham số hóa, cùng với các hướng dẫn về thời điểm sử dụng nó. Một tham số có thể được thay thế bằng các giá trị khác nhau để tạo ra các cộng tác khác nhau. Các tham số thường chỉ định các khe cho các lớp. Khi một mẫu được xác định, các tham số của nó được ràng buộc với các lớp thực tế trong một sơ đồ lớp hoặc với các vai trò trong một sự cộng tác lớn hơn.

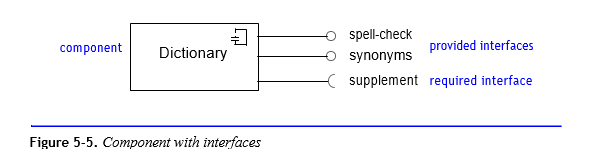
Việc sử dụng một mẫu được hiển thị dưới dạng hình elip đứt quãng được kết nối với từng lớp của nó bằng một đường đứt nét được gắn nhãn với tên của vai trò. Ví dụ, Hình 5-4 cho thấy việc sử dụng mẫu Observer từ [Gamma-95]. Trong trường hợp sử dụng này, CallQueue thay thế vai trò Chủ đề và SlidingBarIcon thay thế vai trò Quan sát viên.  
Các mẫu có thể xuất hiện ở phân tích, kiến trúc, thiết kế chi tiết và mức độ triển khai. Chúng là một cách để nắm bắt các cấu trúc thường xuyên xảy ra để tái sử dụng. Hình 5-4 cho thấy việc sử dụng mẫu Observer.



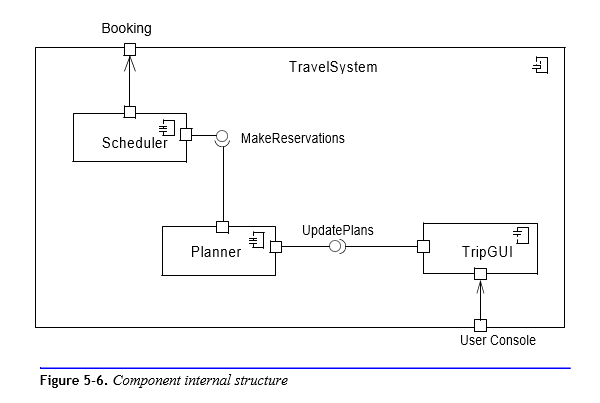
## **5. Component**

Một thành phần đại diện cho một phần mô-đun của một hệ thống logic hoặc vật lý mà hành vi có thể nhìn thấy bên ngoài có thể được mô tả ngắn gọn hơn nhiều so với thực hiện của nó. Các thành phần không phụ thuộc trực tiếp vào các thành phần khác mà trên các giao diện mà các thành phần hỗ trợ. Một thành phần trong một mô hình có thể được thay thế bằng một thành phần khác hỗ trợ các giao diện thích hợp và một thể hiện thành phần trong một cấu hình hệ thống có thể được thay thế bằng một cá thể của bất kỳ thành phần nào hỗ trợ các giao diện tương tự.

Các thành phần (vì chúng là các lớp) có các giao diện mà chúng hỗ trợ (các giao diện được cung cấp) và các giao diện mà chúng yêu cầu từ các thành phần khác (các giao diện bắt buộc). Việc sử dụng các giao diện được đặt tên cho phép phụ thuộc trực tiếp giữa các thành phần cần tránh, tạo điều kiện cho việc thay thế các thành phần mới dễ dàng hơn. Một sơ đồ thành phần cho thấy mạng phụ thuộc giữa các thành phần. Chế độ xem thành phần có thể xuất hiện ở hai dạng. Nó có thể hiển thị một tập hợp các thành phần có sẵn (một thư viện thành phần) với các phụ thuộc của chúng; đây là vật liệu mà trong đó một hệ thống có thể được lắp ráp. Nó cũng có thể hiển thị một hệ thống được cấu hình, với việc lựa chọn các thành phần (trong toàn bộ thư viện) được sử dụng để xây dựng nó. Trong biểu mẫu này, mỗi thành phần được kết nối với các thành phần khác có các dịch vụ mà nó sử dụng; những điều này phải phù hợp với các giao diện của các thành phần.  
Biểu tượng thành phần được vẽ như một hình chữ nhật có hai hình chữ nhật nhỏ ở bên cạnh nó. Một thành phần được vẽ như một hình chữ nhật với một biểu tượng thành phần nhỏ ở góc trên bên phải của nó. Hình chữ nhật chứa tên của thành phần. Một giao diện được cung cấp được vẽ như một vòng tròn nhỏ kết nối với thành phần bằng một đường thẳng. Một giao diện được yêu cầu được vẽ như một hình bán nguyệt nhỏ được kết nối với thành phần bằng một dòng. Hình 5-5 cho thấy một ví dụ.



Các thành phần có thể có cổng. Các tin nhắn nhận được trên các cổng khác nhau được phân biệt bởi thành phần và có thể được thực hiện khác nhau. Một cổng được hiển thị bằng một hình vuông nhỏ trên ranh giới của một biểu tượng thành phần. Biểu tượng giao diện có thể được gắn vào cổng.  
Các thành phần có thể chứa các thành phần khác khi thực hiện chúng. Sự “nối dây” của hai thành phần trong việc thực hiện được thể hiện bằng cách lồng vòng tròn của một giao diện được cung cấp trong hình bán nguyệt của một giao diện được yêu cầu khác theo ký hiệu “bóng và ổ cắm” (Hình 5-6).  
Một thành phần là một trình phân loại cấu trúc. Sự khác biệt ngữ nghĩa giữa một trình phân loại có cấu trúc và một thành phần không phải là rất lớn. Một thành phần là một tuyên bố về ý định thiết kế hơn là một sự khác biệt ngữ nghĩa.



# **CHƯƠNG 5: ACTIVITY VIEW**

## **1. Activity**

Activity là biểu đồ các nút và luồng, cho thấy luồng điều khiển (và tùy chọn dữ liệu) thông qua các bước tính toán. Thi hành các bước có thể vừa đồng thời vừa tuần tự. activity liên quan đến cả đồng bộ hóa và phân nhánh cấu trúc, tương tự nhưng mạnh mẽ hơn flow chart (biểu đồ luồng ) truyền thống, chỉ hỗ trợ các cấu trúc tuần tự và phân nhánh

Định nghĩa activity được hiển thị trong sơ đồ hoạt động (activity diagram) (Hình 8-1).

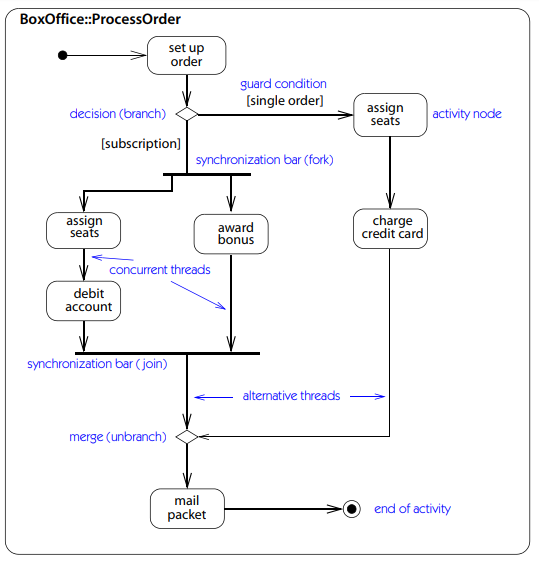
Activity node được hiển thị dưới dạng hộp có các góc được làm tròn chứa mô tả của hoạt động.

Control flow được hiển thị dưới dạng mũi tên.

Branch (nhánh) được hiển thị dưới dạng điều kiện bảo vệ (guard condition) trên luồng điều khiển (control flow) hoặc dưới dạng hình thoi có nhiều mũi tên thoát.

Fork hoặc join kiểm soát được thể hiện bằng nhiều mũi tên đi vào (hoặc đi ra) một thanh đồng bộ đậm.

Hình 8-1 cho thấy một sơ đồ hoạt động (activity diagram) xử lý đặt hàng bởi phòng vé.

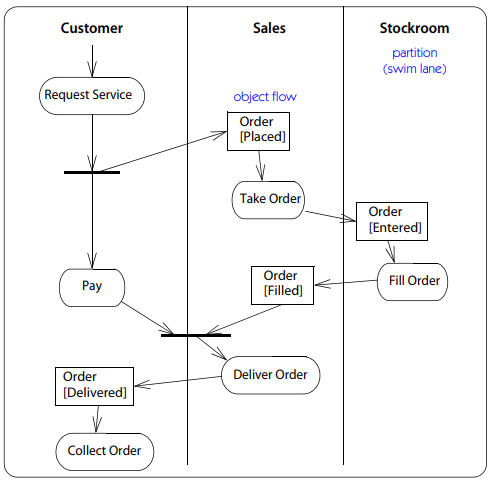


Hình 8-1. Activity diagram

Đối với những tình huống trong đó các sự kiện bên ngoài phải được bao gồm, việc nhận một sự kiện có thể được hiển thị dưới dạng hành động biểu thị việc chờ tín hiệu. Một ký hiệu tương tự cho thấy việc gửi tín hiệu.

## **2. Phân vùng**

Nó thường hữu ích để tổ chức các hoạt động trong một mô hình theo trách nhiệm. Ví dụ, bằng cách nhóm lại với nhau tất cả các hoạt động được xử lý bởi một tổ chức kinh doanh. Loại nhiệm vụ này có thể được hiển thị bằng cách tổ chức các hoạt động thành các vùng riêng biệt (được gọi là partitions ) được phân tách bằng các dòng trong biểu đồ. Bởi vì sự xuất hiện, một khu vực đôi khi được gọi là swimlane. Hình 8-2 hiển thị các phân vùng.



Hình 8-2. Partitions and object flows.

## **3. Object flows**

Biểu đồ hoạt động có thể hiển thị luồng của các giá trị đối tượng, cũng như luồng kiểm soát. Object flow (luồng đối tượng) đại diện cho một đối tượng là đầu vào hoặc đầu ra của một hoạt động.

* Đối với một giá trị đầu ra, một mũi tên được vẽ từ một hoạt động đến một object flow.
* Đối với một giá trị đầu vào, một mũi tên được đi ra từ một object flow đến một hoạt động.

Nếu một hoạt động có nhiều giá trị đầu ra hoặc luồng điều khiển kế tiếp,mũi tên được vẽ từ biểu tượng ngã ba. Tương tự, nhiều đầu vào được vẽ đến tham gia biểu tượng.

Hình 8-2 ​​cho thấy một sơ đồ hoạt động trong đó cả các hoạt động và luồng đối tượng đã được chỉ định cho swimlanes.

## **4. Activities và các chế độ xem khác**

Biểu đồ hoạt động không hiển thị đầy đủ chi tiết về tính toán.

Chúng cho thấy dòng chảy của các hoạt động chứ không phải là các đối tượng thực hiện các hoạt động. Biểu đồ hoạt động là điểm khởi đầu cho thiết kế.

Để hoàn thành một thiết kế, mỗi hoạt động phải được thực hiện như một hoặc nhiều hoạt động, mỗi hoạt động được gán cho một lớp cụ thể để thực hiện.

Kết quả của thiết kế collaboration diagram (sơ đồ cộng tác) dùng để thực hiện biểu đồ hoạt động

## **5. Action**

UML có một tập hợp các hành động nguyên thủy mô hình hóa thao tác của các đối tượng và liên kết cũng như tính toán và giao tiếp giữa các đối tượng. UML không định nghĩa cú pháp cho các hành động bởi vì người ta cho rằng hầu hết các mô hình sẽ sử dụng ngôn ngữ lập trình hoặc ngôn ngữ lập trình hiện hành. Bảng 8-1 cho thấy các loại hành động.

|  |  |  |
| --- | --- | --- |
| Phân loại | Hành động | Mục đích |
| classification | readIsClassifiedObject reclassifyObject  testIdentity | kiểm tra phân loại  thay đổi phân loại  kiểm tra đối tượng-ID |
| communication | broadcastSignal  callOperation  reply  (implicit) return  sendObject  sendSignal | phát sóng  cuộc gọi bình thường  trả lời sau khi chấp nhận rõ ràng  hành động ngầm trên kết thúc hoạt động  gửi tín hiệu làm đối tượng  gửi tín hiệu dưới dạng danh sách đối số |
| computation | accept call  accept event  addVariableValue  applyFunction  callBehavior  clearVariable  readSelf  readVariable removeVariableValue writeVariable | chờ cuộc gọi nội tuyến  chờ đợi nội tuyến cho sự kiện  thêm giá trị bổ sung để đặt  tính toán toán học  hành vi lồng nhau  đặt lại giá trị trong thủ tục  có được bản sắc đối tượng sở hữu  có được giá trị trong thủ tục  xóa giá trị khỏi tập hợp  đặt giá trị trong quy trình |
| control | startOwnedBehavior | kiểm soát rõ ràng |
| creation | createLinkObject  createObject | tạo đối tượng từ liên kết  tạo đối tượng bình thường |
| destruction | destroyObject | Xóa đối tượng |
| exception | raiseException | tăng ngoại lệ trong thủ tục |
| read | readExtent  readLink  readLinkObjectEnd readLinkObjectEndQualifier readStructuralFeature | Lấy tất cả các đối tượng  Lấy giá trị liên kết  lấy giá trị từ lớp kết hợp  lấy giá trị vòng loại  lấy giá trị thuộc tính |
| time | durationObservation timeObservation | đo khoảng thời gian  lấy thời gian hiện tại |
| write | addStructuralFeatureValue clearAssociation clearStructuralFeature createLink  destroyLink removeStructuralFeatureValue | đặt giá trị thuộc tính  liên kết rõ ràng  giá trị thuộc tính rõ ràng  thêm liên kết  xóa liên kết  xóa giá trị khỏi tập hợp |

# **CHƯƠNG 6: INTERACTION VIEW**

## **1. Interaction**

Một structured classifier (phân loại có cấu trúc) định nghĩa một mối quan hệ theo ngữ cảnh. Một structured classifier (bao gồm collaboration) có thể có một tập hợp các hành vi được đính kèm áp dụng cho một tập các đối tượng được liên kết với một cá thể duy nhất của ngữ cảnh. Mô tả hành vi là một trình tự các thông điệp trao đổi bởi các đối tượng liên kết với các vai trò. Mô tả chuỗi thông báo trên structured class (lớp cấu trúc) hoặc collaboration (sự cộng tác) được gọi là interaction (sự tương tác). Một structured class (lớp cấu trúc) hoặc collaboration (sự cộng tác) có thể có bất kỳ số lượng tương tác nào, mỗi phần trong số đó mô tả một loạt các message trao đổi giũa các đối tượng trong bối cảnh để thực hiện một mục tiêu. Rất phổ biến, sự tương tác mô tả việc thực hiện các hoạt động. Các tham số của hoạt động này phục vụ như một vai trò của một sự cộng tác đại diện cho việc thực thi hoạt động.

Interaction (sự tương tác) là một tập hợp các thông điệp trong structured classifier (phân loại có cấu trúc) hoặc collaboration (sự cộng tác) được các role (vai trò) xuất hiện trên các connector (trình kết nối). Một trường hợp của một tương tác tương ứng với một trường hợp về ngữ cảnh của nó, với các object (đối tượng) bị ràng buộc với các vai trò trao đổi các trường hợp message (thông điệp) trên các link (liên kết) bị ràng buộc với các trình kết nối. Một tương tác thường mô hình thực hiện một hoạt động, trường hợp sử dụng hoặc thực thể hành vi khác.

Một message (thông điệp) giao tiếp 1 chiều giữa 2 đối tượng, một luồng kiểm soát với thông tin từ sender (người gửi) đến receiver (người gửi). Một thông điệp có thể có argument (đối số) rằng các giá trị truyền đạt từ người gửi đến người nhận. Một thông điệp có thể là một signal (tín hiệu) hay là một call (cuộc gọi).

Creation (sự tạo) của một đối tượng mói có thể là mô hình hóa như một thông diệp được gửi bởi người tạo đối tượng và được nhận bởi lớp của chính nó. Sự kiện creation được cung cấp cho các trường hợp mới như current event (sự kiện hiện hành) về quá trình chuyển đổi từ trạng thái đầu tiên ở cấp độ đầu.

Trình tự của các sự kiện và các thông điệp có thể thể hiện trong hai loại sơ đồ: sequence diagram (tập trung vào trình tự thời giao của các thông điệp) và communication diagram (tập trung vào mối quan hệ giữa cấc đối tượng mà trao đổi các thông điệp)

## **2. Sequence Diagram**

Sequence diagram hiển thị sự tương tác như một biểu đồ hai chiều.

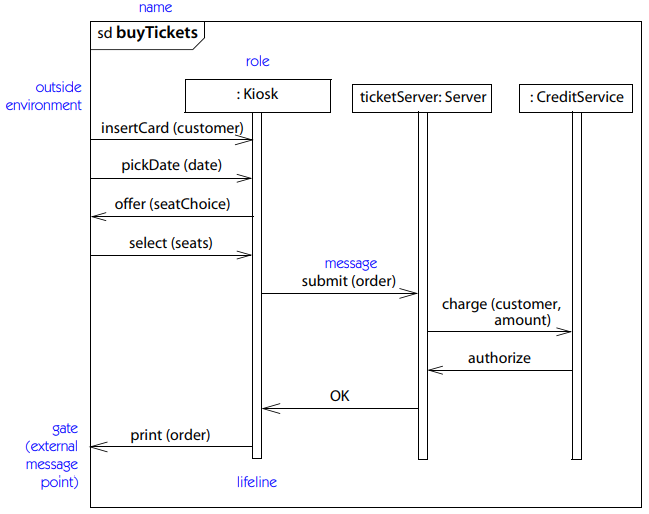
Trục đứng thể hiện thời gian. Trục ngang hiển thị các vai trò đại diện cho các object (đối tượng) riêng lẻ trong cộng tác. mỗi vai trò được đại diện bởi một cột dọc chứa một biểu tượng đầu và một đường thẳng đứng – lifeline.

Trong suốt thời gian đối tượng tồn tại, được thể hiện bằng vạch đứt. Trong thời gian Execution specification (đặc tả thực thi) đối tượng trong trạng thái hoạt động, lifeline được vẽ như đường đôi.

Nhìn chung, sequence diagram chỉ thể hiển trình tự của các thông điệp và không thể hiển chính xác thời gian.

Một message (thông điệp) được thể hiện bằng mũi tên từ lifeline của một đối tượng đến lifeline của một đối tượng khác. Các mũi tên được sắp xếp theo thời gian trình tự xuống sơ đồ. Một asynchronous message (thông điệp bất đồng bộ ) được hiển thị với đầu mũi tên thanh.

Hình 9-1 thể hiện một sequence diagram điển hình với asynchronous message.



Hình 9-1. Sequence Diagram

## **3. Execution Specification (đặc tả thực thi)**

Execution specification (activation – kích hoạt) là việc thực hiện các thủ tục để thực hiện, bao gồm cả thời gian nó đợi các thủ tục lồng nhau để thực hiện. Nó được hiển thị bởi một đường đôi lại đặt một phần của lifeline trong một sequence diagram.

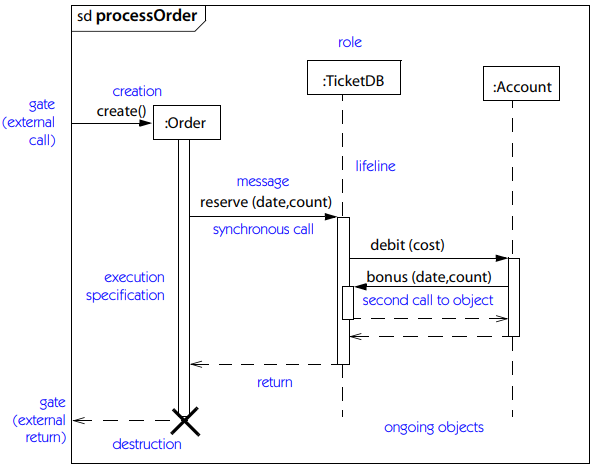
Một call được hiển thị bằng một mũi tên dẫn đến đầu đặc tả thực hiện cuộc gọi. một synchronous call (cuộc gọi đồng bộ )được hiển thị với đầu mũi tên tam giác đầy.

Một cuộc gọi đệ quy xảy ra khi điều khiển nhập lại một hoạt động trên một đối tượng, nhưng cuộc gọi thứ hai là sự thực thi riêng biệt từ lần đầu tiên. đệ quy hoặc một cuộc gọi lồng nhau đến một hoạt động khác trên cùng một đối tượng được hiển thị trong một sequence diagram bằng cách xếp chồng các đường kích hoạt

Trả về của điều khiển từ một cuộc gọi được thể hiển là mũi tên nét đứt.

Hình 9-2 chỉ ra sequence diagram với luồng thủ tục của điều khiển ,bao gồm một cuộc gọi thứ 2 đến một đối tượng lồng nhau trong cuộc gọi khác và sự khởi tạo một đối tương trong lúc tính toán.

Active object là gốc của một chồng các thực thi. Mỗi đối tượng hoạt động được thể hiện bởi đường đôi trên mỗi cạnh của biểu tượng đầu. Đối tượng được gọi bởi active object là passive object, chúng nhận được điều kiển chỉ khi được gọi.



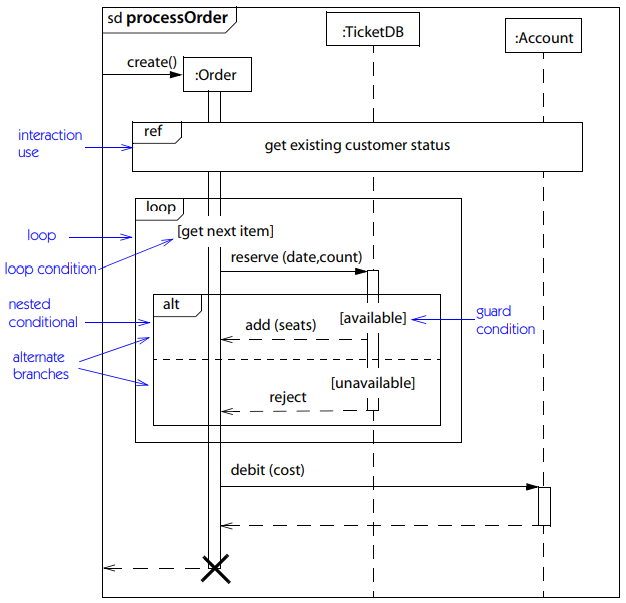
Hình 9-2. Sequence diagram with execution specification

## **4. Structured control constructs ( Cấu trúc điều khiển có cấu trúc )**

Dòng điều khiển tuần tự được hiển thị dễ dàng trong một biểu đồ trình tự như một chuỗi các thông điệp. Dòng điều khiển phức tạp hơn có thể được hiển thị bằng cách sử dụng combined fragments (các đoạn kết hợp)

Combined fragment có một từ khóa và một hoặc nhiều subframent phân lớp (gọi là interaction operands). Con số và ý nghĩa của các subframent (phân lớp) dựa trên từ khóa. Ví dụ ở hình 9-3

Interaction use là một tham chiếu đến sự tương tác khác, thường được xác định trong sequence diagram riêng của nó.



Hình 9-3 Structured control constructs

Loop có một phân lớp, thực thi cho đến khi điều kiện bảo vệ đầu tiên trong phân lớp là đúng

Conditional fragment (từ khóa là alt ) có 2 hoặc nhiều phân lớp,mỗi phân lớp đều có điều kiện bảo vệ ban đầu. Khi đạt điều kiện, phân lớp có điều kiện đó được thực thi. Nếu đúng nhiều hơn một thì một trong số các phân lớp sẽ được chọn là không xác định để thực thi. Nếu không có cái nào đúng thì ko có sự thi hành nào phù hợp với đặc tả.

Optional fragment (từ khóa là opt ) là trường hợp đặc biệt với phân lớp đơn được thực hiện nếu guard condition (điều kiện bảo vệ ) là đúng và bỏ qua nếu điều kiện sai.

Parallel fragment (từ khóa là par ) có 2 hoặc nhiều phân lớp. Khi đến frament thì tất cả các phân lớp được thực hiện đồng thời. Khi tất cả các phân lớp thực thi xong, các thực thi đồng thời nối lại với nhau thành một luồng.

Có một số thứ tự, các cấu trúc có cấu trúc cụ thể hơn.

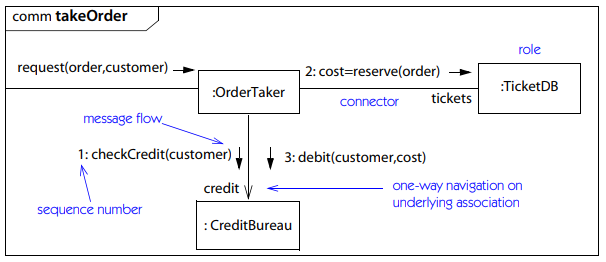
Hình 9-3 chỉ ra một sequence diagram có chứa loop với điều kiện lồng nhau.

## **5. Communication Diagrams**

Communication Diagram dựa trên ngữ cảnh được cung cấp bởi structured classifier ( bao gồm collaboration ). Các vai trò và kết nối mô tả cấu hình của đối tượng và các liên kết có thể xảy ra khi một trường hợp trong một ngữ cảnh được thực thi. Khi ngủ cảnh được khởi tạo, đối tượng bị ràng buộc với vai trò và liên kết bị ràng buộc với các kết nối. Các kết nối cũng có thể bị ràng buộc với các loại liên kết tạm thời khác nhau. Chỉ các đối tượng được tham gia vào ngữ cảnh mới được mô hình hóa.

Thông điệp giữa các vai trò hiển thị dưới dạng mũi tên được gắn với kết nối. Mỗi thông điệp có một sequence number (số thứ tự), một guard condition (điều kiện bảo vệ ) tùy chọn, tên và danh sách argument (đối số ), và một tùy chọn trả về tên dữ liệu . Số thứ tự bao gồm tên của một luồng. tất cả các thông điệp trên cùng luồng theo thứ tự tuần tự. Các thông điệp trên các luồng khác nhau là đồng thời trừ khi có sự phụ thuộc trình tự

Hình 9-4 thể hiện communication diagram



Hình 9-4. Communicate diagram

***Communication và sequence diagrams:*** cả hai đều thể hiện sự tương tác , nhưng thể theo cách khác nhau. Sequence diagram thể hiển thời gian rõ ràng nhưng không thể hiện mối quan hệ giữa các đối tượng rõ ràng. Communicate diagram thì ngược lại. Sequence digram thường hữu ích nhất về việc thể hiện kịch bản; Communication diagram thì hữu ích cho việc thể hiện chi tiết thiết kế các thủ tục.

# **CHƯƠNG 7: DEPLOYMENT VIEW**

## **1. Tổng quan**

Deployment View cho thấy sự sắp xếp vật lý của các node. Một node là một tài nguyên tính toán thời gian chạy, chẳng hạn như một thiết bị máy tính khác. Khi chạy, các node có thể chứa các tạo phẩm, các thực thể vật lý như các tệp. Mối quan hệ biểu hiện giữa yếu tố thiết kế, chẳng hạn như các thành phần và các hiện vật thể hiện, chúng trong hệ thống phần mềm. Khung nhìn triển khai có thể làm nổi bật các nút cổ chai hiệu suất do vị trí của các tạo phẩm biểu hiện các thành phần phụ thuộc lẫn nhau trên các nút khác nhau.

## **2. Node**

Một node mô hình một tài nguyên tính toán thời gian chạy có bộ nhớ ít nhất là một bộ nhớ và thường là khả năng xử lý. Các node có thể có khuôn mẫu để phân biệt các loại tài nguyên khác nhau, chẳng hạn như CPU, thiết bị và bộ nhớ. Các nút có thể chứa các hiện vật.

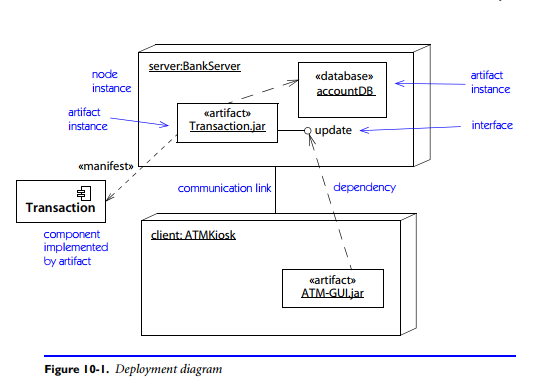
Kiểu node được hiển thị dưới dạng khối lập phương cách điệu với tên được nhúng. Một thể hiện nút được hiển thị dưới dạng khối lập phương với chuỗi tên gạch chân có tên và loại node.

Các liên kết giữa các node thể hiện đường dẫn giao tiếp. Các hiệp hội có thể có khuôn mẫu để phân biệt các loại đường dẫn khác nhau.

Các node có thể có mối quan hệ khái quát hóa để liên kết mô tả chung của một node với một biến thể cụ thể hơn.

## **3. Artifact**

Một tạo phẩm mô hình một thực thể vật lý chẳng hạn như tệp. Một tạo phẩm được hiển thị bằng một hình chữ nhật có từ khóa «artifact». Sự hiện diện của một tạo phẩm được thể hiện bằng cách lồng ghép biểu tượng vật lý bên trong biểu tượng node.



Các loại hiện vật khác nhau, chẳng hạn như cơ sở dữ liệu, trang web, tệp thi hành hoặc tập lệnh, có thể được đánh dấu bằng các khuôn mẫu.

Nếu một tạo phẩm thực hiện một thành phần hoặc lớp ngoài, một mũi tên đứt quãng với từ khóa «manifest» được rút ra từ biểu tượng tạo tác cho biểu tượng của conponent mà nó thực hiện. mối quan hệ này được gọi là biểu hiện.

# **CHƯƠNG 8: MODEL MANAGEMENT VIEW**

## **1. Tổng quan**

Bất kỳ hệ thống lớn nào cũng phải được chia thành các đơn vị nhỏ hơn để con người có thể làm việc với số lượng thông tin hạn chế cùng một lúc và do đó các nhóm làm việc không can thiệp vào công việc của nhau. Mô hình mangement bao gồm các gói (bao gồm các loại gói đặc biệt) và mối quan hệ phụ thuộc giữa các gói.

## **2. Package**

Một gói là một phần của một mô hình. Mỗi phần của mô hình phải thuộc về một gói. Người lập mô hình có thể phân bổ nội dung của một mô hình để thiết lập các gói. Nhưng để có thể thực hiện được, phân phối phải tuân theo một số nguyên tắc hợp lý, chẳng hạn như chức năng chung, triển khai kết hợp chặt chẽ và quan điểm chung. UML không áp đặt một quy tắc để soạn thảo các gói, nhưng một sự phân hủy tốt thành các gói sẽ giúp nâng cao khả năng bảo trì mô hình.

Các gói chứa các phần tử mô hình cấp cao nhất, chẳng hạn như các lớp và các mối quan hệ của chúng, các máy trạng thái, biểu đồ trường hợp sử dụng, các tương tác và các cộng tác - bất kỳ thứ gì không có trong một số phần tử khác. Các phần tử như thuộc tính, hoạt động, trạng thái, đường truyền và thông điệp được chứa trong các phần tử khác và không xuất hiện dưới dạng nội dung trực tiếp của các gói. Mỗi phần tử cấp cao nhất có một gói mà nó được khai báo. Đây là gói "nhà" của nó. Nó có thể được tham chiếu trong các gói khác, nhưng nội dung của phần tử được sở hữu bởi gói nhà. Trong một hệ thống điều khiển cấu hình, người lập mô hình phải có quyền truy cập vào gói hoe để sửa đổi nội dung của một phần tử. Điều này cung cấp một cơ chế kiểm soát truy cập để làm việc với các mô hình lớn. Các gói cũng là các đơn vị cho bất kỳ cơ chế phiên bản nào.

Các gói có thể chứa các gói khác. Có một gói gốc gián tiếp chứa toàn bộ mô hình của một hệ thống. Có một số cách có thể để tổ chức các gói cho hệ thống. Chúng có thể được sắp xếp theo chế độ xem, theo chức năng hoặc theo bất kỳ cơ sở nào khác mà người chọn lựa chọn. Các gói là các đơn vị tổ chức phân cấp mục đích chung của các mô hình UML. Chúng có thể được sử dụng để lưu trữ, kiểm soát truy cập, quản lý cấu hình và các thư viện cấu trúc con chứa các đoạn mô hình có thể tái sử dụng.

Nếu các gói được chọn tốt, để các cấu trúc tổ chức (các gói) tương ứng với các cấu trúc thiết kế (các thành phần), chúng phản ánh kiến ​​trúc mức cao của hệ thống phân hủy của nó trong các hệ thống con và các phụ thuộc của chúng. Các gói cắt ngang qua các hệ thống phụ thường gây ra rắc rối giữa các nhóm thiết kế. Sự phụ thuộc giữa các gói tóm tắt sự phụ thuộc giữa các nội dung gói.phân tách tốt thành các gói sẽ nâng cao khả năng bảo trì mô hình.

Các gói chứa các phần tử mô hình cấp cao nhất, chẳng hạn như các lớp và các mối quan hệ của chúng, các máy trạng thái, biểu đồ trường hợp sử dụng, các lần lặp lại và sự cộng tác - mọi thứ không có trong một phần tử khác. Các phần tử như thuộc tính, opratios, trạng thái, đường truyền và thông điệp được chứa trong các phần tử khác và không xuất hiện dưới dạng nội dung trực tiếp của các gói. Mỗi phần tử cấp cao nhất có một gói mà nó được khai báo. Đây là gói "nhà" của nó. Nó có thể được tham chiếu trong các gói khác, nhưng nội dung của phần tử được sở hữu bởi gói nhà. Trong một hệ thống điều khiển cấu hình, người lập mô hình phải có quyền truy cập vào gói hoe để sửa đổi nội dung của một phần tử. Điều này cung cấp một cơ chế điều khiển truy cập để làm việc với các mô hình lớn. Các gói cũng là các đơn vị cho bất kỳ cơ chế phiên bản nào.

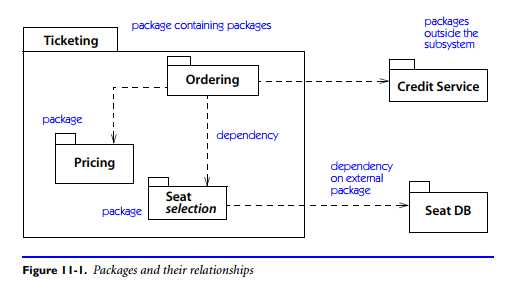
Các gói có thể chứa các gói khác. Có một gói gốc gián tiếp chứa toàn bộ mô hình của một hệ thống. Có các cách có thể có để tổ chức các gói cho hệ thống. Chúng có thể được sắp xếp theo chế độ xem, theo chức năng hoặc theo bất kỳ cơ sở nào khác mà người lập mô hình lựa chọn. Các gói là các đơn vị tổ chức phân cấp mục đích chung của các mô hình UML. Chúng có thể được sử dụng để lưu trữ, kiểm soát truy cập, quản lý cấu hình và các thư viện cấu trúc con chứa các đoạn mô hình có thể tái sử dụng.

Nếu các gói được chọn tốt, để các cấu trúc tổ chức (các gói) tương ứng với các cấu trúc thiết kế (các thành phần), chúng phản ánh kiến ​​trúc mức cao của hệ thống phân hủy của nó trong các hệ thống con và các phụ thuộc của chúng. Các gói cắt ngang qua các hệ thống con gây ra rắc rối giữa các nhóm thiết kế. Sự phụ thuộc giữa các gói tóm tắt sự phụ thuộc giữa các nội dung gói.

## **3. Phụ thuộc vào gói**

Phụ thuộc phát sinh giữa các phần tử riêng lẻ, nhưng trong một hệ thống có kích thước bất kỳ, chúng phải được xem ở mức cao hơn. Các phụ thuộc giữa các gói tóm tắt các phụ thuộc giữa các phần tử trong chúng - nghĩa là các gói phụ thuộc có thể dẫn xuất từ ​​các phụ thuộc giữa các phần tử riêng lẻ.

Sự hiện diện của sự phụ thuộc giữa các gói ngụ ý rằng tồn tại trong một phương pháp từ dưới lên (một tuyên bố tồn tại), hoặc được phép tồn tại sau này trong một phương pháp từ trên xuống (một ràng buộc trong thiết kế trong tương lai), ít nhất một phần tử quan hệ của loại phụ thuộc giữa các phần tử riêng lẻ trong các gói tương ứng. Nó là một "tuyên bố tồn tại" và không ngụ ý rằng tất cả các phần tử của gói có sự phụ thuộc. Đó là một lá cờ cho người lập mô hình rằng có thêm thông tin, nhưng sự phụ thuộc cấp gói không chứa chính thông tin thêm; nó chỉ là một bản tóm tắt.



Cách tiếp cận từ trên xuống phản ánh kiến ​​trúc hệ thống tổng thể. Cách tiếp cận từ dưới lên có thể được tạo tự động từ các phần tử riêng lẻ. Cả hai cách tiếp cận đều có vị trí của họ trong mô hình hóa, ngay cả trên một hệ thống đơn lẻ.

Nhiều phụ thuộc cùng loại giữa các phần tử riêng lẻ được tổng hợp thành một phụ thuộc mức gói đơn trong các phần tử riêng lẻ chứa các khuôn mẫu (chẳng hạn như các kiểu sử dụng khác nhau), khuôn mẫu có thể được bỏ qua trong sự phụ thuộc cấp gói. . Các gói được vẽ như hình chữ nhật với các tab trên chúng (biểu tượng "thư mục" trên máy tính để bàn). Phụ thuộc được hiển thị dưới dạng mũi tên đứt quãng.

## **4. Hiển thị**

Gói là không gian tên cho các phần tử của nó. Nó thiết lập khả năng hiển thị của các yếu tố của nó. Các phần tử được chỉ định bởi các gói có thể có khả năng hiển thị công khai hoặc riêng tư. Một gói chỉ xem các phần tử của các gói khác đã được hiển thị công khai bởi gói chứa chúng. Các phần tử có khả năng hiển thị riêng chỉ hiển thị trong gói chứa chúng bất kỳ gói nào được lồng trong gói đó.

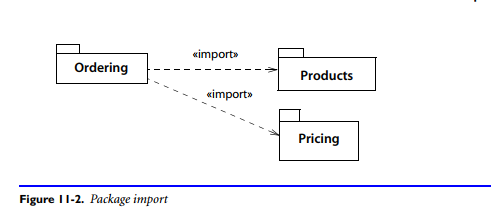
Khả năng hiển thị cũng áp dụng cho các tính năng của các lớp (thuộc tính một hoạt động). Một phần tử có khả năng hiển thị công khai có thể được nhìn thấy bởi các lớp khác trong cùng một gói hoặc trong các gói khác có thể thấy lớp và bởi con cháu của lớp. Một tính năng với khả năng hiển thị riêng chỉ có thể được nhìn thấy bởi lớp riêng của nó. Một tính năng có khả năng hiển thị được bảo vệ có thể được nhìn thấy bởi lớp hoặc sự phụ thuộc của một lớp. Một tính năng với khả năng hiển thị gói có thể được nhìn thấy bởi lớp trong cùng một gói, nhưng không phải bởi các lớp trong các gói khác, ngay cả khi chúng có thể nhìn thấy lớp đó.

Một package được lồng trong một package khác của vùng chứa và có toàn quyền truy cập vào nội dung của nó (quá cảnh). Tuy nhiên, một bộ lọc có thể không nhìn thấy bên trong các package lồng nhau của nó mà không có khả năng hiển thị thích hợp; các nội dung được đóng gói.

## **5. Import**

Một package có thể import các phần tử từ một package khác để thêm tên của chúng vào không gian tên của nó. Import không thêm bất kỳ sức mạnh mô hình nào, nhưng nó đơn giản hóa các biểu thức văn bản xuất hiện trong các ràng buộc và cuối cùng là mã. Một phần tử có thể được nhập khẩu nếu nó được hiển thị cho gói nhập. Một gói cũng có thể nhập toàn bộ gói, tương đương với việc nhập tất cả các phần tử hiển thị trong đó.

Phần tử đã nhập trở thành thành phần hiển thị của gói nhập dưới tên được nhập của nó (có thể khác với tên gốc, mặc dù chúng thường giống nhau). Mối quan hệ truy cập thêm một phần tử vào không gian tên mà không làm cho nó công khai trong gói nhập khẩu.



## **6. Model**

Model là một package bao gồm mô tả đầy đủ về một chế độ xem cụ thể của một hệ thống. Nó cung cấp một mô tả khép kín của một hệ thống từ một quan điểm. Nó không có phụ thuộc mạnh mẽ. Mối quan hệ theo dõi là một dạng phụ thuộc yếu giữa các phần tử trong các mô hình khác nhau, ghi chú sự hiện diện của một số kết nối mà không có các tác động ngữ nghĩa cụ thể.

Thông thường, một model được cấu trúc bằng cây. Các ràng buộc package gốc trong chính nó các package lồng nhau cấu thành toàn bộ chi tiết của hệ thống từ quan điểm đã cho. Một model có thể được hiển thị như là một package với một trang sức tam giác nhưng thường có ít điểm trong việc hiển thị các model như là biểu tượng.