

Phân tích thiết kế Hệ thống

Giảng viên: Nguyễn Bá Ngọc

Chương 5

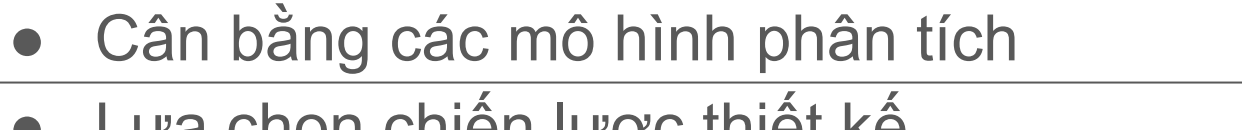
Hà Nội-2021

Thiết kế (phần 1)

Nội dung

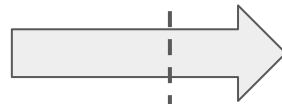
- Cân bằng các mô hình phân tích
- Lựa chọn chiến lược thiết kế
- Thiết kế lớp và phương thức
- Phân mảnh và biểu đồ gói
- Biểu đồ thành phần và biểu đồ triển khai
- Các thành phần kiến trúc Hệ thống
- Mẫu kiến trúc

Nội dung

- 
- Cân bằng các mô hình phân tích
 - Lựa chọn chiến lược thiết kế
 - Thiết kế lớp và phương thức
 - Phân mảnh và biểu đồ gói
 - Biểu đồ thành phần và biểu đồ triển khai
 - Các thành phần kiến trúc Hệ thống
 - Mẫu kiến trúc

Chuyển từ phân tích sang thiết kế

Phân tích



Thiết kế



Tính gắn kết giữa các mô hình

- Chúng ta mô hình hóa 1 hệ thống từ nhiều góc nhìn khác nhau và tạo nhiều mô hình từ mỗi góc nhìn, vì vậy
 - Các mô hình khác nhau có thể có sự gắn kết về nội dung
 - và có những thông tin được biểu diễn nhiều lần trong các mô hình khác nhau,
 - Ví dụ: Các quan hệ trong biểu đồ và trong các đặc tả tương ứng, các hoạt động và các thông điệp, v.v..
- Mỗi mô hình đều có vai trò riêng và có thể tập trung hơn vào 1 khía cạnh cụ thể.
- Tuy nhiên cần đảm bảo tính nhất quán giữa các biểu diễn của 1 nội dung và đảm bảo tính lô-gic giữa những nội dung liên quan.

Cần kiểm tra đồng thời các mô hình phân tích trước khi bắt đầu các hoạt động thiết kế

Cân bằng các mô hình phân tích

- Được thực hiện bởi 1 đội gồm người phân tích, người thiết kế và khách hàng.
- Các mục đích:
 - Kiểm tra tính đúng đắn của các mô hình
 - Tìm lỗi và điều chỉnh kịp thời

Tiềm ẩn nguy cơ người phân tích bị phạt vì những lỗi được phát hiện.

Cân bằng các mô hình chức năng

1. Có thể ánh xạ 2 chiều giữa các sự kiện trong các đặc tả ca sử dụng và các hoạt động trong biểu đồ hoạt động.
2. Các nút đối tượng (nếu có) trong biểu đồ hoạt động phải được sử dụng trong đặc tả ca sử dụng.
3. Quan hệ thứ tự của các sự kiện trong đặc tả ca sử dụng phải nhất quán với quan hệ thứ tự của các hoạt động trong biểu đồ hoạt động.
4. Đảm bảo tương ứng 1-1 giữa các ca sử dụng trong biểu đồ ca sử dụng tổng quan và các đặc tả ca sử dụng.
5. Các tác nhân có trong đặc tả ca sử dụng phải được biểu diễn trên biểu đồ ca sử dụng.
6. Các bên liên quan có trong đặc tả ca sử dụng phải được biểu diễn như các tác nhân trên biểu đồ ca sử dụng.
7. Các mối quan hệ có trong đặc tả ca sử dụng phải nhất quán với các mối quan hệ có trong biểu đồ ca sử dụng.

Cân bằng các mô hình cấu trúc

1. Đảm bảo tương ứng 1-1 giữa các thẻ CRC và các lớp trên biểu đồ lớp.
2. Các trách nhiệm được liệt kê trong thẻ CRC phải được thực hiện bằng các phương thức trong biểu đồ lớp.
3. Các đối tác trong thẻ CRC được thể hiện bằng mối quan hệ liên kết trong biểu đồ lớp.
4. Các thuộc tính được liệt kê trong các thẻ CRC phải được gắn kết với các thuộc tính trong lớp trên biểu đồ lớp.
5. Các thuộc tính có kiểu là lớp khác được biểu diễn bằng quan hệ tổng hợp giữa các lớp.
6. Các mối quan hệ trong các thẻ CRC phải nhất quán với các mối quan hệ được biểu diễn trong biểu đồ lớp.
7. Chỉ sử dụng các lớp liên kết nếu các mối quan hệ có các thuộc tính tiêu biểu không có trong cả 2 lớp.
8. Các biểu đồ đối tượng phải tương thích với biểu đồ lớp: Thuộc tính, cơ sở, liên kết.

Cân bằng các mô hình hành vi

1. Các tác nhân & các đối tượng trong biểu đồ tuần tự cũng là các tác nhân và các đối tượng trong biểu đồ giao tiếp tương ứng và ngược lại.
2. Nếu thông điệp được trao đổi giữa 2 đối tượng trong biểu đồ tuần tự thì phải tồn tại liên kết giữa 2 đối tượng tương ứng trong biểu đồ giao tiếp.
3. Tất cả thông điệp trên biểu đồ tuần tự phải xuất hiện như các thông điệp trong biểu đồ giao tiếp tương ứng và ngược lại.
4. Các điều kiện bảo vệ thông điệp trong các biểu đồ tuần tự cần được biểu diễn như các điều kiện bảo vệ tương đương trên các biểu đồ giao tiếp tương ứng.
5. Số thứ tự trên các nhãn thông điệp trong các biểu đồ giao tiếp phải tương ứng với trật tự từ trên xuống dưới của các thông điệp được gửi trong biểu đồ tuần tự.
6. Các bước chuyển trạng thái trong biểu đồ máy trạng thái phải gắn kết với các thông điệp trong các biểu đồ tương tác.
7. Các giá trị trong ma trận CRUDE phải gắn kết với các thông điệp trong các biểu đồ tương tác.

Chúng ta đang sử dụng 2 dạng biểu đồ tương tác là: Tuần tự và Giao tiếp

Chức năng & Cấu trúc

1. Mỗi lớp trên biểu đồ lớp phải được gắn với ít nhất 1 ca sử dụng
2. Mỗi hoạt động trong biểu đồ hoạt động và mỗi sự kiện trong đặc tả ca sử dụng phải được gắn kết với ít nhất 1 phương thức trên biểu đồ lớp.
3. Mỗi nút đối tượng trên biểu đồ hoạt động phải là 1 thuộc tính hoặc 1 đối tượng thuộc lớp có trong biểu đồ lớp
4. Mỗi thuộc tính hoặc 1 mối liên hệ liên quan/tổng hợp trên biểu đồ lớp phải được gắn kết với ít nhất 1 chủ thể hoặc đối tượng của 1 sự kiện trong đặc tả ca sử dụng.

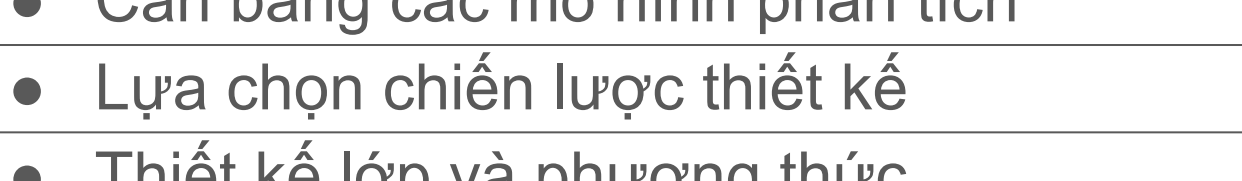
Chức năng & Hành vi

1. Các biểu đồ tương tác phải được gắn với ca sử dụng
2. Các tác nhân trong các biểu đồ tương tác hoặc ma trận CRUDE phải được gắn với các tác nhân trong ca sử dụng
3. Các thông điệp trong các biểu đồ tương tác, các bước chuyển trong máy trạng thái và các giá trị trong ma trận CRUDE phải gắn kết với các hoạt động trong biểu đồ hoạt động và các sự kiện trong đặc tả ca sử dụng.
4. Tất cả các đối tượng phức tạp trong các biểu đồ hoạt động phải được biểu diễn bằng biểu đồ máy trạng thái.

Cấu trúc & Hành vi

1. Các đối tượng trong ma trận CRUDE phải tương thích với biểu đồ lớp.
2. Biểu đồ máy trạng thái phải được gắn với đối tượng thuộc lớp có trong biểu đồ lớp
3. Các đối tượng trong các biểu đồ tương tác phải thuộc lớp có trong biểu đồ lớp.
4. Các thông điệp trong các biểu đồ tương tác và các bước chuyển trong biểu đồ máy trạng thái phải được gắn với phương thức trong lớp.
5. Các trạng thái của đối tượng trong biểu đồ máy trạng thái phải được lưu bởi 1 thuộc tính của lớp tương ứng.

Nội dung

- Cân bằng các mô hình phân tích
 - Lựa chọn chiến lược thiết kế
 - Thiết kế lớp và phương thức
 - Phân mảnh và biểu đồ gói
 - Biểu đồ thành phần và biểu đồ triển khai
 - Các thành phần kiến trúc Hệ thống
 - Mẫu kiến trúc
- 

Các chiến lược thiết kế

- Phát triển riêng - Xây dựng trong phạm vi nội bộ từ đầu
- Mua gói ứng dụng
 - Các ứng dụng văn phòng (trình soạn thảo, bảng tính, v.v..)
 - Hệ thống quản lý nhân sự, quản lý quán cafe, v.v..
- Tích hợp hệ thống
 - Hệ thống bán hàng và hệ thống quản lý hóa đơn, v.v..
- Thuê đối tác (gia công phần mềm)
 - Hợp tác phát triển các tính năng không yêu cầu bảo mật

Phát triển riêng

- Có thể đáp ứng những yêu cầu rất chuyên sâu.
- Có thể linh động và sáng tạo trong giải quyết vấn đề.
- Dễ dàng thay đổi các thành phần
- Phát triển các kỹ năng cá nhân
- Nhưng có thể làm quá tải đội IT
- Có thể tạo thêm rủi ro đáng kể

Mua gói phần mềm

- Mua phần mềm đã được bán trên thị trường (ví dụ phần mềm kế toán).
- Có phạm vi từ 1 thành phần, 1 công cụ đến cả 1 hệ thống thông tin doanh nghiệp.
- Có thể hiệu quả nếu phù hợp với nghiệp vụ
- Có thể đã được kiểm tra và thử nghiệm kỹ hơn
- Nhưng phải chấp nhận chức năng như được cung cấp.
- Có thể phải thay đổi cách công ty hoạt động.
- Có thể phải tùy chỉnh đáng kể hoặc phát triển các thành phần ghép nối/thay thế.

Tích hợp hệ thống

- Kết hợp các gói, hệ thống cũ, và phần mềm mới
 - Không hiếm trường hợp mua phần mềm có sẵn và thuê đối tác tích hợp nó với hệ thống hiện có
- Thách thức chủ yếu là tích hợp dữ liệu
 - Có thể yêu cầu chuyển đổi dữ liệu
 - Có thể phải sử dụng gói phần mềm bổ xung để viết dữ liệu theo cùng định dạng với hệ thống cũ
- Kỹ thuật đóng gói
 - Đóng gói hệ thống cũ và cung cấp các API để hệ thống mới có thể tương tác với nó.
 - Tránh can thiệp vào hệ thống hiện có.

Gia công phần mềm

- Thuê 1 công ty ngoài xây dựng hệ thống
- Có thể mở rộng các tài nguyên và kỹ năng hiện có.
- Yêu cầu điều hướng 2 chiều, trao đổi thông tin và lòng tin.
- Nhưng có thể mất kiểm soát, chia sẻ thông tin quan trọng, chuyển giao kinh nghiệm.
- Thận trọng lựa chọn đối tác, Thận trọng chuẩn bị hợp đồng và phương thức thanh toán.
- Không bao giờ thuê gia công những gì không hiểu

Gia công phần mềm₍₂₎

Các dạng hợp đồng:

- Thời gian-và-Thỏa thuận: Thanh toán theo thời gian và các khoản chi
- Giá cố định: Thanh toán theo giá như đã thỏa thuận
- Giá trị tăng cường: Thanh toán theo % lợi nhuận.

Lựa chọn chiến lược thiết kế

	Phát triển riêng	Sử dụng gói phần mềm	Gia công
Nhu cầu nghiệp vụ	Duy nhất	Phổ biến	Không phải cốt lõi
Chuyên môn hiện có	Có nghiệp vụ và kỹ thuật	Không có kỹ thuật	Không có nghiệp vụ hoặc kỹ thuật
Kỹ năng dự án	Có mong muốn phát triển kỹ năng liên quan	Phát triển kỹ năng không nằm trong chiến lược	Quyết định gia công là 1 quyết định chiến lược
Người quản lý dự án	Có trình độ cao và phương pháp phát triển đã được kiểm chứng	Có thể điều phối công việc của nhà cung cấp	Có trình độ cao trong môi trường của đối tác
Khung thời gian	Linh động	Ngắn	Ngắn hoặc linh động

Lựa chọn chiến lược thực hiện

- Xác định các công cụ và kỹ thuật cần để tự phát triển
- Xác định các gói phù hợp với nhu cầu người dùng
- Tìm các công ty có thể xây dựng hệ thống theo hợp đồng
- Tạo ma trận lựa chọn để cân nhắc các ưu điểm và nhược điểm của mỗi lựa chọn
 - Kết hợp với tính khả thi kỹ thuật, kinh tế, và tổ chức
 - Sử dụng cơ chế yêu cầu đề xuất (RFP) và yêu cầu thông tin (RFI) để thu thập ước lượng kinh phí & thời gian từ những nhà cung cấp tiềm năng

Nội dung

- Cân bằng các mô hình phân tích
- Lựa chọn chiến lược thiết kế
- Thiết kế lớp và phương thức
- Phân mảnh và biểu đồ gói
- Biểu đồ thành phần và biểu đồ triển khai
- Các thành phần kiến trúc Hệ thống
- Mẫu kiến trúc

Đánh giá chất lượng thiết kế

- Đánh giá dựa trên 1 tập chỉ số
- Ghép nối/Coupling - Thể hiện độ gần của mối quan hệ giữa các lớp
- Thống nhất/Cohesion - Thể hiện mức độ các thuộc tính và các phương thức cùng hỗ trợ 1 đối tượng
- Đồng sinh/Connascence - Thể hiện mức độ phụ thuộc giữa các đối tượng
 - (*Connascence means to be born together*)


Tính ghép nối

- Ghép nối gần có nghĩa là thay đổi trong 1 phần của thiết kế có thể yêu cầu thay đổi trong phần khác
- Phân loại
 - Ghép nối tương tác được đo theo trao đổi thông điệp
 - Ghép nối kế thừa được đo theo quan hệ kế thừa trong cây phân cấp lớp
 - [Coad and Edward Yourdon, *Object-Oriented Design*, 1991]
- Có thể hạn chế ghép nối tương tác bằng cách giới hạn các thông điệp
 - Quy tắc Demeter
- Có thể hạn chế ghép nối kế thừa bằng cách chỉ sử dụng khái quát hóa/chi tiết hóa và áp dụng quy tắc khả thay/substitutability

Quy tắc Demeter

- Thông điệp chỉ nên được gửi bởi 1 đối tượng:
 - Tới chính nó
 - Tới đối tượng là thuộc tính của nó hoặc lớp cha
 - Tới đối tượng được truyền qua tham số cho phương thức
 - Tới đối tượng được tạo bởi phương thức
 - Tới đối tượng được lưu trong biến toàn cục
- Ghép nối tương tác được tăng lên theo mỗi trường hợp, ví dụ:
 - Nếu phương thức gọi truyền các thuộc tính cho phương thức được gọi
 - hoặc phương thức gọi phụ thuộc vào giá trị được trả về bởi phương thức được gọi

Các loại ghép nối tương tác

Mức độ	Loại	Mô tả
Tốt	Không có ghép nối trực tiếp	Các phương thức không liên kết với nhau/không gọi lẫn nhau
	Dữ liệu	Phương thức gọi truyền 1 biến cho phương thức được gọi. Nếu đó là 1 đối tượng bao gồm nhiều thuộc tính thì tất cả các thuộc tính đều được sử dụng bởi phương thức được gọi để thực hiện trách nhiệm của nó.
	Dấu ấn	Phương thức gọi truyền 1 đối tượng, nhưng phương thức được gọi chỉ sử dụng 1 phần của đối tượng để thực hiện trách nhiệm của nó.
	Điều khiển	Phương thức gọi truyền 1 biến điều khiển có giá trị được sử dụng để điều khiển phương thức được gọi
	Chung hoặc toàn cục	Phương thức truy cập tới "1 vùng dữ liệu toàn cục" nằm ngoài phạm vi của đối tượng
Không tốt	Nội dung hoặc Biểu diễn	Phương thức của 1 đối tượng truy cập tới biểu diễn bên trong của đối tượng khác. Tạo ngoại lệ đối với quy tắc đóng gói và ẩn dữ liệu. (ví dụ bạn/friend trong C++)

Nguồn: Meilir Page-Jones, *The Practical Guide to Structured Systems Design*, 2nd, 1978.

Ghép nối kế thừa

- Có nhiều luồng quan điểm cho rằng ghép nối kế thừa là cần thiết.
- Một số trường hợp điển hình:
 - Phương thức được định nghĩa trong lớp con gọi phương thức được định nghĩa trong lớp cha
 - Phương thức được định nghĩa trong lớp con sử dụng thuộc tính được định nghĩa trong lớp cha
 - Phương thức ảo được định nghĩa trong lớp cha dựa vào các lớp con của nó để định nghĩa 1 phần xử lý của nó.
 - *(Có thể được triển khai với ngôn ngữ lập trình?)*
- Người lập trình cần cân đối việc phá vỡ quy tắc đóng gói và ẩn dữ liệu và tăng tính ghép nối mong đợi giữa các lớp con và các lớp cha của nó.

Tính ghép nối trong thiết kế

- Nhìn chung, cần cực tiểu tính ghép nối
- Tuy nhiên trong thực tế có thể có những ghép nối kỹ thuật khó tránh khỏi
 - Ví dụ điển hình như 1 lớp tầng HCI phụ thuộc vào lớp lĩnh vực ứng dụng tương ứng:
 - Biểu mẫu thông tin sinh viên hiển thị các thông tin của 1 đối tượng sinh viên thuộc lớp Student (tầng lĩnh vực ứng dụng).

Tính thống nhất

- Tính thống nhất thể hiện 1 lớp, 1 đối tượng, hoặc 1 phương thức chỉ biểu diễn 1 thứ.
- Phân loại:
 - Thống nhất phương thức
 - 1 phương thức có thực hiện nhiều hơn 1 chức năng?
 - Phương thức đa chức năng khó hiểu và khó triển khai hơn
 - Thống nhất lớp
 - Các thuộc tính và phương thức có biểu diễn 1 đối tượng?
 - Lớp không được trộn lẫn các vai trò lớp, lĩnh vực hoặc đối tượng
 - Thống nhất khái quát hóa/chi tiết hóa
 - Các lớp trong 1 cây phân cấp phải biểu diễn quan hệ "1 loại của"/"a-kind-of", không phải tổng hợp hoặc liên quan.
 - Nguyên lý khả thay / Substitutability

Thống nhất phương thức

- Thể hiện tính thống nhất trong 1 phương thức
 - Mỗi phương thức chỉ nên làm 1 việc
- Có 7 trường hợp / loại thống nhất phương thức được xác định với mức chất lượng giảm dần từ tốt đến không tốt
 - (Được cung cấp trong trang tiếp theo)
- Nhìn chung tính thống nhất phương thức cần được cực đại hóa


Các kiểu thống nhất phương thức

Mức độ	Kiểu	Mô tả
Tốt	Chức năng	Mỗi phương thức thực hiện 1 công việc (ví dụ, tính GPA hiện tại)
	Tuần tự	Phương thức kết hợp 2 chức năng, trong đó đầu ra của chức năng đầu tiên được sử dụng như đầu vào của chức năng thứ 2 (ví dụ, định dạng và kiểm tra GPA hiện tại)
	Giao tiếp	Phương thức kết hợp 2 chức năng sử dụng chung tập thuộc tính (ví dụ, tính điểm GPA hiện tại và điểm GPA tích lũy)
	Tiến trình	Phương thức hỗ trợ nhiều chức năng liên kết yếu, ví dụ, phương thức có thể tính GPA của sinh viên, in thông tin sinh viên, tính GPA tích lũy.
	Tạm thời hoặc cổ điển	Phương thức hỗ trợ nhiều chức năng liên quan theo thời gian (ví dụ, khởi tạo tất cả các thuộc tính)
	Lô-gic	Phương thức hỗ trợ nhiều chức năng liên quan, nhưng lựa chọn chức năng cụ thể được thực hiện dựa trên biến điều khiển được cung cấp như tham số. Ví dụ, phương thức có thể mở tài khoản tiết kiệm, tài khoản ghi nợ, hoặc tính lãi dựa theo thông điệp được gửi tới bởi phương thức gọi.
Không tốt	Ngẫu nhiên	Không thể xác định mục đích của phương thức hoặc nó thực hiện nhiều chức năng không liên quan. Ví dụ, phương thức có thể cập nhật hồ sơ khách hàng, tính lãi khoản vay, và phân tích cấu trúc giá của đối thủ.
Nguồn: Page-Jones, <i>The Practical Guide to Structured System</i> , và Myers, <i>Composite/Structured Design</i> .		

Tổng nhất lớp

- Thể hiện tính tổng nhất giữa các thuộc tính và các phương thức của 1 lớp, cùng hỗ trợ biểu diễn 1 thứ
- Tất cả các thuộc tính và các phương thức trong 1 lớp đều cần thiết để biểu diễn lớp
 - Mỗi lớp chỉ nên có các thuộc tính và phương thức cần thiết để biểu diễn các đối tượng của nó
- Ví dụ,
 - Lớp Student cần có các thuộc tính như: MSSV, Họ, Tên, Địa Chỉ
 - Nhưng không có các thuộc tính như: CPU, RAM, OS

Các kiểu thống nhất lớp

Mức	Kiểu	Mô tả
Tốt	Lý tưởng	Lớp không có yếu tố hỗn hợp
	Hỗn hợp vai trò	Lớp có thành phần trực tiếp kết nối đối tượng của lớp với đối tượng thuộc lớp khác trên cùng tầng (ví dụ, tầng lĩnh vực ứng dụng), nhưng các thuộc tính đó không liên quan với ý nghĩa cơ bản của lớp
	Hỗn hợp lĩnh vực	Lớp có thành phần trực tiếp kết nối đối tượng thuộc lớp với đối tượng thuộc lớp khác trên tầng khác, nhưng thành phần đó không liên quan đến ý nghĩa cơ bản của lớp. Trong những trường hợp này, các thành phần lạc hướng thuộc về lớp khác trên tầng khác. Ví dụ, thuộc tính cổng nằm trong 1 lớp lĩnh vực đúng ra phải nằm trong lớp thuộc tầng kiến trúc hệ thống liên quan với lớp lĩnh vực.
Không tốt	Hỗn hợp đối tượng	Lớp biểu diễn nhiều loại đối tượng. Lớp như vậy cần được phân tách thành các lớp riêng biệt. Thông thường, mỗi đối tượng chỉ sử dụng 1 phần định nghĩa của lớp.

Nguồn: Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

Lớp thống nhất lý tưởng

Các yêu cầu được đưa ra bởi Glenford Meyers:

- Chứa nhiều phương thức có thể được nhìn thấy từ bên ngoài lớp và mỗi phương thức chỉ thực hiện 1 chức năng
- Các phương thức chỉ sử dụng các thuộc tính và các phương thức khác được định nghĩa trong lớp hoặc (các) lớp cha của nó
- Không có gắn kết mức điều khiển giữa các phương thức của nó

Loại thống nhất: Hỗn hợp vai trò

- Tính hỗn hợp vai trò làm giảm khả năng tái sử dụng
- Ví dụ 1: lớp Student và Room cùng thuộc tầng lĩnh vực ứng dụng, nhưng phòng học không phải thuộc tính mô tả sinh viên.
 - Thiết kế lớp Student trực tiếp gắn kết với lớp Room có vấn đề hỗn hợp vai trò.
- Ví dụ 2: Lớp Student và lớp Registration cũng cùng thuộc tầng lĩnh vực ứng dụng, và sinh viên là 1 phần trong đăng ký học tập
 - Thiết kế lớp Registration trực tiếp gắn kết với lớp Student không phát sinh vấn đề hỗn hợp vai trò

Loại thống nhất: Hỗn hợp lĩnh vực

- Phát sinh khi lớp chứa thành phần trực tiếp gắn kết với lớp trên 1 tầng khác / không thuộc tầng của nó.
- Kiểm tra tính năng có phải thiết yếu đối với lớp không?
 - Ví dụ 1: Lớp Order và lớp Printer thuộc các tầng khác nhau
 - Triển khai phương thức in hóa đơn trong lớp đơn hàng? Có thể triển khai lớp đơn hàng mà không có phương thức in hay không?
 - Ví dụ 2: Lớp OrderDetailView và lớp Order cũng thuộc các tầng khác nhau
 - Có thể triển khai giao diện đơn hàng mà không đọc dữ liệu đơn hàng hay không?

Loại thống nhất: Hỗn hợp đối tượng

- Lớp biểu diễn nhiều hơn 1 nhóm đối tượng / Có thành phần không được sử dụng cho 1 số đối tượng
- Ví dụ: Giả sử chúng ta đang triển khai 1 lớp Person với các phương thức:
 - GetSalary() trả về lương nếu là nhân viên.
 - GetCashBack() trả về số dư tài khoản hoàn tiền nếu là khách hàng.
 - Với mỗi đối tượng Person là của 1 nhân viên hoặc 1 khách hàng chúng ta đều có tính năng không sử dụng đến
 - Cho thấy lớp Person đang quá rộng.
 - Giải pháp:
 - Tách các tính năng gắn với nhân viên và đưa vào 1 lớp Employee
 - Tách các tính năng gắn với khách hàng và đưa vào 1 lớp Customer
 - Các lớp Employee và Customer kế thừa lớp Person

Đồng sinh

- Các lớp phụ thuộc lẫn nhau tới mức thay đổi trong 1 lớp kéo theo thay đổi trong lớp kia.
- Khái quát hóa và kết hợp tính ghép nối và tính thống nhất trên cơ sở các giới hạn đóng gói
- Có 3 mức đóng gói được sử dụng:
 - Đóng gói mức 0: Đóng gói trong phạm vi 1 dòng lệnh
 - Đóng gói mức 1: Phương thức kết hợp nhiều dòng lệnh
 - Được đánh giá bởi Tính thống nhất phương thức và ghép nối tương tác.
 - Đóng gói mức 2: Lớp chứa cả thuộc tính và phương thức
 - Thống nhất lớp, thống nhất khái quát hóa/chi tiết hóa, và ghép nối kế thừa

Đánh giá chất lượng với tính đồng sinh

- Tính đồng sinh có thể được phân tích với đóng gói mức 1/mức phương thức, và đóng gói mức 2/mức lớp.
- Mã nguồn tốt phải:
 - Cực tiểu hóa đồng sinh giữa các giới hạn đóng gói (ít liên kết phụ thuộc giữa các đơn vị)
 - Cực đại hóa đồng sinh trong phạm vi đóng gói (nhiều quan hệ phụ thuộc trong 1 đơn vị)
 - => Lớp con không nên trực tiếp truy cập thuộc tính hoặc phương thức ẩn của lớp cha

Chi tiết về các loại đồng sinh được cung cấp trong trang tiếp theo

Các kiểu đồng sinh

Kiểu	Mô tả
Tên	Nếu 1 phương thức sử dụng 1 thuộc tính, nó được gắn với tên của thuộc tính. Nếu tên thuộc tính thay đổi, nội dung của phương thức cũng sẽ phải thay đổi.
Kiểu hoặc Lớp	Nếu 1 lớp có 1 thuộc tính thuộc kiểu A, nó được gắn với kiểu của thuộc tính. Nếu kiểu của thuộc tính thay đổi, khai báo thuộc tính cũng sẽ phải thay đổi.
Quy cách	Một lớp có 1 thuộc tính mà miền giá trị có ý nghĩa (ví dụ, số tài khoản với các giá trị trong phạm vi từ 1000 tới 1999 là các tài sản). Nếu khoảng thay đổi, thì tất cả phương thức sử dụng thuộc tính cũng cần phải thay đổi.
Giải thuật	Hai phương thức khác nhau của 1 lớp cùng phụ thuộc vào 1 giải thuật để chạy đúng (ví dụ, kiểm tra tính hợp lệ của địa chỉ email khi tạo tài khoản và khi gửi yêu cầu khôi phục mật khẩu). Nếu giải thuật cơ sở cần thay đổi, thì phương thức thêm và tìm kiếm cũng sẽ phải thay đổi.
Vị trí	Thứ tự mã nguồn trong 1 phương thức hoặc thứ tự các tham số trong 1 phương thức là đặc biệt quan trọng để phương thức chạy đúng. Nếu bất kỳ thứ tự nào sai, thì phương thức, tối thiểu, không hoạt động đúng.
Nguồn: Meilir Page-Jones, <i>Comparing Techniques by Means of Encapsulation and Connascence</i> and Meilir Page-Jones, <i>Fundamentals of Object-Oriented Design in UML</i> .	

Các hoạt động thiết kế đối tượng

- Mở rộng và tiếp tục phát triển các mô hình phân tích.
- Tạo các mô tả cho thiết kế chi tiết lớp và phương thức
 - Bổ xung các đặc tả cho mô hình hiện có
 - Xác định các tiềm năng tái sử dụng
 - Tái cấu trúc thiết kế
 - Tối ưu hóa thiết kế
 - Ánh xạ các lớp lĩnh vực ứng dụng và ngôn ngữ triển khai
- Thay đổi 1 lớp trên 1 tầng có thể khiến các lớp khác (có thể trên các tầng khác) có gắn kết với lớp đó thay đổi theo.

Các đặc tả bổ xung

- Kiểm tra các mô hình cấu trúc và hành vi để đảm bảo các thành phần là đầy đủ và cần thiết.
- Thiết lập giới hạn nhìn của các thuộc tính và phương thức của mỗi lớp
 - Gắn kết với ngôn ngữ triển khai
- Xác định nguyên mẫu của các phương thức, bao gồm:
Tên phương thức, các tham số được truyền cho phương thức, kiểu giá trị được trả về khi phương thức được gọi.
- Xác định các ràng buộc mà đối tượng phải đáp ứng
 - Có 3 loại ràng buộc: Tiền điều kiện, hậu điều kiện, tính chất bất biến.
- Các thông tin bổ xung được mô tả dưới dạng hợp đồng, hoặc được thêm vào các thẻ CRC và biểu đồ lớp.

Các đặc tả bổ xung₍₂₎

- Chúng ta cũng cần quyết định cách xử lý các ngoại lệ:
 - *(phát sinh nếu các ràng buộc không được đảm bảo)*
 - Hệ thống có thể xử lý đơn giản bằng cách bỏ qua?
 - Hoàn tác các thay đổi đã thực hiện dẫn đến ngoại lệ?
 - Để người dùng lựa chọn phương án xử lý ngoại lệ nếu có nhiều phương án?
- Người thiết kế phải thiết kế các lỗi mà hệ thống phải xử lý
 - Gắn kết với ngôn ngữ lập trình
 - Sử dụng mã lỗi
 - Sử dụng cơ chế exception (như trong C++ hoặc Java)

Xác định các tiềm năng tái sử dụng

- Trong pha thiết kế, bên cạnh việc sử dụng các mẫu phân tích, chúng ta có thể sử dụng:
 - Các mẫu thiết kế, các nền tảng, các thư viện, các thành phần.
- Các tiềm năng tái sử dụng có thể thay đổi theo từng tầng
 - Ví dụ: 1 thư viện có thể ít hữu ích trong tầng lĩnh vực ứng dụng, nhưng lại rất hữu ích trong tầng nền tảng.
- Các mẫu thiết kế
 - Rất hữu ích để gom nhóm các lớp tương tác có thể cung cấp giải pháp cho 1 vấn đề phổ biến. Có thể giải quyết “Vấn đề thiết kế điển hình trong 1 ngữ cảnh cụ thể
 - *(Các chi tiết sẽ được cung cấp sau).*

Tiềm năng tái sử dụng: Nền tảng và thư viện

- Các nền tảng cung cấp 1 tập lớp có thể được sử dụng làm cơ sở để phát triển chương trình ứng dụng
 - Có thể triển khai toàn bộ hoặc 1 phần của hệ thống.
 - CORBA, DCOM, Spring Boot, Struts, Qt, v.v..
- Thư viện
 - Bao gồm 1 tập lớp, tương tự như nền tảng, được thiết kế để tái sử dụng
 - Có nhiều thư viện để hỗ trợ nhiều mảng nghiệp vụ:
 - Xử lý số học và thống kê
 - Phát triển giao diện đồ họa (tầng HCI)
 - Kết nối với CSDL (tầng quản lý dữ liệu)
- Nền tảng và thư viện thường cho phép kế thừa các lớp của nó
- Chúng ta có thể áp dụng kế thừa để tái sử dụng lớp hoặc trực tiếp tạo đối tượng của các lớp nền tảng và thư viện
 - Làm tăng tính ghép nối giữa chương trình.

Tiềm năng tái sử dụng: Thành phần

- Thành phần:

- Mảnh phần mềm đầy đủ, được đóng gói để có thể nhúng vào 1 hệ thống để cung cấp 1 tập chức năng cụ thể.
 - Ví dụ các thành phần được triển khai dựa trên các công nghệ ActiveX hoặc JavaBean.
- Cung cấp 1 tập phương thức giao diện để tương tác với các đối tượng có trong nó.
 - Lô-gic hoạt động bên trong thành phần được ẩn sau các API.
- Có thể được triển khai bằng các lớp thư viện và nền tảng, nhưng cũng có thể được sử dụng để triển khai nền tảng.
- Nếu giao diện không thay đổi, thì cập nhật thành phần phiên bản mới thường không yêu cầu thay đổi xử lý mã nguồn
 - Có thể không yêu cầu biên dịch lại.
- Thường được sử dụng để giảm lược các chi tiết triển khai.

Tái cấu trúc thiết kế

- Đóng gói phần chung / factoring
 - Lớp mới có thể được gắn kết với các lớp cũ thông qua kế thừa, tổng hợp hoặc liên kết
 - Lưu ý các vấn đề ghép nối, thống nhất, và đồng sinh.
- Triển khai các mối quan hệ trên biểu đồ lớp như những thuộc tính.
 - Các ngôn ngữ hướng đối tượng hầu như không phân biệt các quan hệ tổng hợp hay liên quan.
 - Các mối quan hệ liên quan và tổng hợp phải được chuyển thành các thuộc tính trong lớp.

Đóng gói phần chung

- Tách và đóng gói phần giống nhau và khác nhau của các thứ được quan tâm
- Các lớp mới được hình thành thông qua:
 - Quan hệ khái quát hóa (thuộc loại), hoặc
 - Tạo lớp bậc cao hơn (ví dụ, tạo lớp Employee từ tập vị trí công việc)
 - Chi tiết hóa, hoặc
 - Tạo lớp cụ thể (ví dụ, tạo lớp Secretary hoặc Bookkeeper từ lớp Employee)
 - Quan hệ tổng hợp (có các thành phần)

Tối ưu hóa thiết kế

Có thể được thực hiện để tạo thiết kế hiệu quả hơn

- Kiểm tra các đường dẫn tới đối tượng:
 - Có những trường hợp thông điệp từ 1 đối tượng tới 1 đối tượng khác phải qua nhiều bước trung gian.
 - Nếu đường dẫn dài và thông điệp được gửi thường xuyên, thì cần cân nhắc rút ngắn đường truyền thông điệp.
 - Có thể bổ xung thuộc tính vào đối tượng gửi thông điệp để có thể truy cập trực tiếp.

Tối ưu hóa thiết kế: Thuộc tính

- Xác định những phương thức sử dụng thuộc tính và những đối tượng sử dụng phương thức.
- Nếu chỉ có phương thức đọc và cập nhật sử dụng thuộc tính, và chỉ có đối tượng thuộc 1 lớp gửi thông điệp đọc và cập nhật đối tượng
 - Thuộc tính đó có thể thuộc về lớp gọi thay vì lớp được gọi
 - Chuyển thuộc tính sang lớp gọi có thể giúp hệ thống hoạt động nhanh hơn.

Tối ưu hóa thiết kế: Luồng ra

- Luồng ra bao gồm các thông điệp được gửi trực tiếp và gián tiếp trong thời gian thực hiện phương thức
 - Thông điệp trực tiếp được gửi bởi chính phương thức đó
 - Thông điệp gián tiếp được gửi bởi phương thức được gọi trong khi thực hiện phương thức đó
- Nếu kích thước luồng ra quá lớn so với các phương thức khác trong hệ thống, thì phương thức đó có thể cần được tối ưu hóa.

Tối ưu hóa thiết kế: Thứ tự thực hiện

- Kiểm tra thứ tự thực hiện các lệnh trong phương thức được sử dụng thường xuyên
 - Có những trường hợp có thể sắp xếp lại các câu lệnh để đạt được hiệu quả cao hơn.
 - Ví dụ: Giả định kịch bản tìm kiếm, trong đó phạm vi tìm kiếm được thu hẹp sau khi tìm kiếm theo 1 thuộc tính.
 - Có thể tìm cách tối ưu hóa xử lý theo 1 thứ tự thuộc tính tối ưu.

Tối ưu hóa thiết kế: Sử dụng bộ nhớ đệm

- Tránh tính toán lại thuộc tính suy diễn:
 - Ví dụ tạo thuộc tính total để lưu tổng giá trị đơn hàng.
 - Thiết lập cơ chế kích hoạt tiến trình tính toán.
 - Chỉ tính lại giá trị của thuộc tính suy diễn khi thay đổi các thuộc tính thành phần được sử dụng trong tính toán.

Ảnh xạ các lớp lĩnh vực ứng dụng

- Tái cấu trúc các thành phần đa kế thừa nếu ngôn ngữ chỉ hỗ trợ đơn kế thừa.
- Tái cấu trúc các thành phần kế thừa nếu ngôn ngữ không hỗ trợ kế thừa.
- Tránh triển khai 1 thiết kế hướng đối tượng với 1 ngôn ngữ lập trình không hỗ trợ lập trình hướng đối tượng.

Các ràng buộc và các hợp đồng

- Hợp đồng là 1 tập các ràng buộc/constraints và các đảm bảo / guarantees
 - Nếu đối tượng gửi (client) đáp ứng các ràng buộc, thì đối tượng cung cấp dịch vụ (server) đảm bảo hành vi mong đợi
- Hợp đồng mô tả thông điệp được gửi giữa các đối tượng
 - Được tạo cho các phương thức công khai của lớp.
 - Cần chứa đủ thông tin để người lập trình có thể hiểu hành vi mong đợi của phương thức.
- Các loại ràng buộc:
 - Tiền điều kiện - Phải đúng trước khi phương thức được thực hiện
 - Hậu điều kiện - Phải đúng sau khi phương thức kết thúc
 - Bất biến - Phải luôn đúng đối với tất cả các đối tượng.

Biểu mẫu hợp đồng thông điệp

Tên phương thức:	Tên lớp:	Mã số:
Mã khách:		
Ca sử dụng liên quan:		
Mô tả các trách nhiệm:		
Các tham số nhận được:		
Kiểu dữ liệu trả về:		
Tiền điều kiện:		
Hậu điều kiện:		

Đặc tả phương thức

- Mô tả các chi tiết của mỗi phương thức
 - Để người lập trình viết mã nguồn
 - *(đây là tài liệu thiết kế rất gần triển khai)*
- Không có quy chuẩn, tuy nhiên có thể bao gồm các thông tin:
 - Thông tin tổng quan (Tên phương thức, tên lớp, v.v...)
 - Sự kiện - Kích hoạt phương thức (ví dụ khi nhấn chuột)
 - Cấu trúc thông điệp được truyền tới bao gồm cả các tham số và giá trị trả về
 - Đặc tả giải thuật
 - Các thông tin liên quan khác

Biểu mẫu đặc tả phương thức

Tên phương thức:	Tên lớp:	ID:
Mã thỏa thuận:	Lập trình viên:	Thời hạn:
Ngôn ngữ lập trình:		
<input type="checkbox"/> Visual Basic <input type="checkbox"/> Smalltalk <input type="checkbox"/> C++ <input type="checkbox"/> Java		
Kích hoạt/Sự kiện:		

Các tham số nhận được:	Ghi chú:
Kiểu dữ liệu:	

Các thông điệp đã gửi & Các tham số đã truyền:	Kiểu dữ liệu của tham số:	Ghi chú:
TênLớp.TênPhươngThức:		

Tham số trả về:	Ghi chú:
Kiểu dữ liệu:	
Đặc tả giải thuật:	
Các ghi chú khác:	

Nội dung

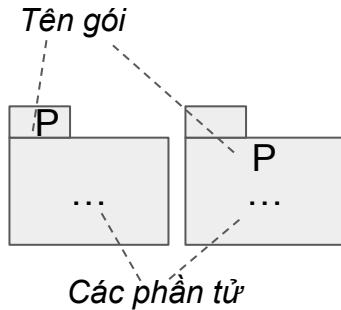
- Cân bằng các mô hình phân tích
- Lựa chọn chiến lược thiết kế
- Thiết kế lớp và phương thức
- Phân mảnh và biểu đồ gói
- Biểu đồ thành phần và biểu đồ triển khai
- Các thành phần kiến trúc Hệ thống
- Mẫu kiến trúc

Phân mảnh

- Phân mảnh: Tạo hệ thống con với các lớp cộng tác.
- Các tương tác và các mối liên hệ có thể là cơ sở để xác định những lớp nào cần được gom cùng nhau
 - Các biểu đồ lớp
 - Các biểu đồ giao tiếp
 - Ví dụ, các thông điệp và liên kết trong biểu đồ giao tiếp
 - Nếu có nhiều thông điệp được gửi giữa 2 lớp thì đưa cả 2 lớp vào cùng 1 mảnh
 - Ma trận CRUDE
 - V.V..
- Tạo mảnh từ các lớp có bậc liên kết cao.

Biểu đồ gói: Hệ ký hiệu

Biểu đồ gói biểu diễn các gói cùng với các mối quan hệ.

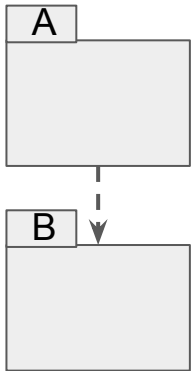


- **Gói/Package:**

- Nhóm lô-gic của nhiều phần tử, là không gian tên của các phần tử
- Có thể gom nhiều phần tử liên quan thành 1 thành phần bậc cao, giúp giản lược mô hình

- **Quan hệ phụ thuộc/Dependency:**

- Nếu B thay đổi thì A cũng sẽ thay đổi theo.
- A phụ thuộc vào B được biểu diễn bằng mũi tên nét đứt từ A tới B



Biểu đồ gói: Hệ ký hiệu₍₂₎

<<merge>>
----->

- Quan hệ hợp nhất/Merge:
 - Nội dung của gói đích (theo mũi tên) được hợp nhất với nội dung của gói nguồn.
 - Các thành phần cùng tên cũng được hợp nhất.

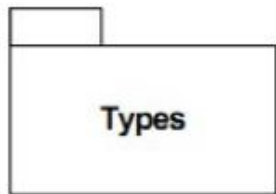
<<import>>
----->

- Quan hệ nhập/Import
 - Thêm các phần tử của gói đích vào gói nguồn.
 - Các phần tử được thêm vào có thể được nhìn thấy từ bên ngoài / nhập công khai.

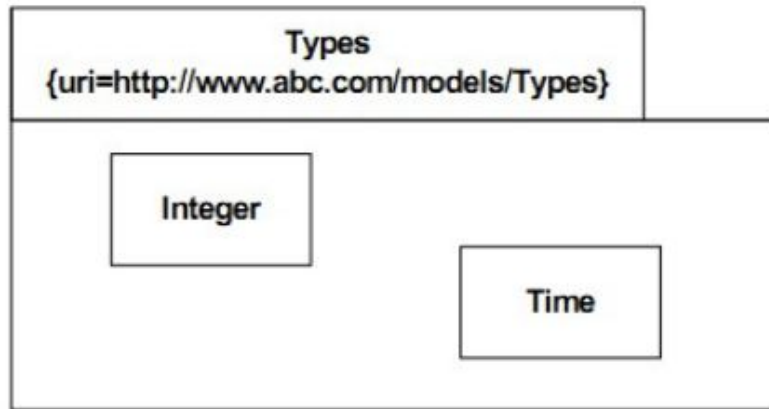
<<access>>
----->

- Quan hệ truy cập/access
 - Thêm các phần tử của gói đích vào gói nguồn
 - Các phần tử được thêm vào không được nhìn thấy từ bên ngoài / nhập riêng tư

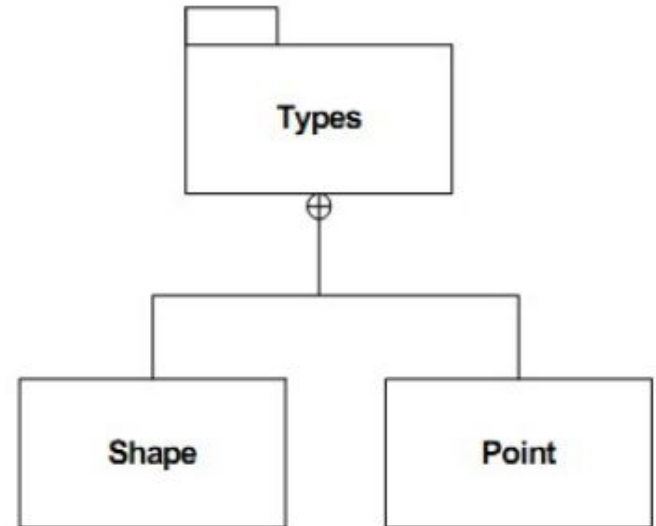
Biểu diễn phần tử thuộc gói



Ẩn nội dung
của gói Types



Các lớp Integer và Time thuộc gói
Types

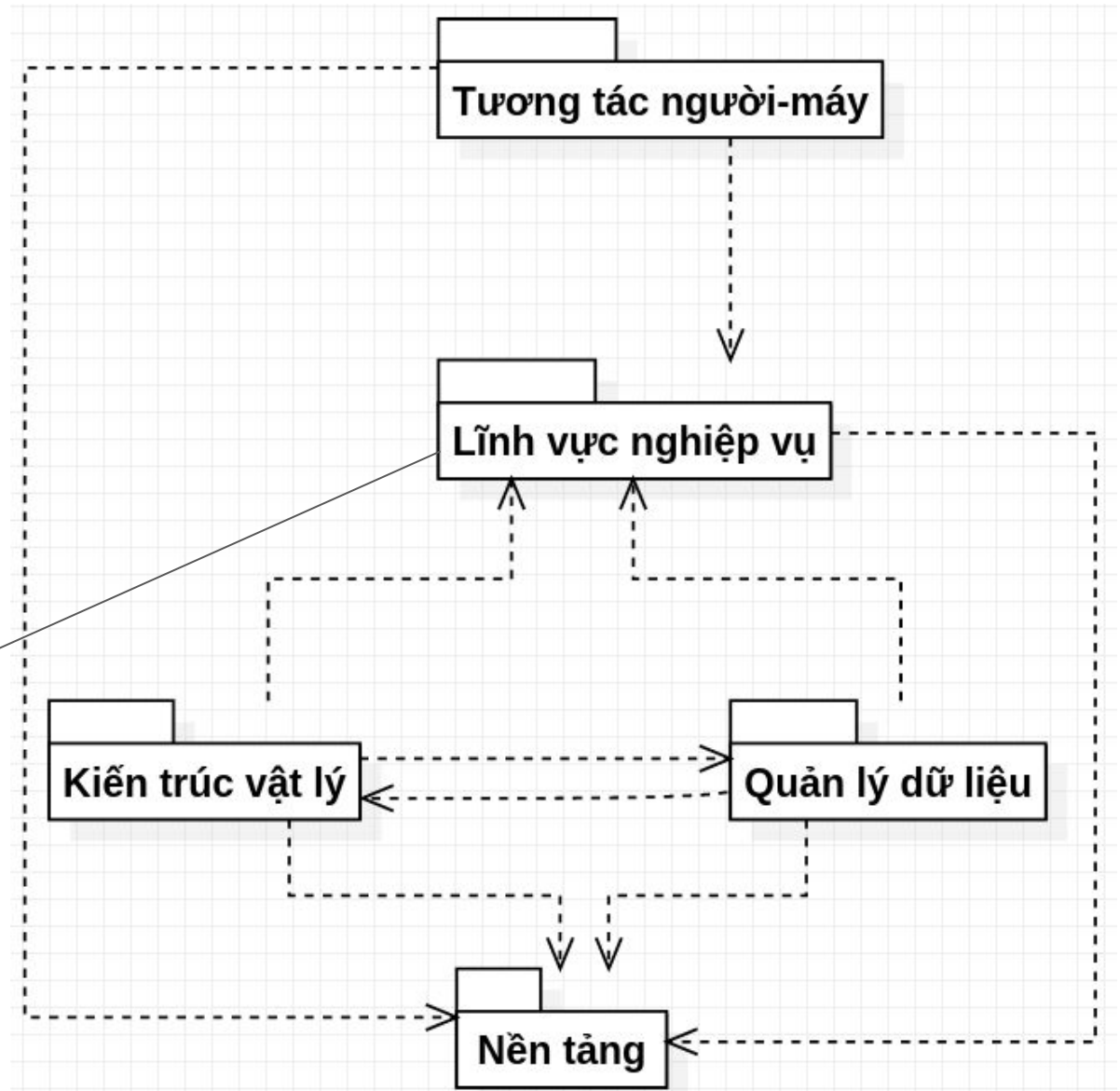


(1 trường hợp khác với) Các lớp
Shape và Point thuộc gói Types

Tổng quan kiến trúc 5 tầng

Hệ thống được tạo thành từ 5 tầng, mỗi tầng được biểu diễn như 1 gói khái quát trên biểu đồ.

*Mô hình
cấu trúc*



Tổng quan kiến trúc 5 tầng₍₂₎

- **Tầng nền tảng:** Cung cấp các thành phần cơ bản để xây dựng các ứng dụng HĐT. Ví dụ lớp: Array, Map
- **Tầng nghiệp vụ:** Các xử lý nhằm đáp ứng các hoạt động nghiệp vụ, đã được đề cập tới ở bước phân tích, và tiếp tục được phát triển ở pha thiết kế. Ví dụ lớp: Order, Customer
- **Quản lý dữ liệu:** Các xử lý liên quan đến lưu trữ cố định. Ví dụ lớp: CustomerDAM, FileInputStream
- **Tương tác người - máy:** Triển khai giao diện tách rời với nghiệp vụ. Ví dụ lớp: Form, Button

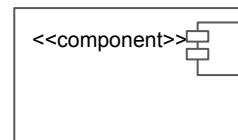
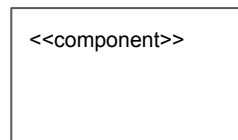
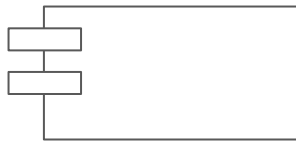
Nội dung

- Cân bằng các mô hình phân tích
- Lựa chọn chiến lược thiết kế
- Thiết kế lớp và phương thức
- Phân mảnh và biểu đồ gói
- Biểu đồ thành phần và biểu đồ triển khai
- Các thành phần kiến trúc Hệ thống
- Mẫu kiến trúc

Biểu đồ thành phần: Các ký hiệu

Biểu đồ thành phần biểu diễn các thành phần khác nhau của hệ thống cùng với các giao diện và các mối quan hệ, và cho biết cấu trúc lô-gic của hệ thống

- Có nhiều cách biểu diễn thành phần:



- Các giao diện được cung cấp và được yêu cầu được biểu diễn bằng các đường tròn và nửa đường tròn.



Giao diện được cung cấp

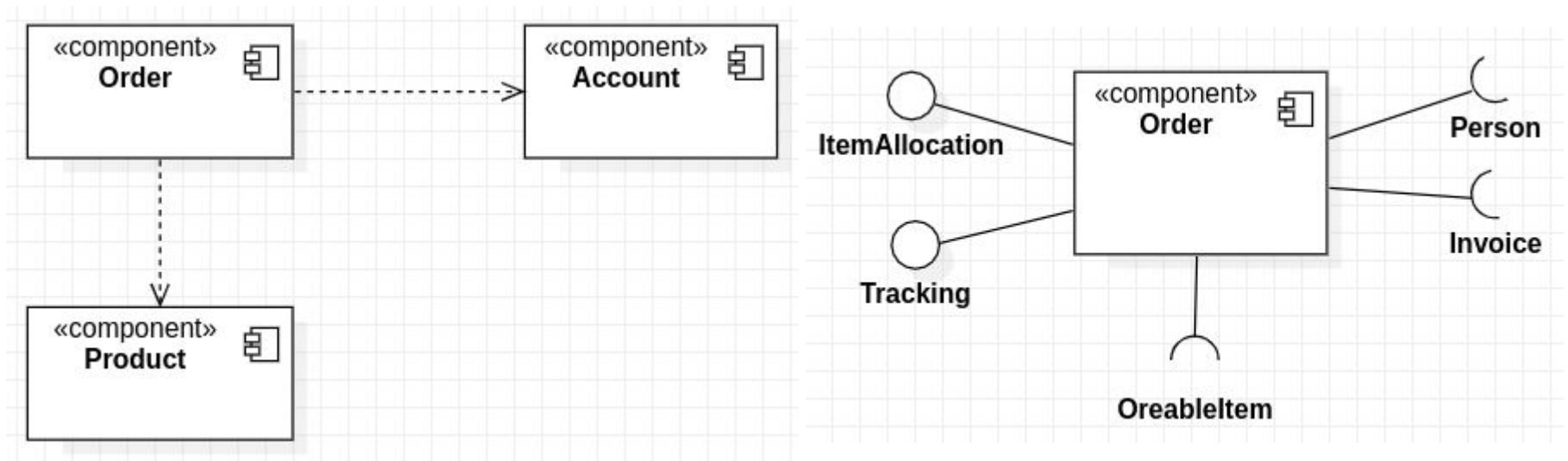


Giao diện được yêu cầu

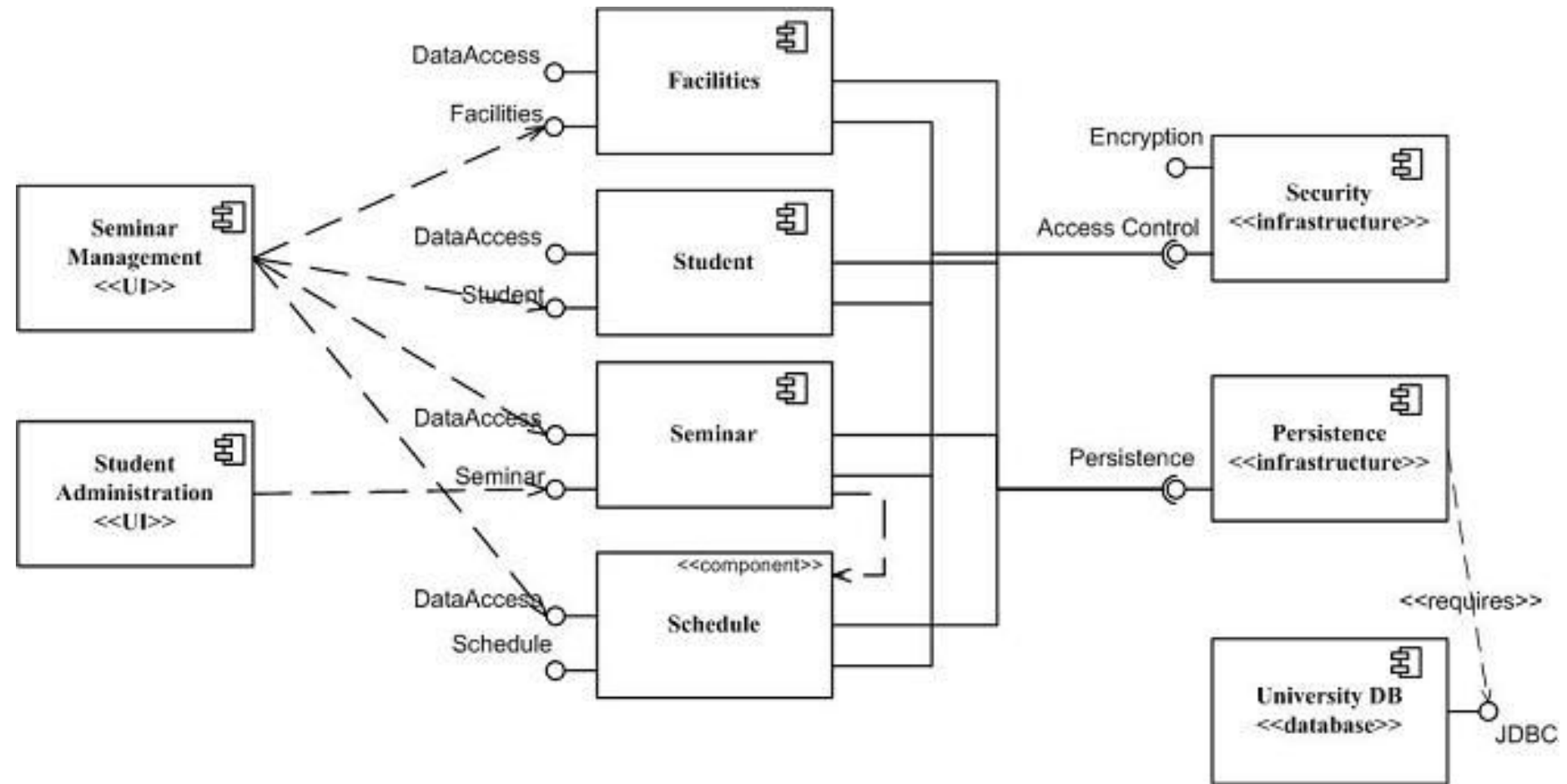
- Quan hệ phụ thuộc được biểu diễn bằng mũi tên nét đứt



Biểu diễn thành phần và các giao diện



Biểu đồ thành phần trong UML 2.x



UML 2.0

Biểu đồ triển khai

- Biểu diễn kiến trúc vật lý của phần cứng và phần mềm của hệ thống được triển khai
- Các nút
 - Thường là các thiết bị tính toán: Máy vi tính, điện thoại thông minh, v.v..
 - Môi trường thực thi: Linux, macOS, Windows, ...
 - Chứa các thành phần phần mềm
- Biểu diễn mối quan hệ vật lý giữa phần cứng và phần mềm trong hệ thống được triển khai
 - Mô tả cách hệ thống tương tác với môi trường bên ngoài

Biểu đồ triển khai: Artifact

Artifact biểu diễn một thành phần thực được triển khai trong các nút.



Một số artifact tiêu chuẩn:

- Tập - `<<file>>` một tập cụ thể trong hệ thống tập
- Tài liệu - `<<document>>` một tài liệu, có thể là mã nguồn hoặc tập thực thi
- Thư viện - `<<library>>` một thư viện tĩnh hoặc động
- Tập thực thi - `<<executable>>` một chương trình có thể được thực thi trên máy tính.

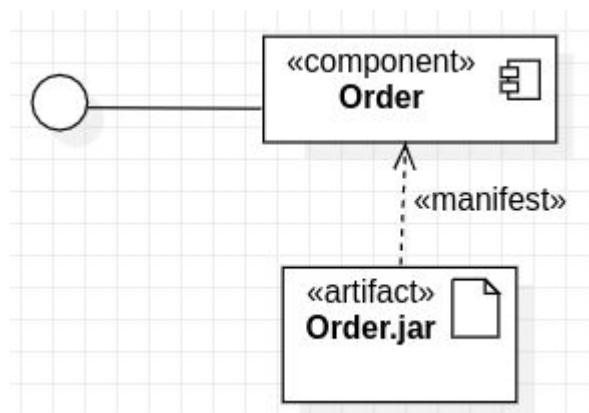
Biểu đồ triển khai: Hệ ký hiệu



- Nút là tài nguyên tính toán:
 - Thiết bị
 - Môi trường thực thi
 - Có thể được kết nối để biểu diễn các kênh trao đổi thông tin theo hình trạng mạng
- Thành phẩm / Artifact:
 - Thành phần cụ thể của hệ thống
 - Các phân kiểu:
 - <<mã nguồn>> / <<source>>
 - <<tệp thực thi>> / <<executable>>
 - <<jar>>
- Đường truyền được biểu diễn như liên kết
- Triển khai thành phẩm trên nút

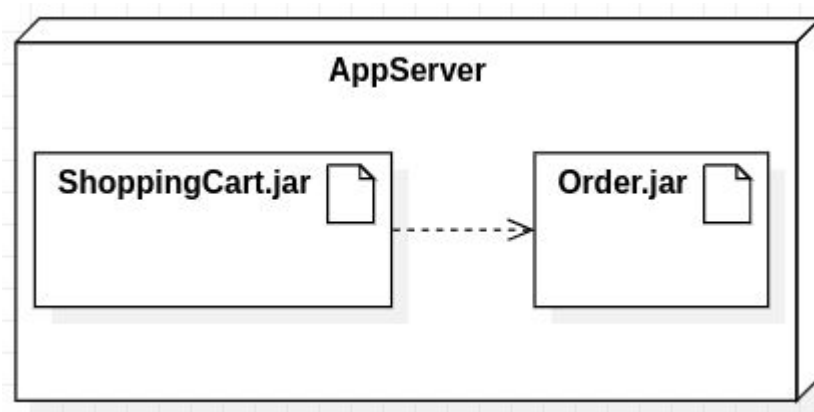
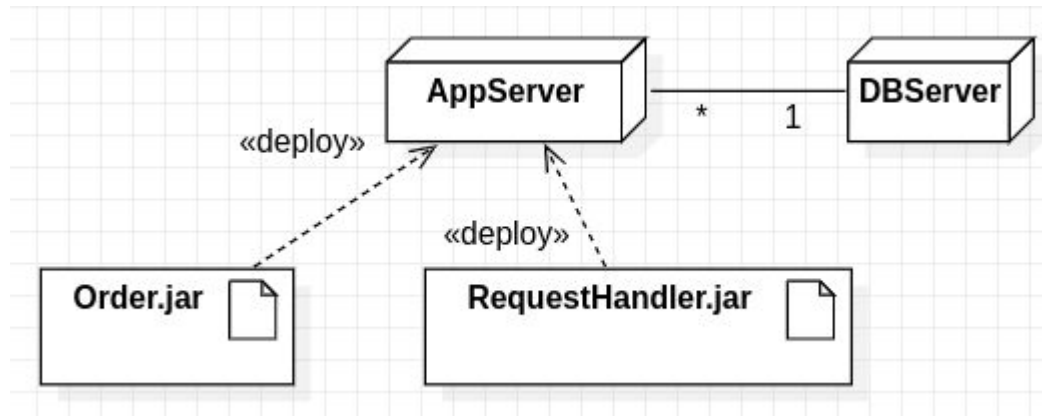
Quan hệ manifest

- Biểu diễn triển khai vật lý của 1 hoặc nhiều phần tử của mô hình dưới dạng thành phẩm.
- Thường được sử dụng để biểu diễn triển khai của thành phần dưới dạng thành phẩm
 - Mỗi thành phần có thể được triển khai như 1 thành phẩm
 - Mỗi thành phẩm có thể bao gồm nhiều thành phần



- Ngoài ra có thể sử dụng với bất kỳ phần tử nào khác nếu có thể là thành phần của gói / PackageableElement

Biểu diễn cách triển khai thành phẩm



Nội dung

- Cân bằng các mô hình phân tích
- Lựa chọn chiến lược thiết kế
- Thiết kế lớp và phương thức
- Phân mảnh và biểu đồ gói
- Biểu đồ thành phần và biểu đồ triển khai
- Các thành phần kiến trúc Hệ thống
- Mẫu kiến trúc

Các thành phần kiến trúc

- Kiến trúc công nghệ
 - Máy tính, mạng máy tính và hình trạng mạng, và phần mềm hệ thống
 - Hình thành hạ tầng hỗ trợ các phần mềm và dịch vụ được công ty cung cấp
- Kiến trúc ứng dụng
 - Các chương trình phần mềm / các hệ thống thông tin công ty cần để hỗ trợ các hoạt động của nó.
 - Được triển khai trên kiến trúc công nghệ bằng cách cài đặt các thành phần phần mềm trên các thiết bị phần cứng và kết nối chúng bằng các giao thức và đường truyền mạng.

Mục đích thiết kế kiến trúc tổng quan là xác định cách bố trí các thành phần phần mềm trong các phần cứng.

Các thành phần kiến trúc cơ bản

- Thành phần phần cứng
 - Máy khách / máy tính cá nhân
 - Máy chủ
 - Hạ tầng mạng
- Thành phần phần mềm
 - Thành phần lưu trữ dữ liệu
 - Thành phần truy cập dữ liệu
 - Thành phần ứng dụng
 - Thành phần trình diễn

Môi trường thực thi

- Máy chủ - Quản lý các tài nguyên dùng chung và cho phép người dùng và các máy tính khác truy cập đến những tài nguyên đó.
- Máy khách / Máy tính cá nhân
 - Máy bàn, máy tính xách tay, máy tính bảng, điện thoại thông minh, v.v..

Điện toán đám mây

- Cho thuê hạ tầng phần cứng
 - Máy chủ ở trên "Đám mây"
 - Máy khách là thiết bị của người dùng
- "Đám mây"
 - Trung tâm dữ liệu, nội hoặc ngoại; hoặc
 - Dịch vụ được cung cấp bởi nhà cung cấp
 - Tích hợp nhiều công nghệ:
 - Ảo hóa
 - Kiến trúc hướng dịch vụ
 - Mạng lưới tính toán / Grid computing

Hạ tầng mạng

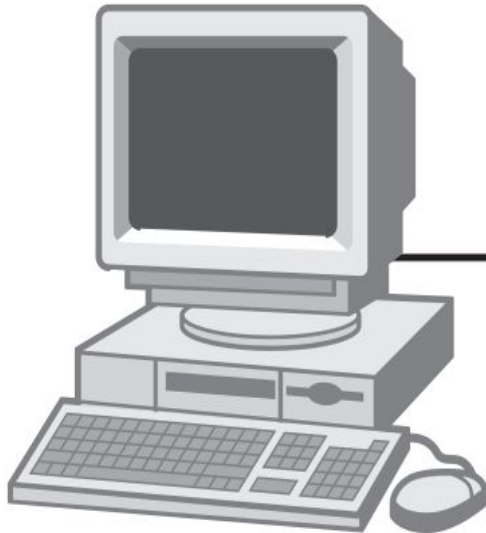
- Có thể bao gồm phần cứng, phần mềm, phương tiện truyền tín hiệu
- Hạ tầng Internet
 - Các đường truyền băng thông lớn và máy tính tốc độ cao
 - Được sở hữu bởi các chính phủ và các công ty viễn thông
- Mạng địa phương (LAN)
 - Mạng nhỏ cho 1 hệ thống thông tin
- Mạng toàn cầu (WWW)
 - Tất cả các tài nguyên được kết nối và truy cập qua Internet

Các giao thức

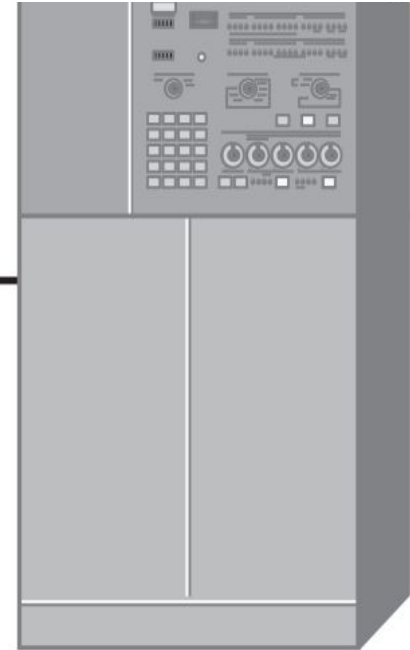
- Giao thức:
 - Một tập ngôn ngữ và quy tắc để đảm bảo giao tiếp và trao đổi dữ liệu giữa phần cứng và phần mềm
- Các giao thức mạng:
 - Mạng riêng tư ảo (VPN)
 - Tạo mạng riêng tư trên Internet bằng cách sử dụng các công nghệ bảo mật và mã hóa

Kiến trúc dựa trên máy chủ

Máy khách/Cổng
vận hành



Máy chủ



Trình diễn
Ứng dụng
Truy cập dữ liệu
Lưu trữ dữ liệu

Phần mềm như 1 dịch vụ (SaaS)

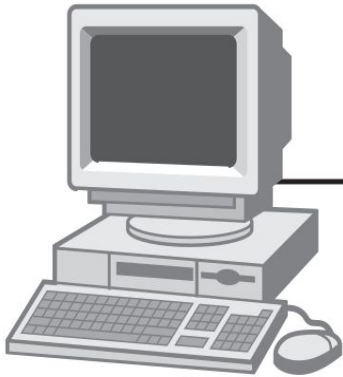
- Thường được thực hiện với hạ tầng điện toán đám mây
- Phần mềm không được cài trên thiết bị của người dùng
- Các dịch vụ ứng dụng được truy cập từ xa (thường qua trình duyệt)
- Dữ liệu người dùng được cô lập và được lưu trên máy chủ dùng chung

Dịch vụ Web

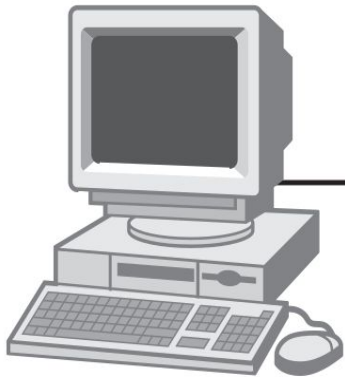
- Chức năng phần mềm được thực hiện bằng các quy chuẩn Web
 - Truy cập qua URL
 - Dữ liệu đầu vào được gửi qua URL
 - Được thực thi từ xa
 - Dữ liệu được trả về trong trang Web

Kiến trúc dựa trên máy khách

Các máy khách

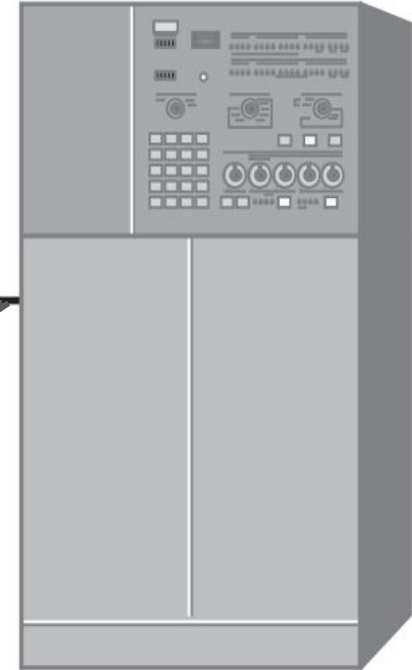


Trình diễn, ứng dụng
Truy cập dữ liệu



Trình diễn, ứng dụng
Truy cập dữ liệu

Máy chủ



Lưu trữ dữ liệu

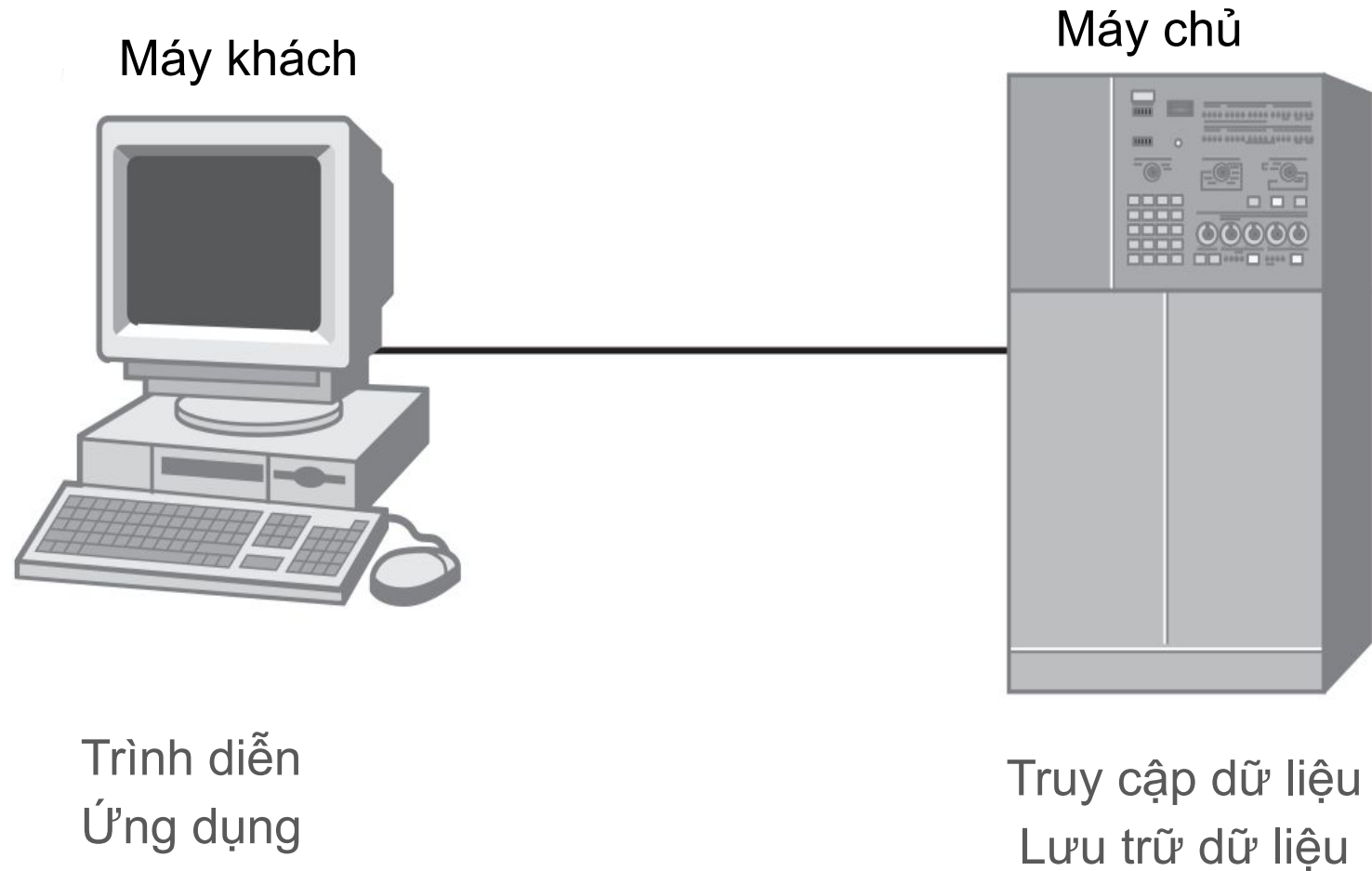
Kiến trúc máy khách-máy chủ

- Hệ thống được thiết kế với 1 phần ở máy chủ và 1 phần ở máy khách, cố gắng phân chia hợp lý khối lượng tính toán
- Kiến trúc phổ biến nhất trong các hệ thống hiện đại
- Khối lượng tính toán của máy khách dao động
 - Máy khách mỏng chỉ thực hiện lô-gic trình diễn
 - Máy khách dày thực hiện lô-gic trình diễn và lô-gic ứng dụng
- Khả năng mở rộng cao với chi phí từng phần
- Phức tạp hơn phải phát triển ứng dụng cho máy khách, máy chủ và đảm bảo tương tác phân tán.

Các dãy máy khách-máy chủ

- Các dãy máy khách-máy chủ được định nghĩa dựa trên cách phân mảnh lô-gic:
 - 2-dãy: 1 máy chủ chịu trách nhiệm lưu trữ và truy cập dữ liệu; máy khách có trách nhiệm xử lý lô-gic ứng dụng và lô-gic trình diễn
 - 3-dãy: Lô-gic truy cập và lưu trữ dữ liệu trên 1 máy chủ; lô-gic ứng dụng trên 1 máy chủ khác; máy khách có trách nhiệm xử lý lô-gic trình diễn.
 - n-dãy: Lô-gic ứng dụng được phân chia trên nhiều máy chủ, lô-gic dữ liệu trên 1 máy khác chủ khác
 - Phổ biến trong các ứng dụng thương mại điện tử
 - Cân bằng tải tốt hơn
 - Khả năng mở rộng cao hơn hệ thống 2 hoặc 3 dãy
 - Nhu cầu sử dụng mạng cao hơn

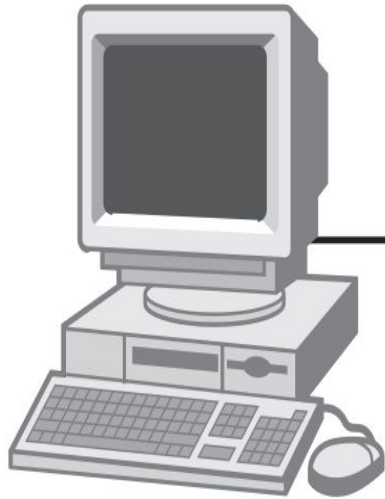
Kiến trúc máy khách-máy chủ



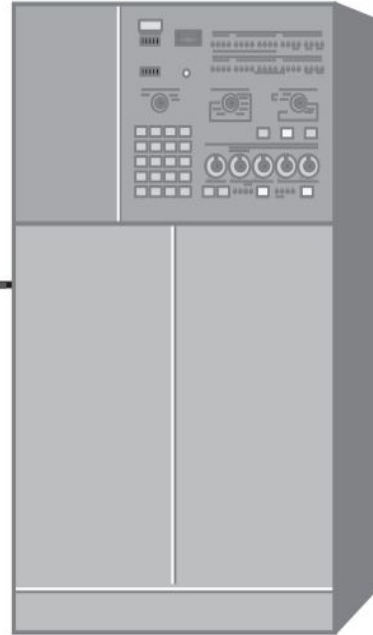
Kiến trúc 2 dây (sử dụng 2 nhóm máy tính)

Kiến trúc máy khách-máy chủ: 3 dãy

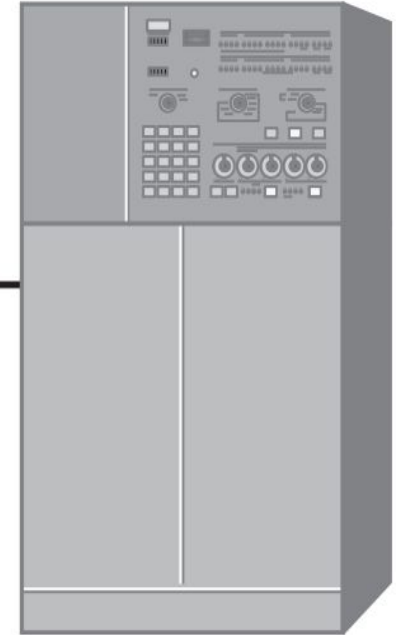
Máy khách



Máy chủ ứng dụng



Máy chủ CSDL

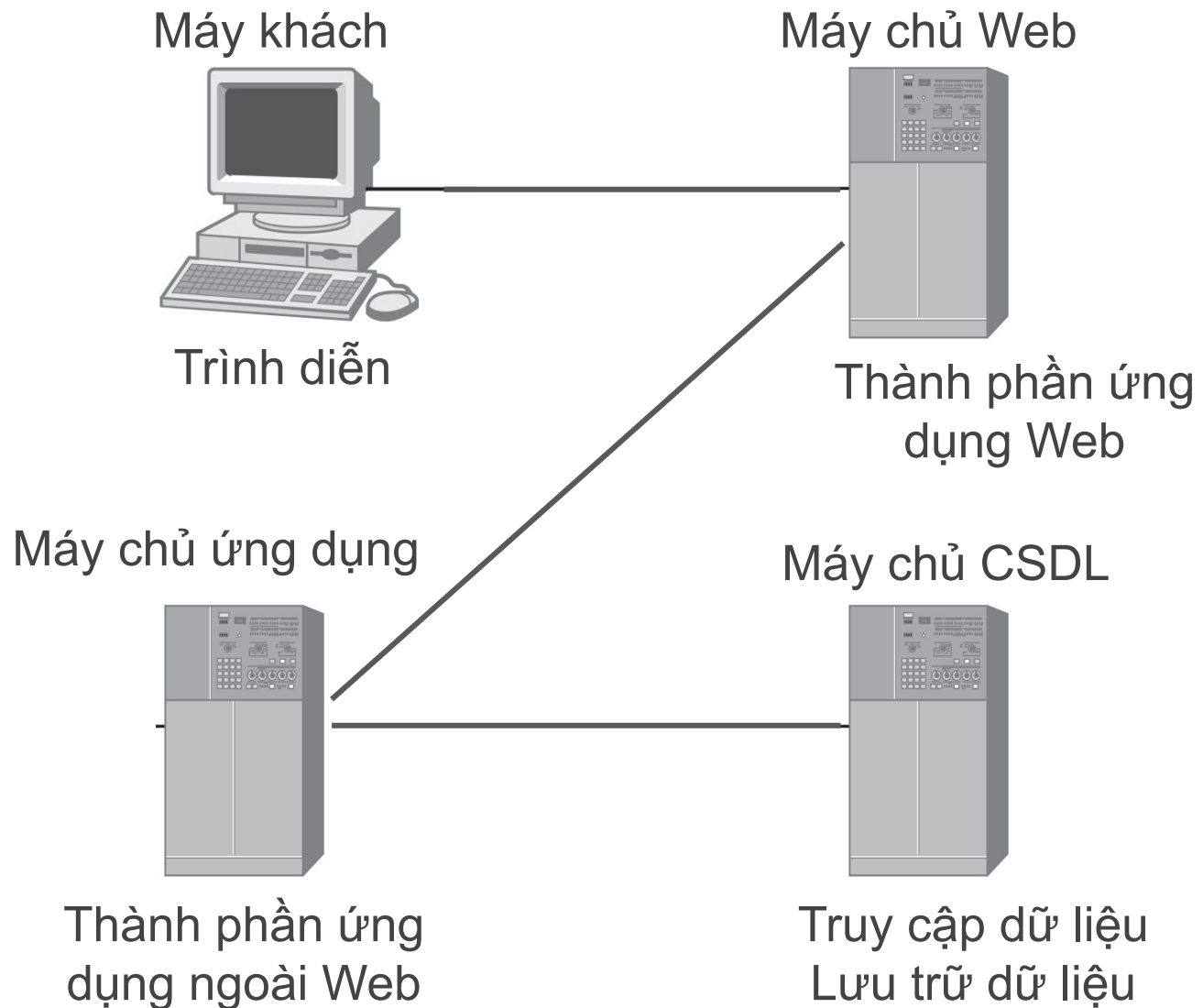


Trình diễn

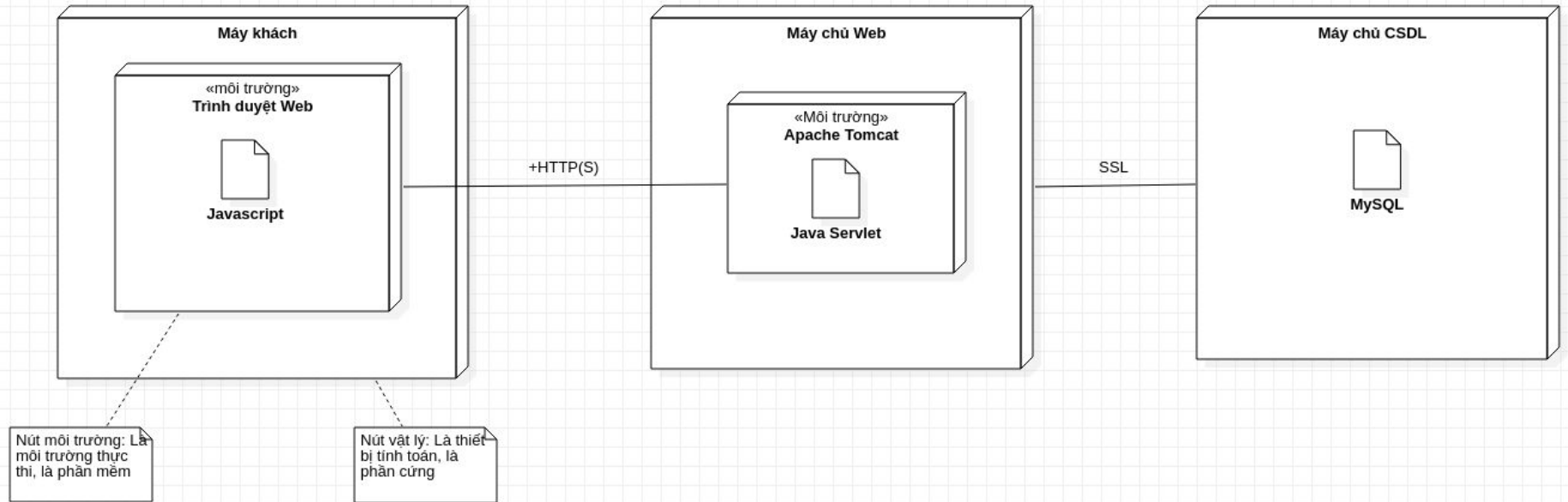
Ứng dụng

Truy cập dữ liệu
Lưu trữ dữ liệu

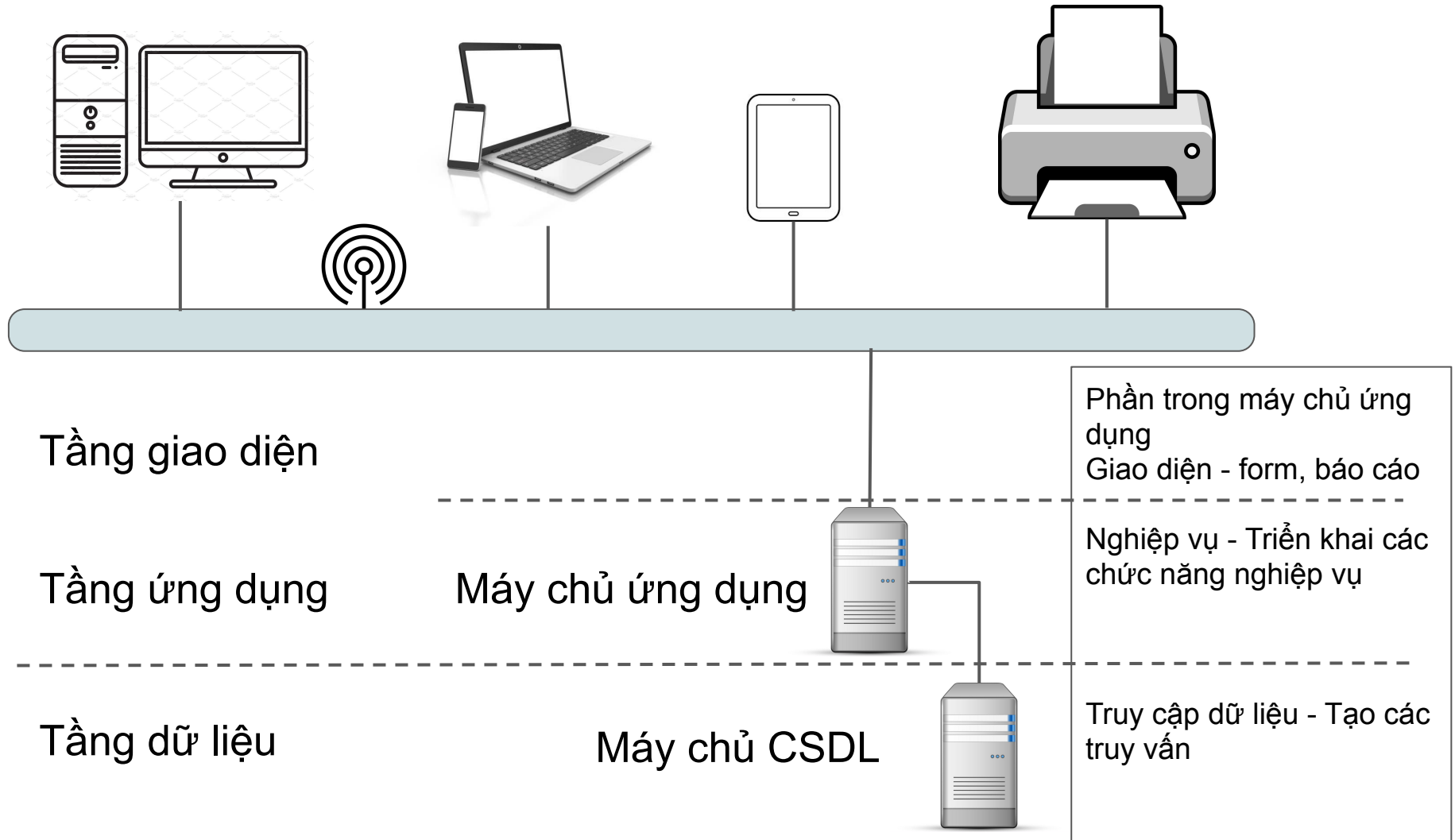
Kiến trúc máy khách-máy chủ: 4-dãy



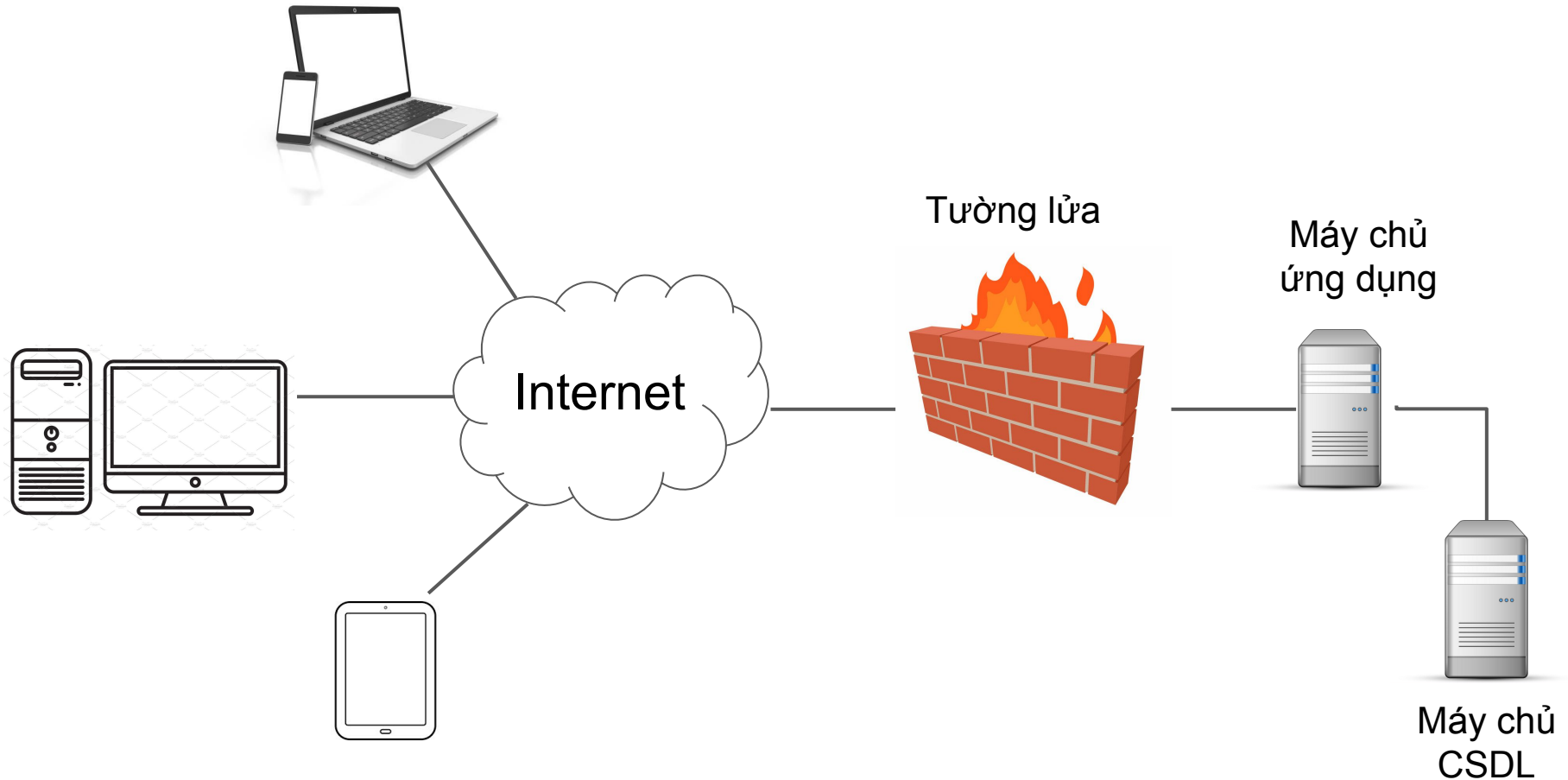
Triển khai hệ thống



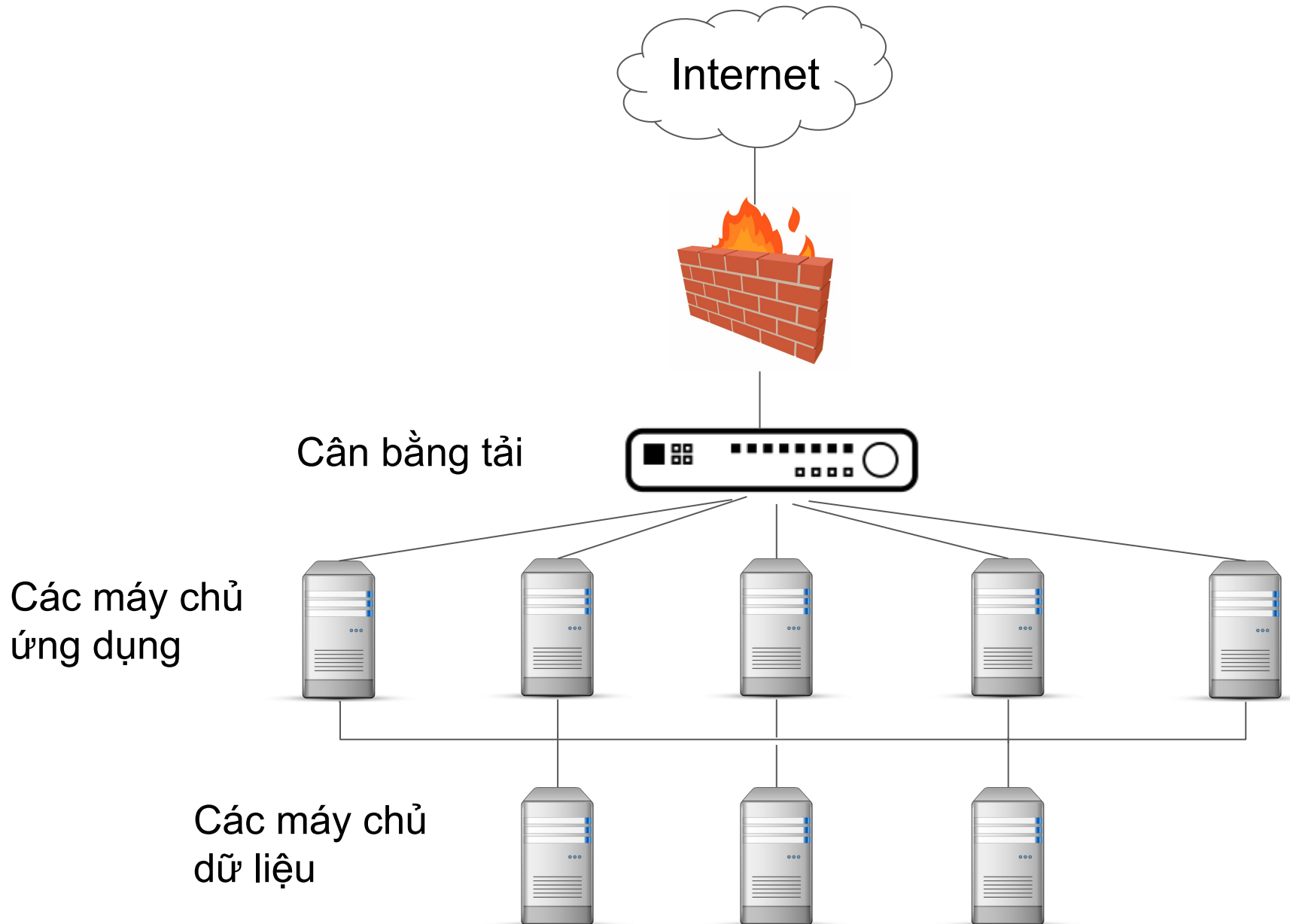
Kiến trúc 3 tầng: Triển khai nội bộ




Kiến trúc 3 tầng: Triển khai trên internet



Kiến trúc 3 tầng: Mở rộng hệ thống



Nội dung

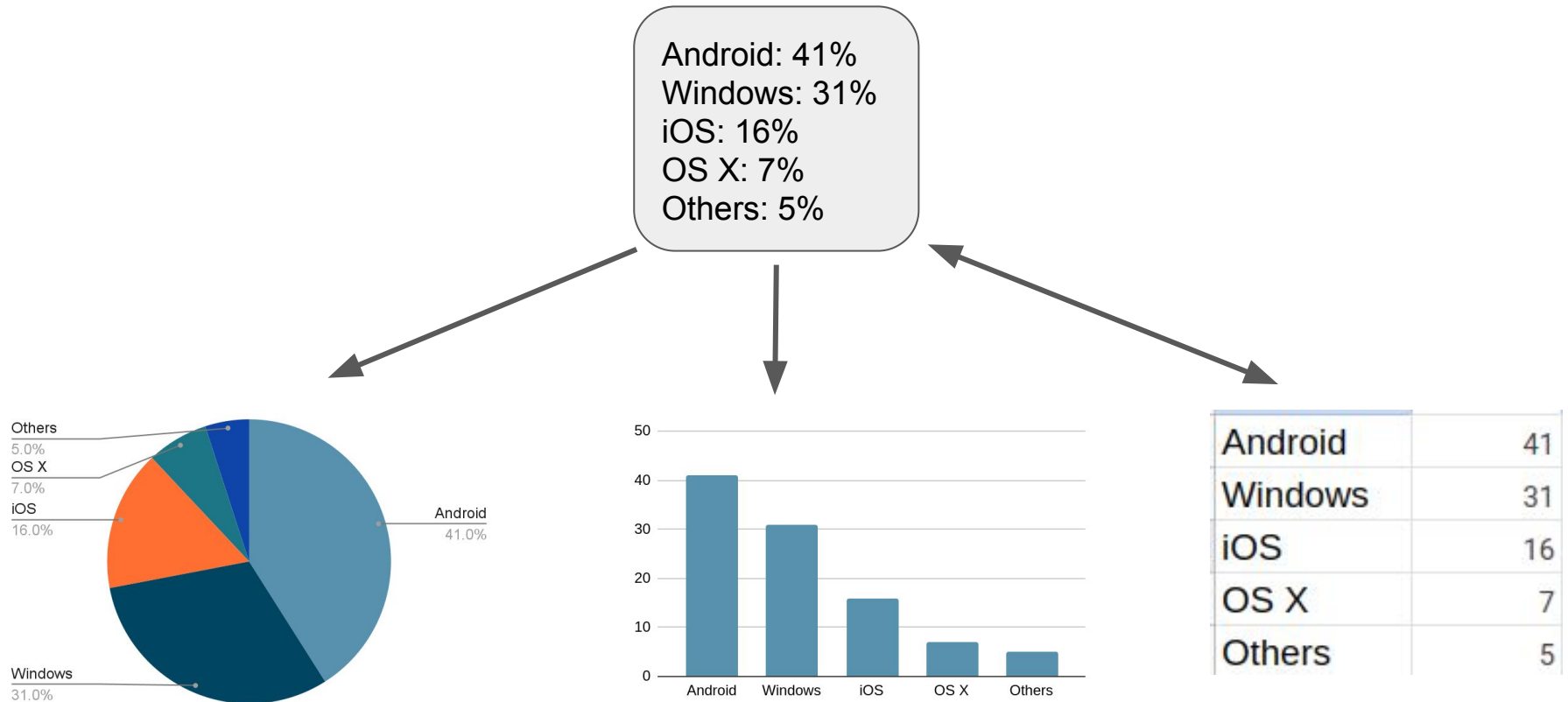
- Cân bằng các mô hình phân tích
 - Lựa chọn chiến lược thiết kế
 - Thiết kế lớp và phương thức
 - Phân mảnh và biểu đồ gói
 - Biểu đồ thành phần và biểu đồ triển khai
 - Các thành phần kiến trúc Hệ thống
 - Mẫu kiến trúc
- 

Kiến trúc MVC

- Mẫu kiến trúc Model-View-Controller / MVC phân mảnh 1 chương trình tương tác thành 3 thành phần.
 - Mô hình/Model chứa dữ liệu và triển khai chức năng lõi
 - Khung nhìn/View hiển thị thông tin cho người dùng
 - Điều khiển/Controller tiếp nhận dữ liệu vào của người dùng
- Khung nhìn và điều khiển kết hợp lại thành giao diện người dùng.
- Cơ chế lan truyền thay đổi đảm bảo tính nhất quán giữa giao diện người dùng và mô hình.

Trình diễn thông tin

Ví dụ dữ liệu thị phần hệ điều hành

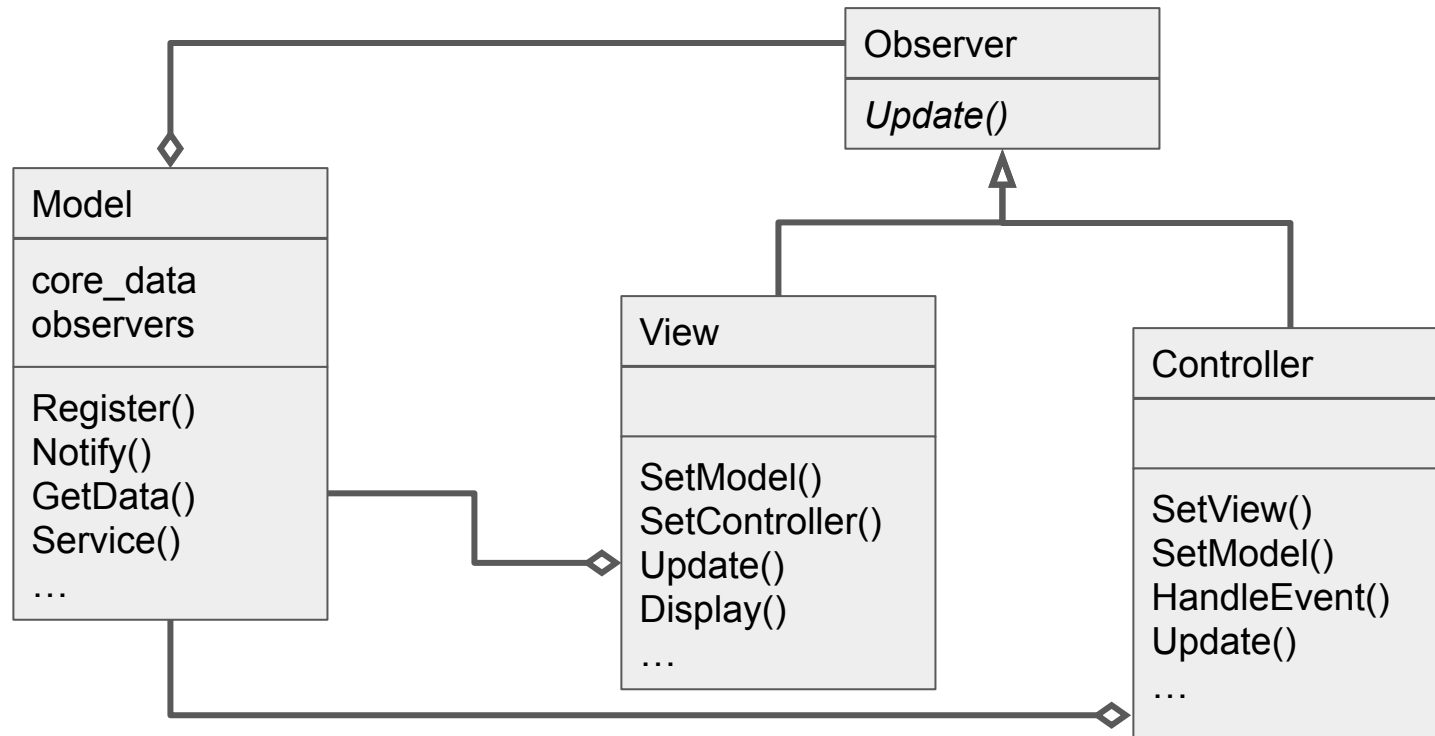


Ứng dụng tương tác với giao diện linh động

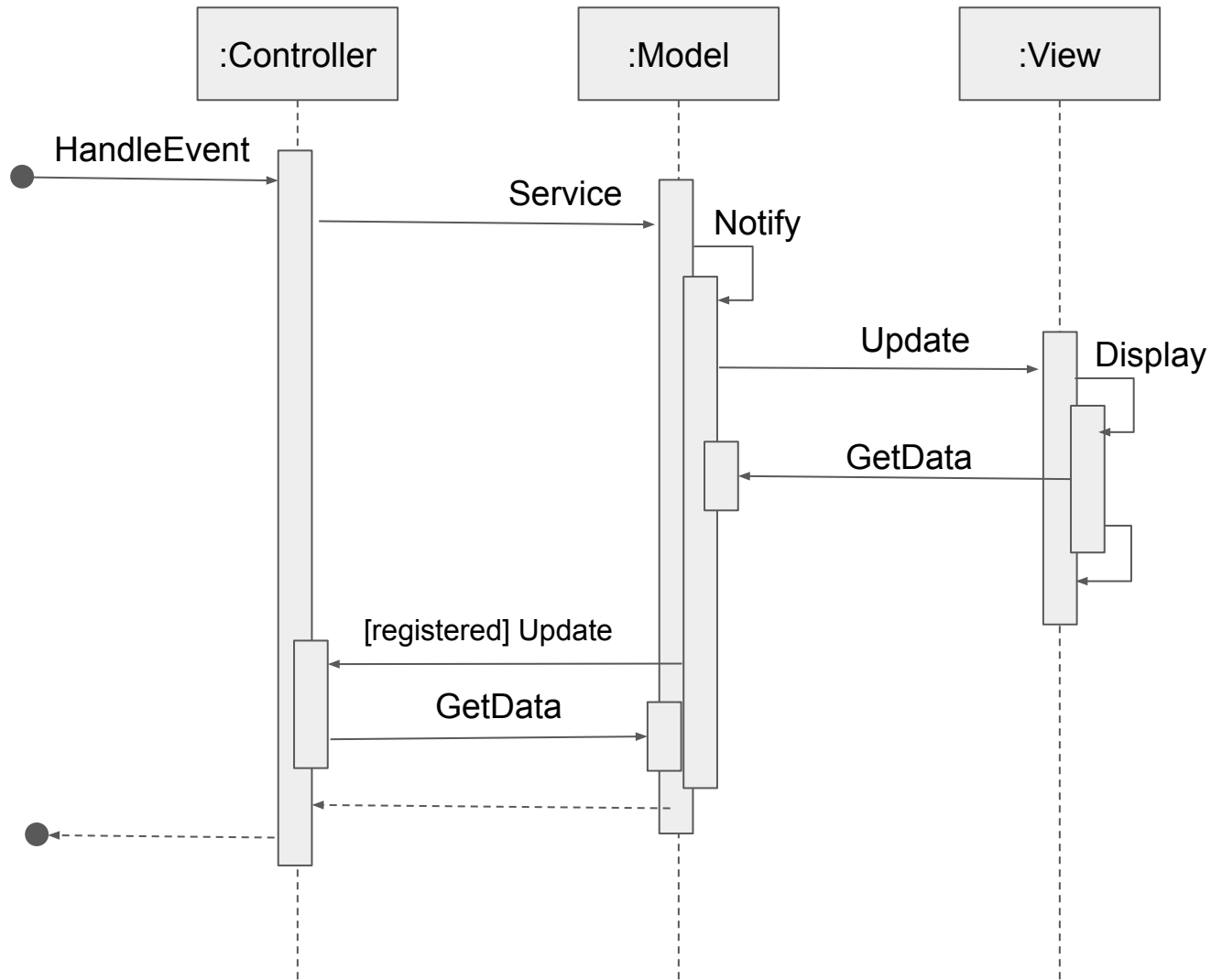
- Giao diện người dùng là thành phần có thể thay đổi vì nhiều nguyên nhân khác nhau:
 - Khi mở rộng chức năng của chương trình,
 - Các yêu cầu khác nhau của người dùng, v.v.
- Phân mảnh chương trình theo MVC cho phép giữ nguyên lô-gic ứng dụng (thành phần Mô hình) trong khi thay đổi thành phần giao diện hoặc sử dụng đồng thời nhiều giao diện.

Triển khai MVC với cơ chế lan truyền thay đổi

Trong triển khai hướng đối tượng của MVC, các thành phần được triển khai bằng các lớp riêng. Ví dụ 1 triển khai:

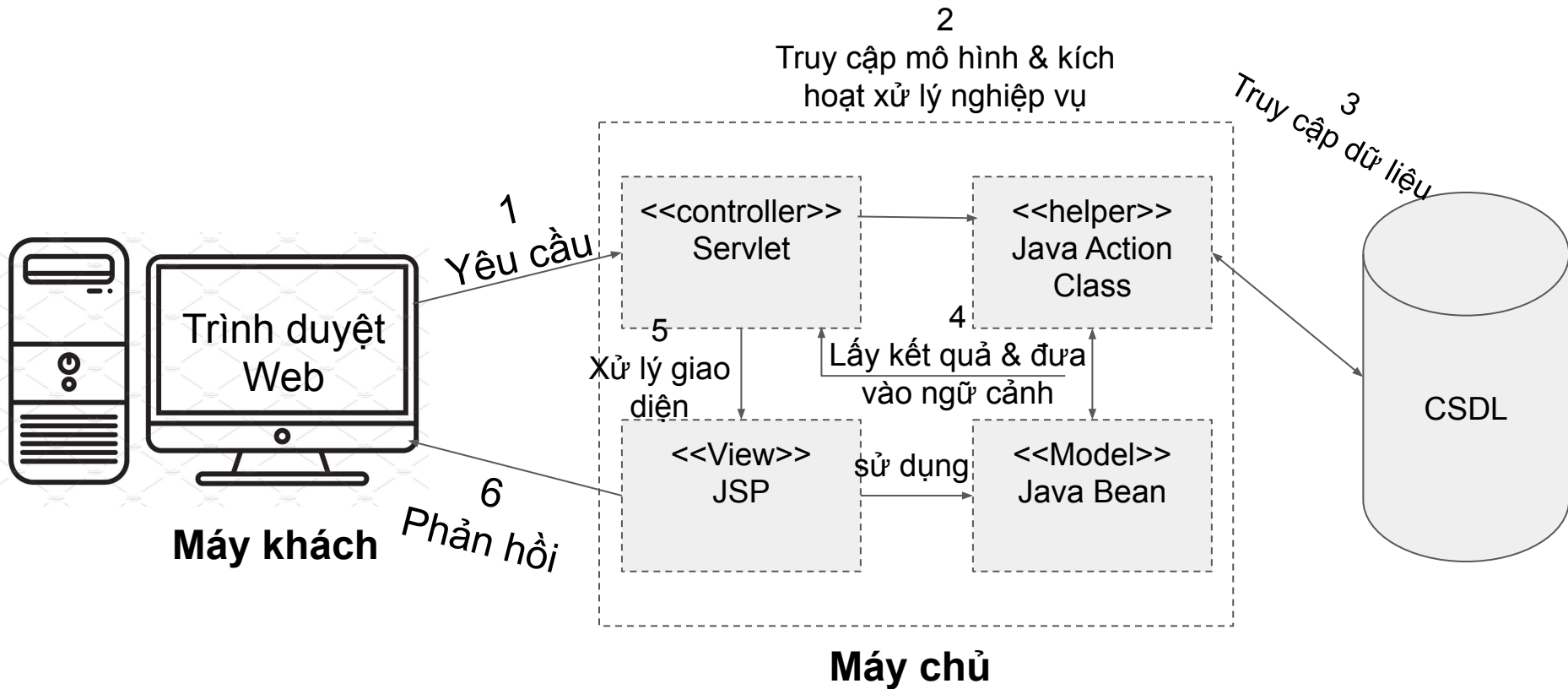


Kịch bản truyền thông điệp thường



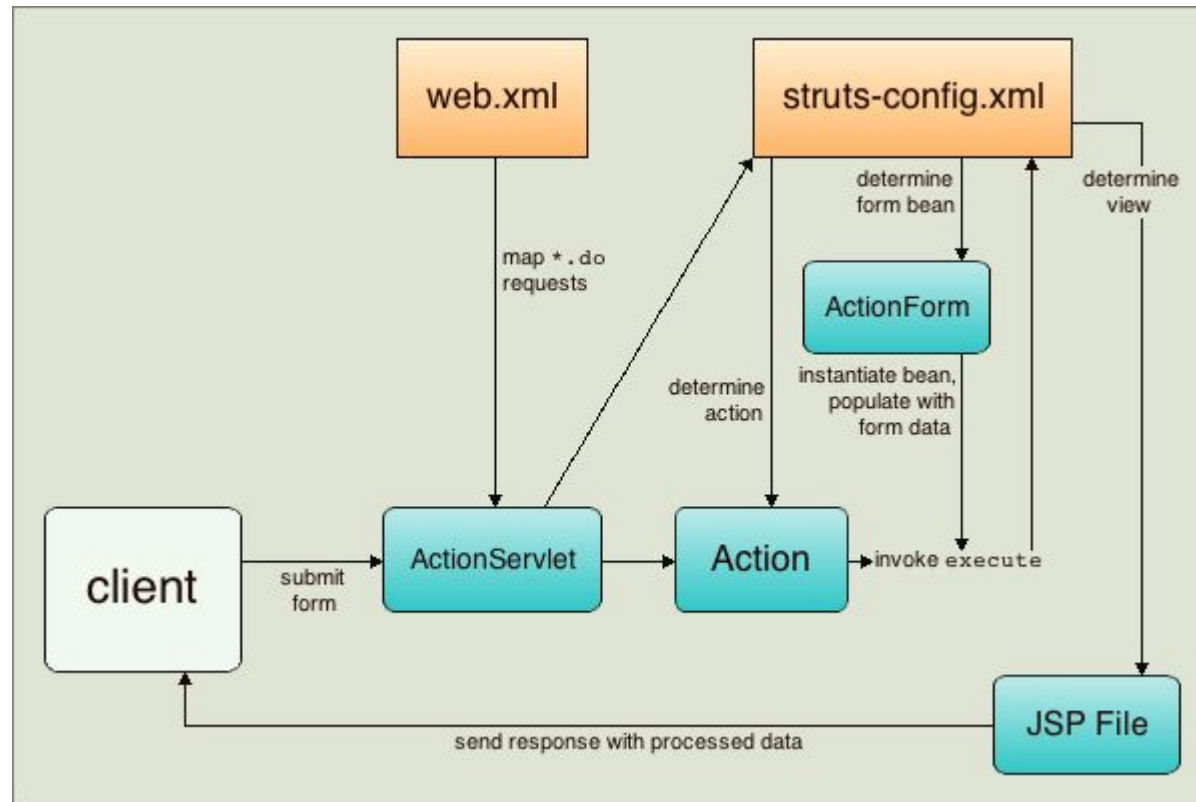
Nền tảng Struts

Kiến trúc JSP Model 2: Tách biệt lô-gic trình diễn và lô-gic ứng dụng tương tự như trong MVC.



** JSP Model 2 không định nghĩa định dạng cụ thể của Model*

Kiến trúc ứng dụng Web: Nền tảng Struts



[apache.org]

JSP Model 2 và MVC

