Unified Modeling Language
UML 2.0

Phân công .....

Trần Thanh Tùng - 03110276
Giới thiệu
Use Cases Diagrams
Activity Diagrams
Class Diagrams
Advanced Class Diagrams

Vũ Thành Huy - 03110098
Object Diagrams
Sequence Diagrams
Communication Diagrams
Timing Diagrams
Interaction Overview Diagrams

Nguyễn Xuân Thanh - 03110213
Composite Structures
Component Diagrams
Packages
State Machine Diagrams
Deployment Diagrams

Mục lục

I. Giới thiệu

UML là ngôn ngữ mô hình hóa chuẩn dùng trong phát triển chương trình, hệ thống.

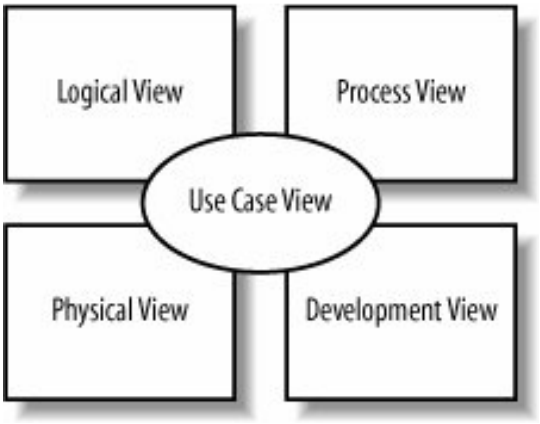
UML giúp người thiết kế, người lập trình hiểu được các thành phần cần thiết, các yêu cầu, các qui trình của chương trình, hệ thống một cách chính xác.

UML 2.0 sử dụng các kiểu lưu đồ sau

Các kiểu lưu đồ (lược đồ)	Có khả năng mô tả
Use case	Tương tác giữa hệ thống và người dùng hoặc hệ thống ngoài. Cũng có thể ánh xạ các yêu cầu vào hệ thống
Activity	Các hoạt động tuần tự hoặc song song của hệ thống
Class	Lớp, kiểu, interface và mối liên hệ giữa chúng
Object	Object là thể hiện của lớp được định nghĩa trong Class Diagram
Sequence	Tương tác giữa các đối tượng khi thứ tự tương tác là quan trọng
Communication	Cách mà các đối tượng tương tác và các kết nối cần thiết để hỗ trợ sự tương tác đó
Timing	Tương tác giữa các đối tượng khi yếu tố thời gian được quan tâm
Interaction Overview	Dùng để gộp các lưu đồ sequence, communication và timing lại để thể hiện sự tương tác quan trọng trong hệ thống
Composite Structure	Các thành phần bên trong của lớp, thành phần và có thể mô tả các mối quan hệ của lớp trong một ngữ cảnh
Component	Các thành phần quan trọng trong hệ thống và giao diện dùng để tương tác giữa các thành phần
Package	Tổ chức phân cấp của nhóm các lớp và thành phần
State Machine	Trạng thái của đối tượng xuyên suốt quá trình sống và các biến cố có thể làm thay đổi trạng thái này
Deployment	Cách hệ thống cuối cùng được sắp xếp (cài đặt) trong tình huống cụ thể

UML không phải chỉ gồm các lược đồ, mà UML được dùng để mô hình hóa hệ thống của bạn. Một lược đồ (diagram) chỉ mà phần trong mô hình hệ thống của bạn. Một lược đồ cụ thể sẽ thể hiện được một vài phần chứ không phải tất cả hệ thống của bạn.
Tập hợp tất cả các yếu tố mô tả hệ thống của bạn cùng với các mối liên kết giữa chúng mới tạo thành mô hình của bạn. Vì vậy, khi bạn dùng các công cụ UML để làm việc với các khái niệm của UML thì cần nhớ rằng bạn chỉ đang thao tác trên khung nhìn của mô hình của bạn.

Có rất nhiều cách để chia mô hình của bạn thành các góc độ nhìn. Ở đây ta sử dụng mô hình Kruchten's 4+1 để cho bạn thấy được vai trò của từng lược đồ trong mô hình chung.



Mô hình Kruchten's 4 + 1

Logical View :

Mô tả tóm tắt các phần của hệ thống. Được dùng để mô hình hóa hệ thống bao gồm những cái gì? và cách tương tác giữa các phần của hệ thống. Các lược đồ được sử dụng : Lược đồ class, object, state machine và interaction

Process View:

Mô tả các qui trình trong hệ thống. Đặc biệt hữu ích khi muốn thể hiện những gì phải xảy ra trong hệ thống. Lược đồ được dùng : Lược đồ Activity.

Development View:

Mô tả cách các phần của hệ thống được tổ chức thành module và component. Rất tiện trong việc quản lý các lớp (layer) trong cấu trúc hệ thống. Các lược đồ sử dụng : Lược đồ Package và Component.

Physical View:

Mô tả cách phần thiết kế của bạn hệ thống, như được mô tả trong 3 góc độ nhìn trên, được đem và thực tế. Lược đồ trong góc nhìn này thể hiện cách các phần được trừu tượng hóa ánh xạ và hệ thống thực tế. Lược đồ được sử dụng : Lược đồ Deploment.

Use case View:

Mô tả các tính năng của hệ thống đang được mô hình hóa theo hướng từ ngoài nhìn vào. Góc nhìn này cần để mô tả các vấn đề hệ thống sẽ giải quyết. Các lược đồ còn lại dựa trên lược đồ này. Lược đồ được sử dụng : Lược đồ Use case, phần mô tả, và lược đồ tổng quát.

II. Mô hình hóa các yêu cầu : Use Cases Diagram

1. Giới thiệu :

Use case là trường hợp hay tình huống mà hệ thống được sử dụng để đáp ứng yêu cầu của người dùng.

2. Các khái niệm :

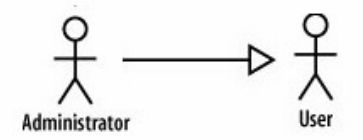
a. Actor

Là tác nhân ngoài tác động vào hệ thống. Actor có thể là một người, một hệ thống khác hoặc một đối tượng

Cần chú ý khi xác định Actor. Không phải lúc nào Actor cũng dễ nhận ra như một hệ thống ngoài, một người. Actor có thể là giờ hệ thống
Ký hiệu :

b. Mối liên hệ giữa các actor

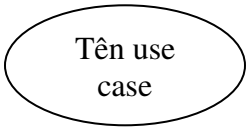
Sử dụng mũi tên tổng quát hoá để thể hiện sự thừa kế giữa các actor.
Ký hiệu :



Actor Administrator
thừa kế từ Actor User

c. Use case

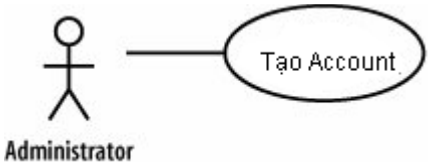
Ký hiệu :



d. Communication Lines - Đường tương tác

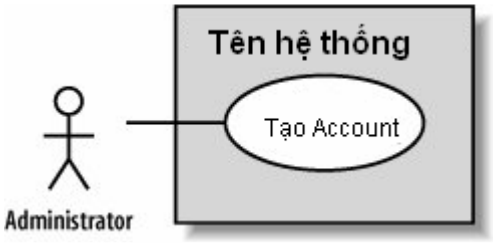
Dùng để nối giữa Actor và Use Case thể hiện Actor tham gia vào Use case

Ký hiệu



Administrator tham gia vào use case tạo Account

System Boundaries
Theo mặc định Actor (tác nhân ngoài) và Use case (thành phần trong của hệ thống) đã tách rời nhau. Để thể hiện rõ, UML cung cấp khái niệm System Boundaries (Biên của hệ thống).
Ký hiệu



e. Use Case Descriptions - Bảng mô tả của Use case

Lược đồ bao gồm Actor và Use case biểu diễn được các chức năng của hệ thống và các tương tác của Actor với các Use case này.
Để cung cấp đầy đủ thông tin về các chức năng của hệ thống như các bước thực hiện, các điều kiện thực hiện … UML cung cấp khái niệm Use Case Description

Các thông tin trong bảng mô tả

Thông tin	**Ý nghĩa và tầm quan trọng**
Các yêu cầu liên quan	Chỉ ra yêu cầu mà use case đáp ứng một phần hoặc toàn bộ
Hoàn cảnh	Chỉ ra vị trí của use case trong hệ thống và vì sao cần use case này
Điều kiện tiền đề	Điều gì cần phải có trước khi use case thực hiện
Trạng thái của hệ thống khi use case thành công	Trạng thái của hệ thống khi use case thực hiện thành công
Trạng thái của hệ thống khi use case thất bại	Trạng thái của hệ thống khi use case thực hiện không thành công
Tác nhân chính	Tác nhân chính tham gia vào use case. Thường gồm những tác nhân kích hoạt hoặc trực tiếp nhận thông tin từ việc thực hiện use case
Tác nhân phụ	Tác nhân tham gia nhưng không giữ vai trò chính trong việc thực hiện use case
Điều kiện kích hoạt	Biến cố được kích hoạt bởi tác nhân để làm cho use case thực hiện
Luồng chính	Phần mô tả các bước quan trọng khi thực hiện use case trong điều

Thông tin	**Ý nghĩa và tầm quan trọng**
	kiện bình thường
Phần mở rộng	Phần mô tả các bước rẽ nhánh của phần mô tả trên

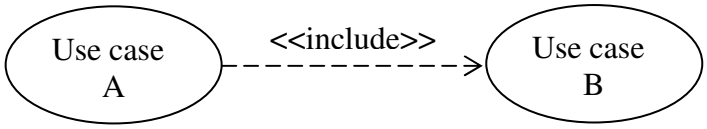
Ví dụ:

Use case name	**Đăng ký môn học đúng tiến độ**	
Related Requirements	Yêu cầu về đăng ký môn học đúng tiến độ do Khoa hay PĐT xếp lịch	
Goal In Context	SV đăng ký môn học đúng tiến độ	
Preconditions	SV đang đăng ký môn học	
Successful End Condition	SV có PĐKMH đúng tiến độ các môn do PĐT xếp lịch hoặc các môn do Khoa xếp lịch	
Failed End Condition	SV không có PĐKMH đúng tiến độ các môn do PĐT xếp lịch hoặc các môn do Khoa xếp lịch	
Primary Actors	SV	
Secondary Actors		
Trigger	SV chọn chức năng đăng ký môn học đúng tiến độ	
Include Cases	Không	
Main Flow	Step	Action
	1	SV tra cứu lịch học vụ
	2	SV chọn nhóm cho từng môn
	3	SV hủy các môn không học
	4	Submit
	5	Hệ thống tạo PĐKMH
	6	Lưu PĐKMH vào cơ sở dữ liệu PĐK SV
	7	Thông báo PĐKMH được lập
Extentions	Step	Branching Action
	6.1	Lưu không thành công
	6.2	Hệ thống thông báo đăng ký không thành công

f. Mối quan hệ giữa các use case

Trong thiết kế, bạn sẽ thấy xuất hiện các công việc được thực hiện giống nhau, lặp đi lặp lại trong các chức năng. Hoặc trong một chức năng có quá nhiều bước, bạn muốn thể hiện các bước quan trọng một cách rõ ràng nhất.
Với UML bạn có thể sử dụng lại, lựa chọn hoặc đặc biệt hóa các bước giữa các use case.

Quan hệ <<include>>
Thể hiện một use case sử dụng lại toàn bộ các bước của một use case khác.
Ký hiệu :

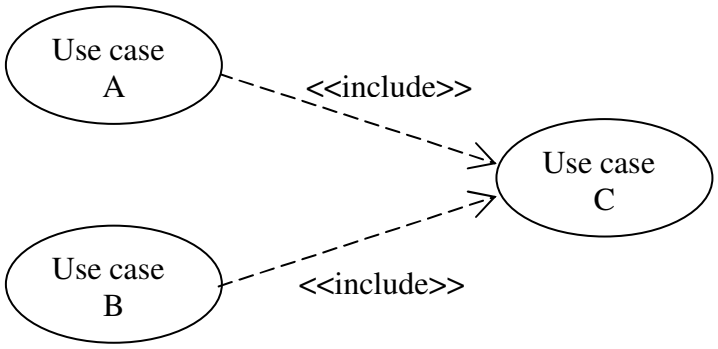


Use case A sử dụng toàn bộ các bước của use case B

Trong bảng mô tả phần Include được thể hiện như sau :

Main Flow	Step	Action
	…	…
	n include::A	Mô tả chức năng A
	…	…

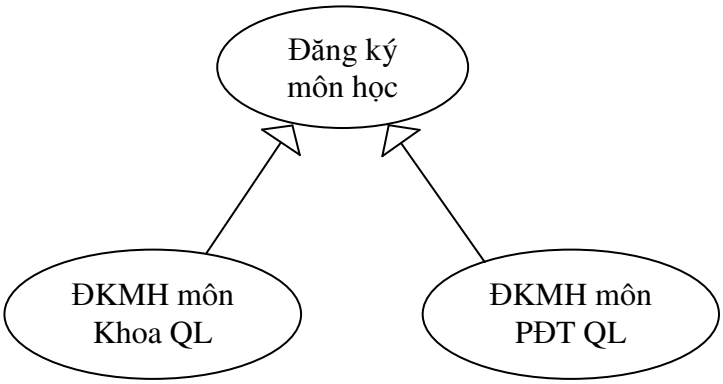
Ví dụ 2:



Use case A, B sử dụng chung use case C

g. Đặc biệt hoá use case

Mối quan hệ thừa kế, và có chỉnh sửa
Ký hiệu – Ví dụ:

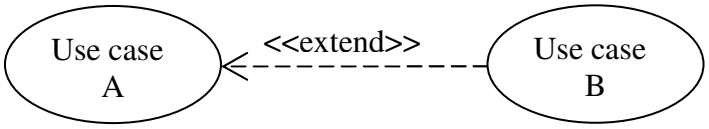


Mối quan hệ thừa kế

Cần lưu ý: Mọi bước, mọi mối liên hệ với các tác nhân ngoài trong use case "cha" bắt buộc phải có trong use case "con". Nếu bạn không muốn use case cụ thể phải thực hiện tất cả các bước của use case tổng quát thì nên sử dụng quan hệ <<include>> hoặc là <extend>> trong phần tiếp theo.

Quan hệ <<extend>>

Khác với quan hệ <<include>>, <<extend>> thể hiện một use case có thể sử dụng lại toàn bộ một use case khác hoặc là không sử dụng gì.
Ký hiệu :



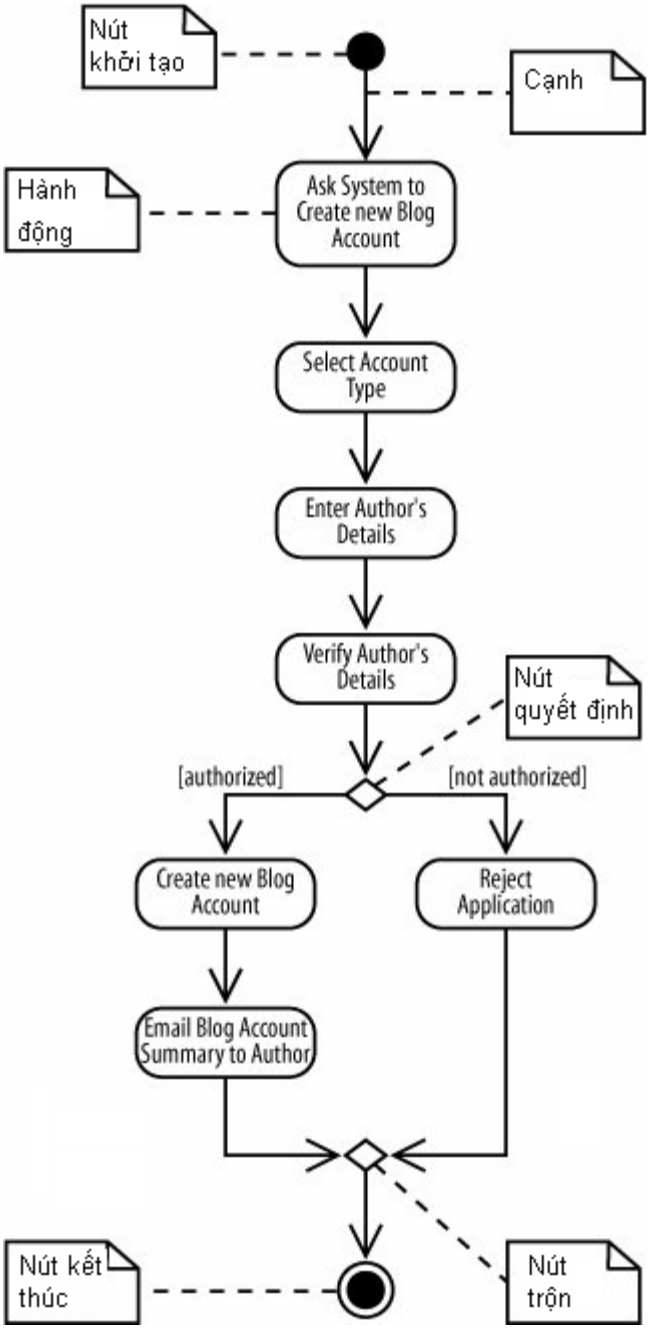
Use case A extend Use case B

III. Mô hình hóa luồng công việc của hệ thống : Activity Diagrams

1. Giới thiệu

Lược đồ Use case mô tả hệ thống cần làm gì (What). Lược đồ hoạt động cho phép chỉ rõ hệ thống làm như thế nào để hoàn thành các nhiệm vụ đó (How).
Trong Process View chỉ sử dụng lược đồ hoạt động.
Lược đồ hoạt động thể hiện chi tiết các bước trong bảng mô tả của use case.

2. Các khái niệm:

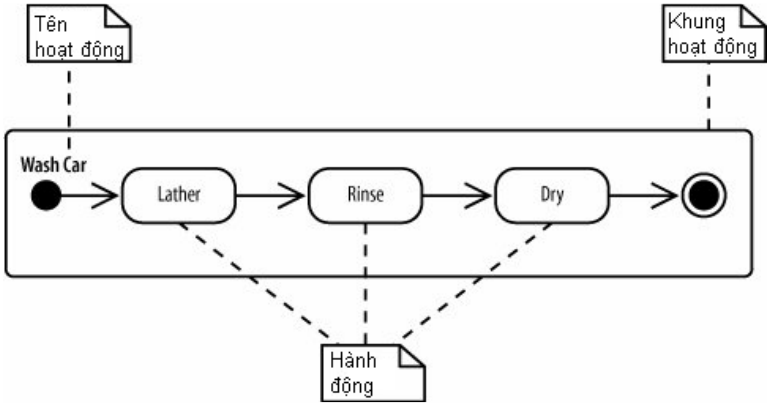


a. Hoạt động và hành động

Một hoạt động bao gồm nhiều hành động.. Một hoạt động là một quá trình, qui trình. Một hành động là một bước của qui trình dó

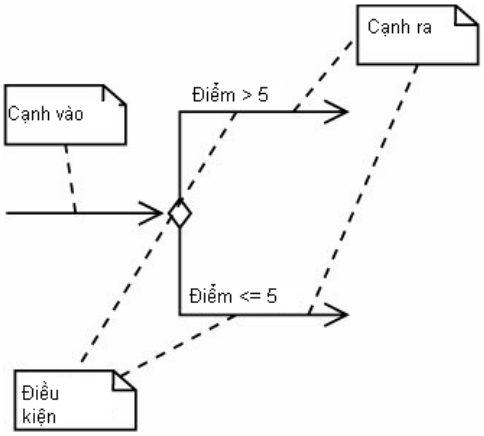
b. Khung hoạt động – Activity Frame

Toàn bộ một hoạt động được bao bởi một hình chữ nhật, tròn góc với tên của hoạt động nằm ở phía trên bên trái. Khung hoạt động thường sử dụng khi trong một sơ đồ có nhiều hoạt động.
Tuy nhiên, khung hoạt động là không bắt buộc, ta có thể bỏ đi.
Ví dụ:



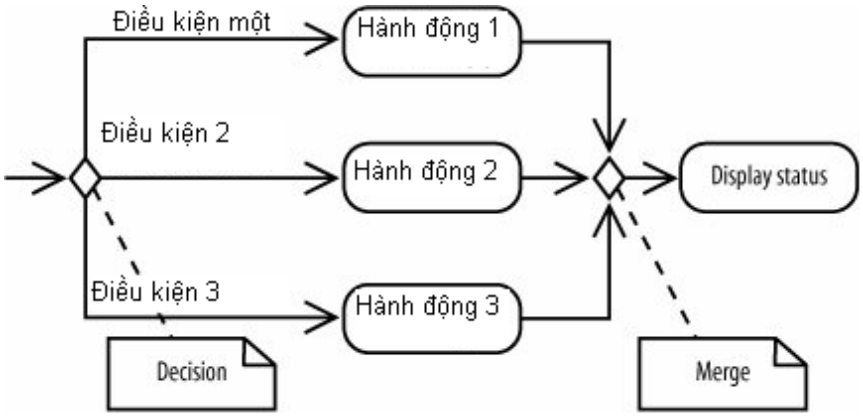
c. Decisions và Merges

Decisions được dùng khi ta muốn thực hiện các chuỗi hành động khác nhau tùy vào từng trường hợp cụ thể.
Ký hiệu



Được ký hiệu là hình thoi. Có một cạnh vào và nhiều cạnh ra. Trên mỗi cạnh có biểu thức điều kiện để trong ngoặc vuông. Biểu thức điều kiện này sẽ quyết định cạnh được chọn.

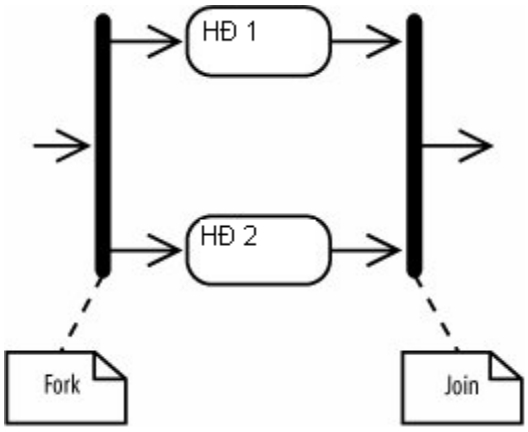
Các cạnh ra sẽ hội về nút trộn (Merge). Nút trộn đánh dấu kết thúc hành động bắt đầu ở nút quyết định.
Ký hiệu



Các lưu ý:
Nếu có 2 biểu thức điều kiện cùng đúng, UML sẽ chọn đường bất kỳ. Vì vậy nên đưa ra các biểu thức tách bạch, loại trừ nhau.
Nếu điều kiện không đầy đủ trường hợp, khi đó hoạt động sẽ bị kẹt tại nút quyết định. Ta có thể sử dụng biểu thức "else" để tránh trường hợp này.

d. Biểu diễn các hành động diễn ra song song

UML sử dụng khái niệm "fork" và "join" để biểu diễn các hành động diễn ra song song
Ký hiệu



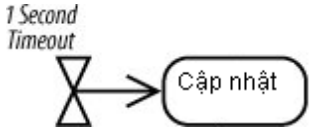
Các hành động nằm giữa "Fork" và "Join" không nhất thiết phải kết thúc đồng thời. Nhưng các hành động này phải thực hiện xong phải hoàn tất thì hành động sau "Join" mới được thực hiện

e. Biến cố thời gian

Dùng để biểu diễn các hành động diễn ra sau hành động trước một khoảng thời gian hoặc để biểu diễn hành động diễn ra sau một khoảng thời gian nhất định (chu kỳ)

Hành động 2 thực hiện sau khi hành động 1 kết thúc 3 ngày

Hành động thực hiện sau một khoảng thời gian nhất định

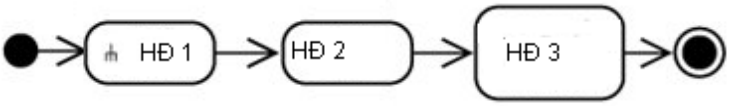


1 giây "Cập nhật" một lần

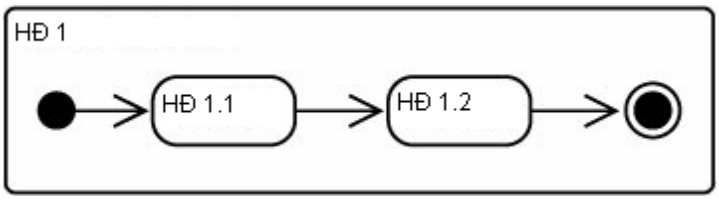
f. Gọi một hoạt động khác

Ta có thể sử dụng khái niệm một hành động gọi một hành động khác để lược đồ được gọn hơn
Ký hiệu : 

Ví dụ :



Node "HĐ1" gọi hoạt động "HĐ 1"



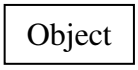
Lược đồ của hoạt động "HĐ"

Hoạt động được gọi vẫn có các nút khởi tạo và nút kết thúc. Tuy nhiên khi kết thúc, sẽ trả quyền cho nút sau nút đã gọi nó. Nên dùng frame activity và frame name trong trường hợp này.

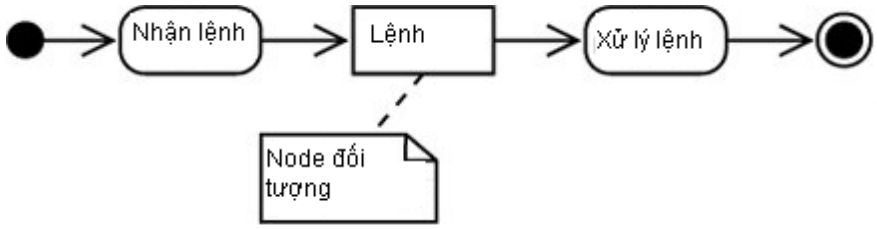
g. Đối tượng

Đối tượng dữ liệu là một phần quan trọng của hệ thống. UML cung cấp khái niệm nút đối tượng - Node Object để mô tả các đối tượng.

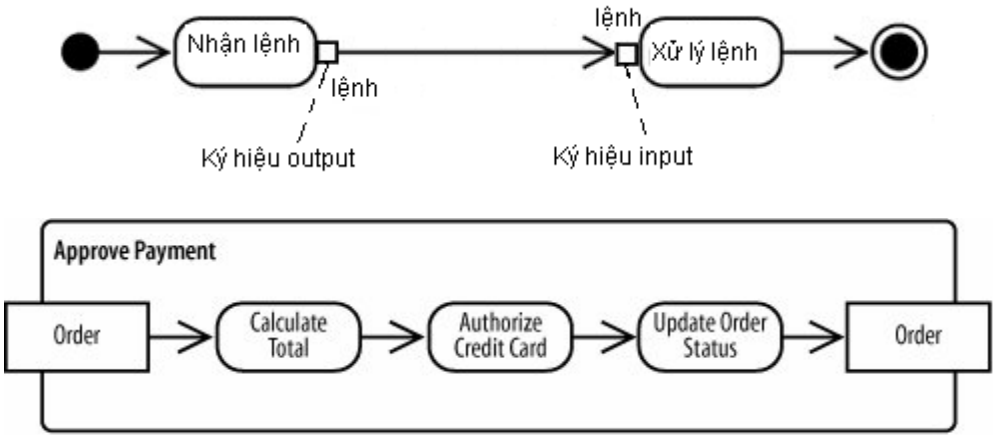
Lược đồ hoạt động cho phép ta biểu diễn các đối tượng trong tiến trình.
Các đối tượng không buộc phải là đối tường phần mền. Ta có thể dùng nút đối tượng để biểu diễn một câu ra lệnh trên thực tế.

Ký hiệu: 

Các cách biểu diễn đối tượng trong tiến trình



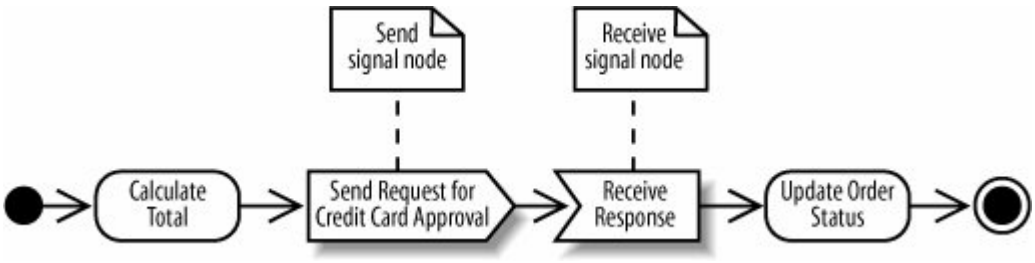
Đối tượng "Lệnh" được truyền giữa hai hành động



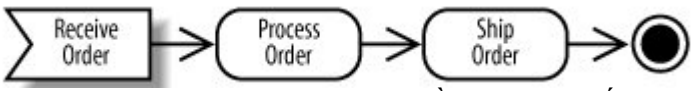
Đối tượng và input và output của một hoạt động

h. Gởi và nhận tín hiệu

Tín hiệu dùng để biểu diễn tương tác với các đối tượng ngoài. Ví dụ như tín hiệu click chuột
Ký hiệu :



Việc nhận tín hiệu có thể kích hoạt một hành động trong lược đồ hoạt động



Dùng node gởi và nhận tín hiệu giúp cho việc đồng bộ hóa tiến trình.
Đối với node nhận tín hiệu có cạnh vào, hệ thống sẽ bắt đầu đợi tín hiệu sau khi hành động trước hoàn thành.

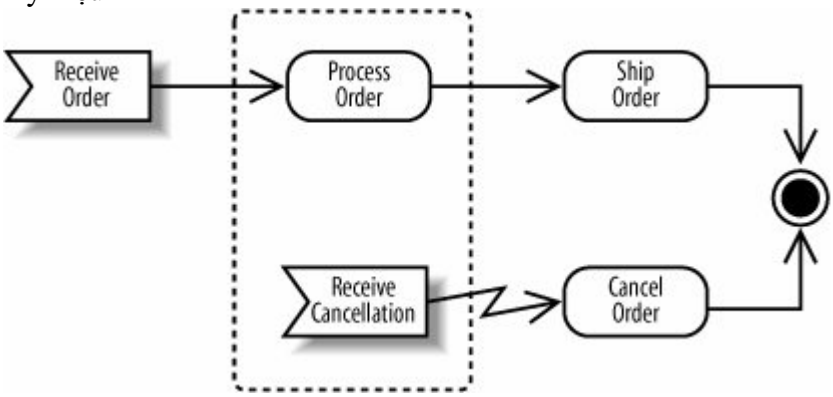
Đối với node nhận tín hiệu không có cạnh vào, nghĩa là hệ thống luôn luôn đợi tín hiệu khi hoạt động chứa node này được khích hoạt.

i. Bắt đầu và kết thúc một hoạt động

Bắt đầu
Một hoạt động có thể được kích hoạt bởi
Node bắt đầu – Init Node
Việc nhận dữ liệu vào
Biến cố thời gian
Việc nhận tín hiệu

Kết thúc
Kết thúc bởi node kết thúc

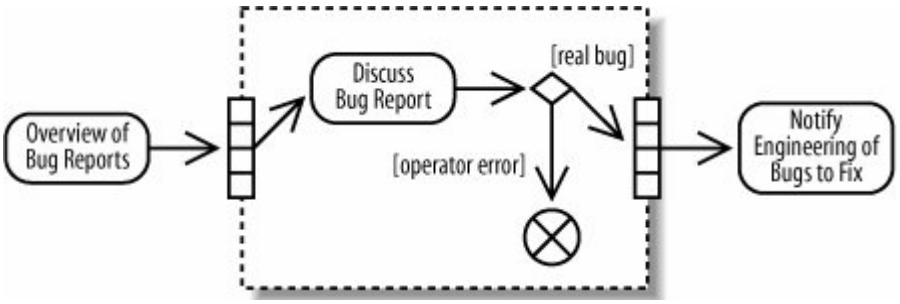
Khái niệm chen ngang một hành động
UML sử dụng Vùng chen ngang - Interruption Region để thể hiện việc chen ngang một hành động bởi một biến cố.
Vùng này được ký hiệu bởi hình chữ nhật đứt nét, tròn góc.
Ký hiệu



j. Các khái niệm mở rộng

Kết thúc luồng
Dùng để kết thúc một luồng các hành động
Ký hiệu 

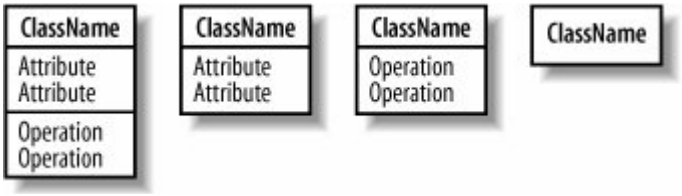
Expansion Region
Vùng này biểu diễn các hành động trong vùng sẽ được thực hiện trên từng thành phần của tập dữ liệu đầu vào.
Ký hiệu : Hình chữ nhật đứt nét, tròn góc, mỗi bên có 4 khối vuông


Overview of Bug Reports
Discuss Bug Report
[real bug]
[operator error]
Notify Engineering of Bugs to Fix

IV. Mô hình hóa cấu trúc luận lý của hệ thống: Class Diagram

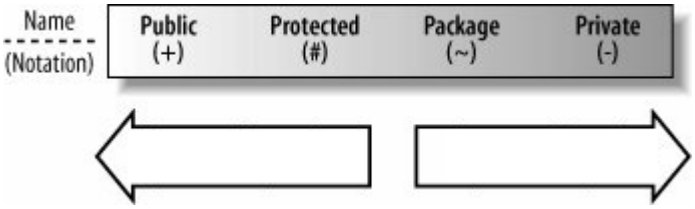
1. Giới thiệu:

Lớp là trung tâm của bất cứ hệ thống hướng đối tượng nào vì thế lược đồ lớp – class diagram là một lược đồ thông dụng nhất trong UML. Class diagram mô hình hóa các lớp trong hệ thống của bạn và quan hệ giữa chúng.
Lớp trong UML được ký hiệu



2. Quyền truy xuất các thành phần

Trong UML sử dụng khái niệm Visibility tương đương với quyền truy xuất trong lập trình hướng đối tượng. Visibility có 4 cấp độ, từ thấp đến cao.

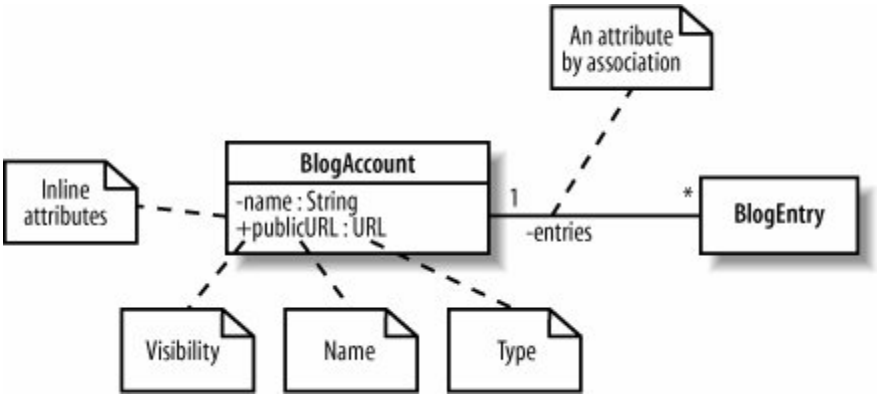


Khác với lập trình hướng đối tượng, ở đây có thêm khái niệm Package. Đây là quyền chỉ cho các lớp cùng package truy xuất (Khái niệm package được giải thích sau)

3. Các thuộc tính

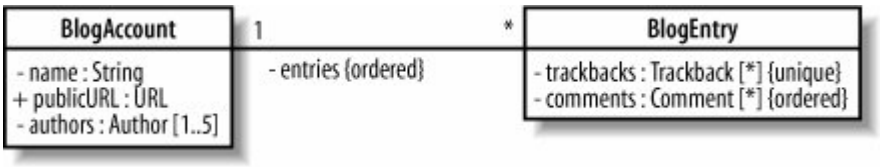
a. Thuộc tính cơ bản

Các thuộc tính của lớp được biểu diễn bằng cách đặt trực tiếp vào khung của lớp - gọi là inline attribute hoặc được biểu diễn bởi liên kết giữa các lớp.



b. Multiplicity

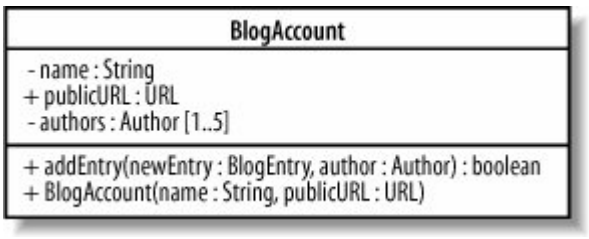
UML sử dụng khái niệm này để biểu diễn một thuộc tính của lớp là một mảng các đối tượng của lớp khác. Có hai cách thể hiện multiplicity : Đặt trực tiếpvào khung của lớp, hoặc thông qua liên kết



Trong ví dụ, thuộc tính authors có kiểu là Author và là một mảng 5 phần tử. Thuộc tính trackbacks có kiểu là Trackback và là một mảng không giới hạn số phần tử. Tương tự cho thuộc tính comments.
Ngoài thông tin với số lượng phần tử, ta có thêm thuộc tính của đối tượng như : unique, ordered.

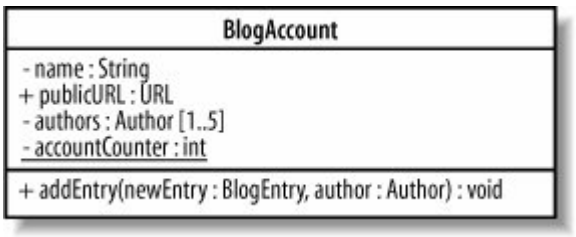
4. Phương thức

a. Cách biểu diễn



b. Thuộc tính static

Phương thức, thuộc tính static được gạch chân


BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
- accountCounter : int
+ addEntry(newEntry : BlogEntry, author : Author) : void

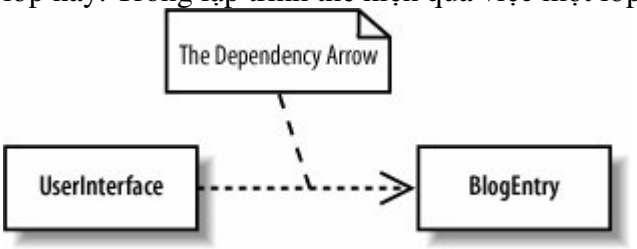
V. Mô hình hóa cấu trúc luận lý của hệ thống: Advanced Class Diagrams

1. Quan hệ giữa các lớp

Độ mạnh của mối quan hệ giữa 2 lớp được tính dựa vào mức độ phụ thuộc giữa 2 lớp. UML đưa ra 5 mức độ quan hệ.

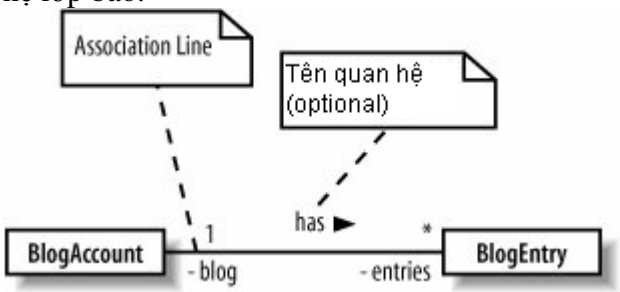
a. Dependency:

Mối quan hệ này thể hiện một lớp cần biết thông tin về lớp khác để sử dụng các thành phần của lớp này. Trong lập trình thể hiện qua việc một lớp inlucde header của một lớp khác.



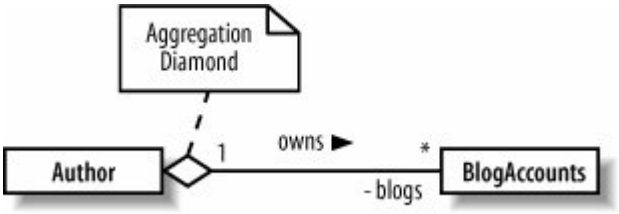
b. Association

Mối quan hệ này biểu diễn một lớp sử dụng đối tượng của lớp khác. Trong lập trình đây là quan hệ lớp bao.



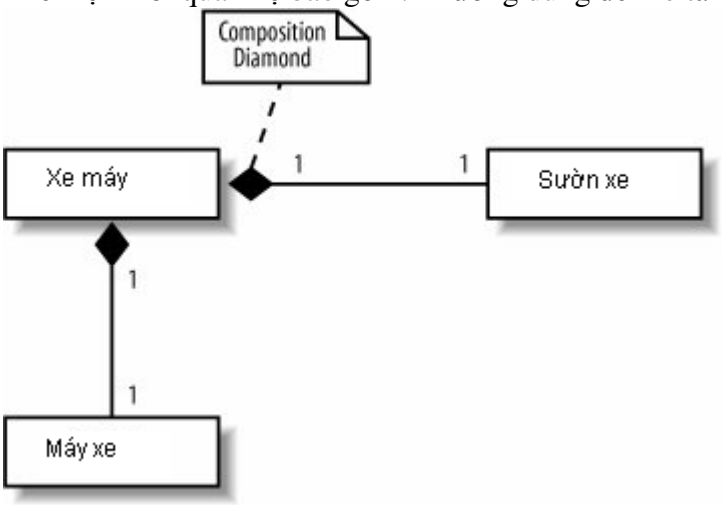
c. Aggregation

Thể hiện mối quan hệ một lớp sở hữu một lớp khác và có thể chia sẽ với các lớp khác
Ví dụ : Mối quan hệ giữa tác giả và blog. Tác giả sở hữu blog và có thể chia sẽ blog này với người khác.



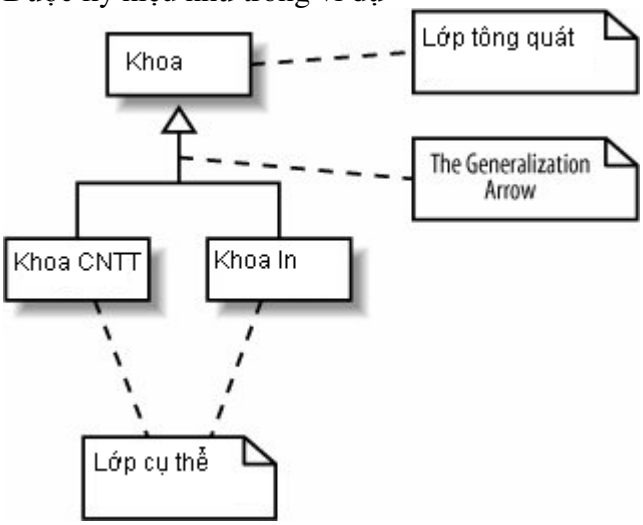
d. Composition

Thể hiện mối quan hệ bao gồm. Thường dùng để mô tả các thành phần của đối tượng.



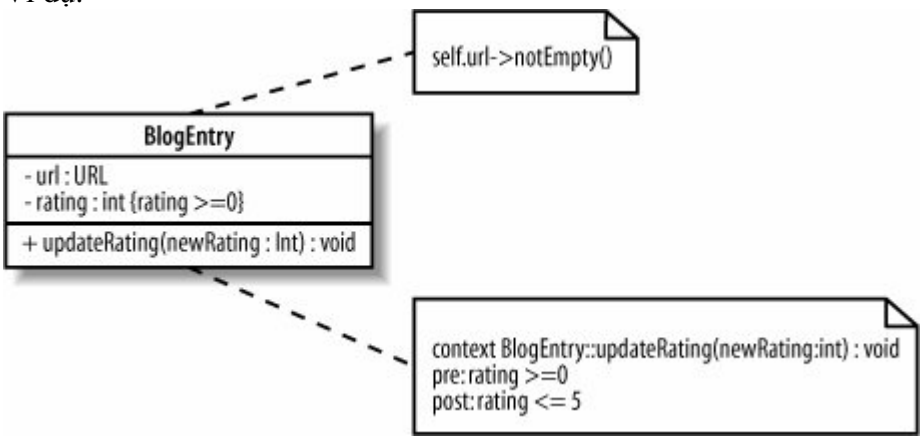
2. Thừa kế

Ta có thể biểu diễn đơn thừa kế hoặc đa thừa kế.
Được ký hiệu như trong ví dụ



3. Các ràng buộc trên lớp

Ta có thể định nghĩa các ràng buộc cho các thuộc tính, các phương thức của lớp.
Bạn có thể xem thêm phần Object Constraint Language - OCL để hiểu thêm về ngôn ngữ ràng buộc
Ví dụ:

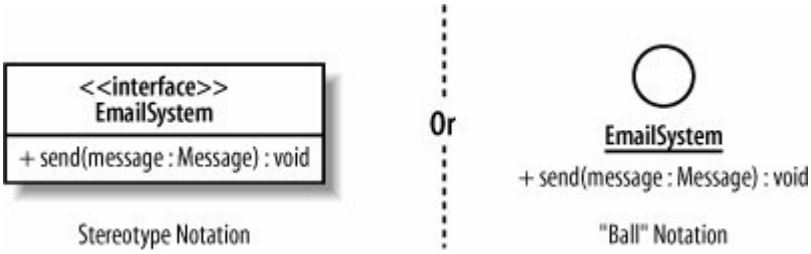


4. Lớp trừu tượng

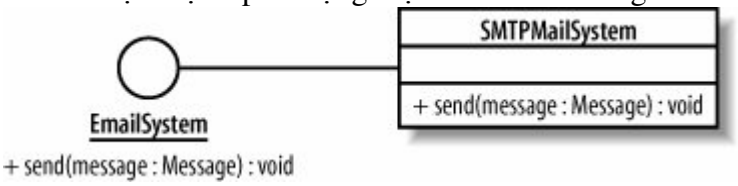
Lớp trừ tượng là lớp có hoặc được thừa kế phương thức trừu tượng
Trong UML tên lớp trừu tượng và tên phương thức trừu tượng được in nghiêng

5. Interface

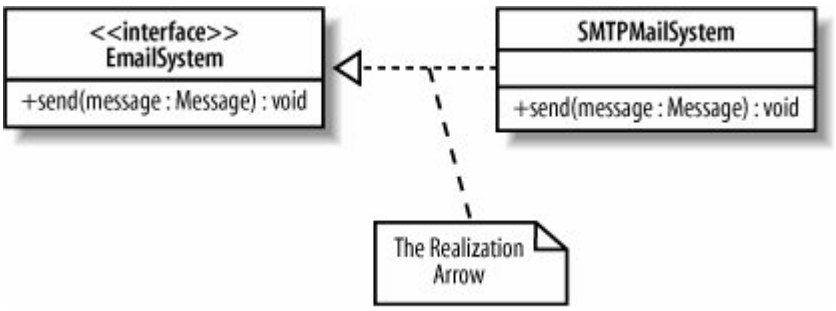
Có hai cách biểu diễn interface



Để thể hiện một lớp sử dụng một interface ta cũng có 2 cách tương ứng

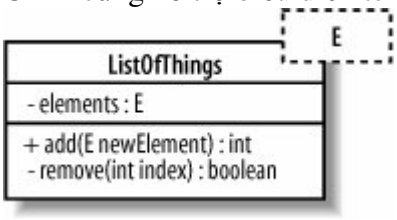


hoặc

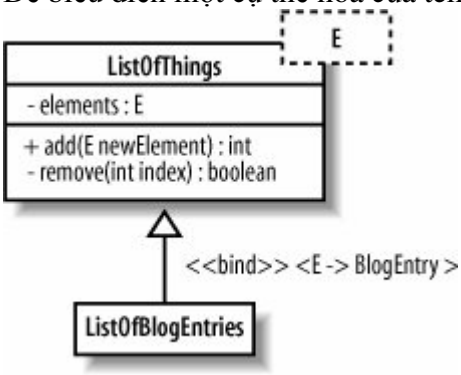


6. Template

UML cũng hỗ trợ biểu diễn template



Để biểu diễn một cụ thể hóa của template ta sử dụng ký hiệu



VI. Mô hình hóa các đối tượng: Object Diagrams

1. Giới thiệu:

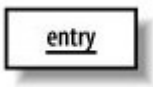
Các đối tượng là trung tâm của bất kì hệ thống hướng đối tượng nào vào lúc runtime. Khi hệ thống mà bạn thiết kế thật sự được sử dụng, các đối tượng hợp các phần của nó lại và đem tất cả các lớp bạn đã design một cách cẩn thận vào hoạt động.

So với biểu đồ lớp thì hệ thống kí hiệu biểu đồ đối tượng khá đơn giản. Mặc dù có ngôn từ khá hạn chế nhưng các biểu đố đối tượng đặc biệt hữu dụng khi bạn muốn mô tả bằng cách nào các đối tượng trong hệ thống làm việc với nhau. Biểu đồ lớp mô tả bằng cách nào các loại đối tượng khác nhau trong hệ thống tương tác với đối tượng khác. Chúng cũng chỉ ra cách mà đối tượng sẽ tồn tại và giao tiếp với hệ thống vào thời điểm chạy. Thêm vào các biểu đồ lớp, các biểu đồ đối tượng giúp bạn có cái nhìn logic về mô hình của bạn.

2. Các khái niệm:

a. Object Instances

Để vẽ một biểu đồ đối tượng, điều đầu tiên mà bạn cần thêm vào chính là các đối tượng. Kí hiệu các đối tượng thì thật sự rất đơn giản nếu bạn đã quen với các kí hiệu lớp. Đối tượng được biểu diễn bởi một hình chữ nhật, giống như lớp, nhưng để chỉ ra đây là thể hiện của lớp chứ không phải lớp thì tiêu đề bạn sẽ gạch dưới như hình 6-2

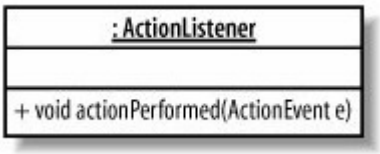


Hình 6-2 đối tượng được biểu diễn bằng 1 hình chữ nhật, tên được gạch dưới

Đối tượng entry trong hình 6-2 chỉ xác định tên đối tượng. Tuy nhiên nếu bạn biết lớp mà đối tượng thể hiện, bạn cũng có thể thêm vào các tên lớp vào tên đối tượng.

Hình 6-3 entry là đối tượng của lớp BlogEntry

Có một loại đối tượng nữa bạn muốn sử dụng trong mô hình đối tượng của bạn đó là đối tượng ẩn danh



Hình 6-4 đối tượng thuộc lớp ActionListener, không có tên

Chúng ta sử dụng đối tượng ẩn danh khi không quan tâm tới tên của nó mà chỉ cần biết nó sẽ làm gì . Giống khi bạn tạo ra một sự kiện trong Java sử dụng một đối tượng nặc danh bởi vì bạn

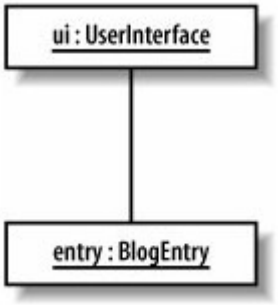
không cần quan tâm tới tên đối tượng là gì? Chỉ cần quan tâm tới việc nó được đưa vào source làm gì như đoạn code sau:

```
with a JButton
public void initialiseUI(  ) {
   //... khai báo phương thức khác ...

   JButton button = new Jbutton("Submit");
   button.addActionListener(new ActionListener{
      public void actionPerformed(ActionEvent e)
      {
         System.out.println("Khi nhan nut thi lam gi do ...");
      }
   });

   //... khai báo phương thức khác ...
}
```

Ví dụ 6-1 : sử dụng đối tượng ẩn danh trong Java

b. Links

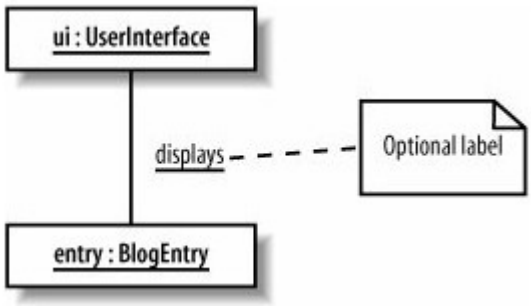
Để các đối tượng có thể làm việc với nhau ta sử dụng các link.



Hình 6-5 link là các đường thẳng nối giữa 2 đối tượng

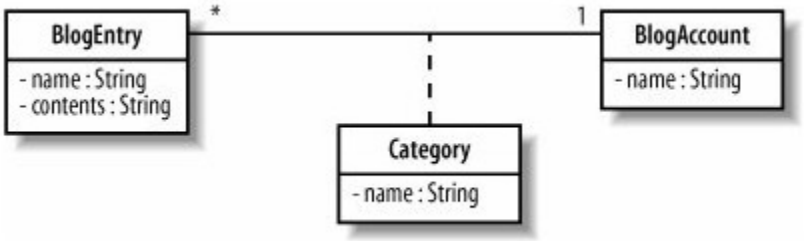
Link giữa các đối tượng trong biểu đồ đối tượng chỉ ra rằng 2 đối tượng có thể được giao tiếp với nhau như thế nào. Tuy nhiên, không phải đối tượng nào cũng có thể link với nhau. Nếu bạn tạo ra một link giữa 2 đối tượng, thì chúng phải thuộc 2 lớp có sự kết hợp tương ứng.
Các link trên biểu đồ đối tượng hoạt động tương tự như link trong biểu đồ truyền tin (chương 8). Tuy nhiên, không giống như biểu đồ truyền tin, thì phần thông tin trên link chỉ là 1 cái nhãn cho biết mục đích link.

- **Link và các ràng buộc**
  Link giữa các đối tượng phải phù hợp với các kết hợp giữa các lớp của đối tượng. Điều này có nghĩa là các luật ràng buộc được áp dụng ở sự kết hợp nào thì link phải tuân thủ theo những ràng buộc này.

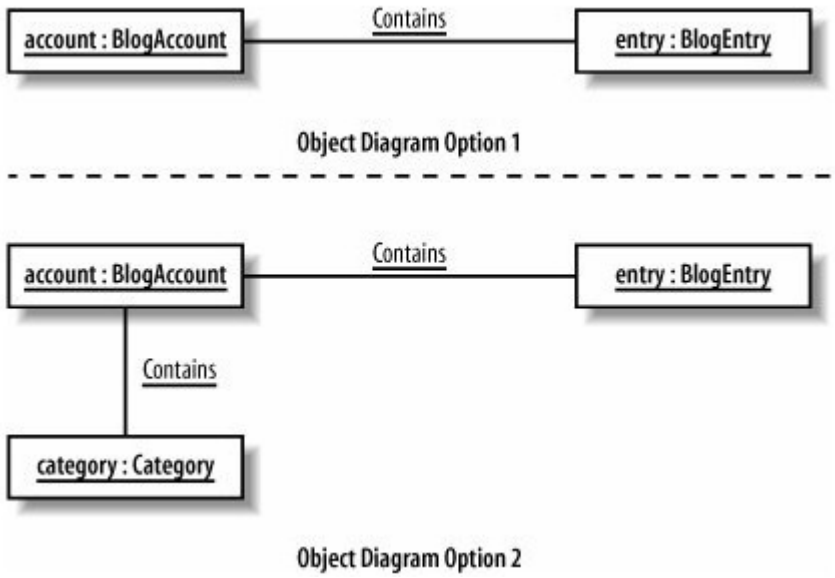


Hình 6-6 đối tượng BlogEntry được kết nối với đối tượng UserInterface

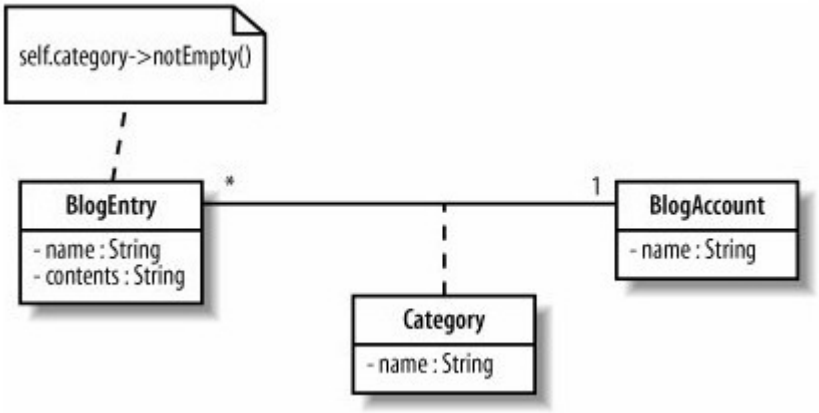
Trong chương 5, mối kết hợp giữa BlogEntry, BlogAccount và Category được mô hình hóa sử dụng một lớp thích hợp. Hình 6-7



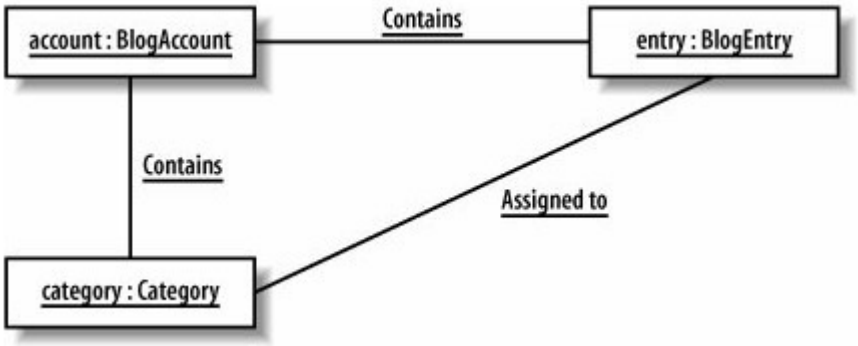
Hình 6-7 lớp Category thì được kết nối với mối quan hệ giữa BlogEntry và BlogAccount



1 entry có thể kết hợp với 1 BlogAccount hoặc 1 bộ category, nhưng không có 1 luật nào nói rằng BlogEntry phải kết hợp với Category
Nếu bạn muốn chỉ ra rằng 1 entry phải được kết hợp với 1 category khi nó được chứa trong 1 blog account bạn cần phải thêm vào 1 vài OCL vào biểu đồ lớp gốc.



Ràng buộc nói rằng trong phạm vi của BlogEntry, 1 đối tượng BlogEntry nên được gắn với 1 category và sự gắn kết không rỗng; nói cách khác, 1 BlogEntry để được thêm vào 1 BlogAccount nó phải có 1 category đính kèm.
Các đối tượng và link trên biểu đồ đối tượng phải tồn tại bởi các luật được phát biểu của OCL. Đây là một trong những lí do tại sao OCL được gọi là ngôn ngữ ràng buộc đối tượng mà không phải là ngôn ngữ ràng buộc lớp. Với các ràng buộc được áp dụng cho biểu đồ lớp, chọn lựa có thể cho 1 biểu đồ đối tượng dựa trên lớp này thì giảm một vài thứ cho có nghĩa hơn.

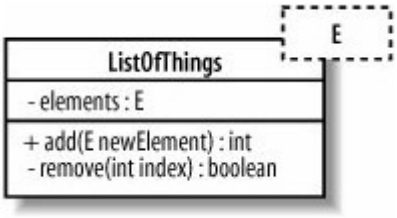


Ràng buộc *self.category->notEmpty()* ảnh hưởng tới các chọn lựa có sẵn khi bạn tạo 1 biểu đồ đối tượng, hãy chắc rằng 1 entry được kết hợp với 1 category để được thêm vào *BlogAccount*.

c. Binding Class Templates

Trong chương 5, chúng ta đã biết các đối số mà được khai báo trong class template được sử dụng như thế nào bằng cách dùng 1 subclassing trong biểu đồ lớp. Mặc dù cách này làm việc tốt, sự mạnh thật sự của template là khi bạn bind đối số template vào lúc chạy. Để làm điều này, bạn lấy 1 template và gắn nó vào các khuôn mẫu mà đối số của nó được cấu trúc thành 1 đối tượng.
Biểu đố đối tượng là tư tưởng cho việc mô hình sự thay thế các gắn kết được thực hiện như thế nào. Khi bạn sử dụng các đối số là các template được thay thế khi chạy, có nghĩa là bạn đang đề cập đến các đối tượng hơn là các lớp, vì vậy bạn không thể mô hình hóa các thông tin này bằng biểu đồ lớp thông thường
Ghi chú: mặc dù bài này nói về UML trong giới hạn các loại biểu đồ, nhưng việc làm rõ về UML không ràng buộc vào 1 loại biểu đồ cụ thể nào. Thật vậy, bạn có thể trình bày các kí hiệu biểu đồ

đối tượng trên 1 biểu đồ lớp nếu bạn muốn gom các lớp của bạn và các sự gắn kết lúc chạy lại thành 1 biểu đồ.
Hình dưới trình bày 1 class template đơn giản:



Để tạo mô hình là đối số E của template ListOfThing thì được gắn vào 1 lớp cụ thể vào lúc runtime, tất cả những gì bạn cần làm là têhm chi tiết đối số binding vào cuối của mô tả lớp của đối tượng, như sau:

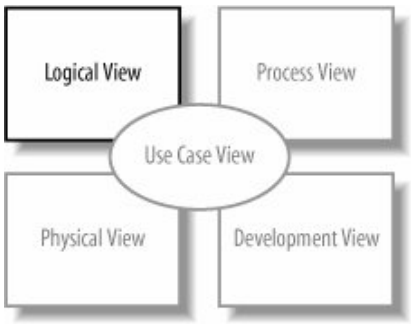
Trước đây, việc thể hiện các gắn kết của Java là điều không thể, ngôn ngữ không hỗ trợ template. Tuy nhiên, với Java 5 và các đặc tính ngôn ngữ mới, sự gắn kết của lớp BlogEntry với tham số E trong template ListOfThings bây giờ có thể được implement:

```
public class ListOfThings<E> {
   public ListOfThings {
      elements = new ArrayList<E>();
   }

   public int add(E object) {
      return elements.add(object);
   }

   public E remove(int index) {
      return elements.remove(index);
   }
}

public class Application {
   public static void main(String[] args){
      // Binding the E parameter on the ListOfThings template to a
Musician class
      // to create a ListOfThings that will only store Musician
objects.
      ListOfThings <BlogEntry>listOfBlogEntries= new
ListOfThings<BlogEntry>(  );
   }
}
```

VII. Mô hình hóa thứ tự các tương tác: Sequence Diagrams

1. Giới thiệu:

Các use case cho phép mô hình của bạn mô tả hệ thống có thể làm gì. Các lớp cho phép mô hình của bạn mô tả sự khác nhau giữa các loại thành phần cấu hình nên hệ thống. Nếu các use case và class đứng một mình thì bạn không thể lập mô hình hệ thống mà cho biết thật sự chúng làm những công việc gì? Từ đó biểu đồ tuần tự xuất hiện.
Biểu đồ tuần tự là 1 thành viên quan trọng của nhóm mà được hiểu là các biểu đồ tương tác. Các biểu đố tương tác lập nên mô hình tương tác giữa các phần tạo nên hệ thống và tạo ra cái nhìn logic cho mô hình của bạn. Hình 7-1



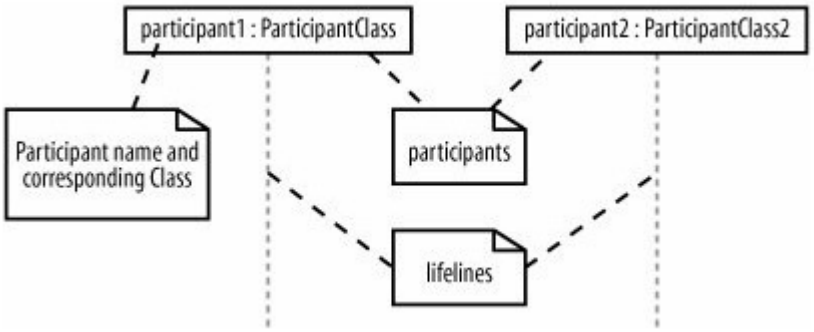
Hình 7-1 Các phần của hệ thống, gồm các tương tác giữa các phần

Biểu đồ tuần tự không đứng một mình trong nhóm này, chúng làm việc cùng với biểu đồ truyền tin và các biểu đồ thời để giúp bạn mô hình hóa một cách chính xác các phần của hệ thống tương tác thế nào?
Biểu đồ tuần tự là biểu đồ phổ biến nhất trong 3 loại biểu đồ tương tác. Điều này có thể là bởi vì chúng biểu diễn thông tin đúng thứ tự hoặc đơn giản bởi vì chúng có khuynh hướng làm cho người mới biết UML cảm thấy thoải mái.
Biểu đồ tuần tự, tất cả chỉ là ràng buộc thứ tự tương tác giữa các phần của hệ thống. Sử dụng biểu đồ tuần tự, bạn có thể chỉ ra các tương tác nào sẽ thực hiện khi 1 use case cụ thể được thực thi và các tương tác này diễn ra theo thứ tự nào. Biểu đồ tuần tự cho biết nhiều thông tin hơn về một tương tác nhưng điểm mạnh của chúng là cái cách đơn giản và hiệu quả mà chúng truyền thông tin theo thứ tự tương tác.

2. Các khái niệm:

a. Participant:

Một biểu đồ tuần tự được tạo thành từ một tập các participant. 1 participant được đặt ở đâu trong biểu đồ tuần tự là rất quan trọng:



Hình 7-2: 1 biểu đồ tuần tự đơn giản. Chỉ có 1participant thì lạ nhưng cũng không sai cú pháp UML

Mỗi participant có 1 đường sinh tồn chạy xuống dưới. Đường sinh tồn biểu diễn thời gian tồn tại của đối tượng

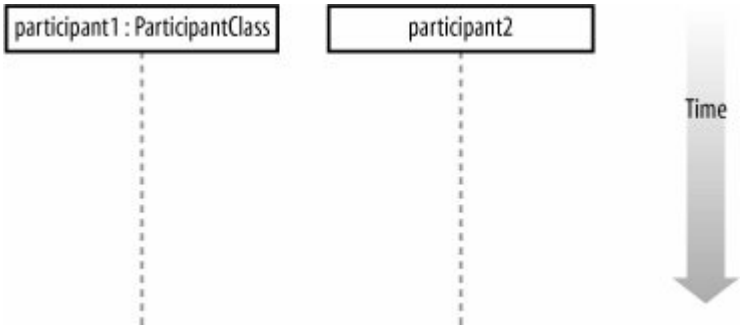
Các participant trên biểu đồ tuần tự có thể được đặt tên bằng 1 số cách khác nhau, bạn có thể chọn dựa vào định dạng chuẩn sau :
name [selector] : *class_name ref decomposition*
Các phần mà bạn lấy để sử dụng cho 1 participant cụ thể sẽ lệ thuộc vào thông tin về participant vào thời điểm đưa ra.

Ví dụ về tên participant	Mô tả
admin	1 part được đặt tên là admin, nhưng tại thời điểm này thì part chưa được gắn với class
: ContentManagementSystem	Lớp của participant là ContentManagementSystem, nhưng part hiện hành chưa có tên riêng
admin : Administrator	Có 1 part tên admin thuộc lớp Administrator
eventHandlers [2] : EventHandler	1 part được truy cập với mảng gồm 2 phần tử thuộc lớp EventHandler
: ContentManagementSystem ref cmsInteraction	Participant thuộc lớp ContentManagementSystem và có biểu đồ tương tác khác là cmsInteraction chỉ ra bên trong participant làm việc như thế nào

Định dạng được sử dụng để tạo ra tên của các participant thì hoàn toàn lệ thuộc vào bạn hoặc có lẽ là theo phong cách của công ty bạn. Chúng ta sẽ đặt tên cho participant với chữ cái đầu tiên viết thường để tránh lẫn lộn với tên lớp.

b. Time

1 biểu đồ tuần tự mô tả thứ tự các tương tác diễn ra, vì vậy, thời gian là một nhân tố quan trọng. sự liên hệ giữa thời gian với biểu đồ tuần tự như sau:

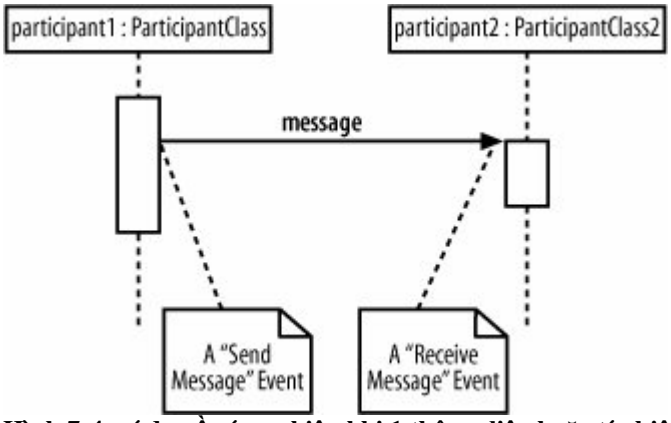


Hình 7-3: thời gian đi từ trên xuống dưới

Thời gian trên biểu đồ tuần tự bắt đầu trên đỉnh trang, chỉ sau đề mục participant ở trên đỉnh. Thứ tự các tương tác diễn ra từ trên xuống dưới trang trên biểu đồ tuần tự chỉ ra thứ tự các tương tác sẽ diễn ra theo thời gian.
Thời gian trên biểu đồ đối tượng thực chất là nói về thứ tự chứ không đơn thuần là thời gian bình thường. Biểu đồ tuần tự chỉ là bước đầu về thứ tự tượng tác giữa các participant, thông tin về thời gian chi tiết hơn sẽ được đề cập rõ ràng hơn trong biểu đồ thời gian.

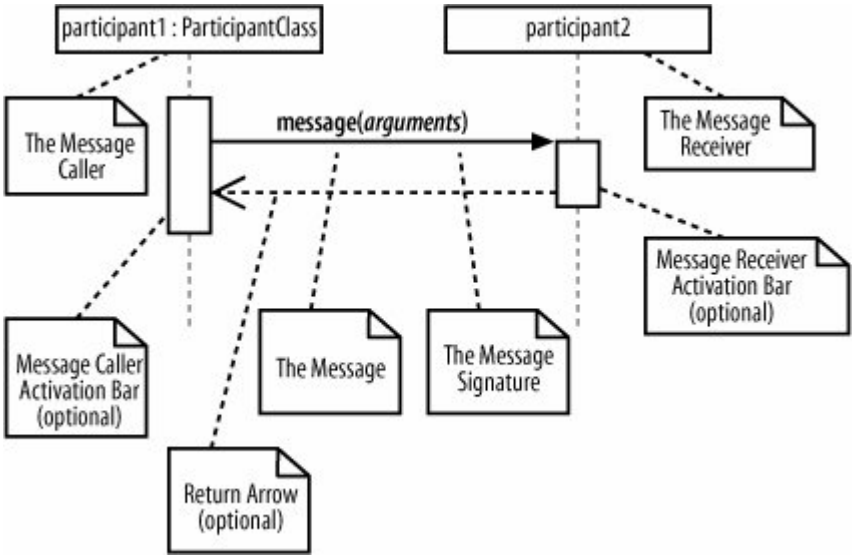
c. Events, Signals, và Messages

Phần nhỏ nhất của 1 tương tác là sự kiện. Sự kiện là bất kì điểm nào trong 1 tương tác nơi mà diễn ra một điều gì đó. Hình 7-4



Hình 7-4: ví dụ về các sự kiện khi 1 thông điệp hoặc tín hiệu được gửi hoặc nhận

Tín hiệu và thông điệp là 2 từ khác nhau nhưng có cùng khái niệm. Tín hiệu là thuật ngữ được sử dụng bởi các nhà thiết kế hệ thống, trong khi các nhà thiết kế phần mềm thì thích gọi là thông điệp hơn.
Trong giới hạn của biểu đồ tuần tự, thì tín hiệu và thông điệp được xem là như nhau, vì vậy chúng ta qui ước sử dụng từ thông điệp sau này.
1 tương tác trong biểu đồ đối tượng diễn ra khi 1 participant quyết định gởi 1 thông điệp tới 1 participant khác.



Hình 7-5: các tương tác trên biểu đồ tuần tự giữa các participant

Các thông điệp trên biểu đồ tuần tự được chỉ ra bằng cách sử dụng 1 mũi tên từ participant mà muốn gửi thông điệp tới participant nhận thông điệp. Các thông điệp có thể theo bất cứ hướng nào lệ thuộc vào yêu cầu tương tác, từ trái qua phải, từ phải qua trái thậm chí trở lại chính nó. Hãy xem thông điệp như 1 sự kiện mà được được gởi bởi nơi gởi yêu cầu bên nhận làm 1 việc gì đó.

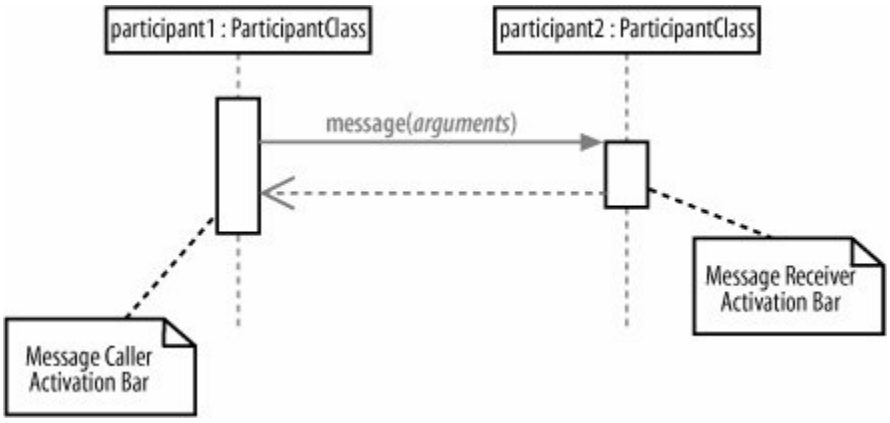
d. Message Signatures

Mũi tên mang thông điệp thường có 1 mô tả hoặc kí hiệu. Theo dạng sau:
Thuộc tính = tên thông điệp hoặc kí hiệu(các đối số) : loại trả về
Bạn có thể chỉ ra số đối số khác nhau trên 1 thông điệp, được ngăn cách bởi dấu phẩy. Theo dạng sau:
<tên>:<lớp>
Các phần bạn sử dụng cho 1 thông điệp cụ thể sẽ lệ thuộc vào thông tin về các thông điệp cụ thể vào thời gian đưa ra.

Ví dụ về kí hiệu thông điệp	Mô tả
doSomething( )	Tên của thông điệp là doSomething, không còn thông tin nào được biết nữa
doSomething(number1 : Number, number2 : Number)	Tên của thông điệp là doSomething, nó mang 2 đối số, number1 và number2 đều thuộc lớp number
doSomething( ) : ReturnClass	Tên của thông điệp là doSomething, không có đối số và trả về 1 đối tượng thuộc ReturnClass
myVar = doSomething( ) : ReturnClass	Tên của thông điệp là doSomething, không có đối số và nó trả về 1 đối tượng thuộc lớp ReturnClass và được gắn cho thuộc tính myVar của nơi gởi thông điệp

e. Activation Bars

Khi 1 thông điệp được gởi tới 1 participant, yêu cầu paticipant nhận làm 1 việc gì đó. Lúc này, participant được nói là đang active. Để chỉ ra 1 participant đang active, đang làm gì đó, bạn có thể sử dụng 1 activation bar

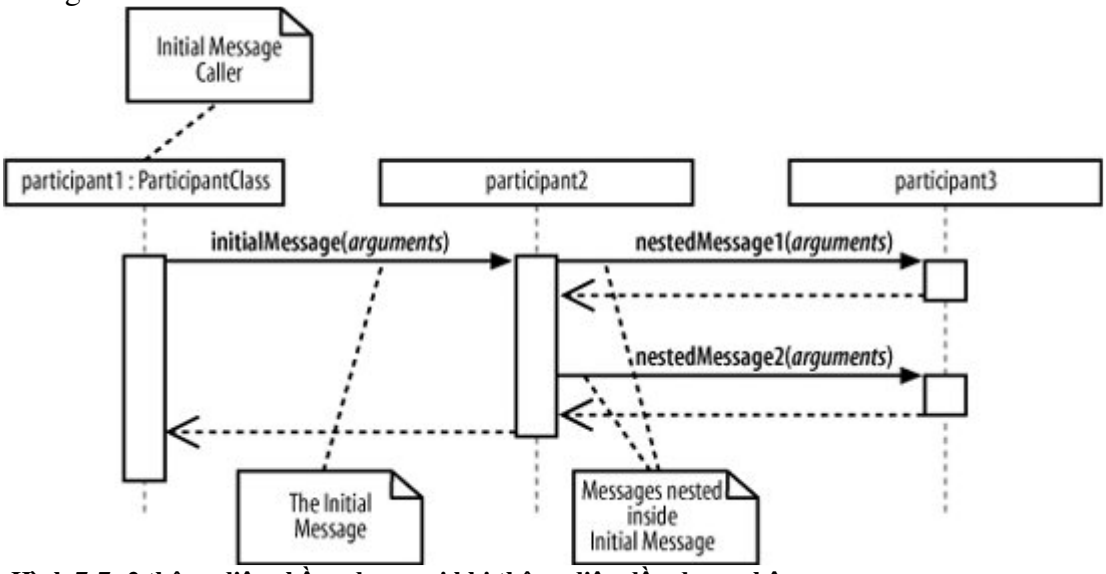


Hình 7-6: các thanh hoạt động chỉ ra là các participant đang bận

1 activation bar được đặt ở cuối của thông điệp gửi hay nhận. Nó chỉ ra rằng participant đang gửi trong trạng thái bận khi nó gửi thông điệp và participant nhận ở trạng thái bận sau khi thông điệp được nhận.
Các activation bar là tùy ý, nhưng chúng có thể làm rối biểu đồ của bạn.

f. Nested Messages:

Khi participant1 gởi 1 thông điệp cho participant2, participant2 nhận thông điệp sau đó lại gởi cho participant3 1 thông điệp hoặc nhiều hơn thì khi đó thông điệp từ participant2 gởi cho participant3 có được do paticipant1 gởi thông điệp cho participant2 được gọi là các thông điệp chồng.

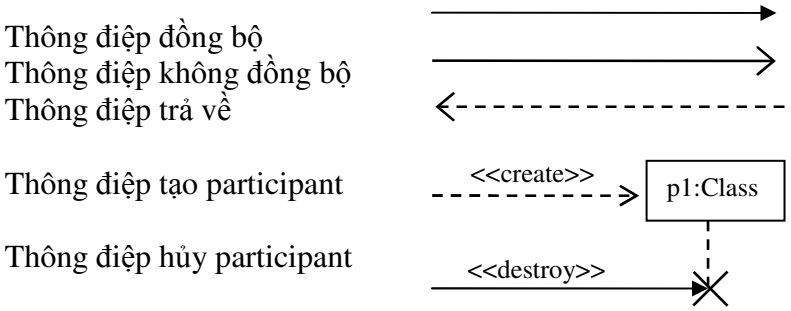


Hình 7-7: 2 thông điệp chồng được gọi khi thông điệp đầu được nhận

Trong hình 7-7, participant1 đã gởi thông điệp initialMessage(..) tới participant2. Khi participant2 nhận thông điệp initialMessage(..), participant2 trở thành active và gởi 2 thông điệp chồng tới participant3. Số lượng thông điệp chồng là tùy ý và cấp của thông điệp chồng trong biểu đồ tuần tự cũng vậy.

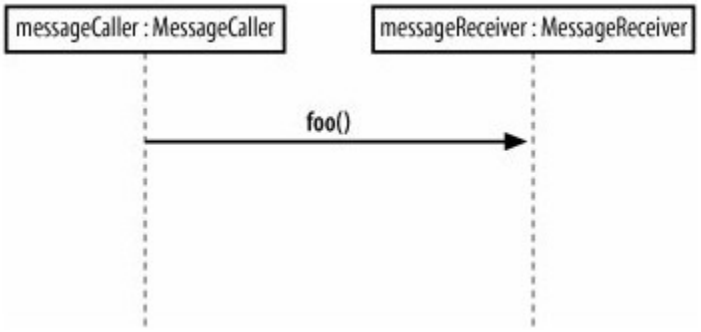
g. Message Arrows

Đầu mũi tên trên 1 thông điệp thì khá quan trọng, nó nói cho ta biết loại thông điệp nào vừa được gởi. Ví dụ, Message Caller muốn đợi một thông điệp được gởi trả lại trước khi tiếp tục công việc là thông điệp đồng bộ. Hoặc khi nó gởi thông điệp đi nhưng không cần đợi để thông điệp được đáp trả mà tiếp tục thực hiện công việc của mình thì là thông điệp không đồng bộ



h. Thông điệp đồng bộ

Như đã đề cập, thông điệp đồng bộ được sử dụng khi nơi gọi thông điệp chờ cho nơi nhận thông điệp đáp trả lại lời gọi thông điệp.



Tương tác được trình bày ở trên được khai báo trong Java chỉ sử dụng 1 lời gọi thủ tục đơn giản Ví dụ 7-1

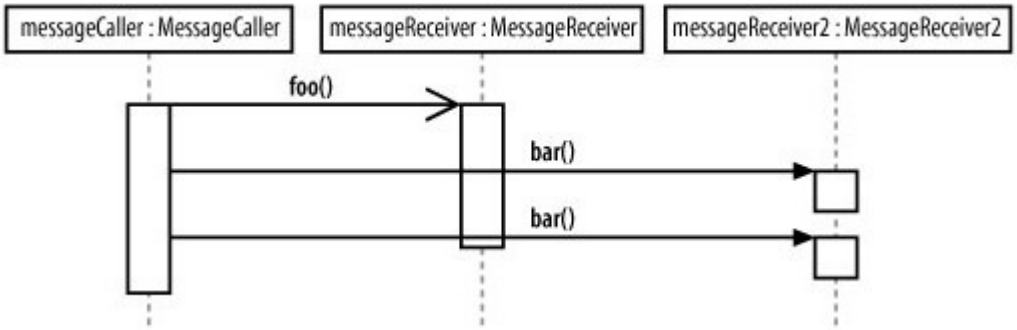
```
public class MessageReceiver
{
   public void foo(  )
   {
      // Do something inside foo.
   }
}

public class MessageCaller
{
   private MessageReceiver messageReceiver;

   // Other Methods and Attributes of the class are declared here

   // The messageRecevier attribute is initialized elsewhere in
   // the class.

   public doSomething(String[] args)
   {
      // The MessageCaller invokes the foo() method

      this.messageReceiver.foo(  ); // then waits for the method to
return

      // before carrying on here with the rest of its work
   }
}
```

Đối tượng *messageCaller* tạo ra một phương thức như thông thường gọi tới phương thức *foo()* trên đối tượng *messageReceiver* sau đó đợi phương thức *messageReceiver.foo()* trả về trước khi tiếp tục các bước khác trong tương tác.

i. Thông điệp không đồng bộ

Sẽ rất tuyệt nếu tất cả các tương tác của bạn diễn ra sau 1 tương tác khác theo 1 thứ tự đơn giản. Mỗi 1 participant sẽ gởi 1 thông điệp tới 1 participant khác và yên lặng đợi thông điệp trả về trước khi tiếp tục. Thật không may, hầu hết các hệ thống không làm việc như vậy. Các tương tác có thể là diễn ra tại 1 điểm trong cùng 1 lúc, và đôi khi bạn sẽ muốn để khai báo 1 tập các tương tác vào cùng thời điểm mà không cần đợi chúng trả về.
Ví dụ, bạn đang thiết kế 1 phần mềm với giao diện người dùng hỗ trợ việc chỉnh sửa và in bộ tài liệu. Ứng dụng của bạn đưa cho người dùng 1 nút để in tài liệu. Việc in ấn sẽ mất một thời gian vì vậy bạn muốn rằng sau khi nhấn in, tài liệu đang được in, người dùng có thể tiếp tục làm việc với các tính năng khác trong ứng dụng. Các mũi tên thông điệp đồng bộ thì không đủ để thể hiện các loại tương tác như thế này. Bạn cần có một loại mũi tên thông điệp khác : mũi tên thông điệp không đồng bộ.
Mũi tên thông điệp không đồng bộ được gọi từ nơi gọi thông điệp đến nơi nhận thông điệp, nhưng nơi gọi thông điệp không đợi cho lời gọi thông điệp được đáp trả trước khi tiếp tục các bước tương tác khác. Điều này có nghĩa là nơi gọi thông điệp sẽ gởi thông điệp tới nơi nhận

thông điệp và nơi gởi thông điệp sẽ tiếp tục trạng thái bận thực hiện các tương tác khác trước khi thông điệp gốc trả về.



Hình 7-10: Trong khi thông điệp foo() đang hoạt động bởi đối tượng messageReceiver, đối tượng messageCaller tiếp tục các tương tác bằng cách thực thi các thông điệp đồng bộ trên đối tượng khác.

Một cách thông thường để implement các thông điệp không đồng bộ trong Java:

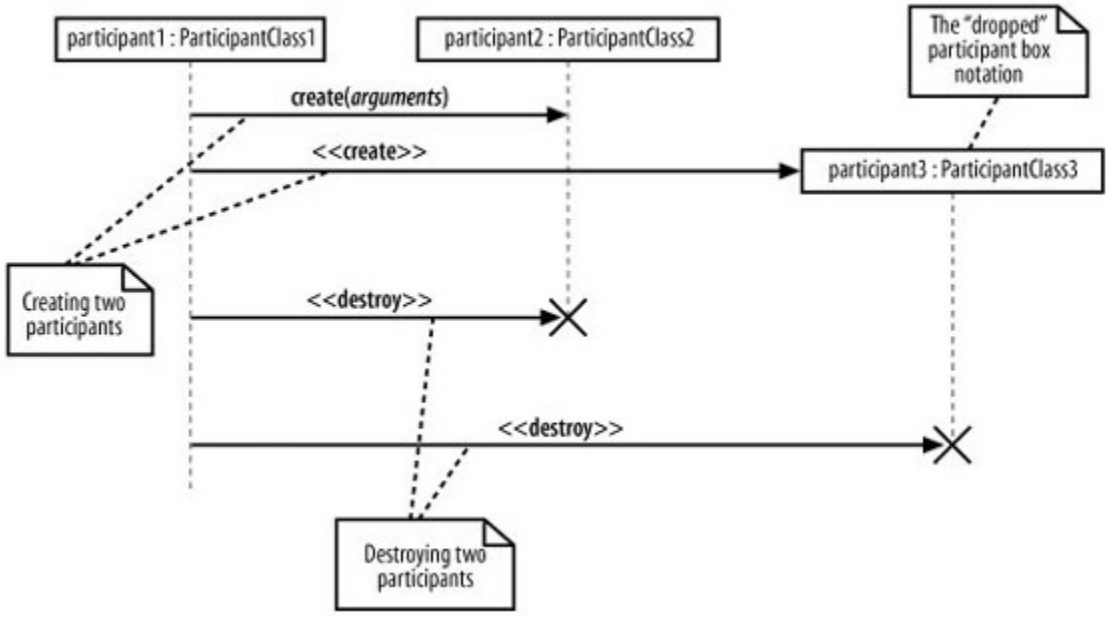
```
public class MessageReceiver implements Runable {

   public void operation1(  ) {
      // nhận thông điệp và tạo thread

      Thread fooWorker = new Thread(this);
      fooWorker.start(); // Lời gọi này bắt đầu 1 thread mới, gọi
phương thức run bên dưới
      // Ngay khi thread được bắt đầu, lời gọi foo() được trả
   }

   public void run(  ) {
      // đây là nơi công việc cho lời gọi foo() được thực
   }
}

public class MessageCaller
{
   private MessageReceiver messageReceiver;

   // Các phương thức và các thuộc tính khac được khai bào ở đây

   // thuộc tính messageRecevier được khởi tạo bất cứ nơi nào trong lớp
   //lớp

   public void doSomething(String[] args) {
      // MessageCaller gọi operation1(  )

      this.messageReceiver.operation1(  );

      // sau đó tiếp tục công việc của nó ngay lập tức
   }
}
```

j. Thông điệp trả về

Thông điệp trả về là một kí hiệu tùy chọn, bạn có thể sử dụng ở cuối activation bar để chỉ ra luồng điều khiển trả về cho participant đã gởi thông điệp ban đầu. Trong code, 1 mũi tên trả về tương tự như khi chúng ta ở cuối của phương thức hoặc khi gọi tường minh phát biểu return. Bạn không phải sử dụng thông điệp trả về nếu bạn thích, chúng có thể làm cho biểu đồ của bạn rất lộn xộn. Bạn không nên làm cho biểu đồ tuần tự của mình rối tung lên với mỗi thông điệp trả về cho mỗi activation bar bởi vì bất cứ activation bar nào được gọi sử dụng thông điệp đồng bộ cũng ngụ ý là có 1 mũi tên trả về rồi.
Mặc dù, 1 thông điệp thường được gởi giữa 2 participant khác nhau, nhưng nếu 1 thông điệp được participant nào đó gởi cho chính nó cũng rất bình thường. Trong giới hạn phần mềm, điều này tương tự như gọi tham chiếu this trong Java và C#

k. Thông điệp tạo và hủy 1 participant

Các participant có thể được tạo và hủy tùy theo thông điệp đã được gởi

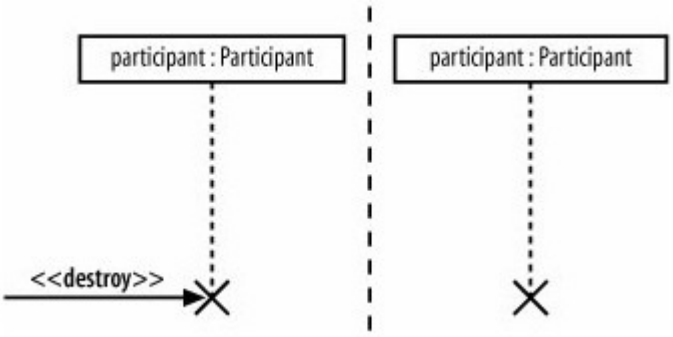


Hình 7-11: participant2 và participant3 đều được tạo trong phạm vi của biểu đồ tuần tự này

Để chỉ ra rằng 1 participant đã được tạo bạn cũng có thể gởi 1 thông điệp create(..) tới đường sinh tồn của participant hoặc sử dụng kí hiệu hộp nơi participant chưa tồn tại lời gọi tạo participant được gọi. Việc xóa được trình bày ở cuối của đường sinh tồn bằng dấu chữ thập

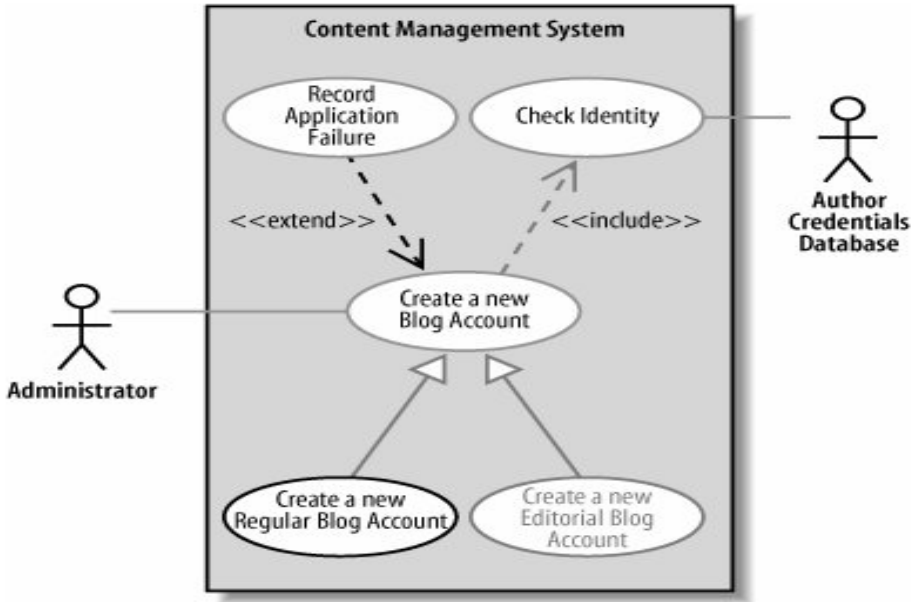
```
public class MessageReceiver {
   // các thuộc tính và phương thức của MessageReceiver
}

   public class MessageCaller {

   // các thuộc tính và các phương thức khác được khai báo ở đây

      public void doSomething(  ) {
      // đối tượng MessageReceiver được tạo
      MessageReceiver messageReceiver = new MessageReceiver(  );
      }
}
```

MessageCaller tạo ra 1 đối tượng *MessageReceiver* mới đơn giản chỉ với từ khóa new
Với một vài ngôn ngữ, chẳng hạn như Java, bạn không có phương thức hủy tường minh, vì vậy nó không trình bày trực quan trên các biểu đồ của bạn. Ở ví dụ trên, đối tượng *messageReceiver* được bật cờ để hủy khi phương thức *doSomething* hoàn thành. Tuy nhiên, không cần gửi thông điệp nào nữa tới *messageReceiver* để hủy chính nó bởi vì tất cả các điều này được làm bởi bộ thu gom rác ngầm định của Java



3. Thực hiện use case với biểu đồ tuần tự:

Bây giờ chúng ta sẽ có cái nhìn sát hơn về biểu đồ tuần tự. Chúng ta sẽ xem 1 biểu đồ tuần tự mà mô hình hóa các tương tác cần có để tạo ra 1 use case Create a new Regular Blog Account:



Tóm lại, use case Create a new Regular Blog Account là một trường hợp đặc biệt của use case Create a new Blog Account. Nó cũng bao gồm tất cả các bước được cấp bởi use case Check Identity và có thể tùy chọn thực thi các bước được cấp bởi use case Record Application Failure, nếu các ứng dụng cho tài khoản mới bị từ chối.
Việc hỗ trợ kỹ thuật hộp tiêu đề ngẫu nhiên
Nhiều công cụ UML chuẩn không hỗ trợ kỹ thuật hộp tiêu đề ngẫu nhiên để chỉ ra việc tạo participant hoặc ký hiệu chữ thập cho việc hủy participant. Ví dụ, bạn sẽ nhận ra rằng công cụ của mình không cho phép đặt hộp tiêu đề của participant bất cứ đâu ngoại trừ đỉnh của biểu đồ. Trong trường hợp này, cách tốt nhất là dựa vào các thông điệp tạo và xóa gọi các đối tượng mà đang được tạo hoặc là hi vọng người đọc biểu đồ nhận ra ý bạn là participant đang được tạo. Thật đáng tiếc là đây không phải là cách tốt nhất của UML, nhưng thỉnh thoảng đó là tất cả những gì bạn có thể làm.

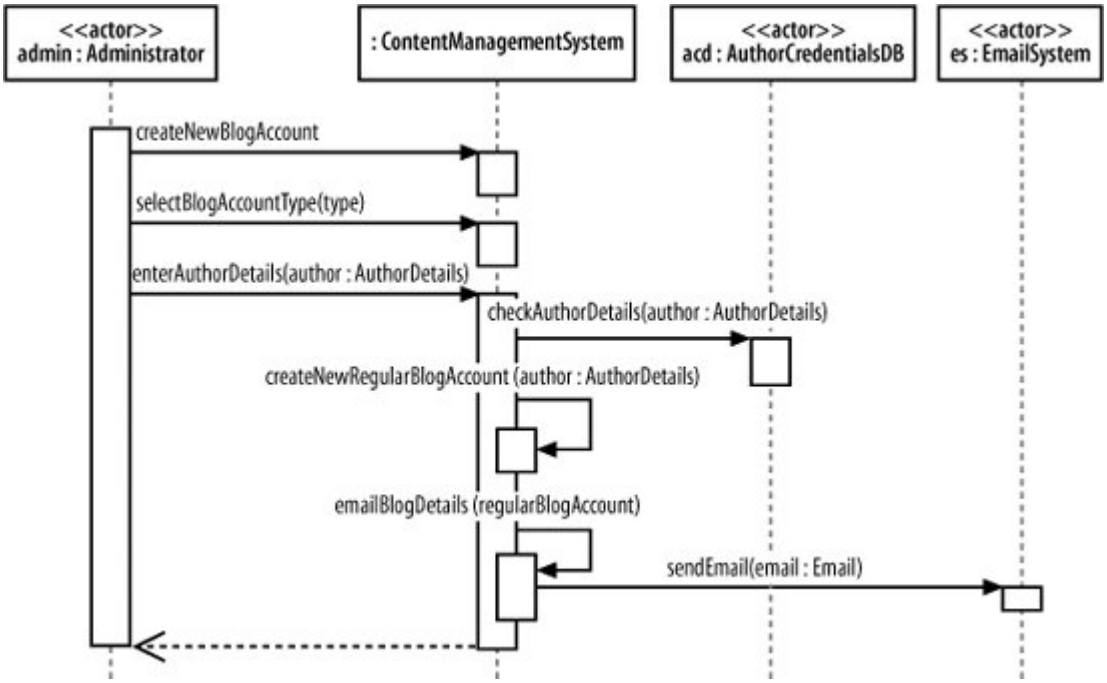
a. Biểu đồ tuần tự cấp cao

Trước khi bạn có thể chỉ ra các tương tác sẽ diễn ra khi 1 use case thực thi, bạn cần có một mô tả chi tiết hơn về các việc use case làm. Nếu bạn có được bản mô tả use case, thì việc tham chiếu các thông tin chi tiết này là khá hữu ích.
Hình dưới trình bày các bước diễn ra trong use case Create a new Regular Blog Account theo mô tả chi tiết của nó
Hầu hết thông tin chi tiết mà bạn cần để bắt đầu cấu trúc 1 biểu đồ tuần tự cho 1 use case nên có sẵn như dòng chính trong mô tả của use case

Dòng chính	Bước	Hoạt động
	1	Nhà quản trị yêu cầu hệ thống tạo 1 blog account mới
	2	Nhà quản trị chọn đúng loại blog account
	3	Nhà quản trị điền thông tin chi tiết của tác giả
	4	Chi tiết tác giả được kiểm tra dựa vào CSDL Author Credentials

	5	Blog account mới được tạo
	6	1 bản tóm tắt các chi tiết của blog account mới được gởi tới tác giả

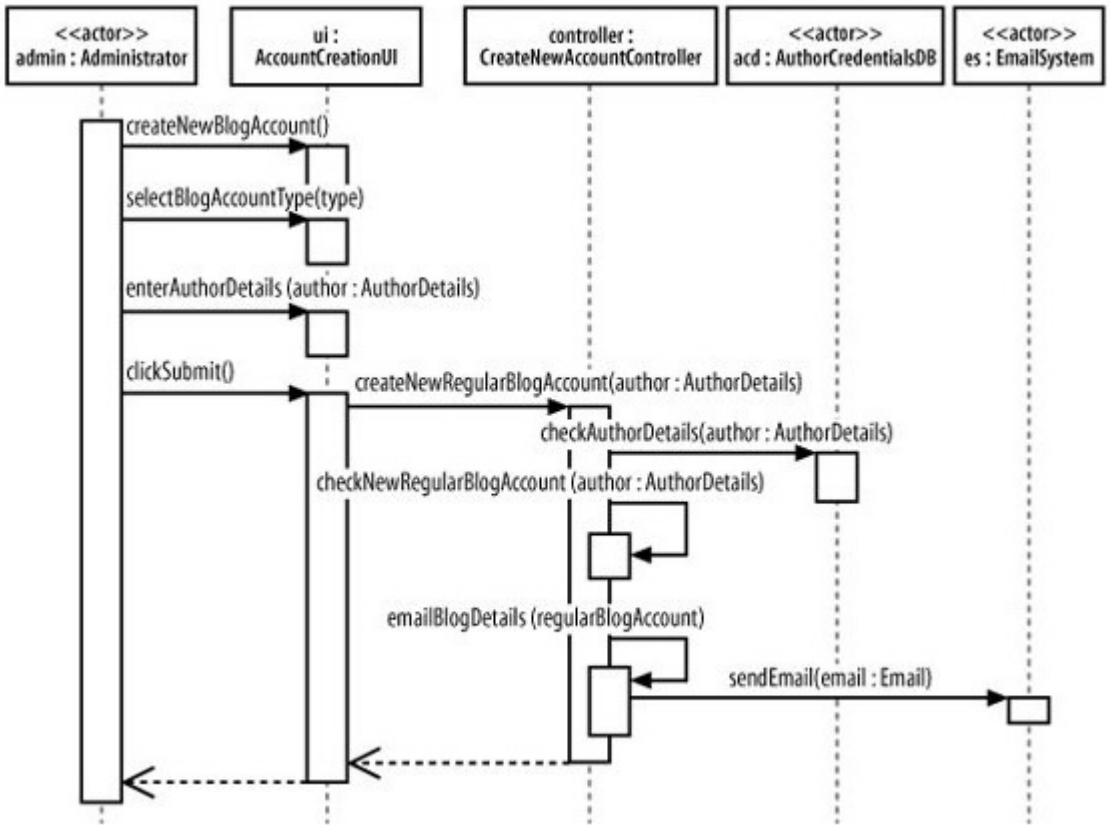
Bảng trên trình bày tất cả các bước trong use case Create a new Regular Blog Account bao gồm các bước nó thừa kế từ Create a new Blog Account hoặc sử dụng lại từ Check Identity. Cái này được làm vì bạn có thể dễ dàng nhìn thấy tất cả các bước Main Flow ở cùng 1 chỗ.
Thực tế thì bạn có thể tìm các thông tin chi tiết về 3 use case một cách riêng rẽ và gom chúng lại mà không cảm thấy khó chịu gì hết.
Bảng trên chỉ trình bày các main flow là các bước diễn ra mà không quan tâm về các phần mở rộng, nhưng như thế cũng đủ để tạo ra 1 biểu đồ tuần tự cấp cao



Biểu đồ tuần tự chỉ ra các actor tương tác với hệ thống và hệ thống của bạn được trình bày đơn giản
Hình trên tập trung vào các participant và các thông điệp liên quan trong use case. Use case tương tự được mô hình hóa trong chương 3 như một biểu đồ hoạt động, tập trung vào các tiến trình liên quan hơn là các participant

b. Việc phá vỡ các tương tác thành các participant riêng rẽ

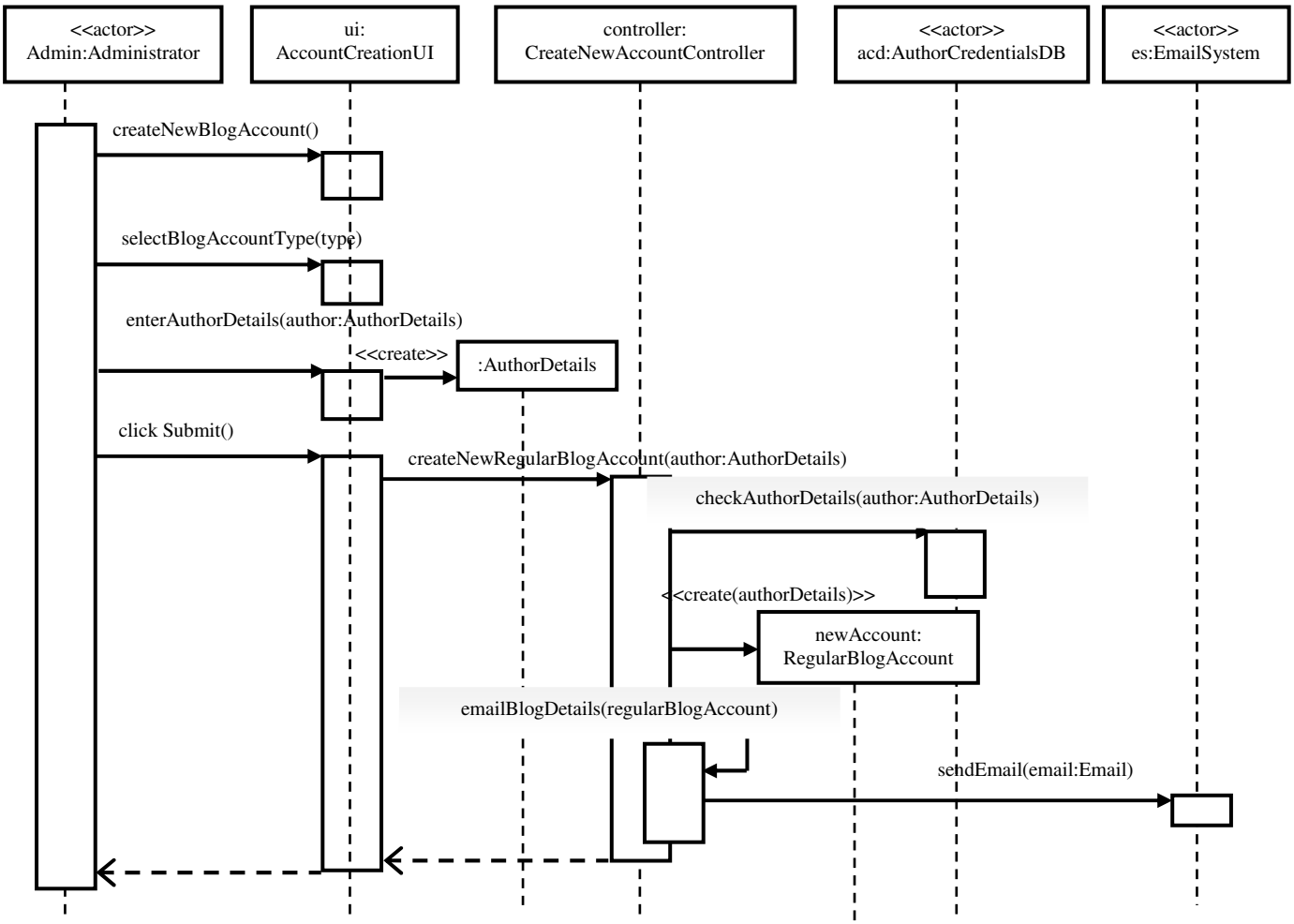
Ở điểm này, hình trên chỉ chỉ ra các tương tác phải diễn ra giữa các actor bên ngoài và hệ thống của bạn bởi vì đó là mức các bước của mô tả use case được viết. Trong biểu đồ tuần tự, hệ thống của bạn được xem như 1 participant đơn, ContentManagementSystem. Tuy nhiên, nếu bạn không có ý định implement hệ thống quản lý nội dung của bạn thành 1 đoạn code cố định (thường thì đây không phải là ý hay), thì đây là lúc phân rã ContentManagementSystem để cho thấy các phần bên trong như hình sau :



Biểu đồ đối tượng có thể phức tạp hơn bằng cách thêm vào 1 cặp participant phụ và một vài tương tác nhỏ hơn. Ở hình trên, biểu đồ đối tượng gốc đã được lọc lại, participant ContentManagementSystem đã được thay thế bởi các participant thực sự liên quan.
Đặt các participant và các tương tác đúng trong biểu đồ tuần tự chi tiết ngay từ lúc đầu là 1 việc làm khó. Update biểu đồ cũng là 1 thử thách. Vì vậy, hãy bỏ ra nhiều thời gian làm việc với biểu đồ tuần tự của bạn cho đến khi mọi thứ đều đúng.

c. Ứng dụng việc tạo participant

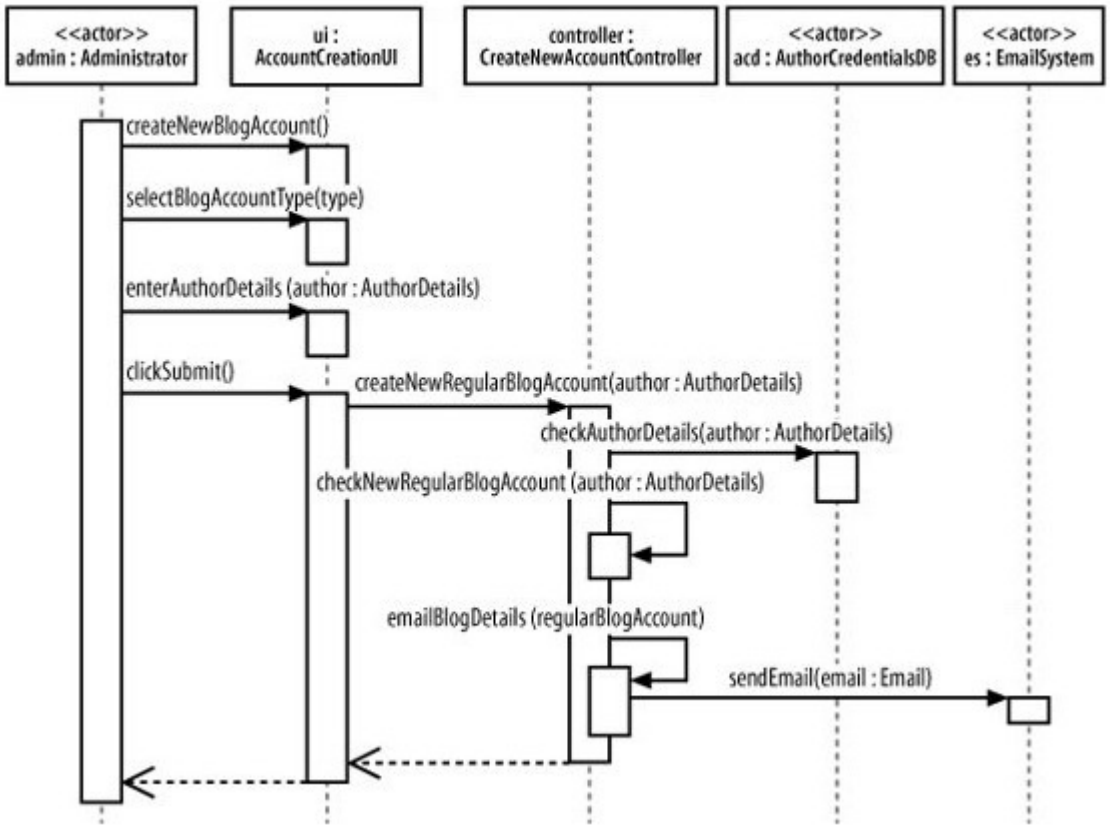
Có một vài thiếu xót nghiêm trọng trong biểu đồ tuần tự bên trên. Tiêu đề của biểu đồ là Create a new Regular Blog Account nhưng việc tạo blog account thật sự là ở đâu? Chúng ta sẽ thêm vào biểu đồ một vài phần để chỉ ra việc tạo blog account thật sự



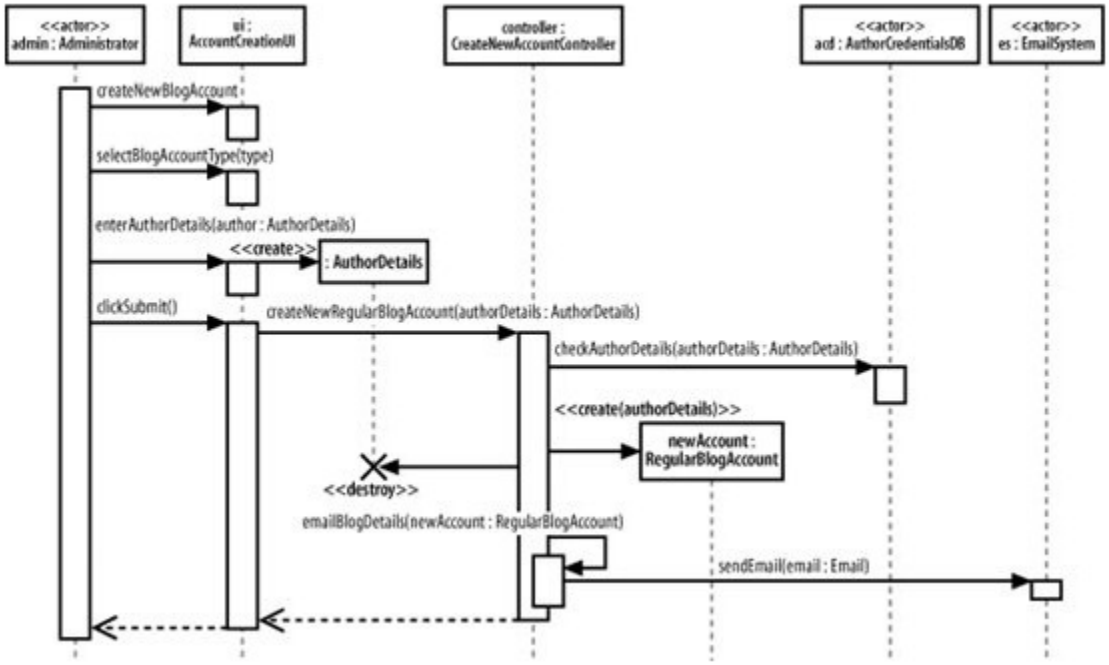
Đường sinh tồn của participant hữu ích một cách đặc biệt khi dùng để chỉ ra một participant vừa được tạo. Trong hình trên, các participant AuthorDetails và RegularBlogAccount không tồn tại khi biểu đồ tuần tự bắt đầu, nhưng chúng được tạo ra trong suốt quá trinh thực thi của nó.
Các participant AuthorDetails và newAccount:RegularBlogAccount được tạo bằng cách đáp lại thông điệp create. Mỗi thông điệp create được kết nối trực tiếp tới hộp tiêu đề cho cái participant mà đạng được tạo, gửi đi bất cứ thông tin nào cần khi tạo ra 1 participant mới. Bằng cách đặt các hộp tiêu đề của participant vào vị trí nơi thông điệp create được gửi, biểu đồ có thể chỉ ra điểm bắt đầu đường sinh tồn của participant một cách rõ ràng.

d. Ứng dụng việc xóa 1 participant:

Khi participant authorDetails:AuthorDetails không còn cần nữa khi newAccount:RegularBlogAccount được tạo. Để hủy participant authorDetails:AuthorDetails tại chỗ này, bạn có thể sử dụng thông điệp destroy tường minh kết hợp với kí hiệu dấu thập hủy.



e. Ứng dụng của thông điệp không đồng bộ:

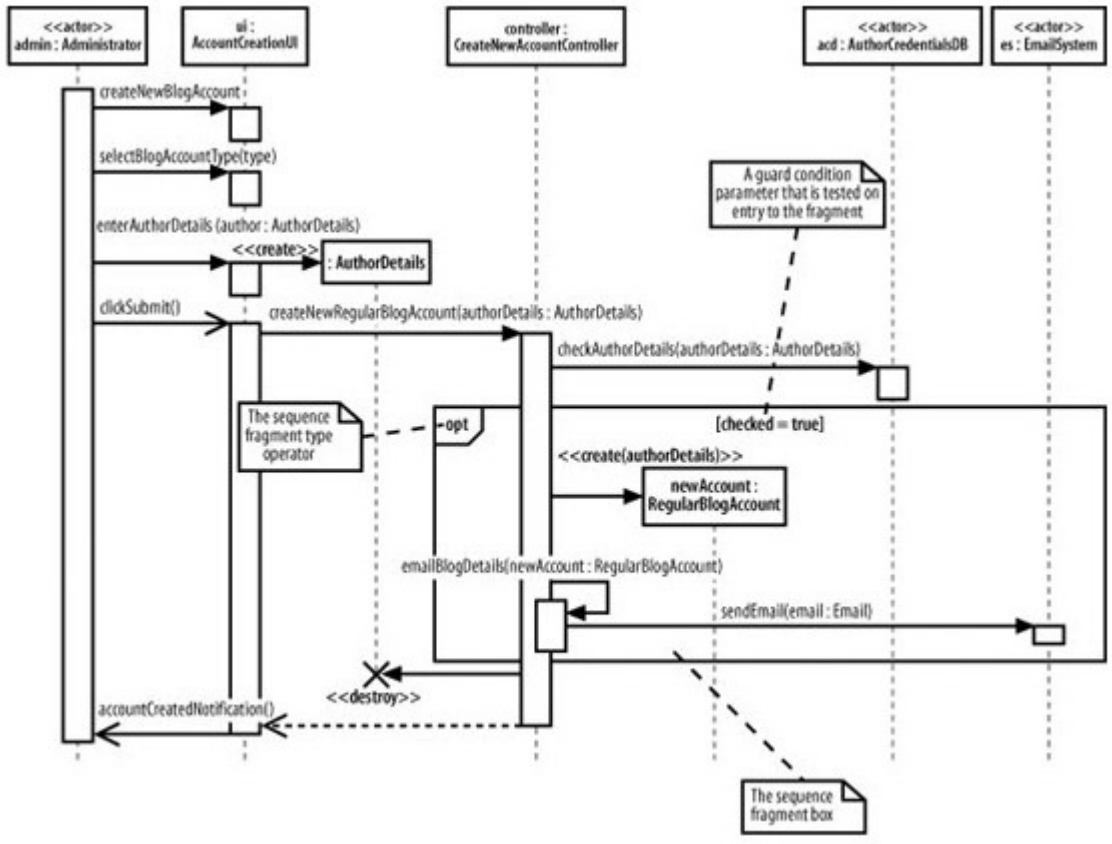


Khi thông điệp clickSubmit là đồng bộ thì khi administrator nhấn nút submit, hệ thống sẽ những hoạt động cho đến khi blog account mới được tạo. Khi chuyển thông điệp clickSubmit từ đồng bộ sang không đồng bộ thì trong khi hệ thống quản lý nội dung tạo 1 blog account mới,

administrator có thể thao tác với các tác vụ khác, hệ thống sẽ không phải đợi blog account mới được tạo.
Đối với administrator để nhận thông tin phản hồi việc blog account mới có tạo được hay không, mũi tên trả về đơn giản phải được thay thế bằng thông điệp không đồng bộ accountCreationNotification( ) bởi vì thông điệp không đồng bộ không có giá trị trả về.

4. Quản lý các tương tác phức tạp với khúc tuần tự

Hầu hết những gì bạn đã thấy trong chương này thì khá quen thuộc với những người đã sử dụng biểu đồ tuần tự trong UML 1.x. Nhưng từ lúc này trở đi thì khác hoàn toàn.
Trong những phiên bản trước UML 2.0, biểu đồ tuần tự trở thành quá lớn và lộn xộn, và nó chứa quá nhiều chi tiết đến nỗi không thể hiểu và quản lý được. Không có cách chuẩn nào để trình bày các dòng loop và chuyển đổi vì vậy bạn phải tự tìm cách giải quyết cho chình mình. Điều này dẫn đến tăng kích cỡ và độ phức tạp của biểu đồ hơn là giúp tacó thể quản lý nó.
Từ đó ta cần có một cái gì đó cho phép tạo ra biểu đồ tuần tự có cấu trúc có thể trình bày các tương tác phức tạp, chẳng hạn các dòng loop hay chuyển đổi. Câu trả lời từ những nhà thiết kế UML 2.0 là khúc tuần tự.
1 khúc tuần tự đại diện là 1 hộp mà đóng lại 1 phần của các tương tác trong biểu đồ tuần tự. Đỉnh góc trái của hộp phân khúc chứa 1 operator cho biết đây là loại phân khúc nào.



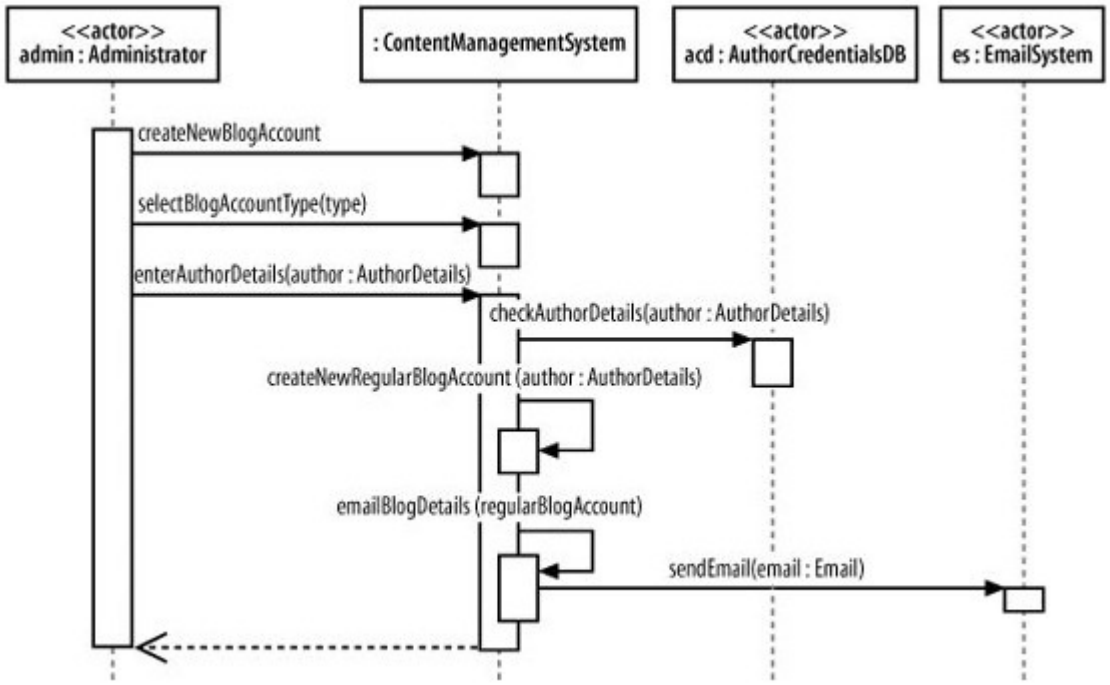
Trong hình operator là opt điều này có nghĩa đây là phân khúc tự chọn, tất cả các tương tác chứa trong khung sẽ thực thi dựa vào kết quả của đối số điều kiện bảo vệ
Một số loại không cần đối số thêm vào chẳng hạn như ref nhưng đối số điều kiện bảo vệ thì cần khi loại phân khúc là opt để quyết định xem có thực hiện tương tác hay không. Trong trường

hợp loại phân khúc là opt, các tương tác mà phân khúc chứa sẽ thực thi chỉ khi điều kiện bảo vệ tương ứng là true.

- **Phân khúc ref**

Loại ref giúp giảm bớt công việc quản lý. Bạn không còn cảm thấy sợ hãi khi nhìn thấy những biểu đồ tuần tự khổng lồ được tạo ra từ các hệ thống phức tạp nữa. Dưới đây là phân khúc ref, 1 phần của biểu đồ lớn hơn.

Các tương tác khi actor administrator chọn loại blog account nào để tạo thì được chứa trong phân khúc được tham chiếu như sau:



Ngoài lợi ích làm giảm kích cỡ của biểu đồ của bạn, phân khúc ref còn cho bạn xài lại 1 bộ các thao tác thông dụng, chỉ cần tham chiếu tới. Phân khúc ref có cách làm tương tự như quan hệ use case <<include>>.

- **Bảng tóm tắt các loại phân khúc của UML 2.0:**

UML 2.0 chứa một bộ lớn các loại phân khúc khác nhau mà bạn có thể đưa vào biểu đồ tuần tự làm cho chúng ý nghĩa hơn

Loại	Đối số	Tại sao nó có ích?
ref	none	Đại diện 1 tương tác được định nghĩa bất cứ đâu trong mô hình. Giúp bạn quản lý 1 biểu đồ lớn bằng cách phân chia và sử dụng lại 1 bộ các tương tác, tương tự như khi mối quan hệ use case <<include>> được áp dụng
assert	none	Chỉ rõ các tương tác chứa trong khung phân khúc phai diễn ra chính xác như chúng đã được khai báo. Mặt khác phân khúc được khai báo không đúng hoặc có ngoại lệ phải được chỉ ra. Các việc này tương tự như phát biểu assert trong Java. Cái này được áp dụng khi ta cần mỗi

		bước trong 1 tương tác phải thực hiện thành công. Ví dụ, khi mô hình hóa 1 công việc kinh doanh
loop	số lần min, số lần max, [điều kiện bảo vệ]	Lặp lại các tương tác trong khung với số lần cụ thể cho đến khi điều kiện = false. Tương tự Java, C# for(..) loop. Nó giúp ích cho bạn rất nhiều khi bạn muốn thực thi một bộ các tương tác với số lần cụ thể
break	none	khi các tương tác trong khung break diễn ra, các tương tác sẽ ngưng ngay. Tương tự phát biểu break trong Java và C#
alt	[điều kiện bảo vệ1] ... [điều kiện bảo vệ2] ... [else]	Lệ thuộc điều kiện bảo vệ đầu tiên được kích hoạt sang true thì các bộ tương tác con tương ứng sẽ được thực thi. Giúp bạn chỉ ra là bộ các tương tác sẽ chỉ thực thi khi các điều kiện đúng. Tương tự như phát biểu if...else
opt	[điều kiện bảo vệ]	Các tương tác trong khung sẽ thực thi nếu điều kiện bảo vệ là true. Tương tự lệnh if nhưng không có else
neg	none	Khai báo các tương tác bên trong khung không được thực thi. Bạn dùng khung này để đánh dấu các tương tác không được thực thi cho đến khi bạn chắc rằng các tương tác này có thể được xóa. Giúp bạn chỉ ra cái gì đó không được làm. Ví dụ, bạn muốn participant không thể gọi được read() sau khi đã gọi close().
par	none	Cho biết các tương tác trong khung có thể thực thi đồng thời

Các khung làm cho việc tạo và quản lý các biểu đồ tuần tự dễ dàng hơn, chính xác hơn. Tuy nhiên, bạn cần nhớ rằng khung không phải cô lập, bạn có thể nối một số khung để mô hình hóa một cách chính xác các tương tác trong biểu đồ. Hãy thận trọng nếu biểu đồ của bạn trở nên lớn và khó sử dụng cho dù bạn có các khung. Lí do đơn giản có thể là do bạn đang mô hình hóa quá nhiều thứ trong 1 chuỗi liên tiếp.

VIII. Tập trung vào liên kết của các tương tác: Communication Diagrams

1. Giới thiệu:

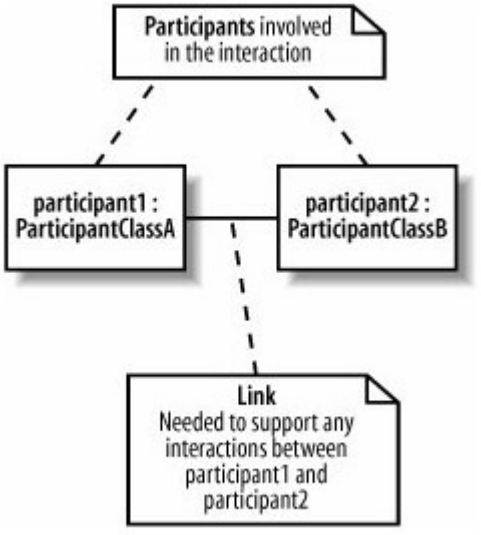
Mục đích chính của biểu đồ tuần tự là trình bày thứ tự các sự kiện giữa các phần của hệ thống liên quan đến 1 tương tác cụ thể. Biểu đồ truyền tin đưa ra cái nhìn khác về tương tác bằng cách tập trung vào các link giữa các paticipant.
Biểu đồ truyền tin thì được sử dụng khi các participant cần các link để gởi các thông điệp của một tương tác. Chỉ cần ngó lướt qua biểu đồ truyền tin, bạn có thể chỉ ra participant nào cần được kết nối để thực hiện 1 tương tác.
Trên biểu đồ tuần tự, các link giữa các participant thì hàm ý là có 1 thông điệp đã được gởi giữa chúng. Trên biểu đồ truyền tin thì thứ tự các sự kiện liên quan 1 tương tác là phần thông tin thứ hai.

Biểu đồ tuần tự và biểu đồ tương tác giống nhau, các công cụ UML có thể chuyển đổi qua lại giữa hai loại biểu đồ này. Khi bạn muốn đề cập đến các thứ tự của các tương tác, bạn sẽ sử dụng biểu đồ tuần tự, ngược lại bạn sẽ sử dụng biểu đồ truyền tin.

2. Các khái niệm:

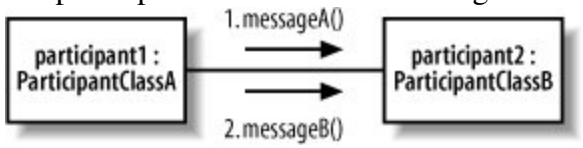
a. Các participant, link và thông điệp:

Biểu đồ truyền tin được cấu thành từ 3 yếu tố : paricipant, các link truyền tin giữa các participant này, và các thông điệp truyền tin được gửi cùng với các link này.



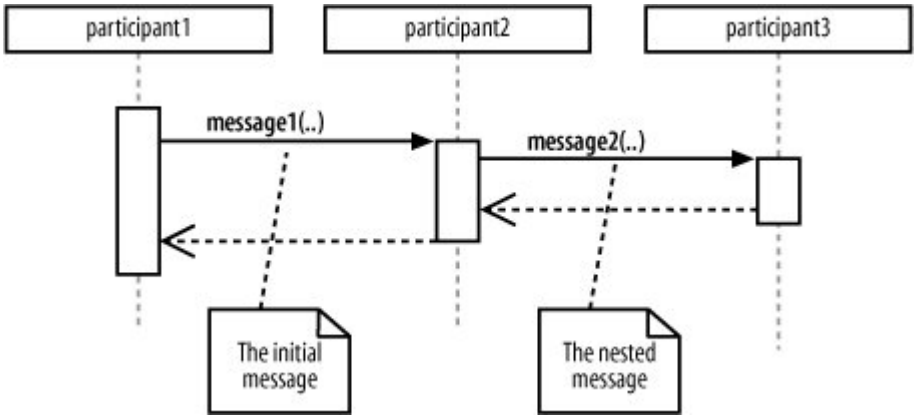
Các participant trong biểu đồ truyền tin được kí hiệu là các hình chữ nhật. Tên participant và tên lớp được đặt ở giữa hình chữ nhật. Tên participant theo định dạng *<tên>:<class>,* tương tự như participant trong biểu đồ tuần tự.

Bạn cần cho biết tên lớp hoặc tên participant (hoặc cả hai). Nếu vì một lí do nào đó, bạn không biết về tên và class khi đó participant là ẩn danh và tên cùng với class sẽ được bỏ trống
1 link truyền tin là 1 đường đơn nét nối giữa 2 participant. Mục đích của link là để truyền tin giữa các participant khác nhau. Nếu không có link, 2 participant không thể tương tác với nhau

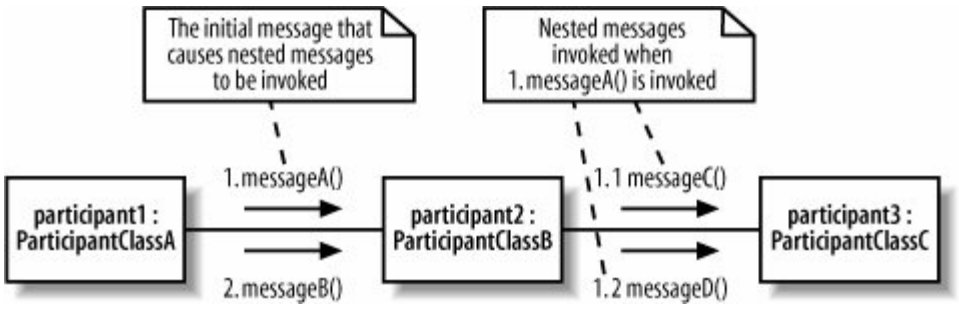


1 thông điệp trên biểu đồ truyền tin được biểu diễn sử dụng 1 mũi tên như hình trên từ nơi gửi đến nơi nhận. Tương tự như các thông điệp trên biểu đồ tuần tự, dạng của thông điệp gồm có tên và 1 danh sách các đối số. Tuy nhiên, không như biểu đồ tuần tự, nếu dạng thông điệp chỉ có vậy thì chưa đủ cho biểu đồ truyền tin, bạn phải cho biết thứ tự các thông điệp được gọi trong quá trình tương tác.

Biểu đồ truyền tin không cần xếp từ trên xuống như biểu đồ tuần tự. Vì vậy thứ tự các thông điệp trên biểu đồ truyền tin đươc biết dựa vào con số đứng trước thông điệp, bắt đầu từ số 1. Do đó theo hình trên, thông điệp A được gọi trước, sau đó là thông điệp B.
Khi một thông điệp gây ra việc gọi 1 thông điệp khác, khi đó thông điệp thứ hai được gọi là chồng trong thông điệp gốc.



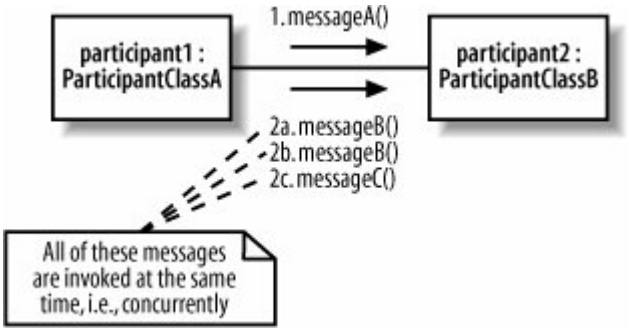
Biểu đồ truyền tin sử dụng 1 giản đồ số thông điệp của chúng để chỉ ra thứ tự các thông điệp. nếu chúng ta có thông điệp đầu là 1., khi đó bất cứ thông điệp chồng nào bên trong thông điệp này sẽ được bắt đầu là 1. thêm 1 số sau cho thứ tự của thông điệp chồng. Nếu thông điệp đầu là 1. Khi đó số của thông điệp chồng đầu tiên sẽ là 1.1



Thông điệp A dẫn tới thông điệp chồng C, sau đó là D, trước khi thông điệp B được gọi

- **Các thông điệp diễn ra cùng lúc**

Biểu đồ truyền tin cho chúng ta 1 câu trả lời đơn giản về vấn đề các thông điệp được gọi cùng lúc. Ở biểu đồ tuần tự thì chúng ta cần các cấu trúc phức tạp, chẳng hạn khung par, còn ở biểu đồ truyền tin thì dễ hơn nhiều, chúng ta sẽ đặt thêm 1 ký tự vào sau số thự tự của thông điệp để chỉ ra rằng các thông điệp này diễn ra cùng lúc.



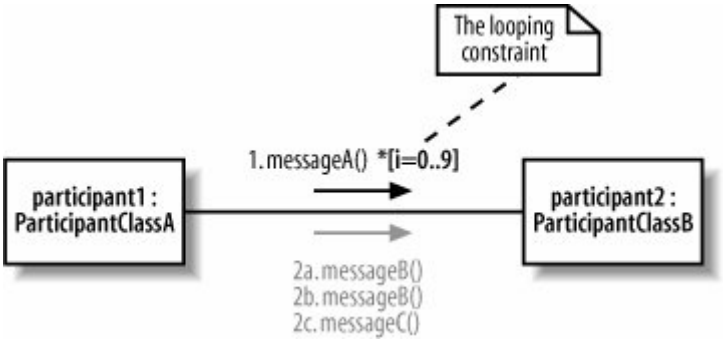
Các thông điệp 2a. messageB( ), 2b. messageB( ), và 2c. messageC() được gọi cùng lúc sau khi thông điệp 1.messageA() được gọi.

- **Gọi lại thông điệp**

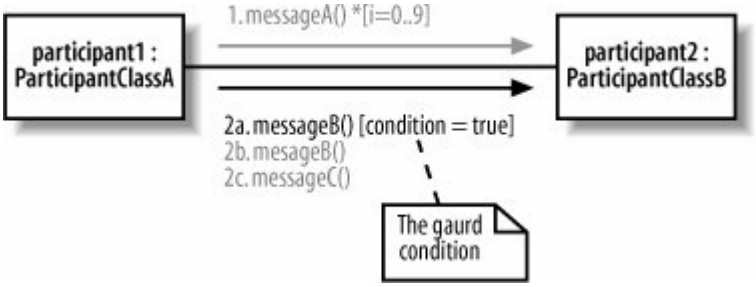
  Khi mô tả các thông điệp trên biểu đồ, bạn sẽ muốn 1 thông điệp được gọi 1 số lần nào đó. Điều này tương tự như thông điệp của bạn được gọi trong vòng lặp for(..) .
  Mặc dù UML không thực sự chỉ cho bạn biết làm cái này như thế nào nhưng nó có phát biểu là 1 dấu sao * nên được sử dụng trước khi ràng buộc lặp được áp dụng. Phát biểu này có nghĩa là **[i=0 .. 9]* là 1 cách đúng để chỉ ra 1 cái gì đó được lặp lại 10 lần.

- **Gửi một thông điệp dựa trên 1 điều kiện nào đó**

  Đôi khi bạn muốn thông điệp của mình gửi đi chỉ khi 1 điều kiện cụ thể nào đó là true. Ví dụ, hệ thống của bạn có 1 thông được sẽ được gọi chỉ khi thông điệp trước đã được thực thi đúng. Giống như các khung của biểu đồ tuần tự, các thông điệp của biểu đồ truyền tin có thể có 1 bộ các bảo vệ để mô tả các điều kiện cần được kích hoạt trước khi thông điệp được gọi



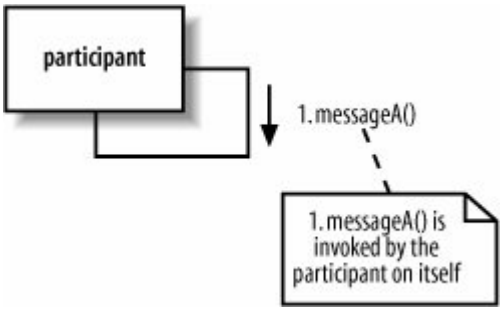
Ở trên thì 1.messageA() được thêm điều kiện là lặp mới là thông điệp phải thực hiện 10 lần rồi các thông điệp 2a. messageB( ), 2b. messageB( ), và 2c. messageC mới được gọi. 1 điều kiện bảo vệ là 1 phát biểu logic. Khi điều kiện là true thì thông điệp tương ứng được thực thi ngược lại thông điệp sẽ bị bỏ qua.



Ở hình trên thì thông điệp 2a. messageB( ) sẽ được gọi cùng lúc với 2b. messageB( ), và 2c. messageC nếu điều kiện biểu thức là true, ngược lại thông điệp 2a. messageB( ) sẽ không được gọi và 2b. messageB( ), và 2c. messageC sẽ thực thi.

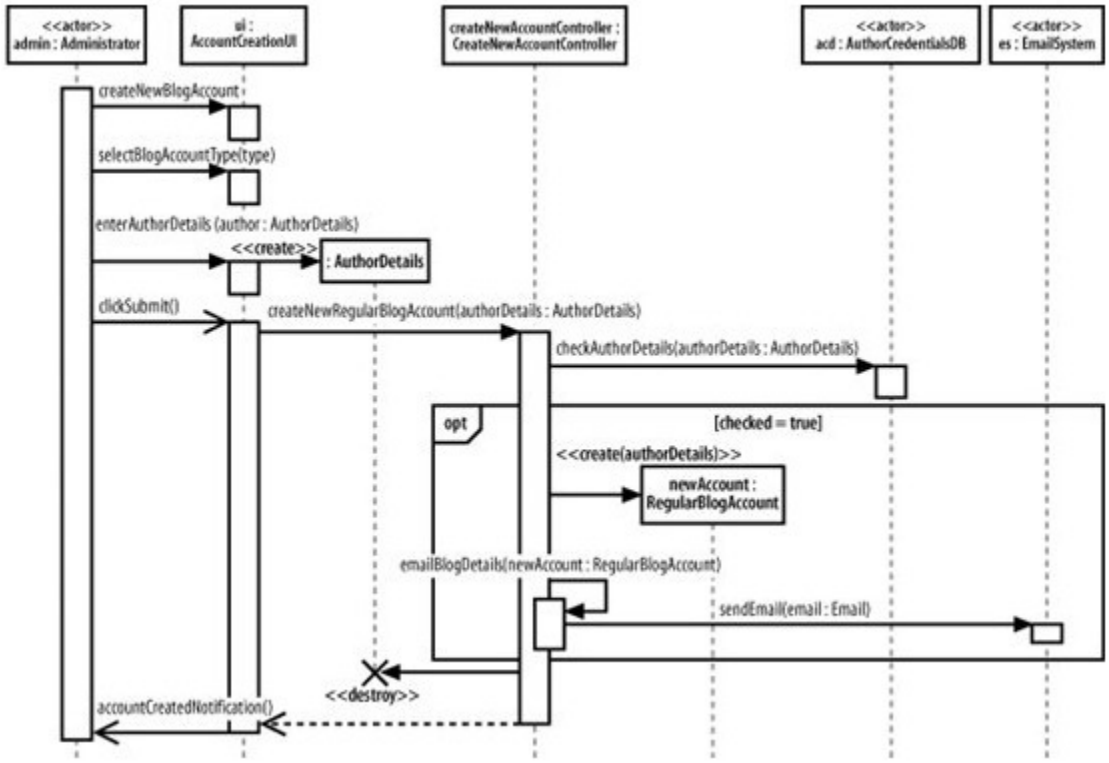
- **Khi một participant gởi thông điệp cho chính nó**

Có lẽ điều này hơi lạ, nhưng bạn thử xét trong phạm vi phần mềm, khi trường hợp 1 đối tượng gọi 1 phương thức riêng của nó thì bạn sẽ hiểu tại sao chúng ta cần cái dạng truyền tin này (và thậm chí là phổ biến). Như biểu đồ tuần tự, 1 participant trên biểu đồ truyền tin cũng có thể gởi 1 thông điệp cho chính nó. Tất cả những gì chúng ta cần là 1 link từ 1 participant tới chính nó.



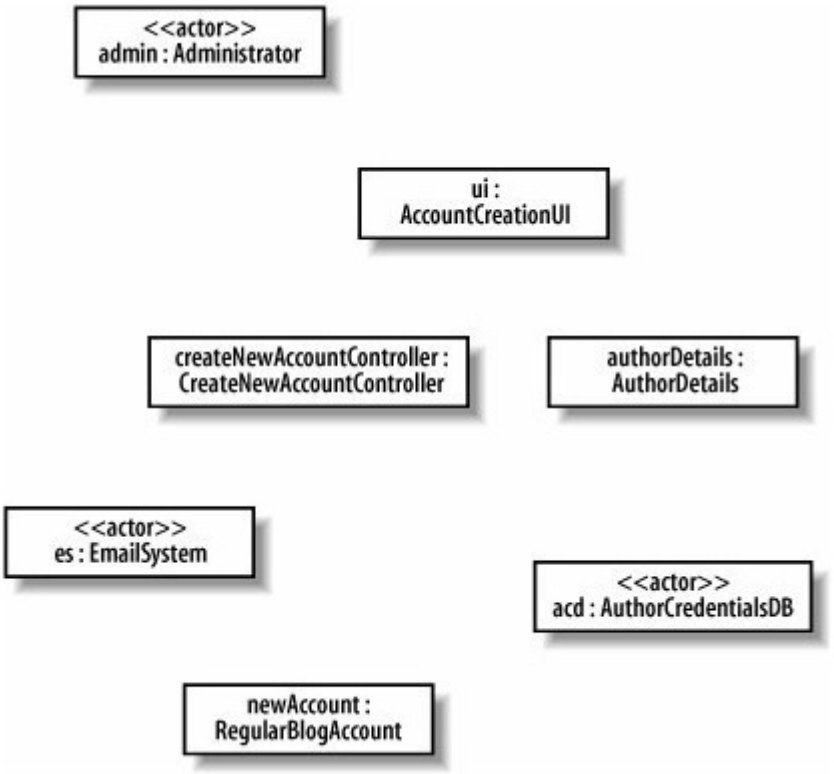
b. Làm quen với tương tác trong biểu đồ truyền tin

Bây giờ chúng ta sẽ xét 1 bài tập ví dụ, chúng ta sẽ lấy 1 trong các biểu đồ tuần tự của chương 7 và trình bày cách các tương tác được mô hình hóa trong biểu đồ truyền tin.

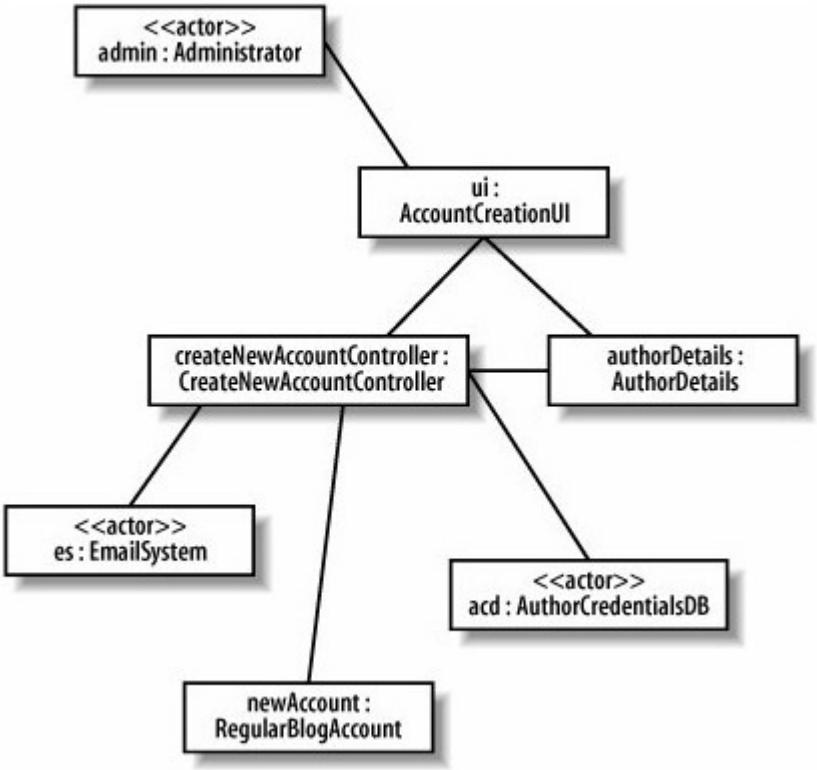


Đây là biểu đồ tuần tự mô tả các tương tác diễn ra trong 1 CMS khi 1 blog account mới được tạo. Hãy xem lại chương 7 nếu bạn chưa nắm kỹ các kí hiệu trên biểu đồ tuần tự. Biểu đồ tuần tự thì có khá nhiều kí hiệu, vì vậy có thể bạn sẽ mất một thời gian để nắm tất cả chúng. Không nhất thiết bạn phải có 1 biểu đồ tuần tự trước khi tạo 1 biểu đồ truyền tin. Bạn có thể tạo biểu đồ tuần tự hay biểu đồ truyền tin cho các tương tác của bạn theo bất cứ thứ tự nào bạn cảm thấy thích hợp.

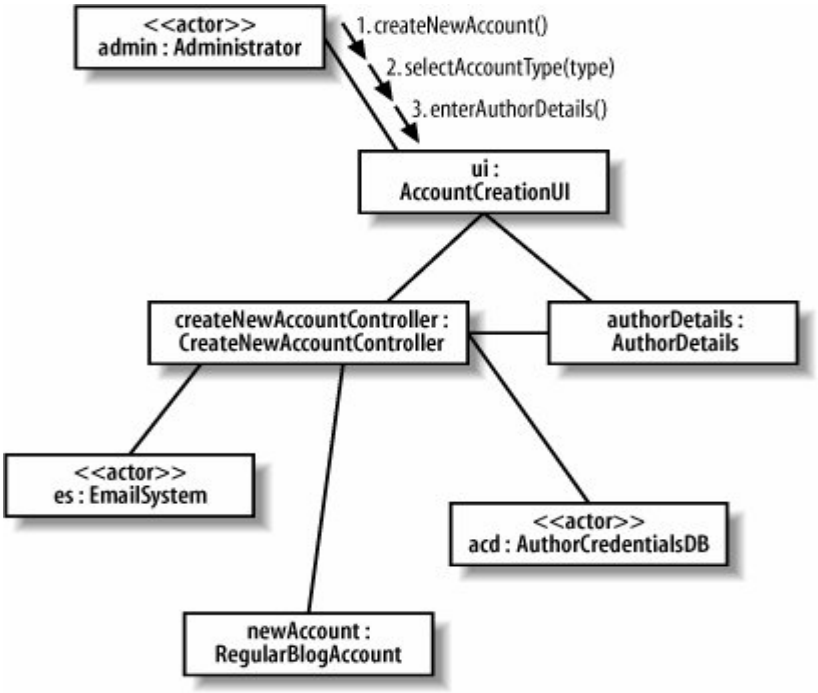
- Trước hết là add các participant từ biểu đồ tuần tự bên trên sang biểu đồ truyền tin ta đang tạo



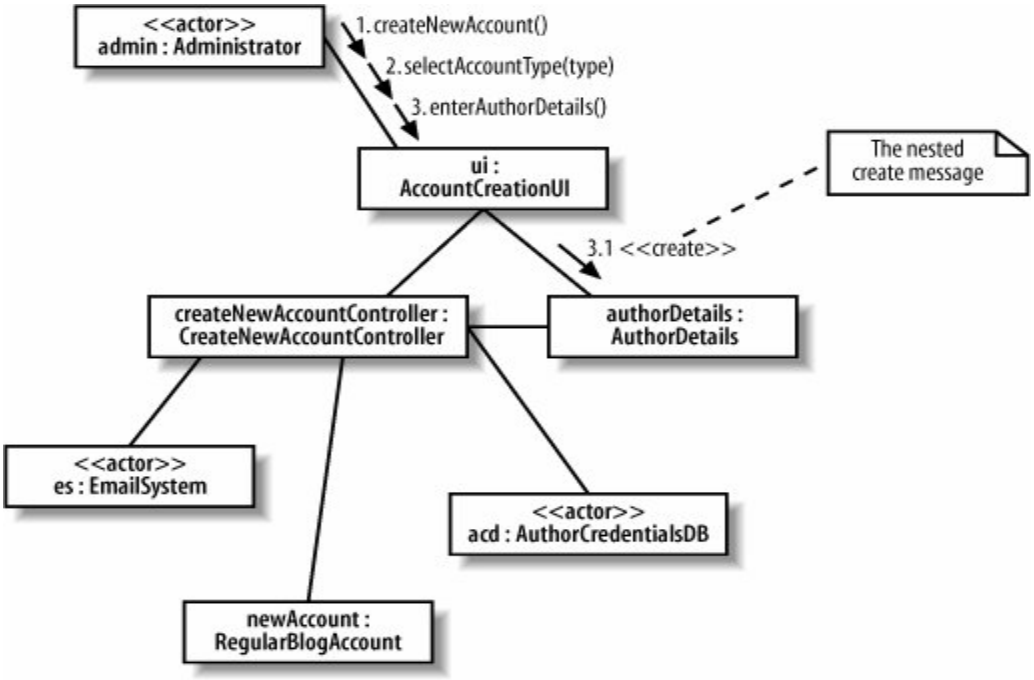
- Kế tiếp, các link giữa các participant được thêm vào để chúng có thể liên lạc với nhau



Các link được xác định dựa vào biểu đồ tuần tự, các link cần để thực hiện truyền tin sẽ được thêm vào biểu đồ truyền tin

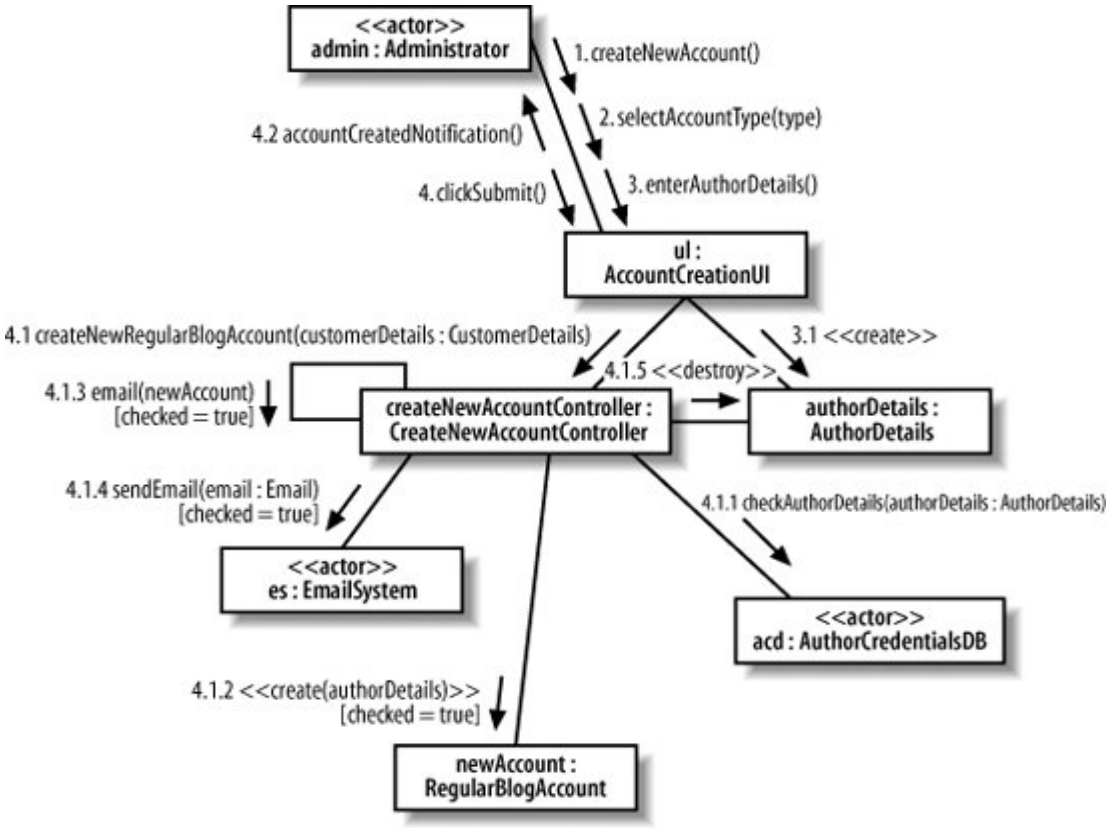


Khi thông điệp ban đầu hoặc các thông điệp được thêm vào biểu đồ truyền tin, mọi thứ bắt đầu phức tạp hơn. Thông điệp 3.enterAuthorDetails( ) tạo ra 1 thông điệp chồng được gửi đi từ participant ui:AccountCreationUI để tạo participant authorDetails:CustomerDetails mới.



Khi thông điệp <<*creat*>> được thêm vào biểu đồ truyền tin, nó sẽ được đánh số là 3.1 để chỉ ra nó được chồng trong thông điệp 3. enterAuthorDetails( )

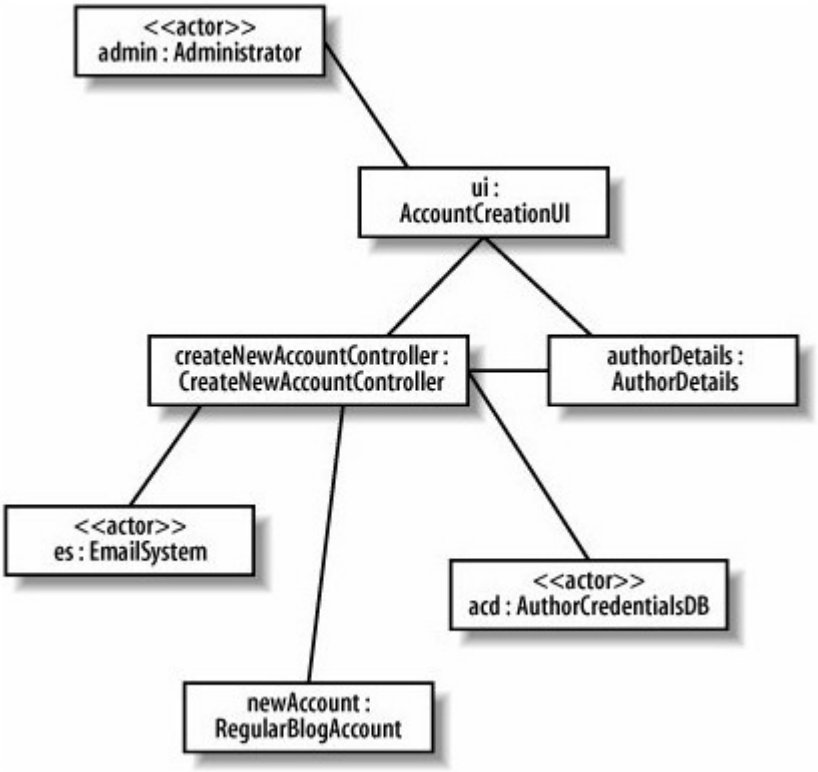
Làm tương tự cho tất cả bạn được biểu đồ như sau :

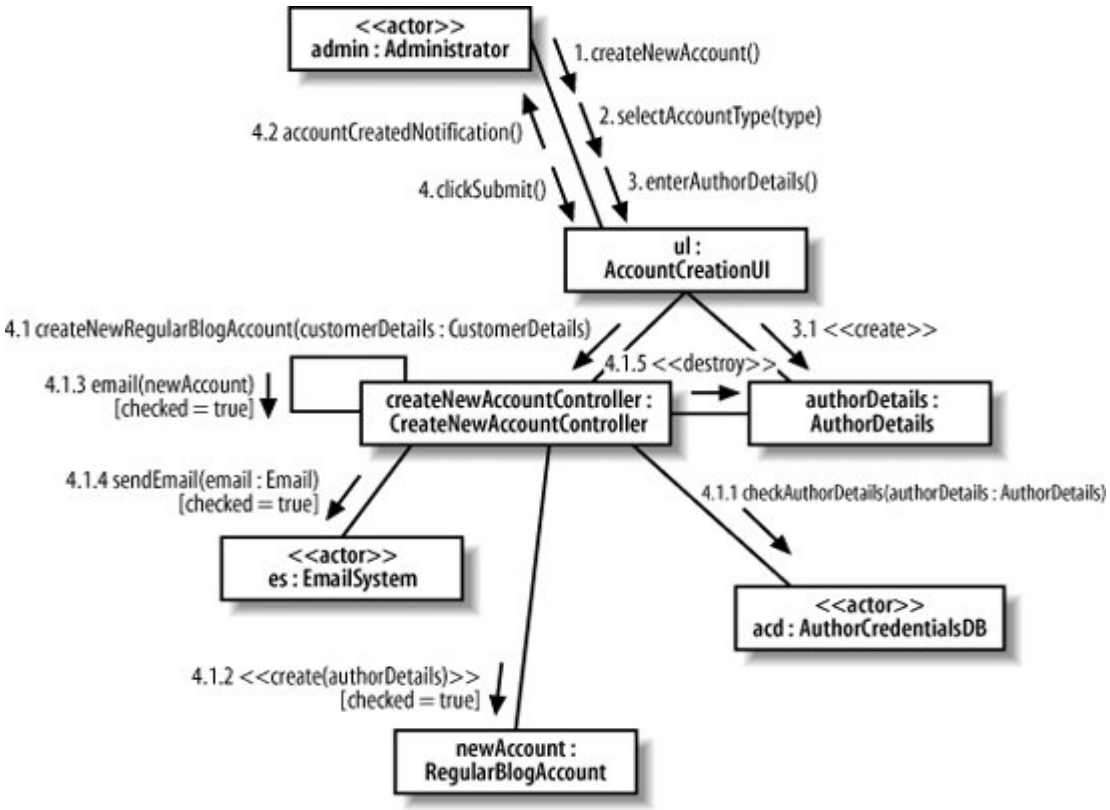


c. So sánh giữa biểu đồ thông tin và biểu đồ tuần tự

Biểu đồ truyền tin và biểu đồ tuần tự thì rất giống nhau, do đó sự so sánh là điều không thể tránh khỏi, nên làm để phân biệt chúng. Khi nào tốt nhất để sử dụng biểu đồ tuần tự, biểu đồ truyền tin hoặc kết hợp cả hai để mô hình hóa 1 tương tác cụ thể.

- **Về hình dạng**:


<<actor>>
admin : Administrator
ui :
AccountCreationUI
createNewAccountController :
CreateNewAccountController
authorDetails :
AuthorDetails
<<actor>>
es : EmailSystem
<<actor>>
acd : AuthorCredentialsDB
newAccount :
RegularBlogAccount


<<actor>>
admin : Administrator
1. createNewAccount()
2. selectAccountType(type)
3. enterAuthorDetails()
4. clickSubmit()
4.2 accountCreatedNotification()
ui :
AccountCreationUI
4.1 createNewRegularBlogAccount(customerDetails : CustomerDetails)
3.1 <<create>>
4.1.5 <<destroy>>
4.1.3 email(newAccount)
[checked = true]
createNewAccountController :
CreateNewAccountController
authorDetails :
AuthorDetails
4.1.4 sendEmail(email : Email)
[checked = true]
4.1.1 checkAuthorDetails(authorDetails : AuthorDetails)
<<actor>>
es : EmailSystem
<<actor>>
acd : AuthorCredentialsDB
4.1.2 <<create(authorDetails)>>
[checked = true]
newAccount :
RegularBlogAccount

- **Các sự kiện chính:**

Bảng dưới so sánh biểu đồ tuần tự và biểu đồ truyền tin để giúp bạn quyết định chọn loại biểu đồ nào phù hợp với mục đích mô hình hóa của bạn

Đặc tính	Biểu đồ tuần tự	Biểu đồ thông tin	Kết quả
Trình bày các participant 1 cách có hiệu quả	Các participant được sắp xếp theo chiều ngang trên đỉnh trang, nếu kí hiệu hộp tạo participant không được sử dụng. Dễ tập hợp các participant liên quan trong 1 tương tác cụ thể	Các participant cũng như các link là tiêu điểm, chúng được trình bày 1 cách rõ rằng bằng các hình chữ nhật	Biểu đồ truyền tin rõ ràng là tốt hơn. Mặc dù cả hai đều có thể trình bày các participant hiệu quả như nhau. Nhưng biểu đồ truyền tin có điểm mạnh là các participant là một trong các tiêu điểm của chúng
Trình bày các link giữa các participant	Các link được hiểu ngầm. Nếu một thông điệp được gửi từ 1 participant này tới 1 participant khác. Khi đó nó hiểu là có một link tồn tại giữa các participant	Trình bày một cách tường minh các link giữa các participant. Thực vậy, đây là mục đích chính của các biểu đồ loại này	Biểu đồ thông tin tốt hơn. Chúng cho thấy một cách rõ ràng các link giữa các participant
Dạng thông điệp	Dạng thông điệp được mô tả một cách đầy đủ	Dạng thông điệp được mô tả một cách đầy đủ	Cả hai loại như nhau
Hỗ trợ các thông điệp đồng thời	Sử dụng các khung	Sử dụng kí hiệu số-kí tự	Cả hai loại như nhau
Hỗ trợ các thông điệp không đồng bộ	Sử dụng các mũi tên không đồng bộ	Biểu đồ truyền tin không có khái niệm về thông điệp không đồng bộ. Bởi vì chúng không tập trung vào thứ tự các thông điệp	Biểu đồ tuần tự rõ ràng có lợi trong việc này bởi vì chúng bởi vì chúng hỗ trợ các thông điệp không đồng bộ
Dễ xem thứ tự các thông điệp	Đây là thế mạnh của biểu đồ tuần tự. Biểu đồ tuần tự trình bày thứ tự các thông điệp một cách rõ ràng dựa vào vị trí các thông điệp trong trang biểu đồ (theo thứ tự từ trên xuống)	Sử dụng kí hiệu số.chồng	Biểu đồ tuần tự rõ ràng sẽ được sử dụng bởi vì chúng trình bày các thông điệp một cách rõ ràng và hệu quả
Dễ tạo và quản lý biểu đồ	Tạo 1 biểu đồ thì khá đơn giản. Tuy nhiên, quản lý thì thật sự rất khó nếu bạn không sử dụng công	Việc tạo ở biểu đồ truyền tin thì cũng rất đơn giản. Tuy nhiên, việc quản lý, đặc biệt	Thật khó chọn, tuy nhiên, biểu đồ thông tin thì có các nguyên tắc quản lý dễ chịu

	cụ trợ giúp	là việc đặt số cho các thông điệp cần được thay đổi, thì ý tưởng vẫn là sử dụng công cụ hỗ trợ	hơn

Tóm lại, bạn sẽ sử dụng biểu đồ tuần tự nếu quan tâm đến các dòng của thông điệp trong suốt 1 tương tác cụ thể. Và sử dụng biểu đồ truyền tin nếu bạn tập trung vào các link giữa các participant khác nhau trong tương tác. Có lẽ điều chúng ta có được quan trọng nhất từ sự so sánh trên là cả 2 loại biểu đồ đều truyền tại thông tin như nhau. Chúng có những thuận lợi khác nhau. Vì vậy cách tốt nhất là sử dụng cả hai.

IX. Tập trung vào thời gian tương tác: Timing Diagrams

1. Giới thiệu:

Biểu đồ thời gian nói về thời gian, dĩ nhiên. Biểu đồ tuần tự (chương 7) tập trung vào thứ tự các thông điệp, biểu đồ truyền tin thì cho biết các link giữa các participant, không có một chỗ nào trên các biểu đồ tương tác này dành để mô tả thông tin thời gian chi tiết. Bạn có 1 tương tác mà cần hơn 10 giây để hoàn thành hoặc 1 thông điệp cần hơn 1 nửa tổng thời gian tương tác để trả về. Nếu các thông tin này cần thiết cho 1 tương tác mà bạn đang mô hình hóa, thì khi đó biểu đồ thời gian sẽ là sự lựa chọn của bạn.

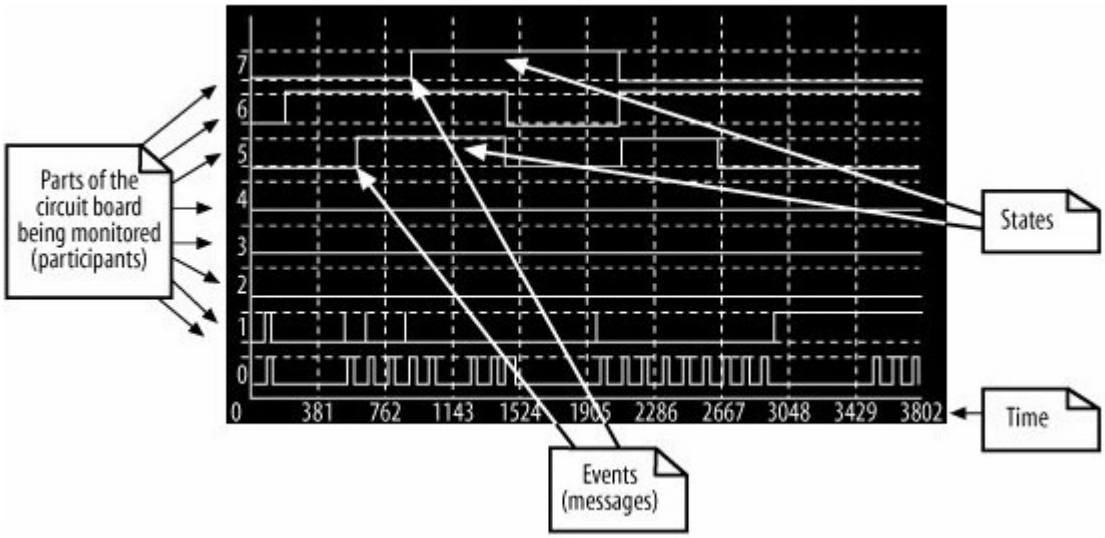
Thời gian tương tác có lẽ thích hợp nhất với các hệ thống thời gian thực, nhưng cũng không có giới hạn nào cho lĩnh vực này. Thật vậy, khi ta có nhu cầu biết được thông tin thời gian chính xác về 1 tương tác có lẽ chúng ta cũng không quan tâm lắm tới loại của hệ thống mà đang được mô hình hóa.

Trong biểu đồ thời gian, mỗi sự kiện có 1 thông tin thời gian được gắn với nó, cho biết một cách chính xác khi nào sự kiện được gọi, và mất bao lâu thì participant kia mới nhận được, mất bao lâu participant nhận cần để nó ở trong 1 trạng thái cụ thể. Không có biểu đồ thời gian cho một tương tác thì giống như ta nói : "tôi biết sự kiện nào cần diễn ra, nhưng tôi thật sự lo lắng chính xác thì chúng diễn ra khi nào hoặc chúng sẽ làm việc nhanh như thế nào"

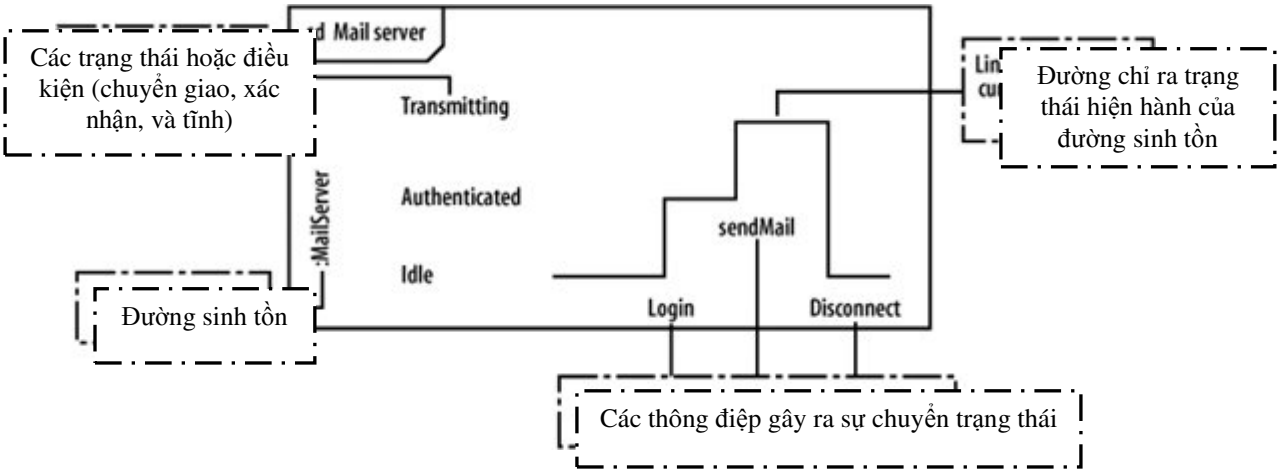
2. Các khái niệm:

a. Biểu đồ thời gian trông thế nào?

Biểu đồ thời gian thì khá lạ đối với những người biết ít về bảng mạch điện. Cái này là do biểu đồ thời gian tương tự như 1 sơ đồ mà bạn nhìn thấy trên 1 máy phân tích logic. Đừng quá lo lắng cho dù bạn chưa từng nhìn thấy nó trước đây

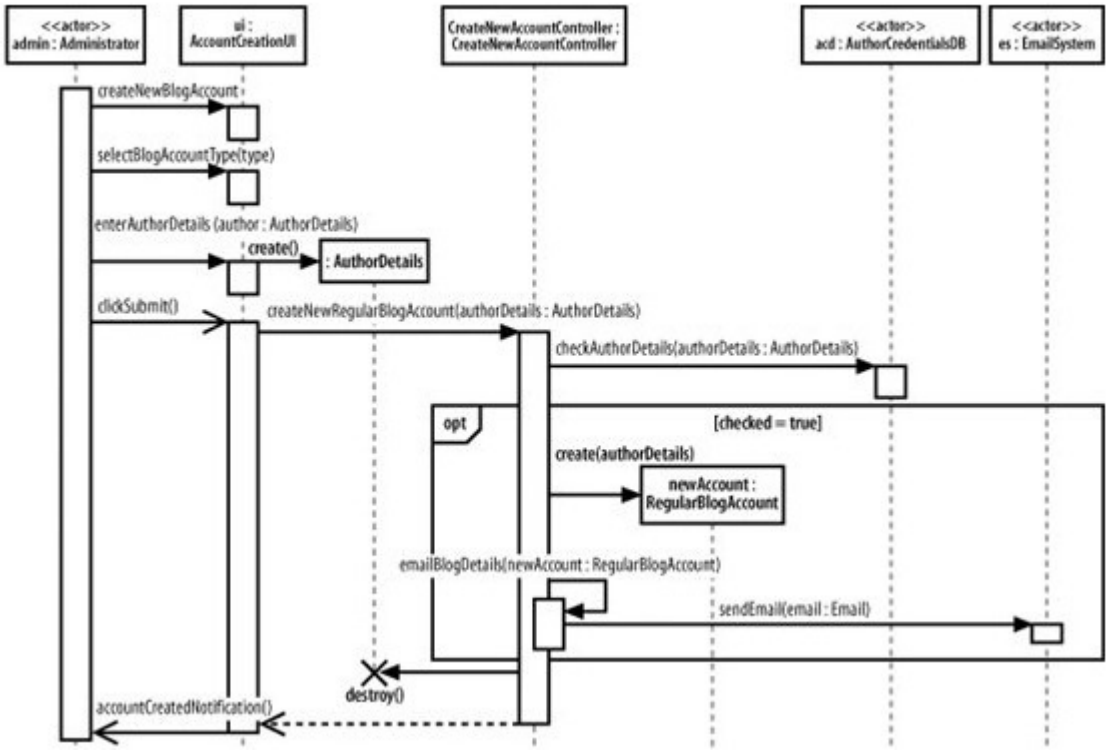


Biểu đồ thời gian cho biết các công việc tương tự cho các participant trong hệ thống của bạn. Trên 1 biểu đồ thời gian, các sự kiện là các tín hiệu của máy phân tích logic, các trạng thái là các trạng thái của participant khi nhận sự kiện. Sự tương tự giữa biểu đồ thời gian và máy phân tích logic là rất rõ ràng bạn có thể so sánh qua 2 hình.



b. Xây dựng 1 biểu đồ thời gian từ biểu đồ tuần tự.

Chúng ta cũng sẽ sử dụng cùng ví dụ trong chương biểu đồ tuần tự và biểu đồ thời gian, tương tác `Create a new Regular Blog Account`:



- **Các ràng buộc thời gian theo yêu cầu của hệ thống**

Tương tác được trình bày ở trên là kết quả của yêu cầu chẳng hạn như:

Hệ thống quản lý nội dung cho phép 1 administrator tạo 1 blog account regular mới, thông tin chi tiết về tác giả mà được cung cấp thì được kiểm tra sử dụng CSDL Author Credentials

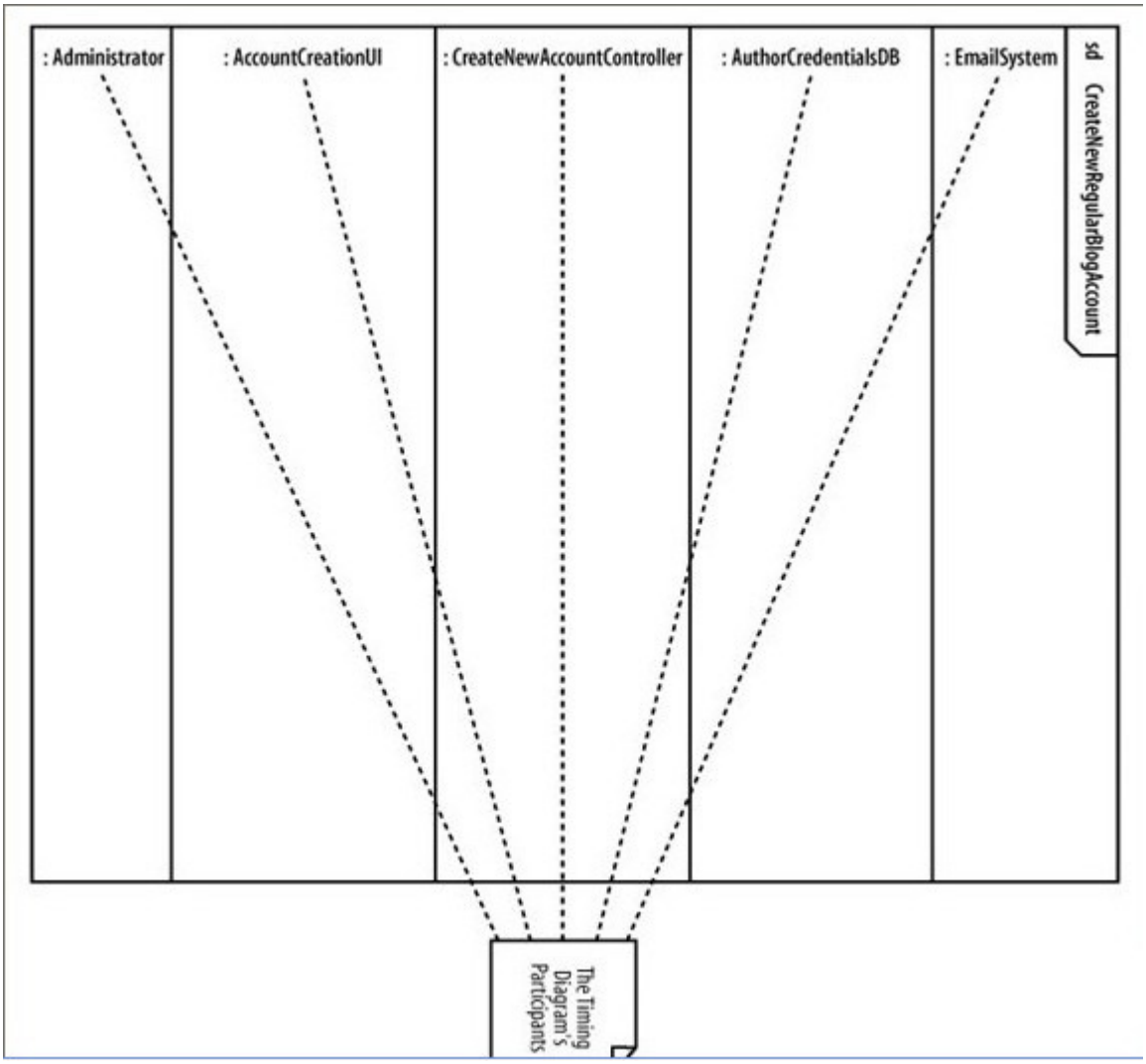
Bây giờ chúng ta sẽ mở rộng ràng buộc ban đầu với một cái gì đó dính tới thời gian.

Hệ thống quản lý nội dung cho phép 1 administrator tạo 1 blog account regular mới trong vòng 5 giây để nhập thông tin, thông tin chi tiết về tác giả mà được cung cấp thì được kiểm tra sử dụng CSDL Author Credentials.

Ràng buộc mới được sửa đổi thêm vào cái ràng buộc về thời gian ra lệnh cho hệ thống nhận, kiểm tra và tạo tài khoản mới trong bao lâu. Giờ chúng ta đã có nhiều thông tin hơn về thời gian để có thể lập mô hình sử dụng bểu đồ thời gian.

c. Đưa các participant vào biểu đồ thời gian

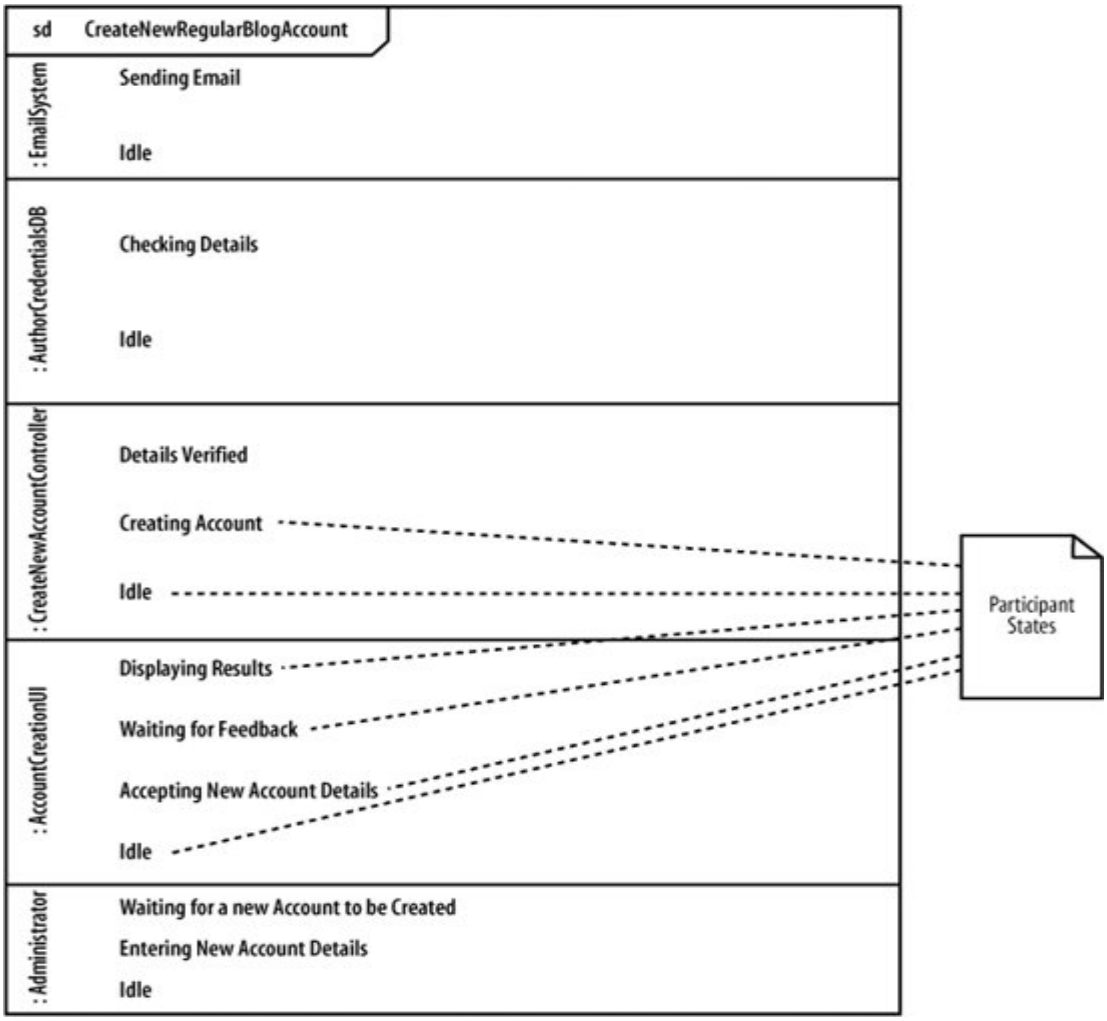
Đầu tiên, bạn tạo ra 1 biểu đồ thời gian kết hợp tất cả các participant liên quan trong tương tác new Regular Blog Account. Chúng ta sẽ bỏ qua tên của các participant, phần tạo và hủy, như các participant :AuthorDetails và :RegularBlogAccount bởi vì ở biểu đồ này điều chúng ta quan tâm nhất là thời gian để chuyển trạng thái.
Trong suốt quá trình lập mô hình, bạn sẽ phải quyết định nên chọn cái gì để đặt trong biểu đồ. Hãy tự hỏi mình "chi tiết này có quan trọng để cho biết là tôi đang lập hình gì hay không?" và "chi tiết có làm cho một vài thứ rõ ràng hơn không". Nếu cả 2 câu bạn đều trả lời là yes thì tốt nhất là bạn nên thêm chi tiết này vào biểu đồ, ngược lại hãy bỏ chúng đi. Điều này sẽ làm cho tình trạng lộn xộn của mô hình giảm tới mức tối thiểu.



d. Các trạng thái

Trong suốt 1 tương tác, 1 participant có thể tồn tại ở nhiều trạng thái. Participant ở một trạng thái cụ thể khi nó nhận 1 sự kiện (chẳng hạn như 1 thông điệp). Nó sẽ ở trạng thái như vậy cho đến khi sự kiện khác diễn ra (chẳng hạn như trả về của thông điệp đó). Chúng ta sẽ nói kĩ vấn đề này ở phần sau.

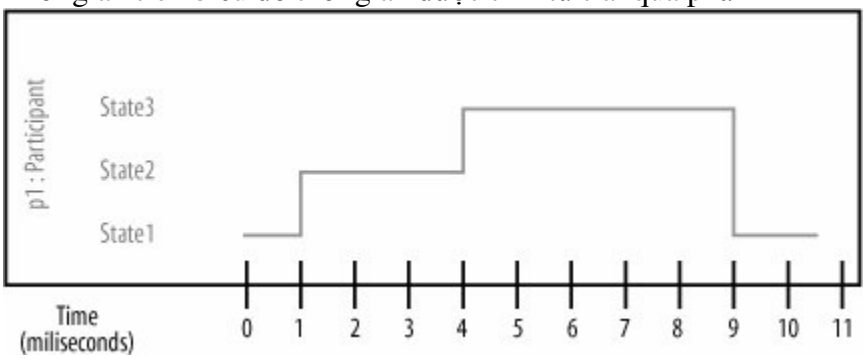
Các trạng thái được đặt kế participant tương ứng của chúng



Các trạng thái được viết theo chiều ngang cạnh participant tương ứng

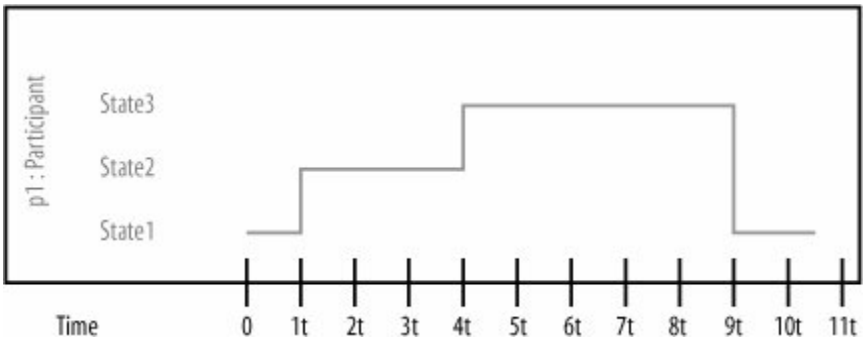
e. Thời gian

Bây giờ chúng ta sẽ thêm thời gian vào
Thời gian trên biểu đồ thời gian được tính từ trái qua phải

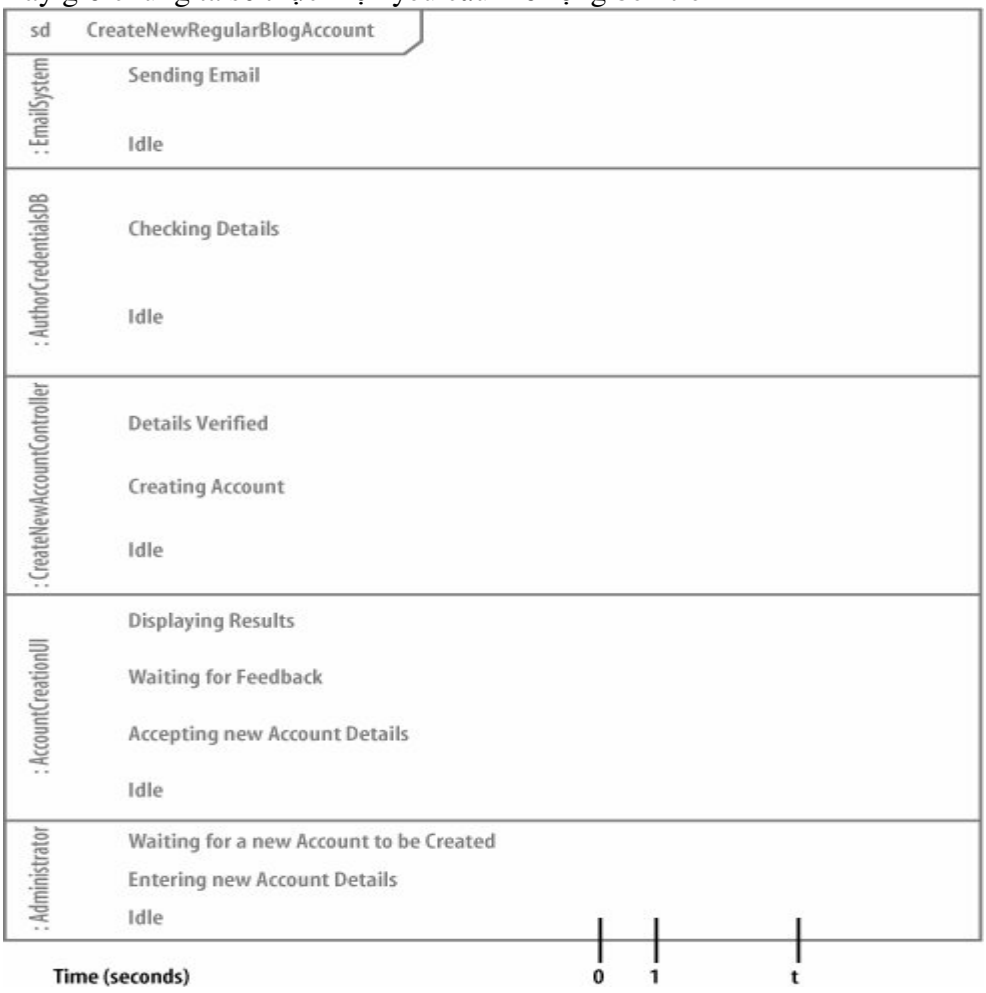


Khung thời gian này được đặt bên dưới biểu đồ xem như 1 thước đo.

Ngoài ra bạn có thể biểu diễn theo cách khác



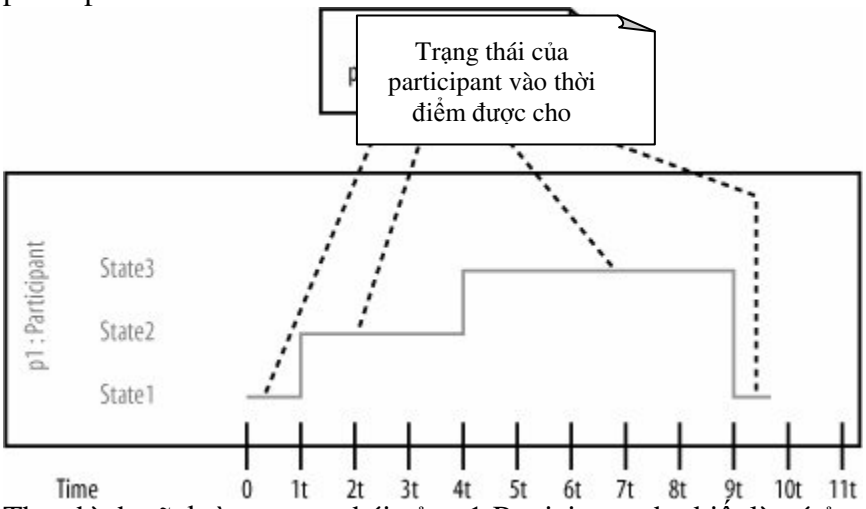
Trong đó t đại diện cho 1 khoảng thời gian nào đó. Bạn sẽ không biết chính xác là nó diễn ra khi nào.
Bây giờ chúng ta sẽ thực hiện yêu cầu mở rộng bên trên



Trong hình trên, thời gian được tính bằng giây. Chúng ta sẽ xem t được sử dụng trên biểu đồ thời gian như thế nào trong phần sau

f. Đường trạng thái của participant

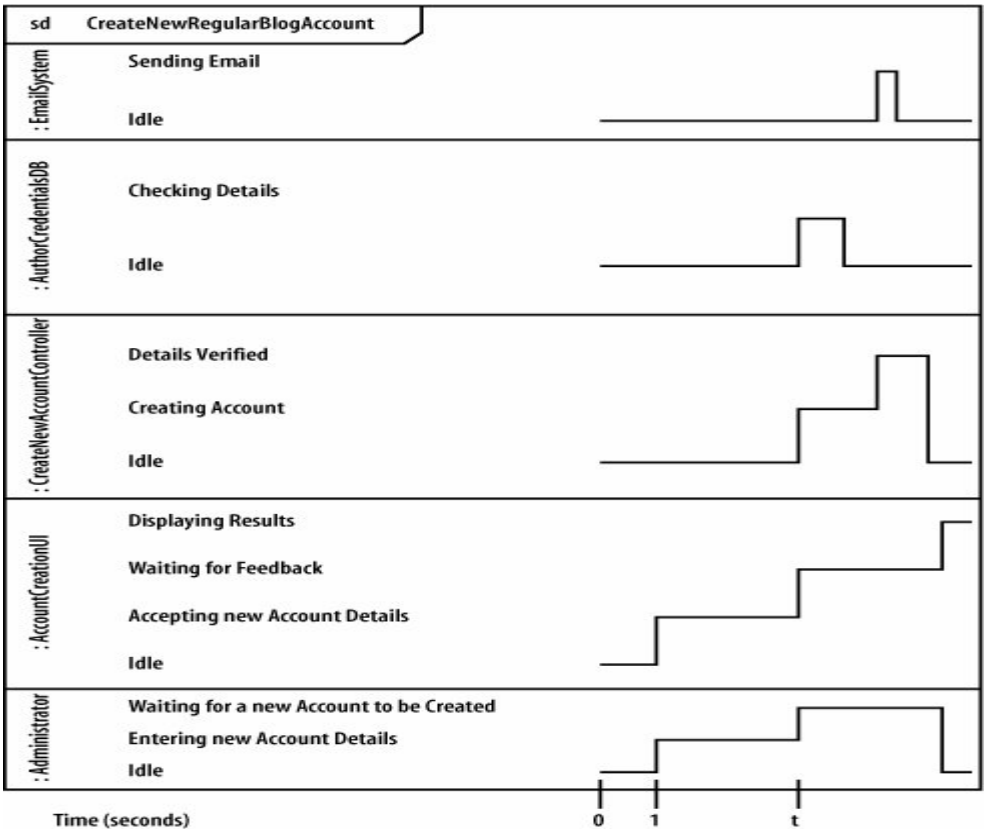
Bạn đã thêm thời gian vào biểu đồ thời gian, bây giờ bạn có thể cho biết trạng thái của một participant vào thời điểm đã cho.



Theo hình vẽ đường trạng thái của p1:Participant cho biết là nó ở trạng thái 1 trong 1 đơn vị thời gian, trạng thái 2 là 3 đơn vị thời gian, và trạng thái 3 là 5 đơn vị thời gian (trước khi trở về trạng thái 1 ở cuối của tương tác.

Trong thực tế, bạn có lẽ sẽ thêm cả trạng thái và sự kiện vào trong biểu đồ thời gian cùng 1 lúc. Chúng ta chỉ phân ra thành 2 để dễ quan sát hệ thống các kí hiệu

Chúng ta sẽ xem biểu đồ thời gian `Create a new Regular Blog Account` đã được update để chỉ ra trạng thái của các participant

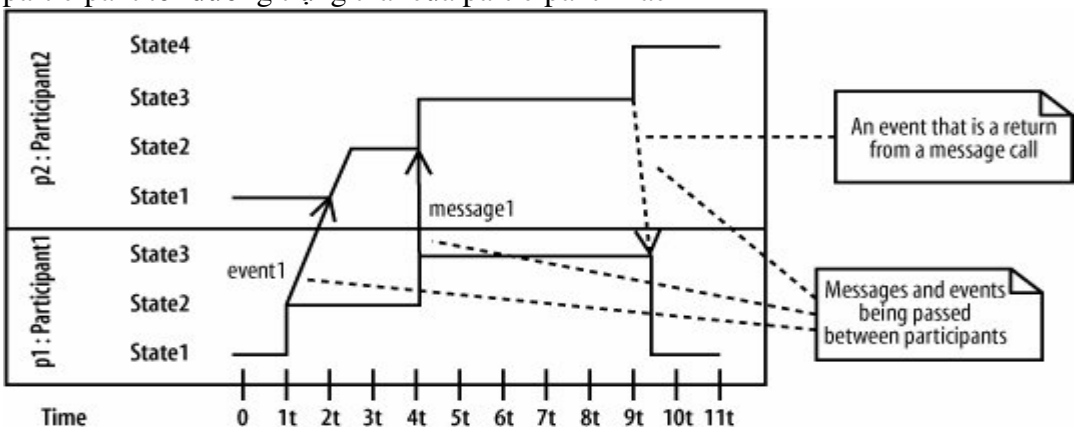


Mỗi particpant cần có 1 đường trạng thài tương ứng để cho biết trạng thái của nó vào thời điểm nào đó.

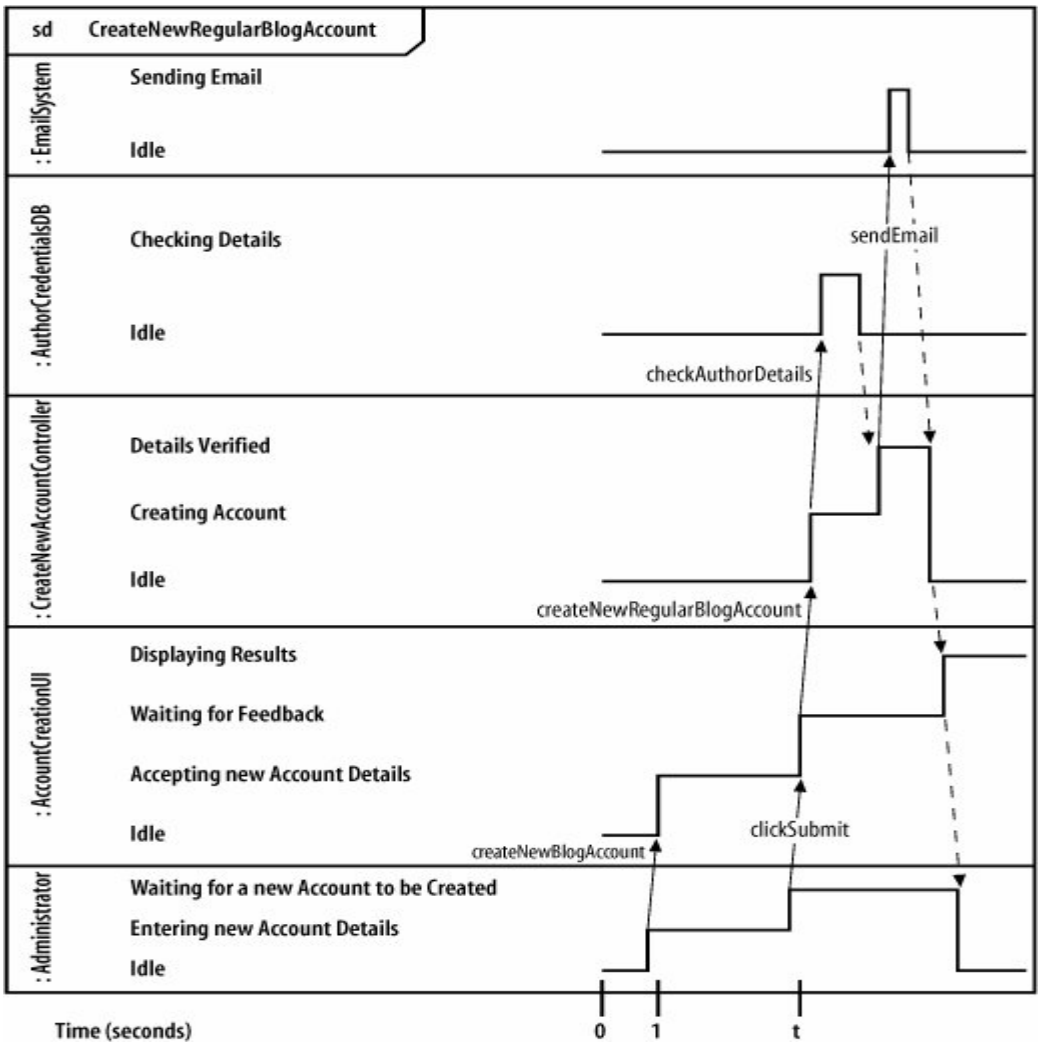
g. Sự kiện và thông điệp

Các participant thay đổi trạng thái để ứng với các sự kiện. những sự kiện này có thể là lời gọi 1 thông điệp, hoặc là làm 1 cái gì đó chặng hạn trả về thông điệp khi nó được gọi. Sự khác biệt giữa thông điệp và sự kiện thì không quan trọng như ở biểu đồ tuần tự. Điều quan trọng là nó được sử dụng để tahy đổi trạng thái của 1 participant

Sự kiện trong biểu đồ thời gian được trình bày bằng 1 dấu mũi tên từ đường trạng thái của 1 participant tới đường trạng thái của participant khác

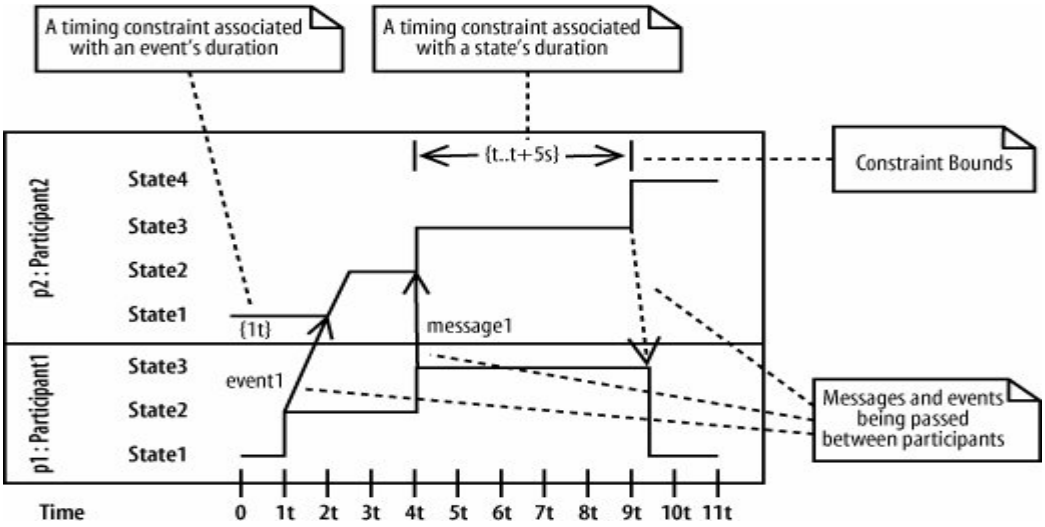


Các sự kiện trên biểu đồ thời gian có thể có khoảng thời gian riêng của chúng, như trình bày thì event1 lấy 1 đơn vị thời gian từ lời gọi bởi p1:Participant1 và nơi nhận là p2:Participant2
Thêm các sự kiện vào biểu đồ thời gian là 1 việc đơn giản bởi vì bạn đã có biểu đồ tuần tự để tham chiếu.



h. Ràng buộc thời gian

Cho đến lúc này thì bạn đã thiết lập được cái nền cho biểu đồ thời gian. Ràng buộc thời gian mô tả chi tiết 1 phần của tương tác được chỉ định thì mất bao lâu. Chúng gắn cho participant 1 lượng thời gian ở 1 trạng thái cụ thể



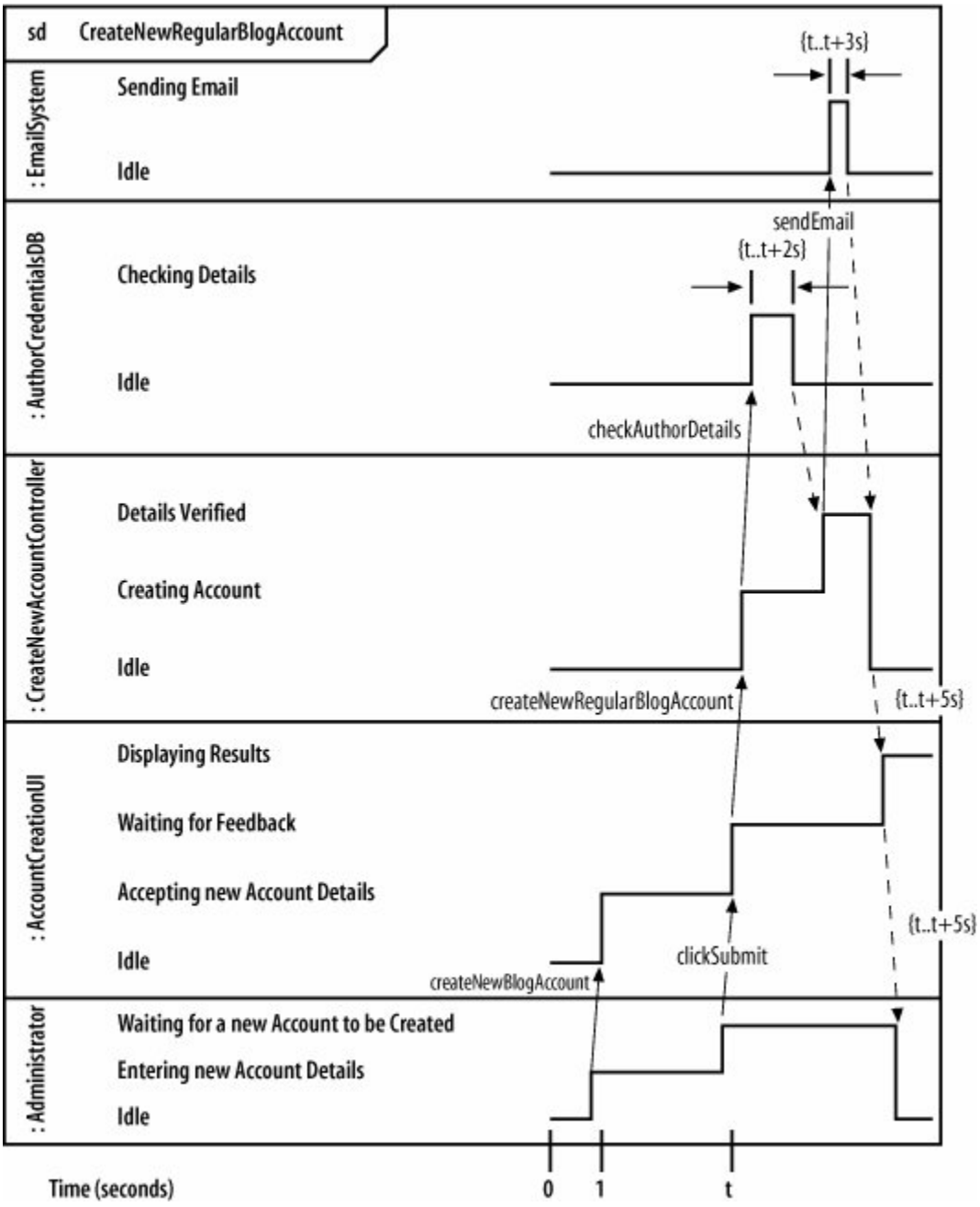
- **Dạng ràng buộc thời gian**

Ràng buộc thời gian có thể được chỉ ra tróng số ccáh khác nhau. Tùy vào thông tin bạn lập mô hình

Ràng buộc thời gian	Mô tả
`{t..t+5s}`	Sự kiện hoặc trạng thái tồn tại phải nhỏ hơn hay = 5 giây
`{<5s}`	Sự kiện hoặc trạng thái tồn tại phải nhỏ hơn 5 giây
`{>5s, <10s}`	Sự kiện hoặc trạng thái tồn tại phải lớn 5 giây nhưng nhỏ hơn 10 giây
`{t}`	Sự kiện hoặc trạng thái tồn tại phải =giá trị t. t có thể là bất cứ giá trị nào
`{t..t*5}`	Sự kiện hoặc trạng thái tồn tại phải nhỏ hơn hay bằng 5 lần t.

- Ứng dụng các ràng buộc vào trong các sự kiện hay thông điệp

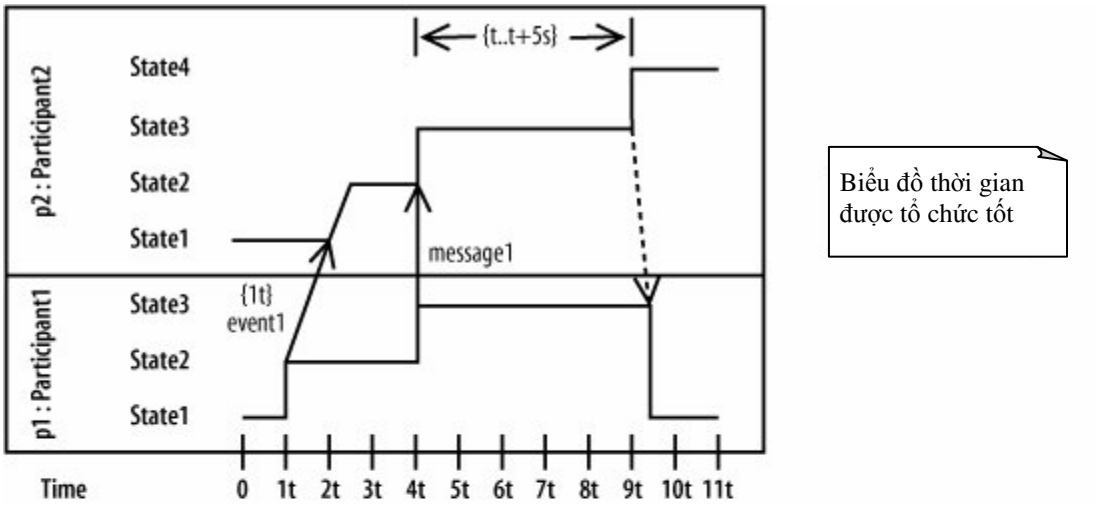
Chúng ta sẽ thực hiện yêu cầu ở đầu chương này

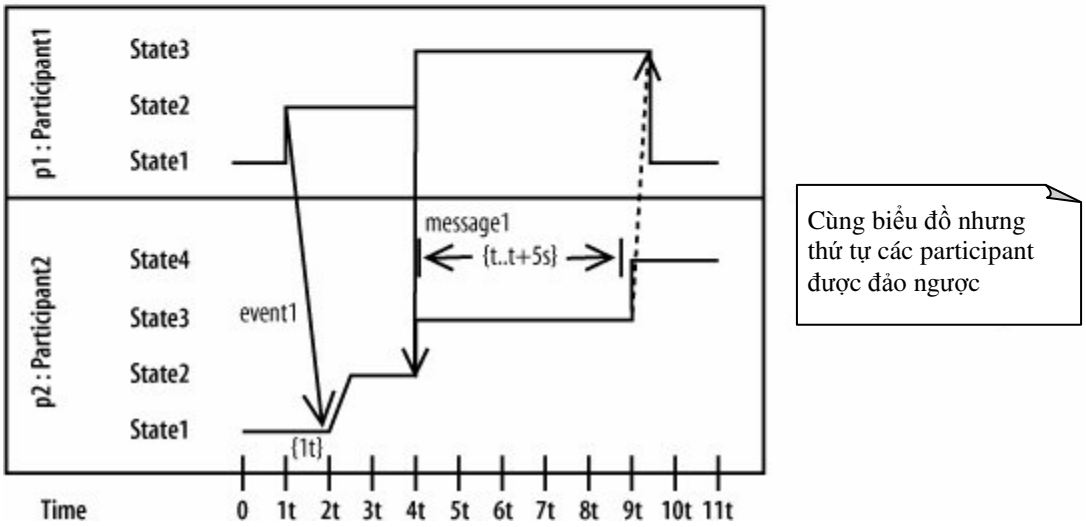


Chúng ta đã thực hiện ràng buộc sao cho việc tạo 1 blog account regular chỉ diễn ra trong vòng 5 giây. Điều này không phải là không phức tạp, bởi vì nó ảnh hưởng đến những tương tác chồng khác giữa các participant. Đây cần kĩ năng của người lập mô hình, bạn cần quyết định sự kiện hay phát biểu nào cần được chia phần của 5 giây để mỗi participant có thể thực hiện công việc của nó.

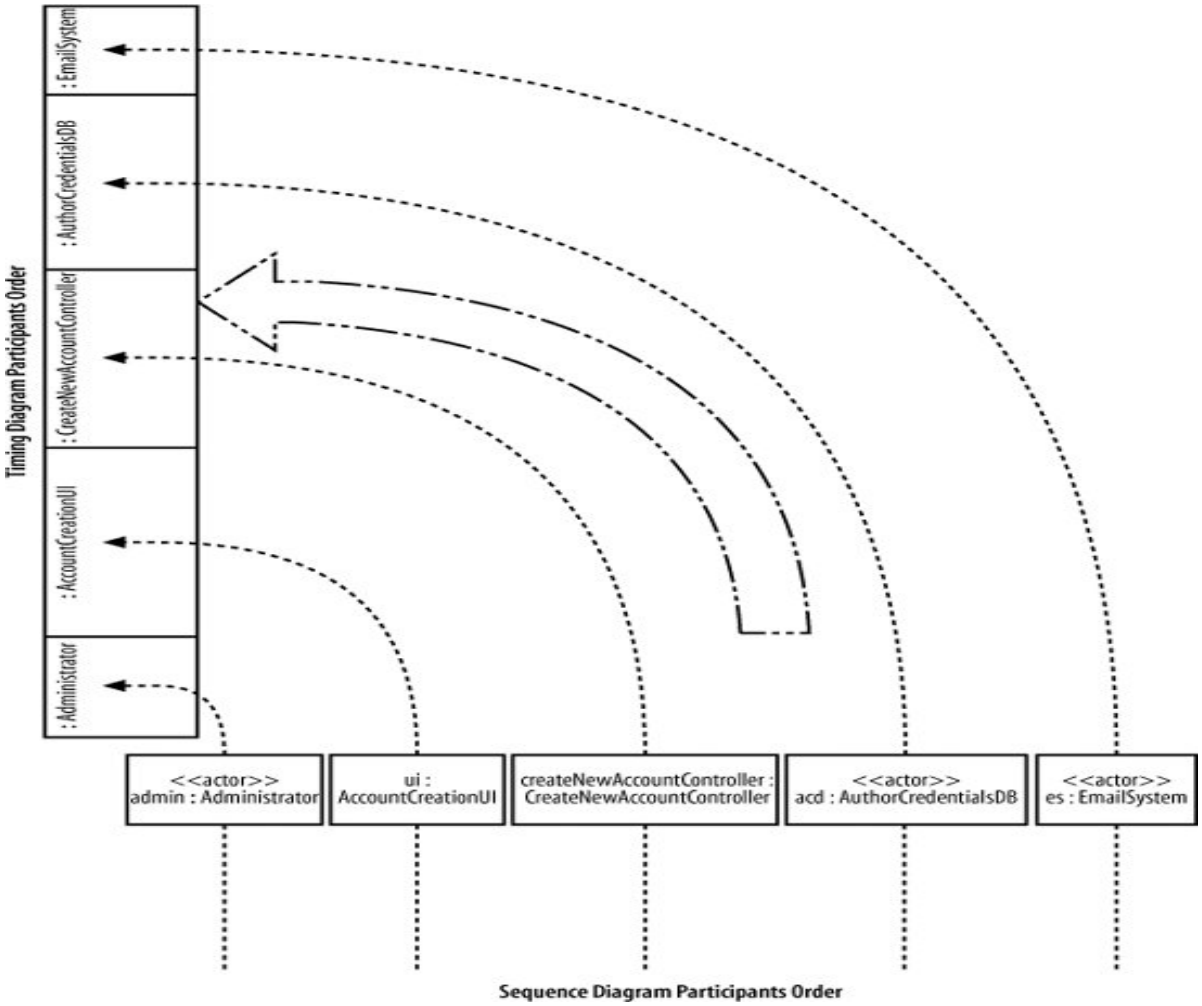
i. Việc tổ chức các participant trong biểu đồ thời gian

Nơi bạn đặt các participant trên biểu đồ thời gian là rất quan trọng, Tuy nhiên, khi bạn thêm thông tin chi tiết về các sự kiện và thời gian, bạn nên sớm nhận ra là nếu participant đứng chỗ đó có gây khó khăn hay không





Biểu đồ dưới thì rất khó đọc và các chi tiết thì trở nên mù mờ
Nếu bạn may mắn có 1 biểu đồ tuần tự cho 1 tương tác thì sẽ có cách rất dễ để sắp xếp các participant lần đầu trên biểu đồ thời gian. Chúng ta sẽ lấy thứ tự các participant ở trên đỉnh trang và quay nó ngược chiều kim đồng hồ 1 góc 90 độ. Nếu biểu đồ tuần tự của bạn được tổ chức tốt, bạn sẽ có 1 thứ tự tốt các participant trên biểu đồ thời gian

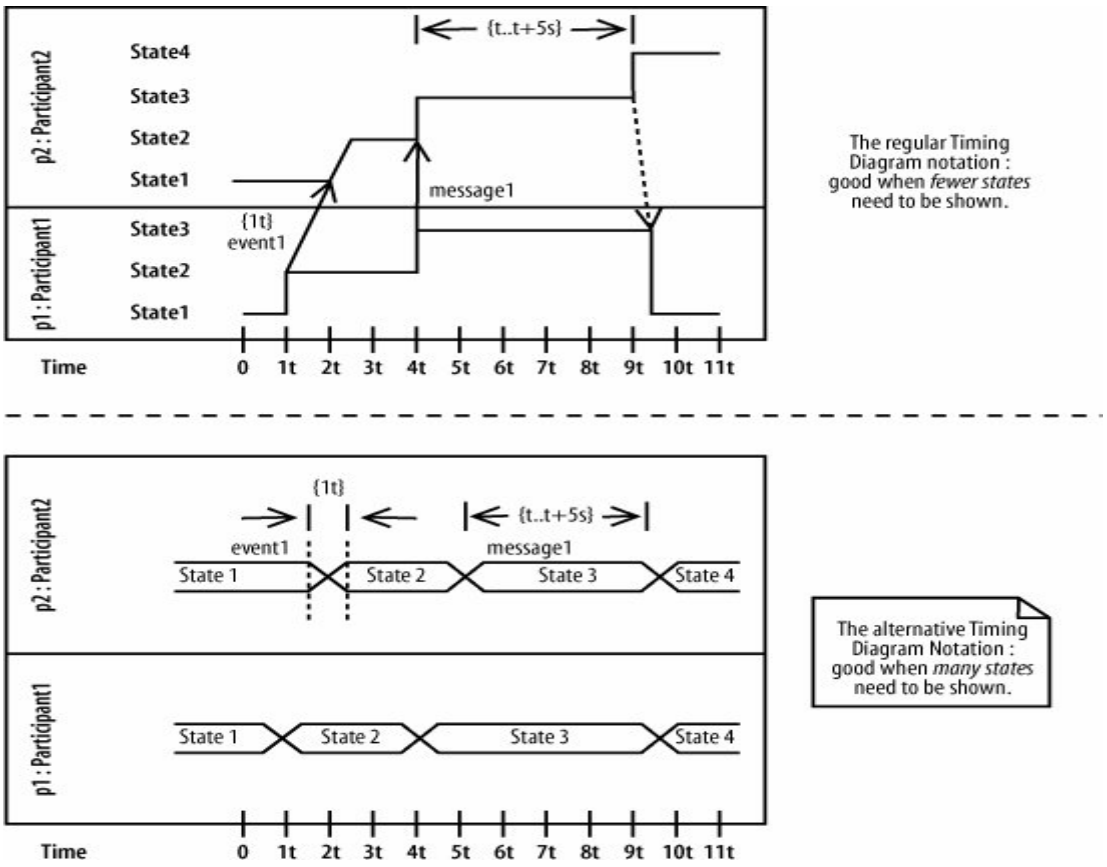


j. Hệ thống kí hiệu chuyển đổi

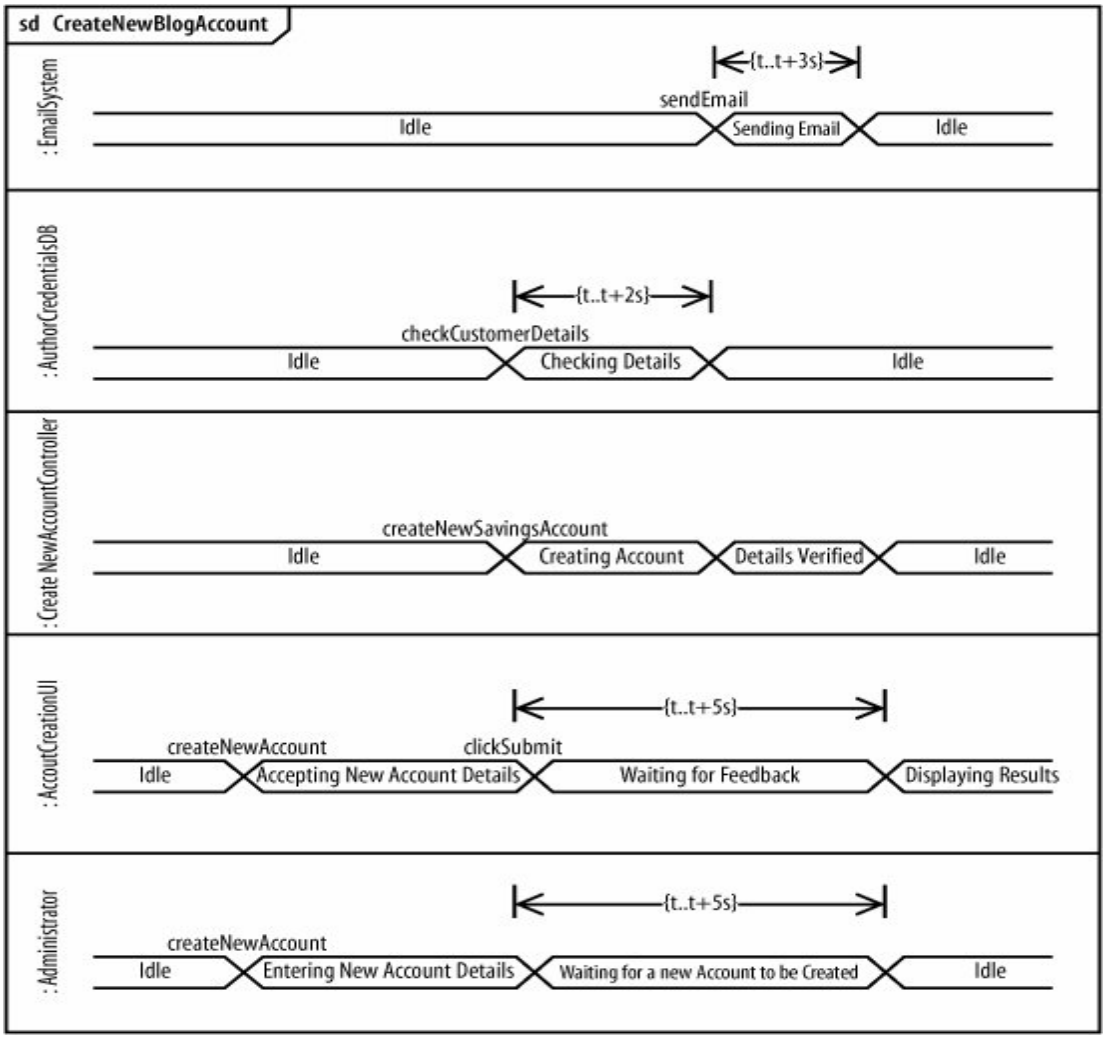
Biểu đồ thời gian của tương tác Create a new Regular Blog Account thật sự là ví dụ đơn giản. Bây giờ thì bạn có thể hình dung xem biểu đồ thời gian có thể lớn như thế nào nếu có 1 số lương lớn các trạng thái. 1 công cụ UML tốt sẽ giúp bạn quản lý và làm việc với biểu đồ thời gian lớn, nhưng cũng chỉ có công cụ có thể làm điều này.

Các nhà phát triển UML đã nhận thấy vấn đề, và họ đã tạo ra 1 hệ thống kí hiệu đơn giản hơn khi bạn có tương tác với số lượng các trạng thái lớn. Thật ra thì nó cũng không khác lắm với hệ kí hiệu cũ, kí hiệu thời gian và các participant thì không thay đổi. Chỉ có 1 thay đổi lớn đó là các trạng thái và sự chuyển trạng thái được trình bày như thế nào.

Với hệ kí hiệu biểu đồ trạng thái thông thường, thì khi bạn có 1 participant với rất nhiều trạng thái, khi đó không gian cần để có thể lập mô hình 1 participant trong biểu đồ thời gian sẽ rất lớn. Hệ kí hiệu đan xen nhau đã khắc phục được vấn đề này. Nó đặt các trạng thái của 1 participant ngay tại mốc thời gian khi participant ở trong trạng thái đó. Vì vậy đường trạng thái không còn cần nữa. Tất cả các trạng thái của 1 participant cụ thể được đặt trên 1 đường ngang qua biểu đồ. Để chỉ ra sự chuyển trạng thái của 1 participant do 1 sự kiện, 1 dấu chữ thập sẽ được đặt giữa 2 trạng thái và sự kiện gây ra thay đổi trạngthái sẽđược viết kế bên. Ràng buộc thời gian vẫn như cũ.



Ngay cả khi không có nhiều trạng thái trong tương tác này, nhưng bạn hãy thử xem xem là hệ kí hiệu xâu chuỗi cô động và dễ quản lý hơn như thế nào trong tình huống có nhiều trạng thái mỗi participant


sd CreateNewBlogAccount
: EmailSystem
{t..t+3s}
sendEmail
Idle
Sending Email
Idle
: AuthorCredentialsDB
{t..t+2s}
checkCustomerDetails
Idle
Checking Details
Idle
: Create NewAccountController
createNewSavingsAccount
Idle
Creating Account
Details Verified
Idle
: AccoutCreationUI
{t..t+5s}
createNewAccount
clickSubmit
Idle
Accepting New Account Details
Waiting for Feedback
Displaying Results
: Administrator
{t..t+5s}
createNewAccount
Idle
Entering New Account Details
Waiting for a new Account to be Created
Idle

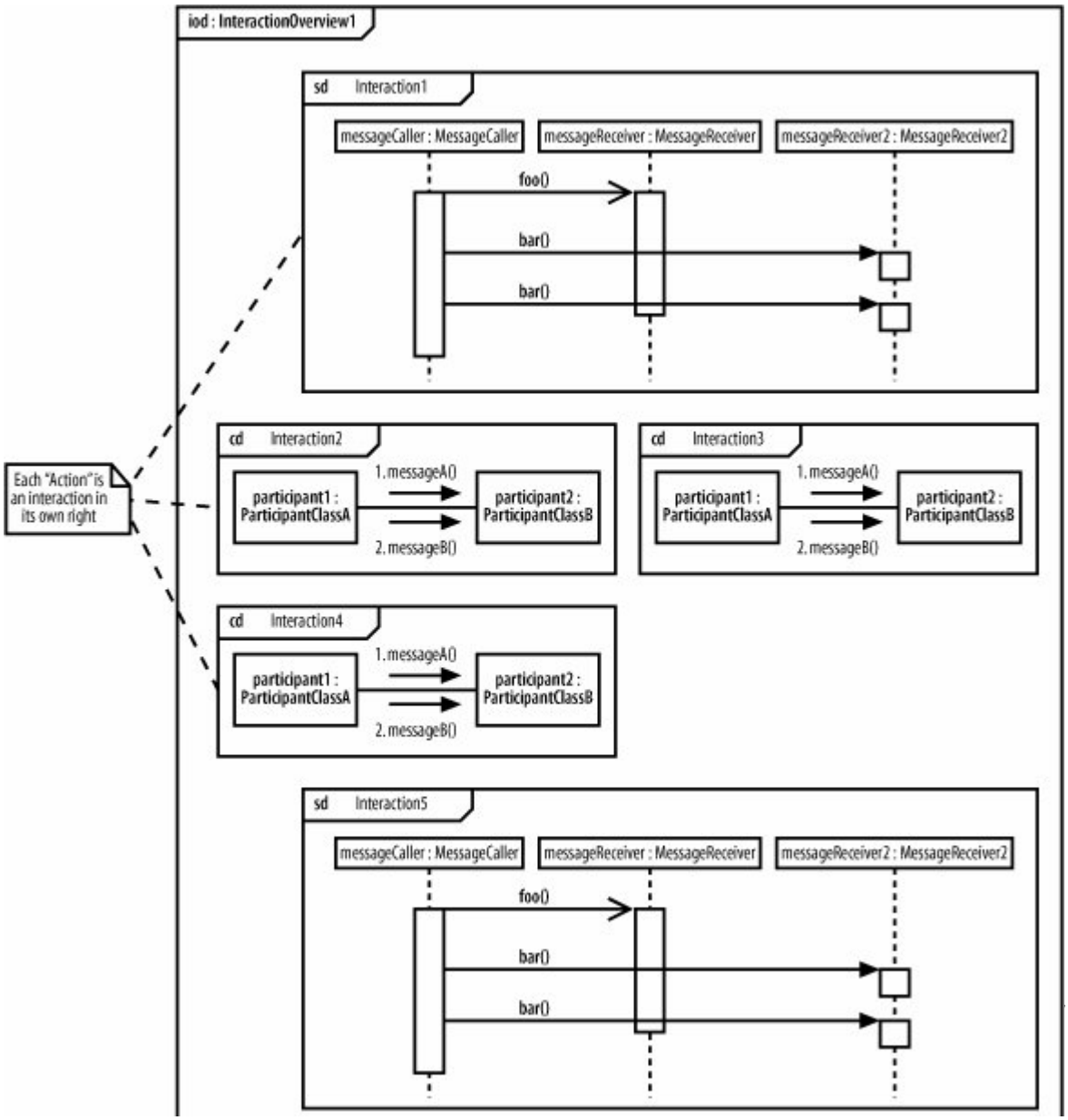
X. Hoàn thành bức tranh tương tác: Interaction Overview Diagrams

1. Giới thiệu:

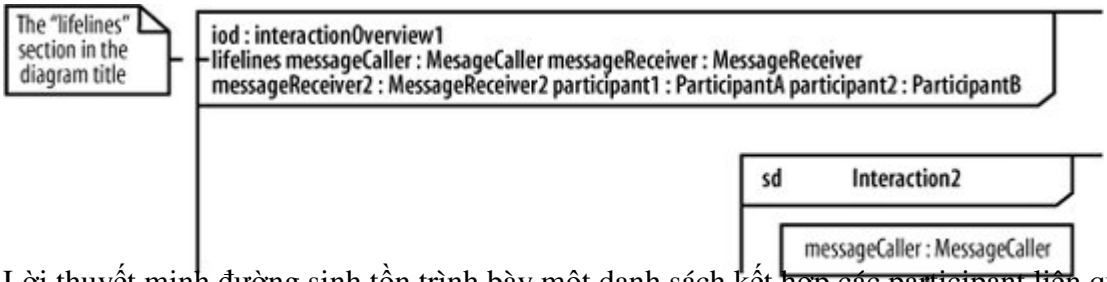
Biểu đồ tổng quát các tương tác cung cấp cho bạn 1 cái nhìn bao quát hơn về việc nhiều tương tác làm việc với nhau như thế nào để implement 1 hệ thống, chẳng hạn như use case. Nó ràng buộc các tương tác khác nhau thành 1 bức tranh hoàn thiện duy nhất của các tương tác cấu thành 1 hệ thống đặc biệt

2. các phần của biểu đồ tổng quát tương tác:

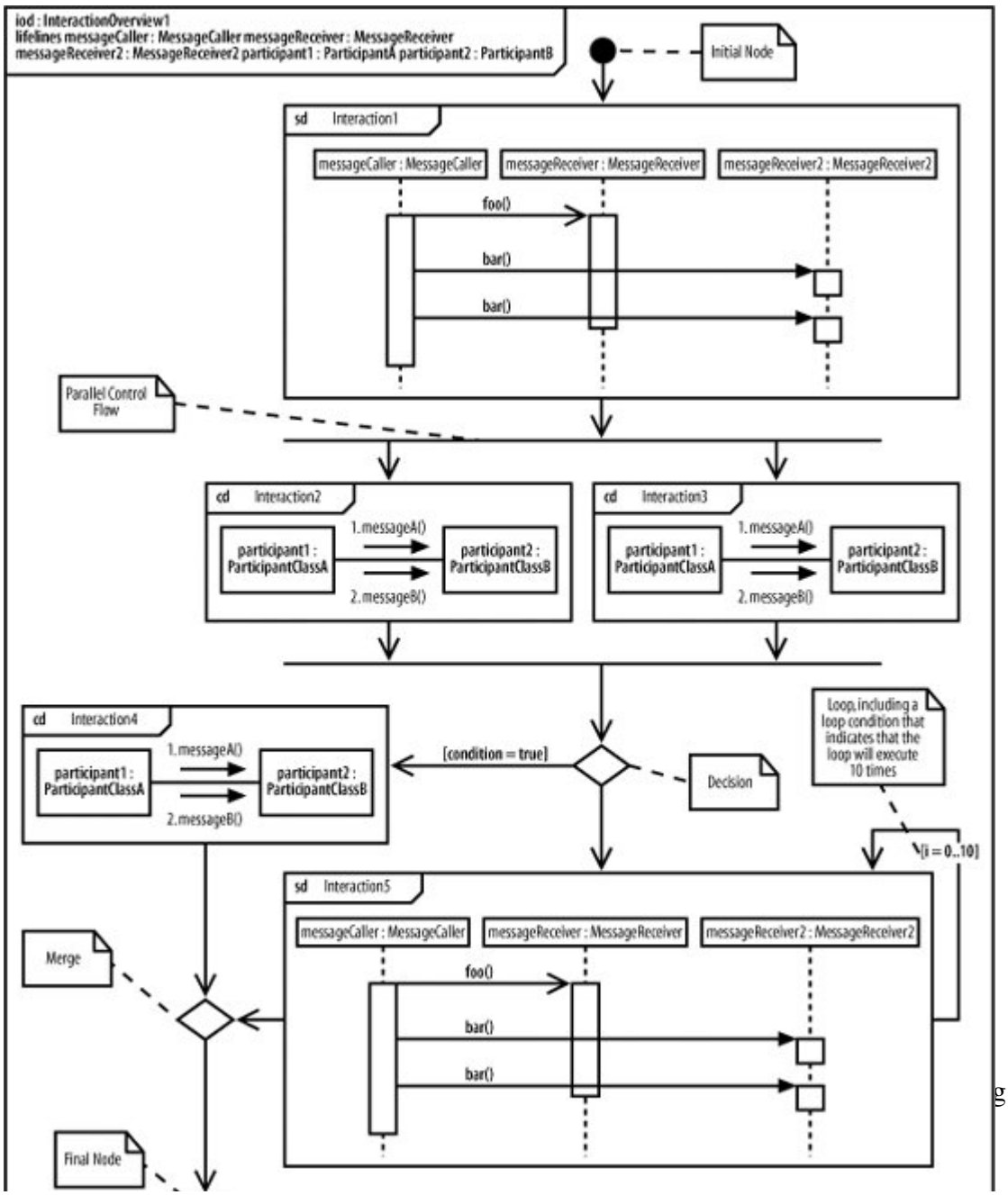
Cách tốt nhất để hiểu 1 hệ kí hiệu Biểu đồ tổng quát các tương tác là xem nó như 1 biểu đồ hoạt động ngoại trừ việc thay vì nó là 1 hành động. 1 tương tác đầy đủ được mô tả bằng cách sử dụng biểu đồ riêng của nó



1 số các participant liên quan trong các tương tác thì diễn ra trong overview. Để xem participant nào liên quan đến toàn bộ overview, 1 lời thuyết minh đường sinh tồn thì được thêm vào tiêu đề của biểu đồ



Lời thuyết minh đường sinh tồn trình bày một danh sách kết hợp các participant liên quan các tương tác trong overview. Tương tự như biểu đồ hoạt động, biểu đồ tổng quát các tương tác bắt đầu với node đầu tiên và kết thúc với node cuối cùng.



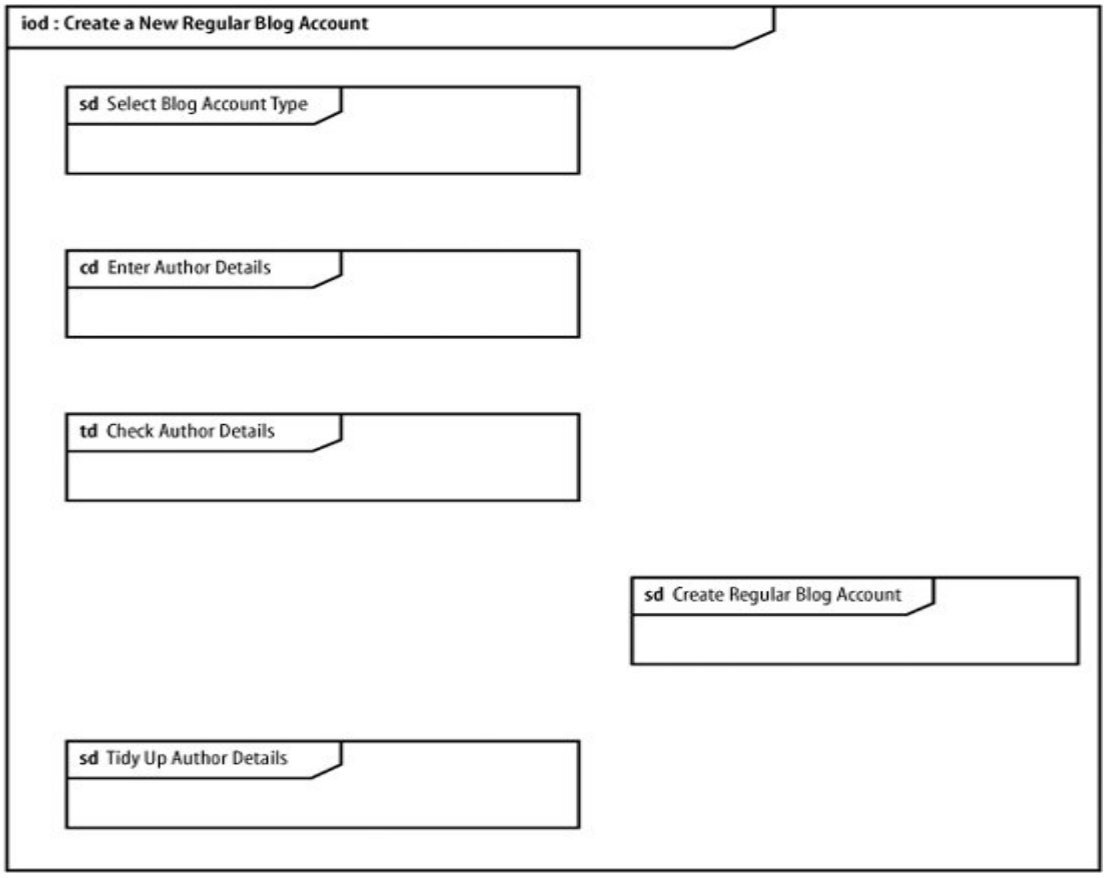
Bắt đầu với node khởi đầu, dòng điều khiển thực thi tương tác 1, theo sau là tương tác 2, 3 được thực hiện đồng thời; tương tác 4 chỉ thực hiện khi điều kiện đúng; mặt khác tương tác 5 được thực hiện 10 lần trước khi dòng điều khiển nhập lại và đi đến node kết thúc.

3. Mô hình hóa 1 use case sử dụng Interaction Overview:

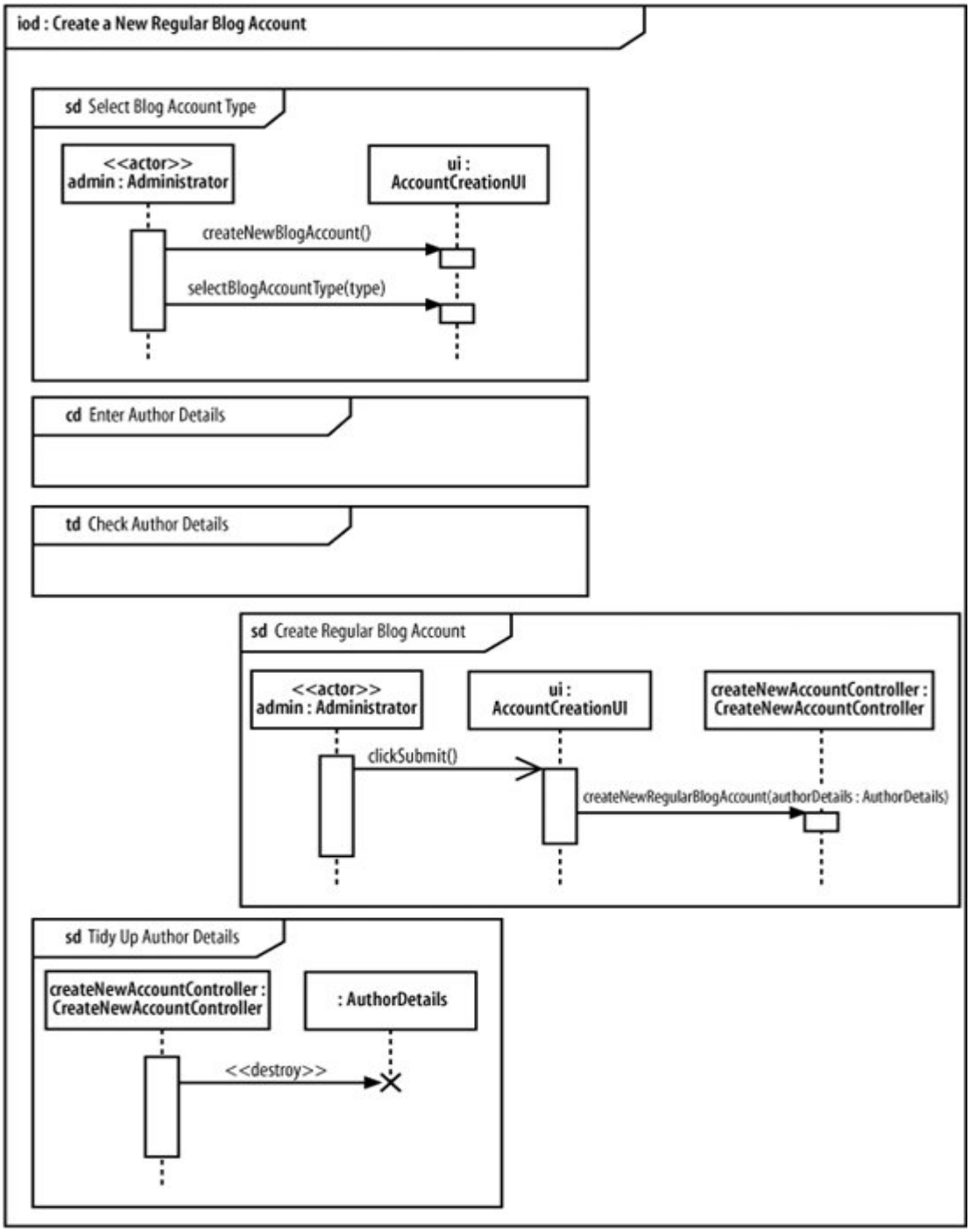
Chúng ta sẽ phát triển một biểu đồ tổng quát các tương tác cho use case Create a New Regular Blog sử dụng lại các phần từ các biểu đồ tương tác đã được tạo trong các chương trước
Khác biệt lớn giữa ví dụ trong chương này và các mô hình trong chương trước, với 1 interaction overview là chúng ta có thể lọc và chọn từ các loại interaction overview khác nhau.

a. Kéo các tương tác lại với nhau:

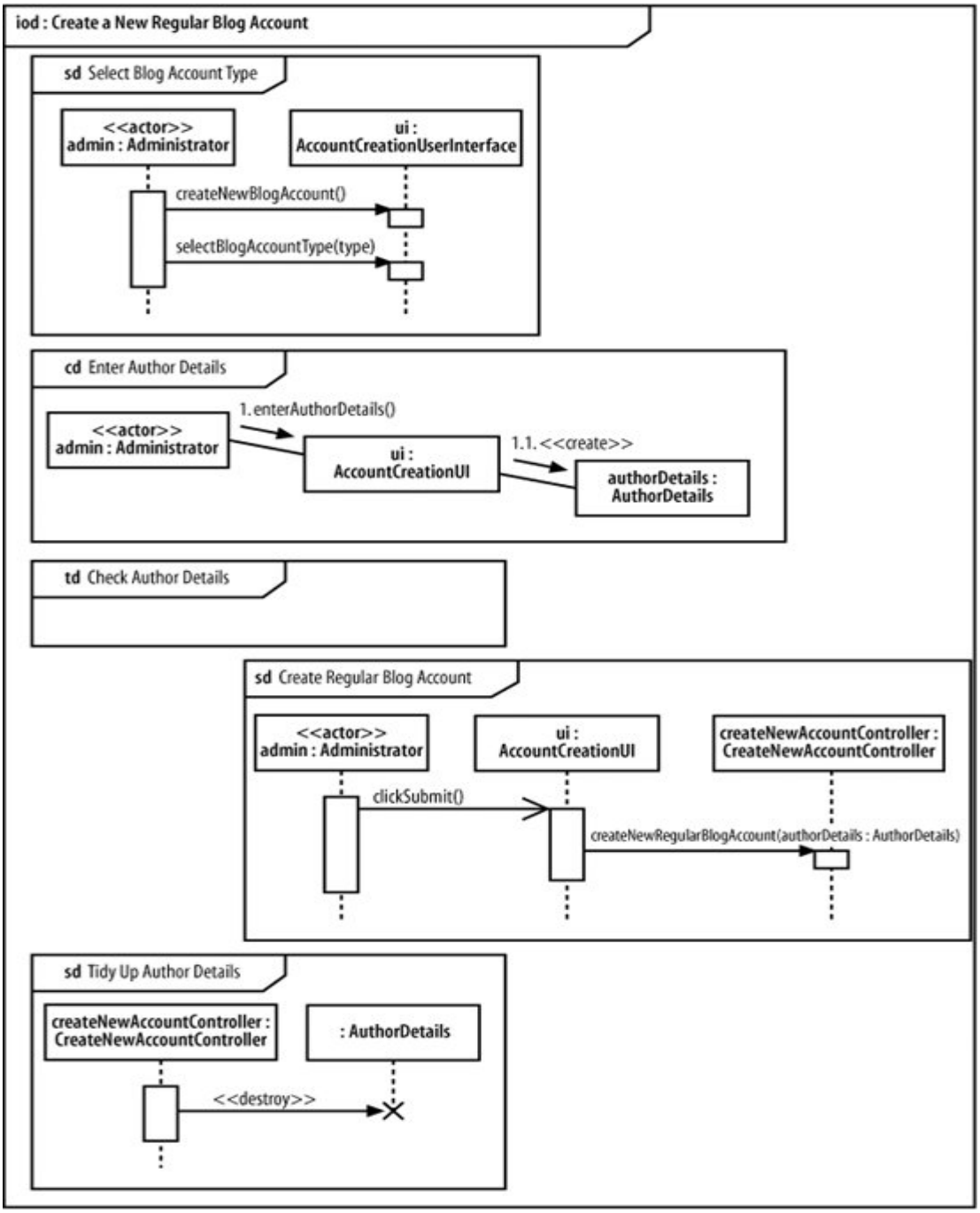
Đầu tiên chúng ta sẽ quyết định interaction overview sẽ được phân ra như thế nào để thành biểu đồ hiệu quả nhất cho mỗi tương tác riêng biệt.



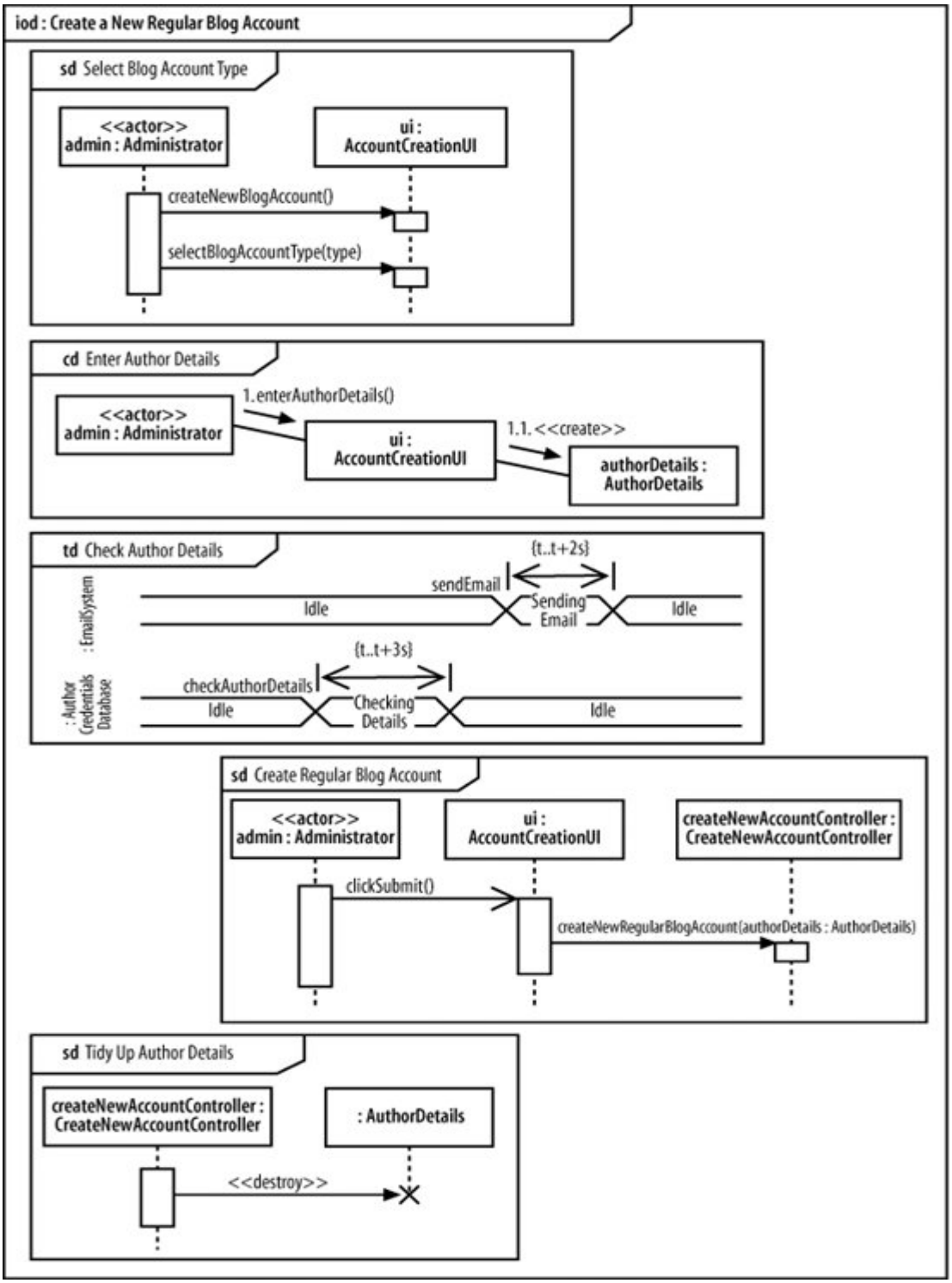
Tất cả các loại tương tác thì được sử dụng trong overview này, sd chỉ biểu đồ tuần tự, cd là biểu đồ truyền tin, td là biểu đồ thời gian
Khi lập mô hình các tương tác Select Blog Account Type, Create Regular Blog Account, và Tidy Up Author Details thì thứ tự các thông điệp là quan trọng hơn hết.



Vì mục đích khác nhau nên Enter Author Details sẽ được trình bày bằng biểu đồ truyền tin

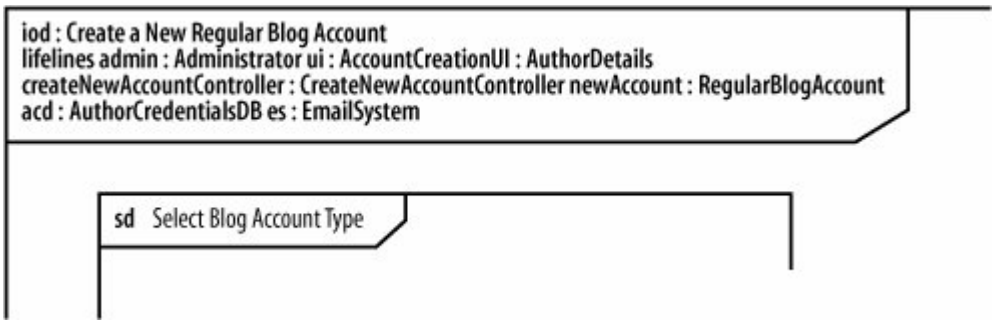


Chúng ta chọn tương tác Enter Author Details được biểu diễn bằng biểu đồ truyền tin đơn giản là vì nó dễ hiểu hơn nhưng có quá nhiều điểm tương tự giữa biểu đồ truyền tin và biểu đồ tuần tự đến nỗi mà nó kết hợp 2 cái thành 1 interaction overview mà thường chúng ta không thấy. Tương tác Check Author Details có ràng buộc thời gian thực hiện trong 5 giây do đó ta sẽ sử dụng biểu đồ thời gian.



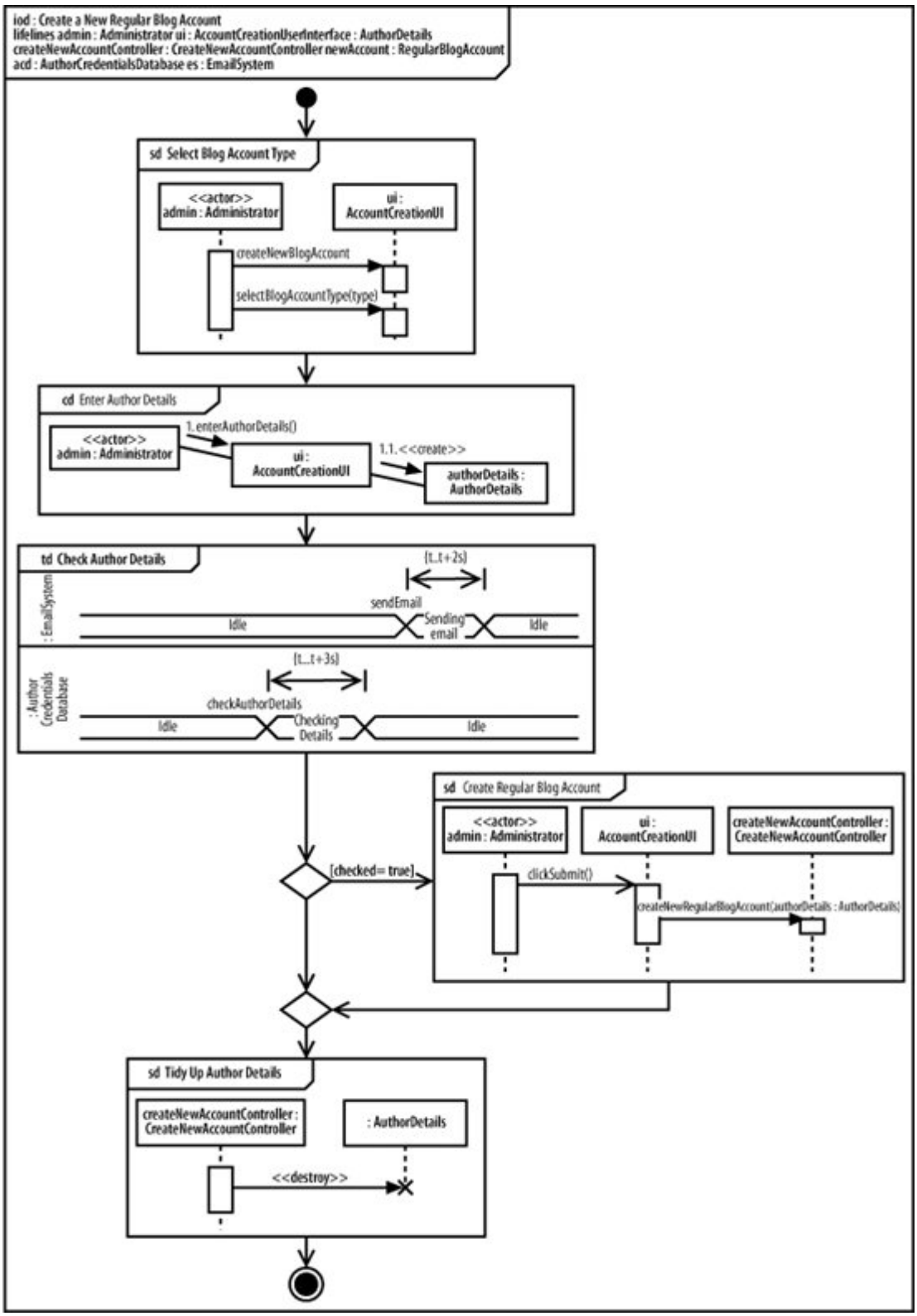
Có đề nghị là chúng ta sẽ sử dụng hệ kí hiệu của biểu đồ thời gian xâu chuỗi. Bởi vì interaction overviews có thể khá lớn khi đó sử dụng kí hiệu xâu chuỗi là tốt nhất.

Đó là tất cả các tương tác được add vào interaction overview, chúng ta đã biết tất cả các participant có liên quan vì vậy chúng ta có thể add tên của chúng vào tiêu đề của biểu đồ



b. Gắn các tương tác lại với nhau

Phần cuối cùng trong interaction overview Create a New Regular Blog Account là dòng thật sự của các control giữa các biểu đồ tương tác riêng biệt



Dòng điều khiển sẽ được thực hiện theo thứ tự. Sự khác biệt là ở tương tác Create a New Regular Blog Account nó chỉ thực hiện nếu nếu chi tiết tác giả đã được kiểm duyệt trong tương tác Check Author Details.

XI. Mô hình hóa cấu trúc bên trong của các lớp : Composite Structures

1. Giới thiệu:

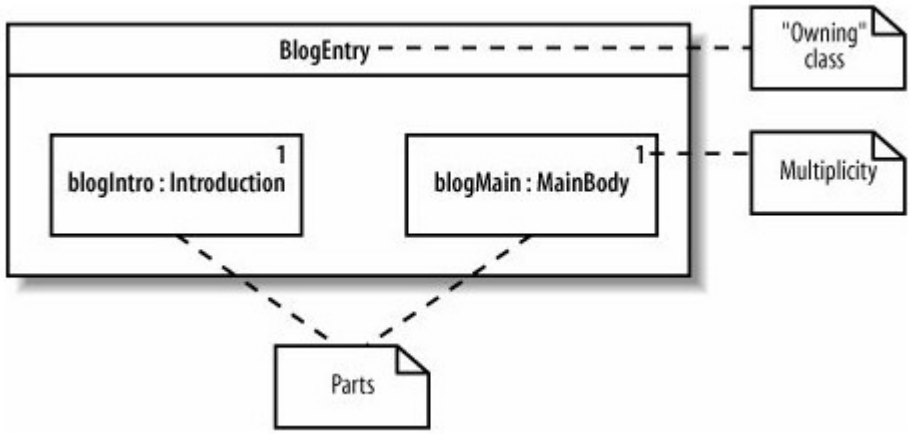
Đôi khi các lược đồ (diagram) chính của UML như class và sequence không thể mô tả chính xác các chi tiết trong hệ thống. Khi đó chúng ta có thể sử dụng các cấu trúc hỗn hợp để làm việc đó, các câu trúc này có thể mô tả cách các đối tượng tương tác với nhau **bên trong một lớp** và cách các đối tượng tiến tới hoàn thành nhiệm vụ của nó. Các cấu trúc hỗn hợp là một khái niệm khá mới tuy nhiên nó rất thích hợp để mô tả các trạng thái chuyên biệt của mô hình bao gồm:

- Internal structure:
  Mô tả các thành phần trong một lớp và mối quan hệ giữa chúng với nhau, cho phép bạn mô tả các mối quan hệ tùy thuộc vào ngữ cảnh.
- Ports:
  Mô tả cách các lớp được sử dụng trong hệ thống với các port.
- Collaborations:
  Mô tả các thiết kế mẫu (pattern) trong phần mềm hay rộng hơn là sự cộng tác giữa các đối tượng để hoàn thành một nhiệm vụ.

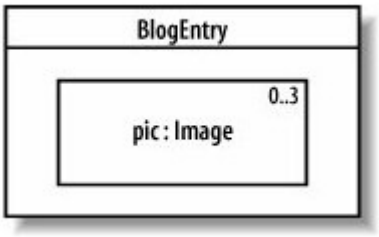
2. Các khái niệm:

a. Internal structure:

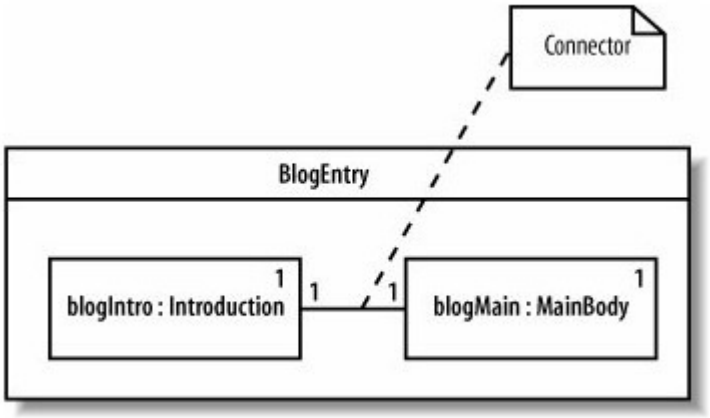
Internal structure cho phép bạn xác định chính xác quan hệ giữa các thành phần bên trong của lớp mang chúng.
Khi thể hiện internal structure của một lớp bạn vẽ các thành phần của nó bên trong. Các thành phần được xác định bởi vai trò của chúng bên trong 1 lớp, được biểu diễn dưới dạng *<roleName> : <type>* trong đó <rolename> là tên của thành phần, <type> là loại của nó, bên góc trên của mỗi phần có 1 *multiplicit* hay hệ số là số thực thể của thành phần này. Xét ví dụ sau:



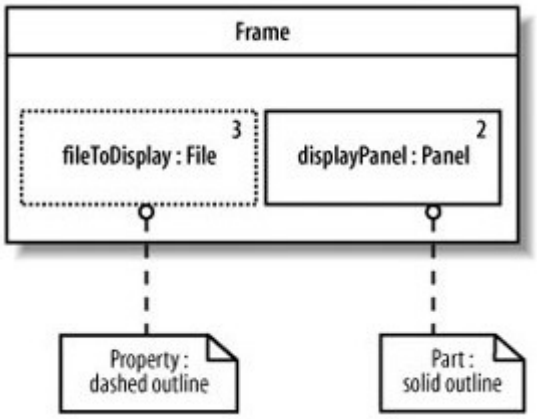
Ta thấy lớp *BlogEntry* có 2 thành phần là *blogInfo* và *blogMain*, *blogInfo* là một thành phần có kiểu là *Introduction* và số thực thể là 1, còn *blogMain* có kiểu là *MainBody* và số thực thể là 1. Số thực thể có thể là 1 khoảng giới hạn ví dụ ta muốn biểu diễn 1 *BlogEntry* chỉ có từ 1 đến 3 hình ảnh bên trong ta vẽ như sau:



Để biểu diễn mối quan hệ giữa các thành phần ta dùng các đường nối (connector) với các hệ số ở 2 đầu:



Nếu thành phần của lớp là một thuộc tính ta biểu diễn bằng đường nét đứt như sau:

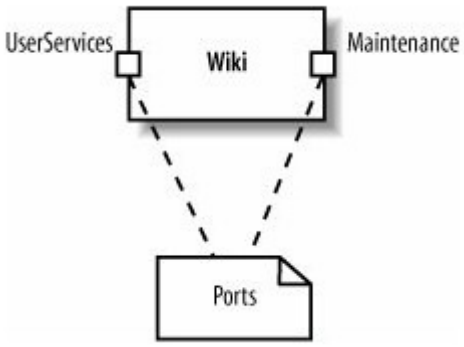


b. Ports:

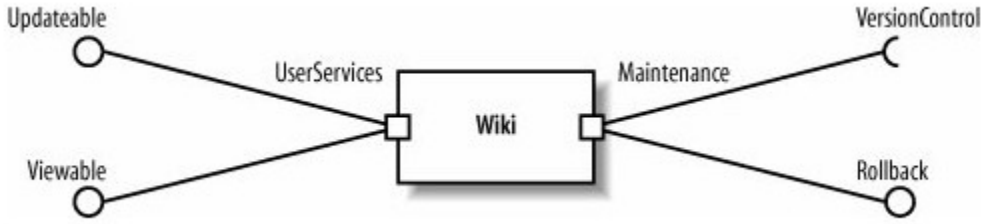
Một port là một điểm tương tác giữa lớp với bên ngoài, nó diễn tả các cách sử dụng lớp khác nhau. Ví dụ lớp Wiki được sử dụng thông qua 2 con đường:

- Cho phép người dùng xem và sửa
- Cung cấp các công cụ để admin bảo trì và thay đổi nội dung

Mỗi cách sử dụng lớp này đại diện bằng 1 port được biểu diễn bằng một hình chữ nhật nhỏ bên cạnh lớp:

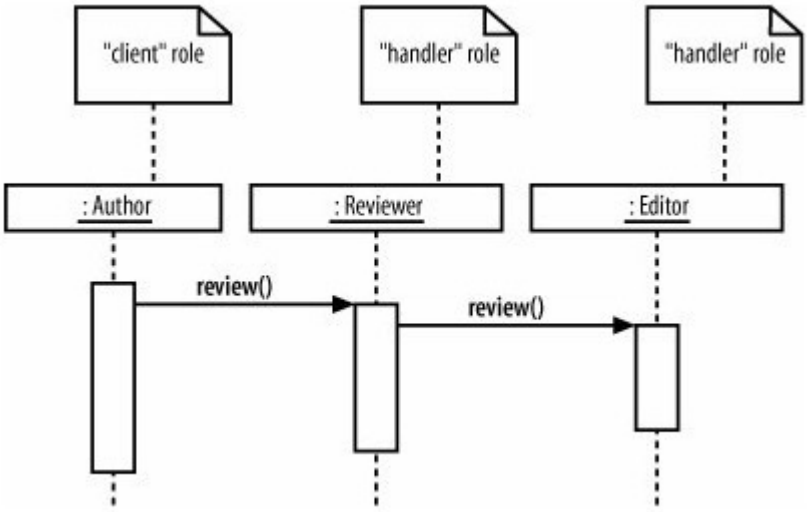


Thông thường các lớp sẽ có các interface liên kết với các port, bạn có thể nhóm các interface có liên quan lại với nhau để biểu thị các dịch vụ được cung cấp ở 1 port.
Ví dụ lớp wiki có implement 2 interface là *Updatable* và *Viewable* cho phép các lớp khác cập nhật và xem wiki thông qua các interface này. Các interface này gắn với *User Service* port.
Thêm vào đó lớp wiki còn có interface *Rollback* cho phép admin trả lại giá trị ban đầu cho wiki (nội dung trước khi sửa), và nó còn yêu cầu 1 interface là *Version Control* để kiểm soát phiên bản, 2 interface này gắn với port Maintenance. Ta biểu diễn các interface gắn với các port như sau:

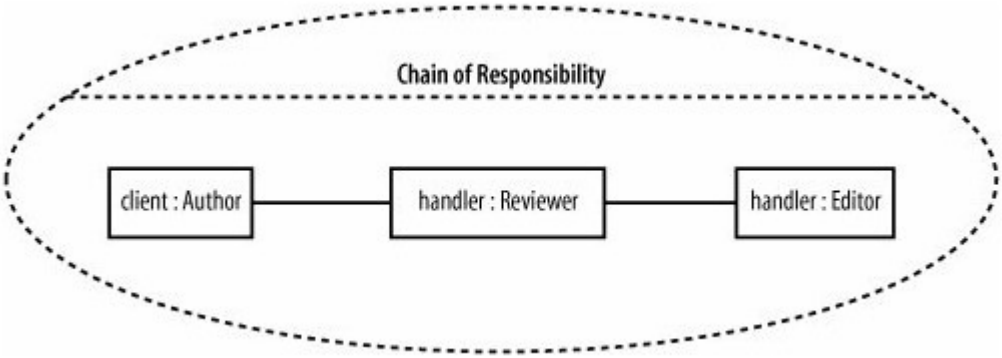


c. Collaborations:

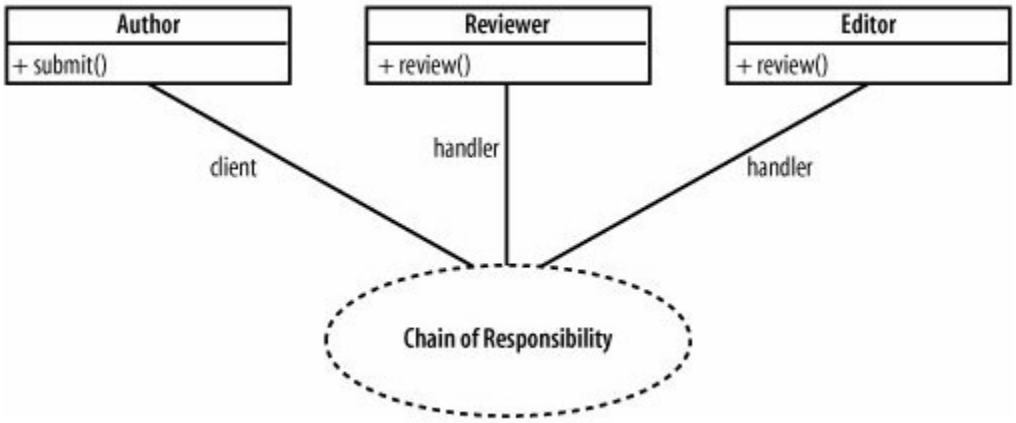
Collaborations thể hiện các đối tượng cùng làm việc chung với nhau một cách tạm thời để hoàn thành một công việc. Nó nghe giống như *object diagrams* nhưng collaborations có một khác biệt: nó tập trung mô tả đối tượng dựa vào vai trò của chúng trong quá trình và cung cấp các mô tả ở một mức cao hơn về cái mà đối tượng đang thực thi.
Ví dụ trong một hệ thống CMS ta có một quá trình xử lý các bài viết như sau: người viết (author) viết một bài và submit nó, người xét duyệt (reviewer) sẽ xem xét và có thể từ chối hoặc chuyển nó lên cho người biên tập (editor), người biên tập có thể từ chối hay chấp nhận nó. Điều này hình thành một mẫu chuỗi trách nhiệm **Chain of Responsibility (COR)**. Mẫu thiết kế COR này cho phép một đối tượng gửi đi các yêu cầu (request) mà không cần quan tâm về đối tượng nào sẽ xử lý các yêu cầu này sau cùng. Trong ví dụ trên người viết đóng vai trò client còn người xét duyệt và người biên tập sẽ đóng vai trò handler. Ta biểu diễn chuỗi này bằng **sequence diagram** như sau:



Còn trong **Collaborations** có 2 cách biểu diễn. Cách thứ nhất là dùng 1 hình oval nét đứt với các thành phần bên trong. Tên các thành phần được viết dưới dạng <*role*> *:* <*type*>, trong đó role là vai trò của thành phần (client hay handler), type là loại của nó. Các thành phần có các đường nối để biểu thị cách chúng giao tiếp với nhau:



Cách thứ 2 là cũng vẽ một hình oval nét đứt nhưng vẽ các thành phần bên ngoài và các đường nối chúng với hình oval có ghi tên vai trò bên cạnh:



Collaborations trông có vẻ không hữu dụng lắm nhưng thực chất nó rất mạnh trong khả năng thể hiện các mẫu (pattern) mà các sơ đồ khác không thể làm.

XII. Quản lý và tái sử dụng các thành phần: Component Diagrams

1. Giới thiệu:

UML component diagram mô tả các thành phần và cách tổ chức các thành phần, các module trong hệ thống.

2. Các khái niệm:

a. Component:

Một thành phần (component) là một phần của hệ thống mà ta có thể tái sử dụng, có thể đóng gói và thay thế dễ dàng. Một thành phần tốt là một thành phần thực hiện một chức năng nào đó của hệ thống mà được sử dụng lại nhiều lần, trong UML một thành phần cũng thực hiện các chức năng gần giống như một lớp: phổ biến và kết hợp với các thành phần khác, có interface và các toán tử…
Một component được biểu diễn bằng một hình chữ nhật với ký hiệu <<component>> và một biểu tượng bên góc trên bên phải như hình:

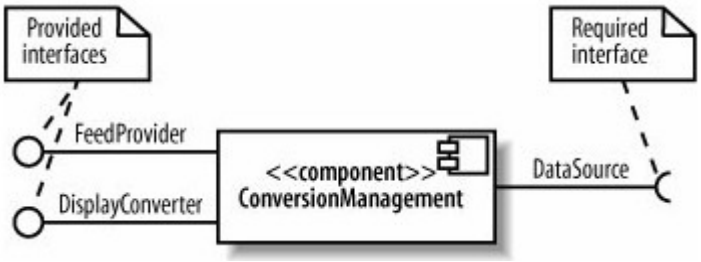
Bạn cũng có thể thay ký hiệu <<component>> bằng ký hiệu <<subsystem>> như hình sau (UML coi subsystem là một component đặc biệt) :

b. Provided và Required Interfaces:

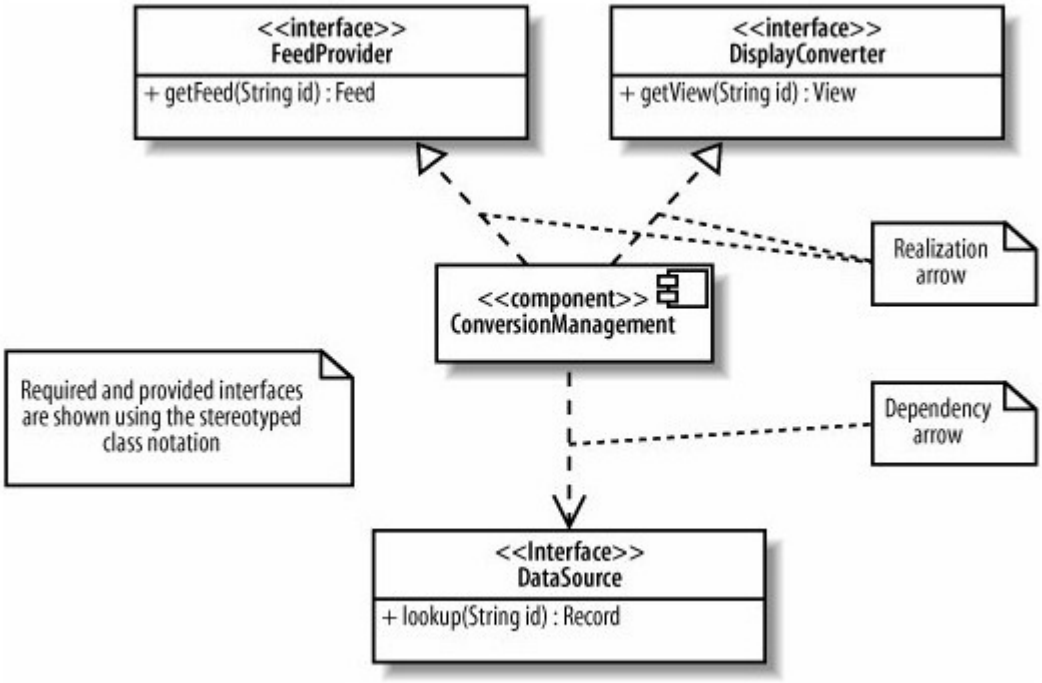
Các component tương tác với nhau thông qua *provided* và *required interfaces* để kiểm soát sự phụ thuộc giữa các component và làm cho chúng có thể trao đổi với nhau.

- Một *provided interface* của component là một giao diện của một component để các component khác có thể nhận ra nó, *provided interface* cung cấp thông tin về các dịch vụ mà một component cung cấp.
- Một *required interfaces* là một giao diện mà một component cần để thực hiện chức năng. Nói cách khác nó liệt kê các dịch vụ mà 1 component cần đến.

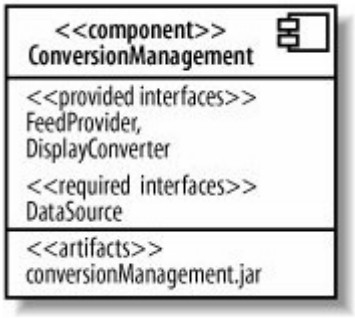
Biểu diễn: một *provided interface* được ký hiệu bằng 1 vòng tròn và một *required interfaces* được ký hiệu bằng một nửa vòng tròn:



Hoặc bạn cũng có thể biểu diễn *provided* và *required interfaces* bằng các mũi tên relize chỉ đến *provided interfaces* và dependency chỉ đến các *required interfaces.*



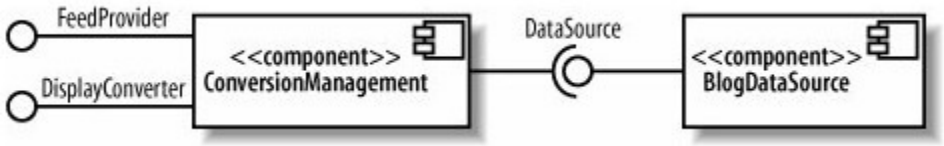
Nhưng cách gọn nhất là liệt kê chúng ra trong component như hình sau:



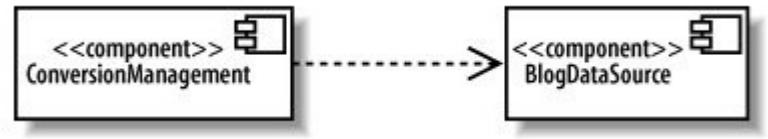
3. Biểu diễn hoạt động của các component:

Khi một component cần các required interface để hoạt động nó cần các component khác cung cấp ta sẽ biểu diễn bằng một mũi tên quan hệ như sau:

Hay như sau:

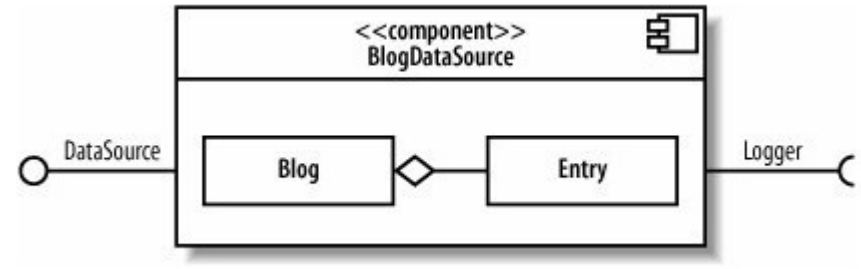


Hay đơn giản chỉ vẽ các mũi tên trực tiếp như sau:

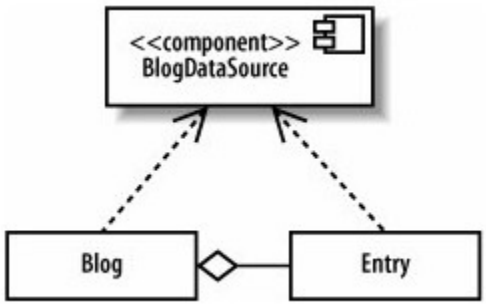


4. Class và component:

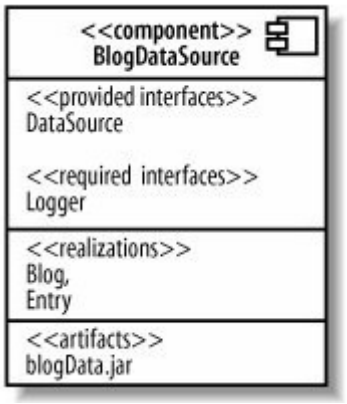
Một component thông thường chứa nhiều class bên trong để thực hiện các chức năng của nó, ta gọi đó là các class thực hiện (realize) component. Để biểu diễn các class này ta vẽ chúng bên trong component như hình vẽ:



Hoặc có thể biểu diễn chúng bằng các mũi tên quan hệ như sau:

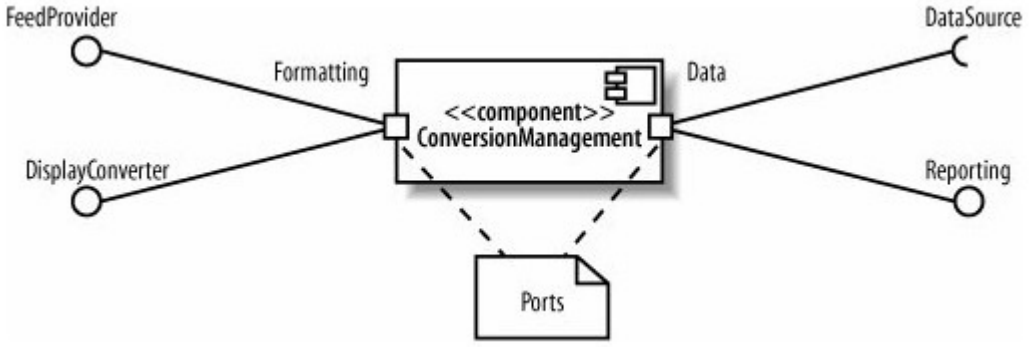


Hay chỉ cần liệt kê các class ra bên trong component sau ký hiệu <<relizatión>> như hình:

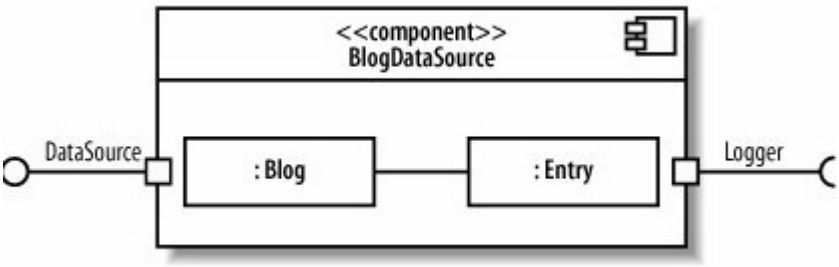


5. Ports và Internal structure của component:

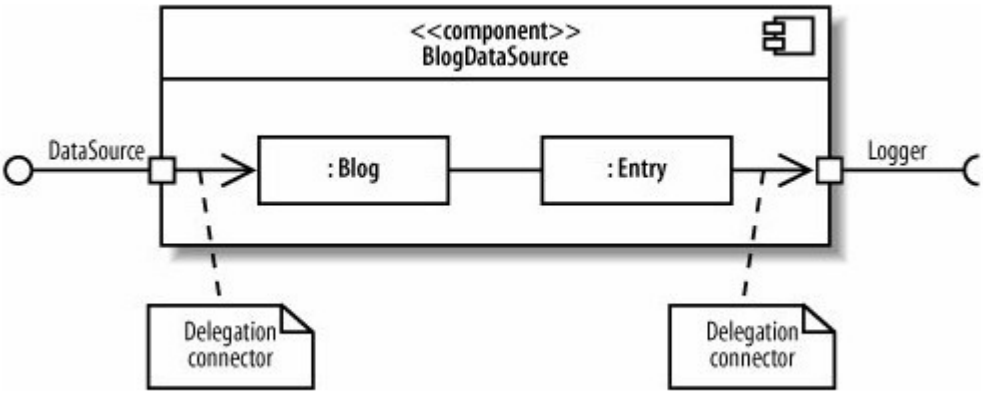
Một component cũng bao gồm các port và internal struture, bạn có thể sử dụng port để mô tả các các riêng biệt mà component được sử dụng liên quan đến port. Ví dụ như hình dưới ConversionManagement component có một Formatting và một Data port, mỗi cái gắn với một interface:



Bạn cũng có thể dùng internal structure (cấu trúc nội) của một component để mô tả các phần, các thuộc tính và các liên kết:



Delegation Connectors: một *provided interface* có thể được nhận diện bởi một thành phần bên trong của component, tương tự một *required interface* có thể được yêu cầu bởi một thành phần bên trong khi đó ta dùng *delegation connectors* để biểu diễn chúng như hình dưới:


<<component>>
BlogDataSource
DataSource
: Blog
: Entry
Logger
Delegation connector
Delegation connector

XIII. Tổ chức mô hình: Packages

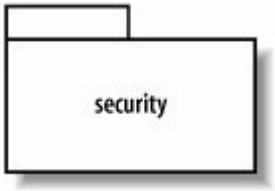
1. Giới thiệu:

Khi một phần mềm phát triển nó ngày càng phức tạp lên, có thể sẽ có hàng trăm lớp. Rất khó để nhận ra và sắp xếp chúng, một cách khả thi là sắp xếp chúng theo các nhóm có quan hệ với nhau. Các lớp có liên quan đễn giao diện có thể vào 1 nhóm, các lớp công cụ có thể vào 1 nhóm…

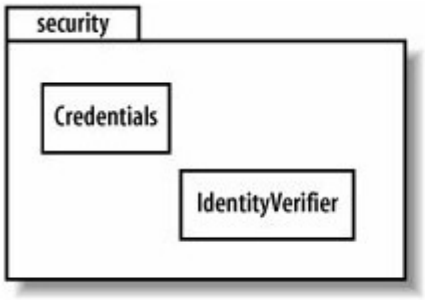
2. Các khái niệm:

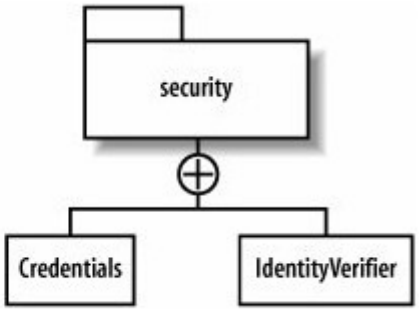
a. Packages:

Như đã package là cách tổ chức các lớp theo các nhóm có quan hệ với nhau.
Ví dụ như khi thiết kế một CMS bạn gom các lớp có liên quan đến bảo mật vào một gói security chúng ta biểu diễn nó bằng bằng biểu tượng một folder với tên của package bên trong:

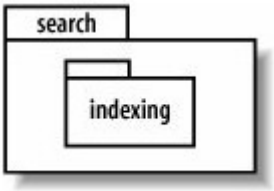


Chúng ta vẽ các yếu tố của package bên trong hoặc bên ngoài với các mũi tên quan hệ:





Một package có thể bao gồm một package khác:

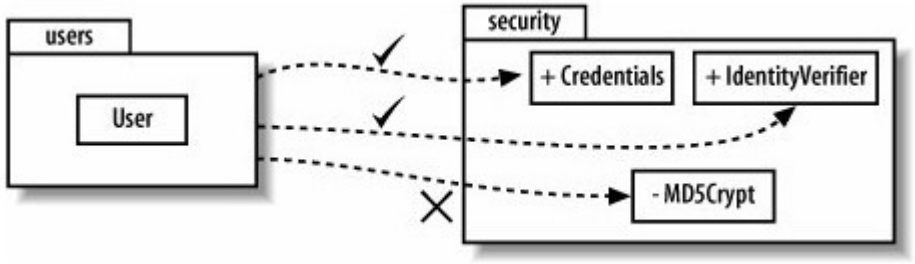


b. Namespace:

Khi một yếu tố trong một package muốn sử dụng một yếu tố trong một package khác nó phải chỉ rõ yếu tố nó cần nằm ở đâu. Do đó khi muốn dùng một yếu tố nào bạn phải bao gồm cả tên của package và yếu tố cần dùng ***packageName***::***className***. Hơn nữa khi có hai lớp giống nhau trong 2 package khác nhau điều này sẽ tránh nhầm lẫn.

c. Tầm nhìn của các yếu tố:

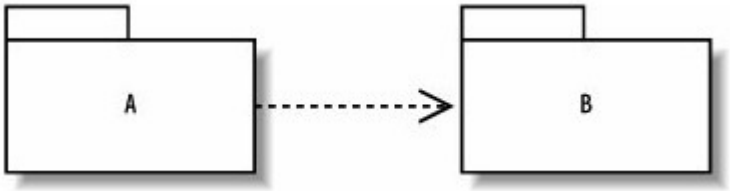
Các yếu tố trong một package có thể có tầm nhìn là public hay private. Nếu có tầm nhìn là public thì các yếu tố có thể tủy cập từ bên ngoài package đó. Các yếu tố có tầm nhìn là private thì không thể truy cập từ bên ngoài mà chỉ có thể truy cập nó tùe bên trong. Để biểu diễn thuộc tính public ta dùng 1 dấu cộng trước yếu tố đó, còn nếu là private ta dùng dấu trừ như hình:



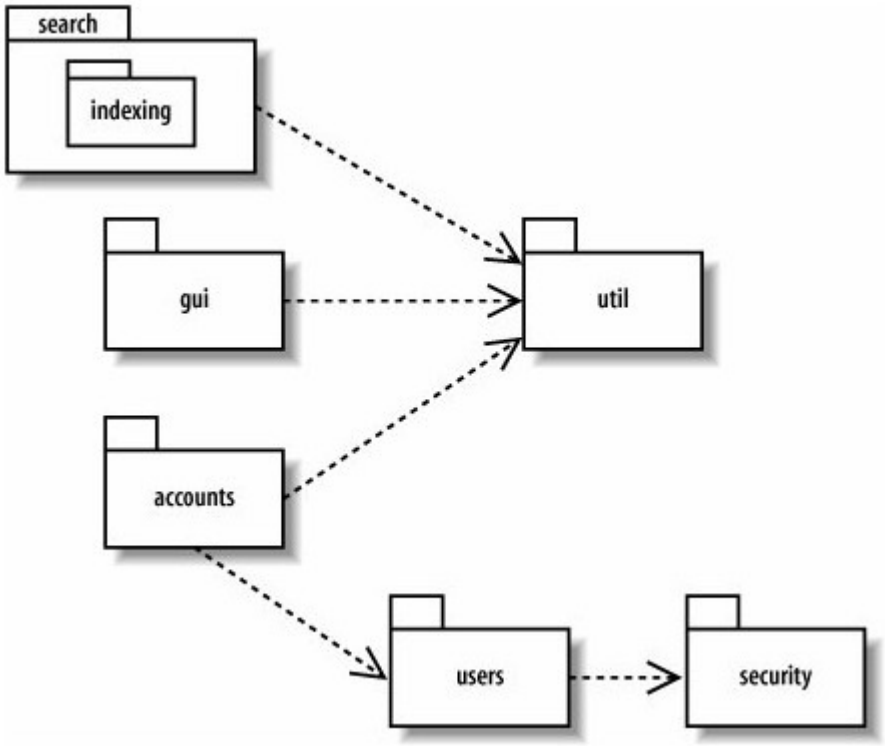
Trong Java một lớp bên trong package có thuộc tính public sẽ được khai báo là public đằng trước: **public** class ClassName {} nếu không có public thì mặc định sẽ là private.

d. Quan hệ giữa các package:

Nếu một yếu tố trong một package A sử dụng một yếu tố trong package B thì ta nói package A phụ thuộc packge B ta vẽ như sau:

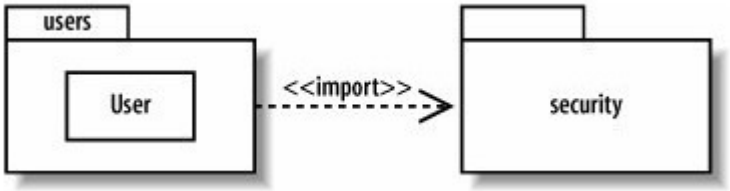


Hiểu được quan hệ giữa các package sẽ rất hữu ích trong việc phân tích sự ổn định của hệ thống.

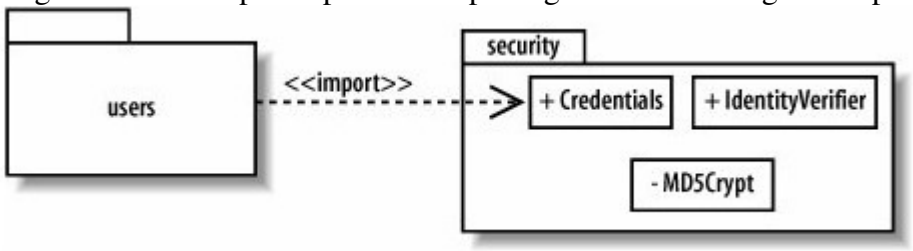


e. Import và truy cập các package:

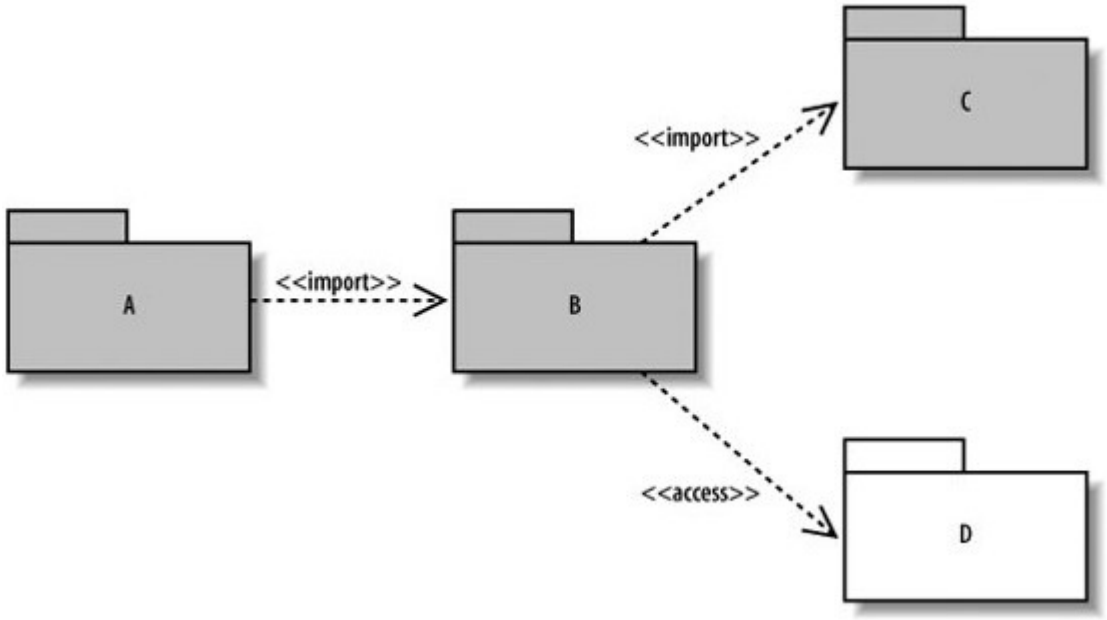
Nếu một package import một package khác khi đó nó có thể sử dụng các yếu tố của package kia mà không cần phải nêu toàn bộ tên đầy đủ (bao gồm cả tên package) của yếu tố đó.
Để biểu diễn quan hệ *import* ta vẽ một mũi tên đến package cần import:



Một package có thể chỉ import 1 phần của 1 package khác mà không cần import toàn bộ:



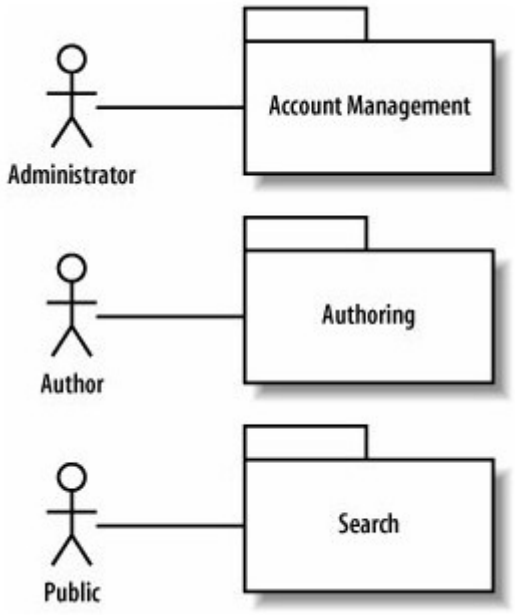
Khi import một package ta chỉ có thể thấy được các yếu tố có thuộc tính là public.
Ta cũng có thể sử dụng kiểu import là *public import* và *private import* (mặc định là public). Public import nghĩa là tất cả các yếu tố được import sẽ mang thuộc tính public, tương tự với private. Ta biểu diễn private import bằng ký hiệu <<access>> như hình:



Hình trên ta thấy A import B, B import C và access D vì vậy A có thể truy cập được các yếu tố public của C nhưng không thể truy cập được các yếu tố public của D vid B đã private import D và các yếu tố này trở thành private trong B.

f. Dùng package để quản lý usecase:

Package không chỉ dùng để quản lý các lớp mà còn dùng để quản lý các usecase nó giúp bạn thấy được các actor sẽ tương tác với các phần nào của hệ thống:


Account Management
Administrator
Authoring
Author
Search
Public

XIV. Mô hình hoá trạng thái các đối tượng: State Machine Diagrams

1. Giới thiệu:

Activity diagrams và interaction diagrams rất thích hợp để mô tả các thuộc tính nhưng chúng vẫn còn thiếu sót. Đôi khi trạng thái của một đối tượng hoặc hệ thống là một nhân tố quan trọng trong hành vi (behavior) của nó. Vì thế cần thiết phải mô tả trạng thái của một đối tượng và các sự kiện có thể thay đổi trạng thái đó khi đó State Machine Diagrams là thích hợp nhất. State machine diagram được sử dụng rất nhiều trong các hệ thống phần mềm và phần cứng như:

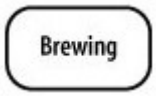
- Hệ thống thời gian thực
- Các thiết bị chuyên môn như máy ATM
- Các game bắn súng góc nhìn thứ I như Doom hay Haftlife

2. Các khái niệm:

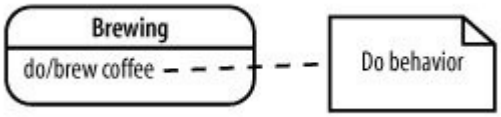
Trong một lược đồ state machine có 2 khái niệm chính đó là state và transition.

a. State:

State là một tình trạng ở một khoảng thời xác định, state có thể là 1 đặc tính thụ động như trạng thái tắt và mở của 1 cái đèn, hay một đặc tính chủ động hoặc một sự việc mà đối tượng đó đang làm như một người pha café sẽ có trạng thái là đang pha café trong suốt quá trình pha café của anh ta.
Một state sẽ được biểu diễn bằng một hình chữ nhật bo tròn các góc với tên của trạng thái nằm bên trong:



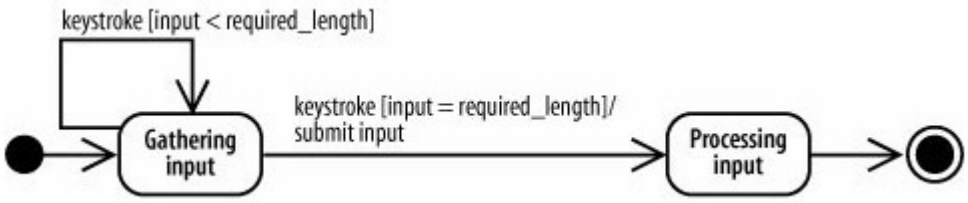
Nếu trạng thái là "đang làm" 1 hành động nào đó nó sẽ được biểu diễn như hình sau :



Điều nàu có nghĩa là đối tượng sẽ thực hiện hành vi *brew coffee* khi đối tượng ở trạng thái *brewing*.

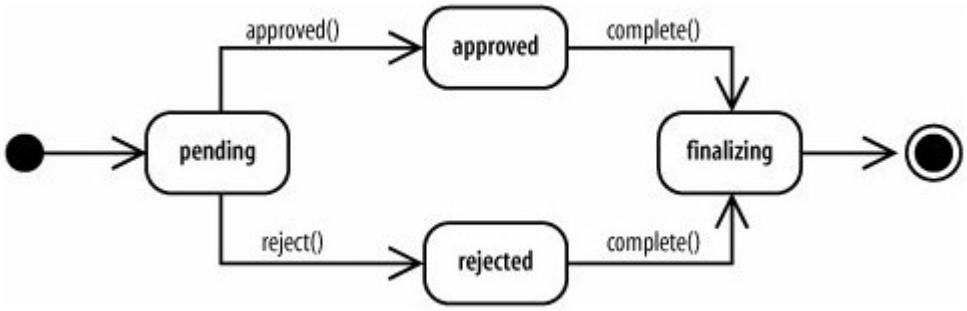
b. Transition:

Transition là sự chuyển trạng thái từ trạng thái nguồn sang một trạng thái đích. Transition được biểu diễn bằng một mũi tên bên trên có ghi bằng một Transition Description mô tả trường hợp đã gây ra sự chuyển trạng thái đó. Một transition description có dạng *trigger[guard] / behavior.*
Trong đó *trigger* là sự kiện dẫn đến 1 transition, một *guard* là một biến bool cho phép hoặc khóa transition tương ứng, *behavior* là một ngắt được thực thi khi transition xảy ra.



3. State Machine diagram trong phần mềm:

Trong một phần mềm state diagram được dùng để mô tả vòng đời của một đối tượng, như trong ví dụ dưới đây nó được dùng để mô tả vòng đời của một *AccountApplication* trong quá trình nó chuyển từ trạng thái *pending* sang *approved* hoặc *rejected* và sau đó là *finalizing*.

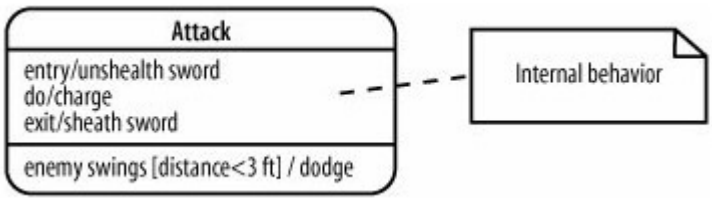


State diagram thích hợp để mô tả những đối tượng có hành vi phụ thuộc vào các trạng thái khác nhau, ví dụ như trong *AccountApplication* ở trên sau khi gọi phương thức *complete()* khi ở trạng thái kết thúc *finalizing* sẽ không phân biệt được trạng thái dẫn đến hành vi này (*approved* hoặc *rejected*). State diagrams là cách hiệu quả nhất để mô tả các thông tin này một cách rõ ràng. Nếu một đối tượng chỉ có một vòng đời đơn giản thì không cần thiết phải mô tả bằng một state diagram. Ví dụ đối tượng *contact information* lưu thông tin về địa chỉ liên lạc của tác giả không thay đổi trạng thái trừ lúc được tạo ra và được hủy nên không cần thiết phải mô tả bằng một state diagram.

4. Nâng cao:

a. Internal Behavior:

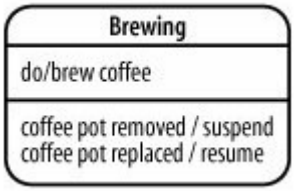
Internal behavior là bất cứ một hành vi nào xảy ra khi một đối tượng đang ở một trạng thái nào đó. Thông thường bạn sẽ thấy 1 hành vi được thực hiện khi một trạng thái ở chế độ kích hoạt, Internal Behavior mô tả chi tiết hơn nó gồm cả sự kiện *entry* và *exit*.



Internal behavior được viết dưới dạng *label / behavior*. Trong đó label mô tả khi nào hành vi *behavior* sẽ được thực thi. Có 3 loại label: entry, exit và do.
Hành vi *entry* sẽ xảy ra ngay khi state của đối tượng được kích hoạt còn exit thì ngược lại. Không giống như *do*, *entry* và *exit* behavior không thì bị ngắt.

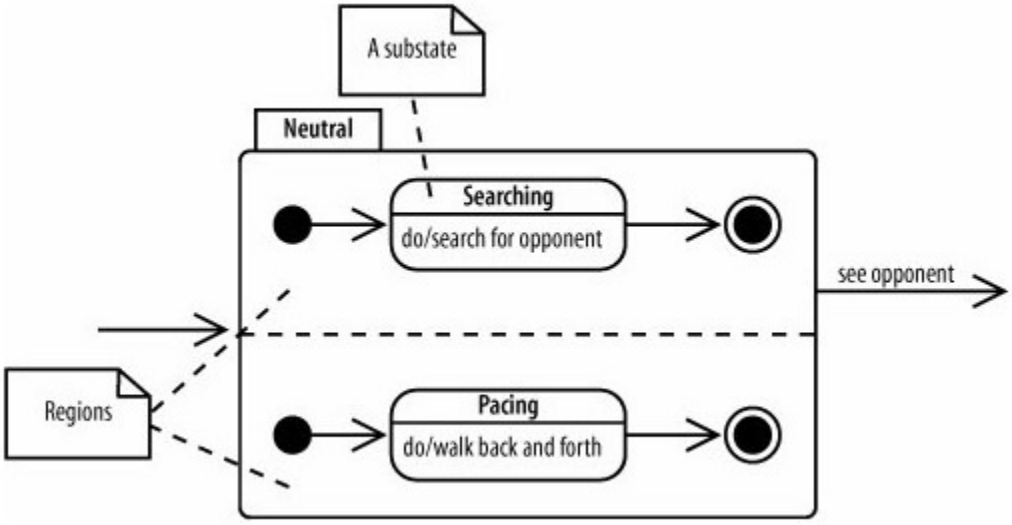
b. Internal Transitions:

Internal Transitions là một transition mà nó gây ra một hành động trong một trạng thái mà không làm thay đổi trạng thái đó, nó khác với việc chuyển thành chính trạng thái đó (self transition) vì nó không kích hoạt hành động *entry* và *exit*. Ví dụ như một người pha café sẽ tạm dừng hành động pha café khi bị lấy mất cái cốc và sẽ tiếp tục khi cốc được trả lại:



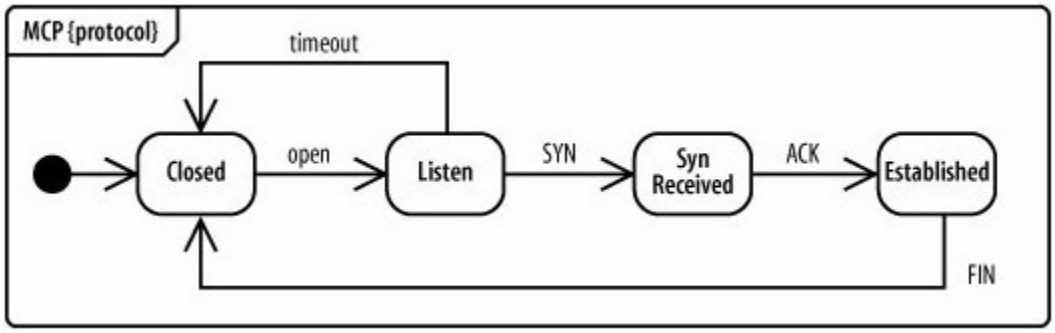
c. Composite States:

Composite state là một state mang nhiều state diagram, mỗi diagram thuộc về một vùng (region) khác nhau và được phân cách bởi một đường nét đứt.



d. Protocol State Machines:

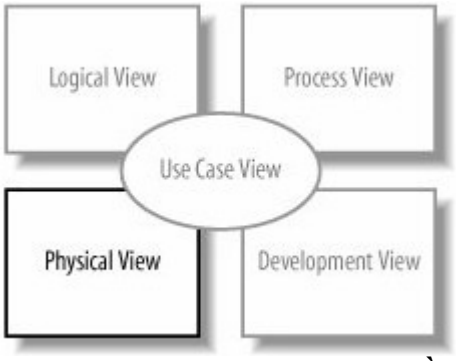
Protocol state machines là một loại đặc biệt của state machine, nó tập trung vào việc mô tả cách một giao thức ví dụ như TCP làm việc. Protocol state machines được vẽ bằng một hình chữ nhật có vát góc với tên giao thức nằm trước ký hiệu {protocol}, như hình:



XV. Mô hình hoá việc triển khai hệ thống: Deployment Diagrams

1. Giới thiệu:

UML deployment diagram nằm trong phần physical view của hệ thống, phần này bao gồm các yếu tố vật lý của hệ thống như các file thực thi và phần cứng mà các file này thực thi trên đó. Lược đồ này sẽ mô tả cách ấn định phần mềm lên một hệ thống phần cứng và cách mà các thành phần trong hệ thống giao tiếp với nhau.



Hình : Deployment diagrams nằm trong phần Physical view

2. Các khái niệm:

Trong một Deployment diagram ta cần tìm hiểu 2 khái niệm chính đó là node và artifact.

a. Nodes:

Trong một UML deployment diagram người ta dùng khái niệm node để mô tả một tài nguyên phần cứng, nhưng node không nhất thiết cứ phải là một phần cứng mà có thể là một loại phần mềm cung cấp môi trường hoạt động cho phần mềm khác hay gọi là môi trươgn thực thi cũng được gọi là node. Như vậy 1 node có thể là 1 thiết bị phần cứng:

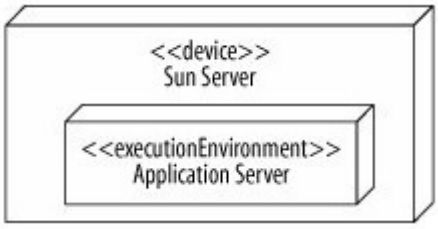
- Một PC
- Một Server
- Một đĩa cứng

Hoặc có thể là 1 môi trường thực thi:

- Một hệ điều hành
- Một Webserver

Một node được biểu diễn bằng một hình hộp (cube) bên trong có một có một ký hiệu **<<device>>** để diễn tả một thiết bị phần cứng hay **<<executionEnvironment>>** để diễn tả một môi trường thực thi, như hình:

Một môi trường thực thi có thể hoạt động trên một hệ thống phần cứng:



Hình: Application Server chạy trên một Sun Server

Để biểu diễn một thực thể của một node ta dùng cấu trúc **name:type**. Như hình:

Hình: svr1 là một thực thể có kiểu là Sun Blade Server

Giao tiếp giữa các node: các node có thể giao tiếp với nhau ví dụ như một ứng dụng trên desktop giao tiếp với một server bằng TCP/IP:

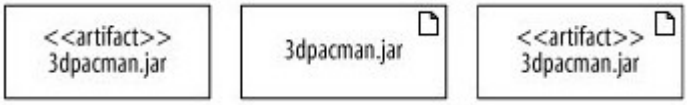
Đường giao tiếp giữa các node được biển diễn bằng 1 đường thẳng bên trên mô tả loại giao tiếp.

b. Artifacts:

Ngược lại các phần mềm được triển khai được gọi là các artifact, các artifcat là các file thực thi và được sử dụng bởi hệ thống. Các artifact thông thường có thể kể ra đây như:

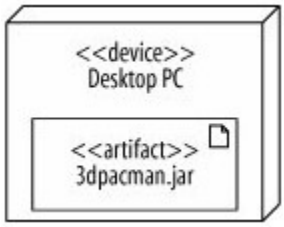
- Các file thực thi như .exe hay .jar
- Các file .dll
- Các file source như: .java hay .cpp
- Các file cấu hình như .xml, .properties, hoặc .txt

Trong Lược đồ một artifact được biểu diễn bằng một hình chữ nhật bên trong có ký hiệu <<artifact>> hoặc một biểu tượng tài liệu nằm ở góc trên bên trái hoặc cả 2 ký hiệu:

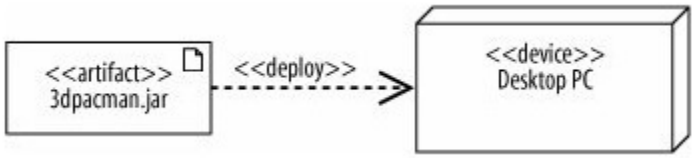


c. Triển khai một artifact trên một node:

Ta biểu diễn việc triển khai 1 artifact trên một node bằng việc đặt các artifact bên trong node tương ứng:

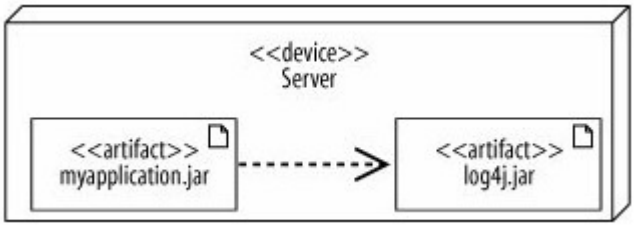


Ta cũng có thể vẽ một mũi tên từ một artifact đến 1 node với ký hiệu <<deploy>> bên trên để biểu diễn:



Nếu có nhiều artifact bên trong một node ta có thể chỉ liệt kê tên các artifact bên trong 1 node để tiết kiệm không gian:

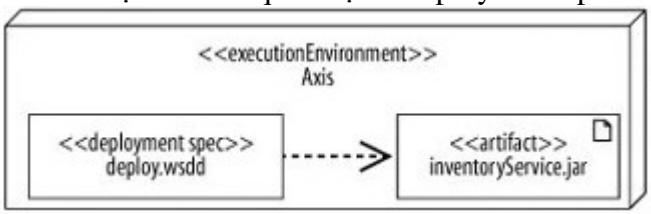
Tuy nhiên để biểu diễn được mối quan hệ giữa các artifact ta không thể liệt kê như trên mà phải vẽ các biểu tượng của artifact với mũi tên phụ thuộc nối các artifact:



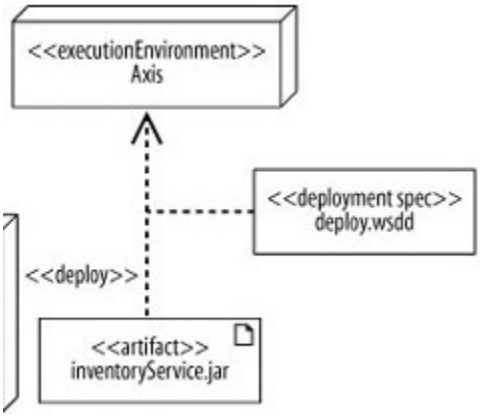
d. Deployment Specifications:

Thông thường triển khai một hệ thống không đơn giản chỉ là bỏ 1 file lên máy mà bạn phải xác định các tham số trước khi phần mềm có thể thực thi. Một **deployment specification** là một artifact đặc biệt dùng để xác định cách mà một artifact khác được triển khai trên một node, nó cung cấp các thông tin cần thiết để một artifact có thể chạy tốt trên môi trường triển khai. Deployment specifications được biểu diễn bằng 1 hình chữ nhật với một ký hiệu <<deployment spec>>. Có 2 cách để gắn một deployment spec vào lược đồ:

- Vẽ một mũi tên quan hệ từ Deployment spec đến Artifact tương ứng:



- Gắn Deployment spec vào mũi tên quan hệ giữa artifact và node:



File deloy.wsdd là file mô tả, nó xác định lớp nào sẽ được thực thi và phương thức nào của lớp sẽ được gọi.

Deployment diagrams rất hữu ích trong việc thiết kế phần mềm, nó sẽ giúp bạn triển khai phần mềm trên một hệ thống dễ dàng và chính xác hơn.