HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Khoa Điện tử Viễn thông

Nội dung 4: Kiến trúc phần mềm

Giảng viên: TS. Lâm Đức Dương

Nội dung 4: Kiến trúc phần mềm

1. Giới thiệu về Kiến trúc Phần mềm
2. Thiết kế Kiến trúc
3. Góc nhìn Kiến trúc (Architectural Views)
4. Các Mẫu Kiến trúc
5. Các Mẫu Kiến trúc cho Hệ thống Phân tán
6. Tầm quan trọng của thiết kế PM

- Giao thức bối cảnh mô hình (Model Context Protocol)

. Tầm quan trọng của Kiến trúc Phần mềm

- Kiến trúc phần mềm ảnh hưởng đến các thuộc tính quan trọng của hệ thống như hiệu suất, tính mạnh mẽ, khả năng phân tán và khả năng bảo trì.
- Nó có ảnh hưởng chi phối đến các đặc tính phi chức năng của hệ thống. Các yêu cầu phi chức năng như hiệu suất và bảo mật có tác động đáng kể nhất đến kiến trúc hệ thống.
- Nó đóng vai trò là một cơ chế giao tiếp giữa các bên liên quan, cung cấp một cái nhìn trừu tượng mà không có các chi tiết không cần thiết.
- Nó hoạt động như tài liệu về kiến trúc đã thiết kế, giúp hệ thống dễ hiểu và phát triển hơn.
- Nó hỗ trợ tái sử dụng phần mềm quy mô lớn bằng cách cung cấp một mô tả ngắn gọn về tổ chức hệ thống và khả năng tương tác giữa các thành phần. Các hệ thống có yêu cầu tương tự thường có thể tái sử dụng cùng một kiến trúc.
- Trong các quy trình Agile, việc tập trung sớm vào thiết kế kiến trúc hệ thống tổng thể thường được chấp nhận, vì việc tái cấu trúc kiến trúc sau này có thể tốn kém.

Quá trình Thiết kế Kiến trúc

- Thiết kế kiến trúc là một quá trình sáng tạo để thiết kế tổ chức hệ thống nhằm đáp ứng các yêu cầu chức năng và phi chức năng.
- Không có quy trình công thức nào; nó phụ thuộc vào loại hệ thống, kinh nghiệm của kiến trúc sư và các yêu cầu cụ thể.
- Hãy nghĩ về nó như một chuỗi các quyết định hơn là một chuỗi các hoạt động.

. Quá trình Thiết kế Kiến trúc

Các khía cạnh chính

- **Loại Ứng dụng**: Đây có phải là hệ thống nhúng, hệ thống dựa trên web, hệ thống thời gian thực, v.v.?.
- **Phân tán Hệ thống**: Nó sẽ là một hệ thống tập trung hay phân tán trên nhiều máy tính?.
- **Kiểu/Mẫu Kiến trúc**: Những cách tổ chức phần mềm đã được kiểm chứng nào sẽ được sử dụng?.
- **Tài liệu hóa và Đánh giá**: Kiến trúc sẽ được tài liệu hóa và đánh giá như thế nào?.
- **Kiến trúc Ứng dụng Chung**: Có mẫu nào có thể được điều chỉnh không?.
- **Phân bổ Phần cứng**: Hệ thống sẽ được phân tán trên phần cứng như thế nào?.

Mức Độ Trừu Tượng

- **Kiến trúc ở Quy mô Nhỏ** (Architecture in the Small):
  - Liên quan đến kiến trúc của các chương trình riêng lẻ.
  - Tập trung vào cách một chương trình duy nhất được phân rã thành các thành phần.

- **Kiến trúc ở Quy mô Lớn** (Architecture in the Large):
  - Giải quyết kiến trúc của các hệ thống doanh nghiệp phức tạp.
  - Bao gồm các hệ thống, chương trình và thành phần chương trình khác.
  - Các hệ thống này có thể phân tán trên các máy tính khác nhau, có thể do các tổ chức khác nhau sở hữu và quản lý.

2. Các Góc nhìn Kiến trúc (Architectural Views)

- Tài liệu hóa các Góc nhìn khác nhau:
  - Kiến trúc có thể được tài liệu hóa từ nhiều góc độ hoặc quan điểm khác nhau.
  - Mỗi góc nhìn giải quyết các mối quan tâm và các bên liên quan cụ thể.

2. Các Góc nhìn Kiến trúc (Architectural Views)

- Các Góc nhìn Kiến trúc Phổ biến:
  - **Góc nhìn Khái niệm (Conceptual View)**: Thể hiện tổ chức cấp cao của hệ thống và các thành phần chính cùng mối quan hệ của chúng. Thường được biểu diễn bằng sơ đồ khối.
  - **Góc nhìn Logic (Logical View)**: Thể hiện các trừu tượng chính trong hệ thống, thường sử dụng các mô hình đối tượng.
  - **Góc nhìn Quy trình (Process View)**: Thể hiện hệ thống bao gồm các quy trình tương tác như thế nào trong thời gian chạy, hữu ích cho việc đánh giá hiệu suất và tính khả dụng.
  - **Góc nhìn Phát triển (Development View)**: Thể hiện phần mềm được phân rã để phát triển thành các thành phần được thực hiện bởi các nhóm phát triển riêng lẻ như thế nào. Hữu ích cho các nhà quản lý và lập trình viên.
  - **Góc nhìn Vật lý (Physical View)**: Thể hiện phần cứng của hệ thống và cách các thành phần phần mềm được phân tán trên các bộ xử lý. Hữu ích cho các kỹ sư hệ thống lập kế hoạch triển khai.
- Mô hình 4+1 view là một cách khác để suy nghĩ về các quan điểm khác nhau này.

3 Các Mẫu Kiến trúc

- **Mẫu Kiến trúc là gì?**
  - Các cách tổ chức kiến trúc phần mềm đã được kiểm chứng có thể được tái sử dụng trong thiết kế hệ thống.
  - Chúng cung cấp một kiến trúc cách điệu có thể nhận ra trên các hệ thống khác nhau.
  - Hữu ích để kích thích các cuộc thảo luận và để tài liệu hóa và giải thích kiến trúc.

3 Các Mẫu Kiến trúc: Ví dụ về các Mẫu Chung

- **Kiến trúc Phân lớp (Layered Architecture)**: Tổ chức hệ thống thành các lớp với chức năng liên quan. Các lớp thấp hơn cung cấp dịch vụ cho các lớp trên. Được sử dụng khi xây dựng các tiện ích mới trên các hệ thống hiện có, khi quá trình phát triển được chia cho nhiều nhóm, hoặc khi có yêu cầu về bảo mật đa cấp.

- **Kiến trúc Kho (Repository Architecture)**: Các thành phần tương tác chia sẻ dữ liệu thông qua một kho lưu trữ trung tâm.

- **Kiến trúc Mô hình-Xem-Bộ điều khiển (Model-View-Controller - MVC)**: Tách biệt giao diện người dùng (View), dữ liệu ứng dụng (Model) và xử lý đầu vào của người dùng (Controller). Thường được sử dụng để quản lý tương tác trong các hệ thống dựa trên web.

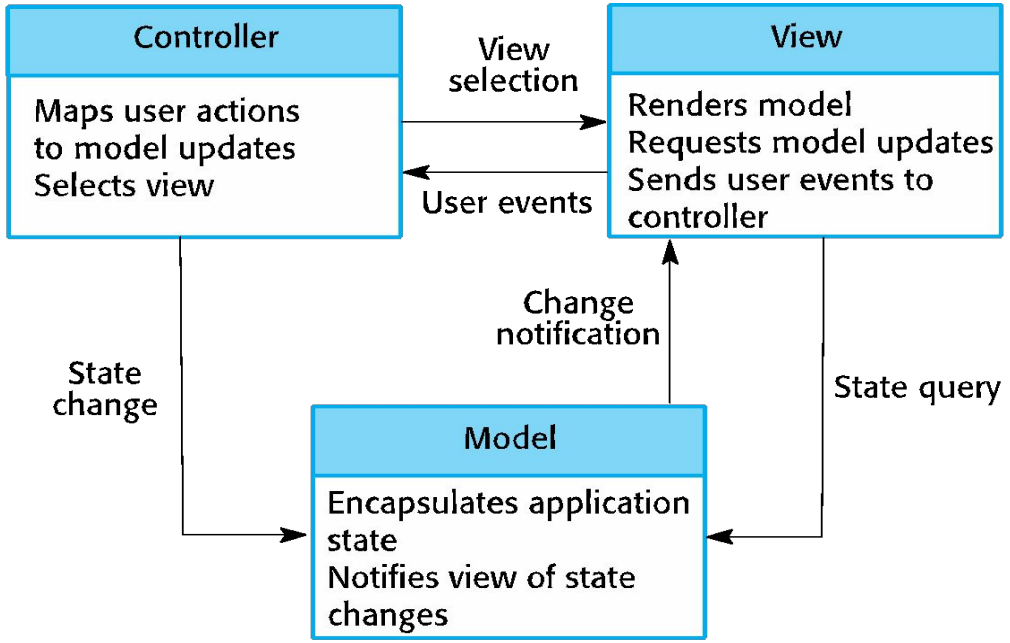
Mẫu Kiến trúc Model-View-Controller (MVC)

Mô tả

Tách biệt phần trình bày (presentation) và tương tác (interaction) khỏi dữ liệu hệ thống (system data). Hệ thống được cấu trúc thành ba thành phần logic tương tác với nhau. Thành phần Model quản lý dữ liệu hệ thống và các hoạt động liên quan đến dữ liệu đó. Thành phần View định nghĩa và quản lý cách dữ liệu được trình bày cho người dùng. Thành phần Controller quản lý tương tác của người dùng (ví dụ: nhấn phím, nhấp chuột, v.v.) và chuyển các tương tác này đến View và Model.

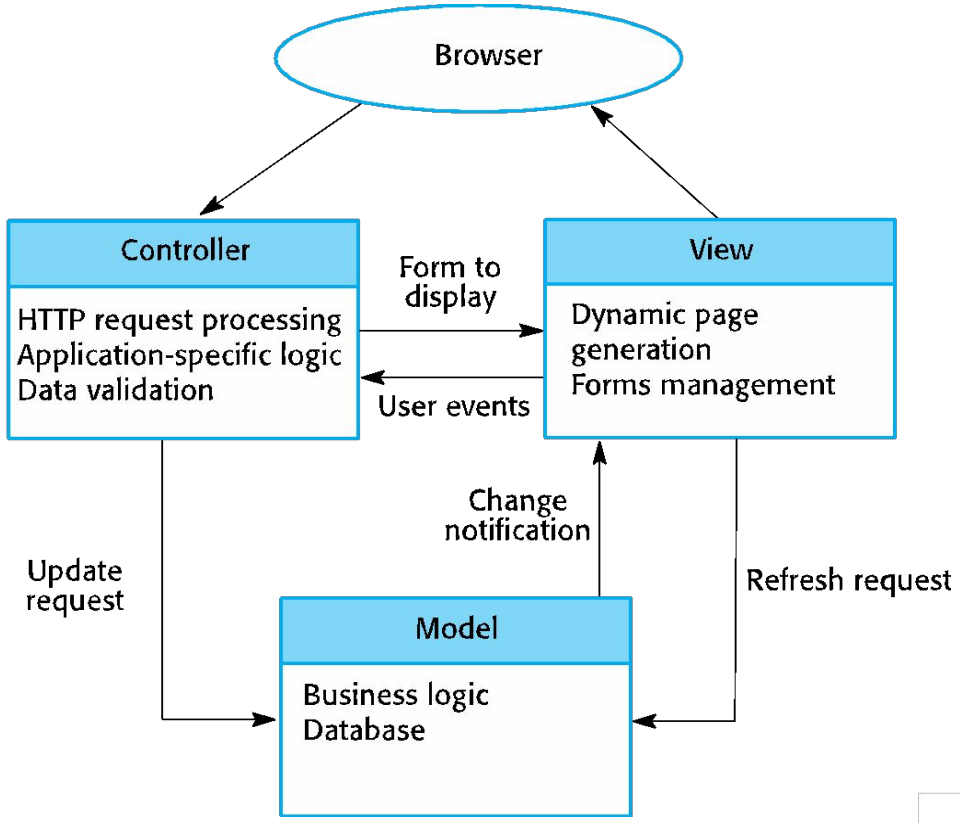
Mẫu Kiến trúc Model-View-Controller (MVC)

Kiến trúc MVC



Ví dụ

Kiến trúc MVC cho Website



Mẫu Kiến trúc lớp Layered

Mô tả: Tổ chức hệ thống thành các lớp với chức năng liên quan được gán cho mỗi lớp. Một lớp cung cấp dịch vụ cho lớp trên nó, vì vậy các lớp cấp thấp nhất đại diện cho các dịch vụ cốt lõi có khả năng được sử dụng trên toàn hệ thống.

Ví dụ: Một mô hình phân lớp của một hệ thống để chia sẻ các tài liệu bản quyền được lưu giữ trong các thư viện khác nhau.

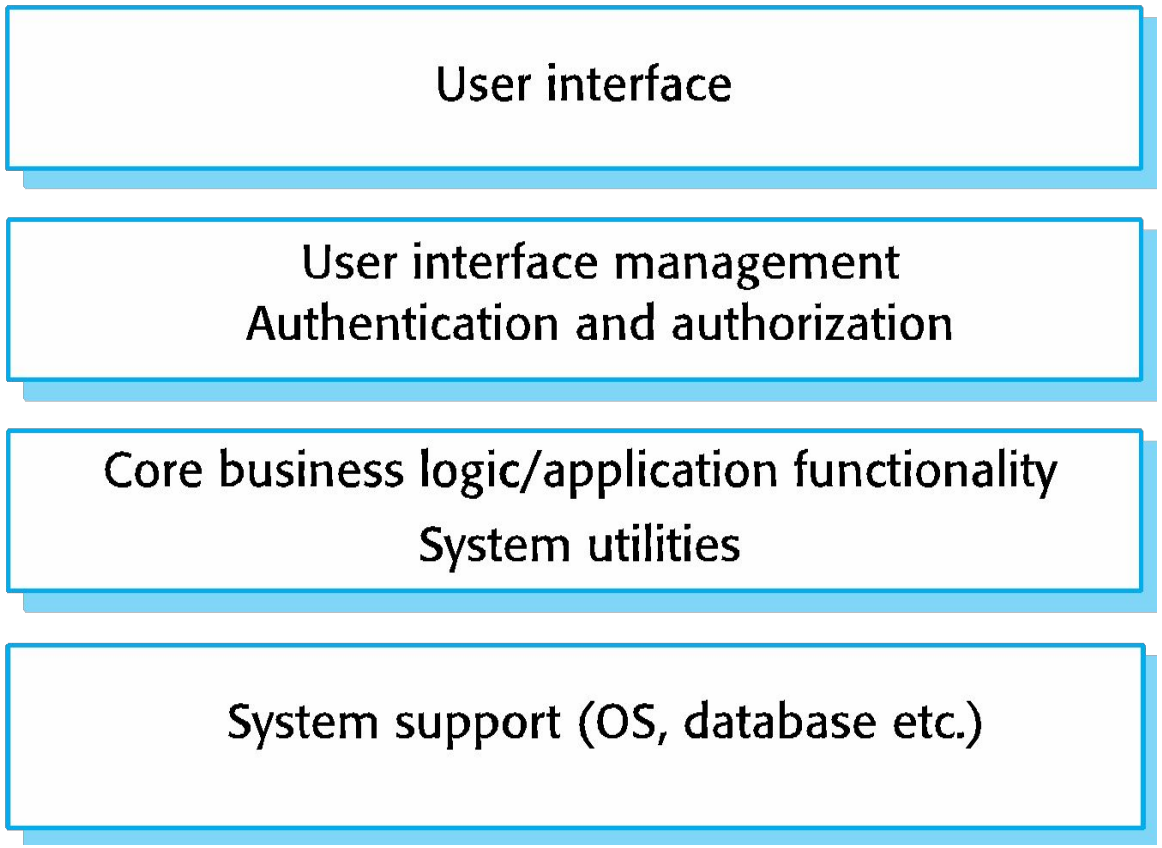
Khi nào sử dụng: Được sử dụng khi xây dựng các tiện ích mới trên các hệ thống hiện có; khi việc phát triển được trải rộng trên nhiều nhóm với mỗi nhóm chịu trách nhiệm về một lớp chức năng; khi có yêu cầu bảo mật đa cấp.

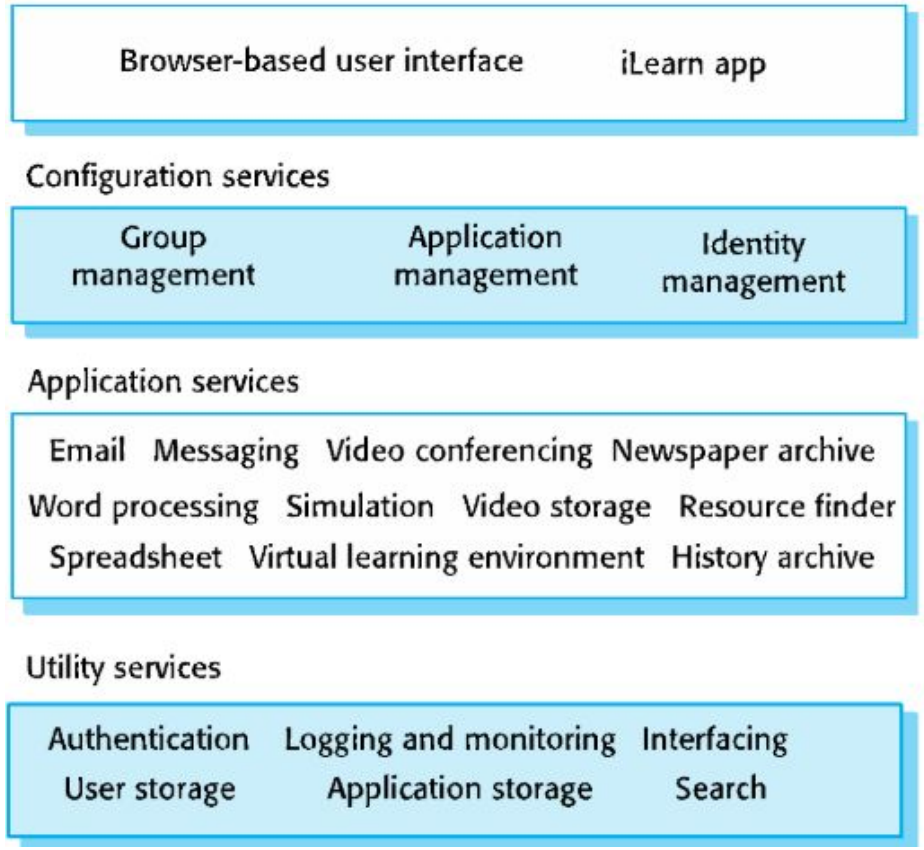
Ưu điểm: Cho phép thay thế toàn bộ các lớp miễn là giao diện được duy trì. Các tiện ích dự phòng (ví dụ: xác thực) có thể được cung cấp trong mỗi lớp để tăng độ tin cậy của hệ thống.

Nhược điểm: Trong thực tế, việc cung cấp sự phân tách rõ ràng giữa các lớp thường khó khăn và một lớp cấp cao có thể phải tương tác trực tiếp với các lớp cấp thấp hơn thay vì thông qua lớp ngay bên dưới nó. Hiệu suất có thể là một vấn đề do nhiều cấp độ diễn giải yêu cầu dịch vụ khi nó được xử lý ở mỗi lớp.

Mẫu Kiến trúc lớp Layered

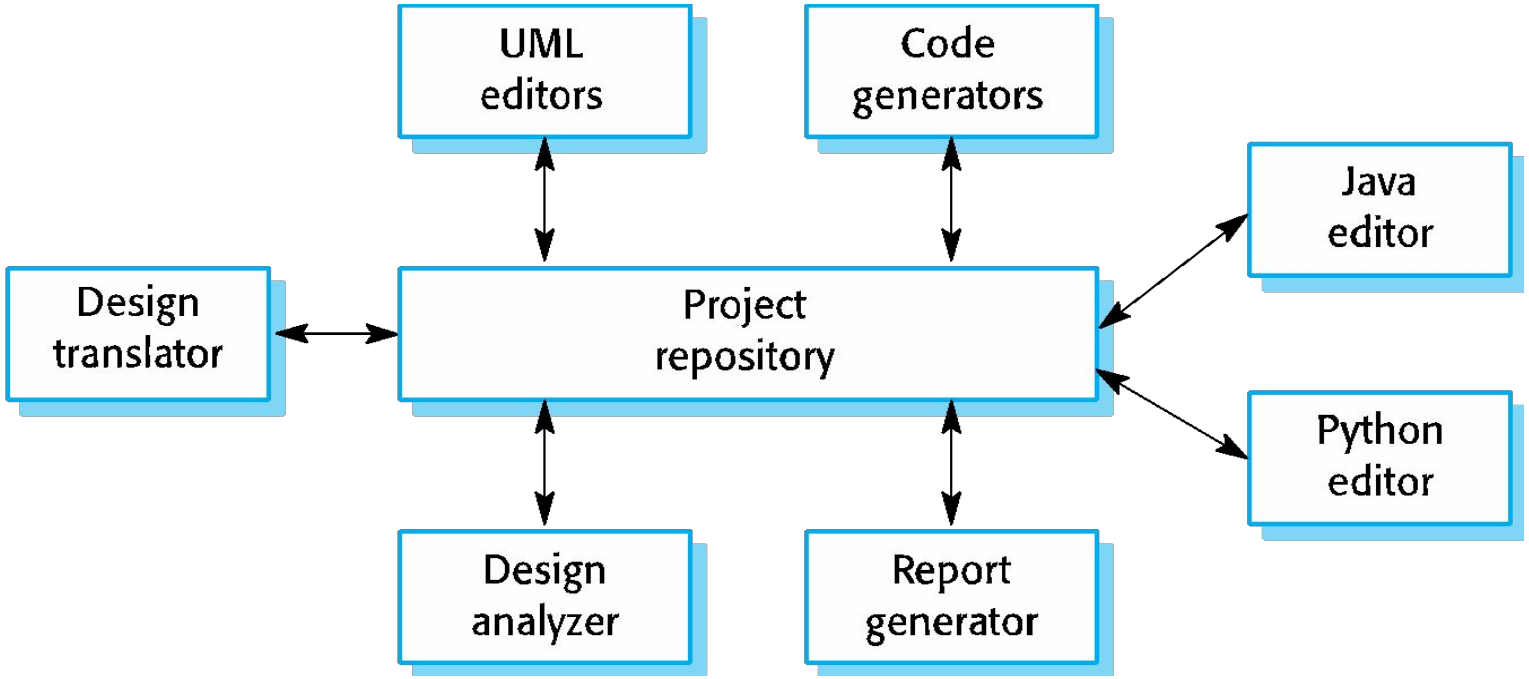
Mô tả





Mẫu Kiến trúc Kho Repository

Các hệ thống con phải trao đổi dữ liệu. Việc này có thể được thực hiện theo hai cách: Dữ liệu được chia sẻ được lưu giữ trong cơ sở dữ liệu trung tâm hoặc kho lưu trữ và có thể được truy cập bởi tất cả các hệ thống con; Mỗi hệ thống con duy trì cơ sở dữ liệu riêng của mình và chuyển dữ liệu một cách rõ ràng đến các hệ thống con khác. Khi một lượng lớn dữ liệu cần được chia sẻ, mô hình kho lưu trữ chia sẻ thường được sử dụng nhất vì đây là một cơ chế chia sẻ dữ liệu hiệu quả.



Mẫu Kiến trúc Kho Repository

Mô tả: Tất cả dữ liệu trong một hệ thống được quản lý trong một kho lưu trữ trung tâm có thể truy cập được đối với tất cả các thành phần hệ thống. Các thành phần không tương tác trực tiếp, chỉ thông qua kho lưu trữ.

Ví dụ: IDE nơi các thành phần sử dụng kho lưu trữ thông tin thiết kế hệ thống. Mỗi công cụ phần mềm tạo ra thông tin, sau đó thông tin này có sẵn để các công cụ khác sử dụng.

Khi nào sử dụng: sử dụng mẫu này khi có một hệ thống tạo ra một lượng lớn thông tin cần được lưu trữ trong một thời gian dài. Có thể sử dụng nó trong các hệ thống hướng dữ liệu, nơi việc đưa dữ liệu vào kho lưu trữ sẽ kích hoạt một hành động hoặc công cụ.

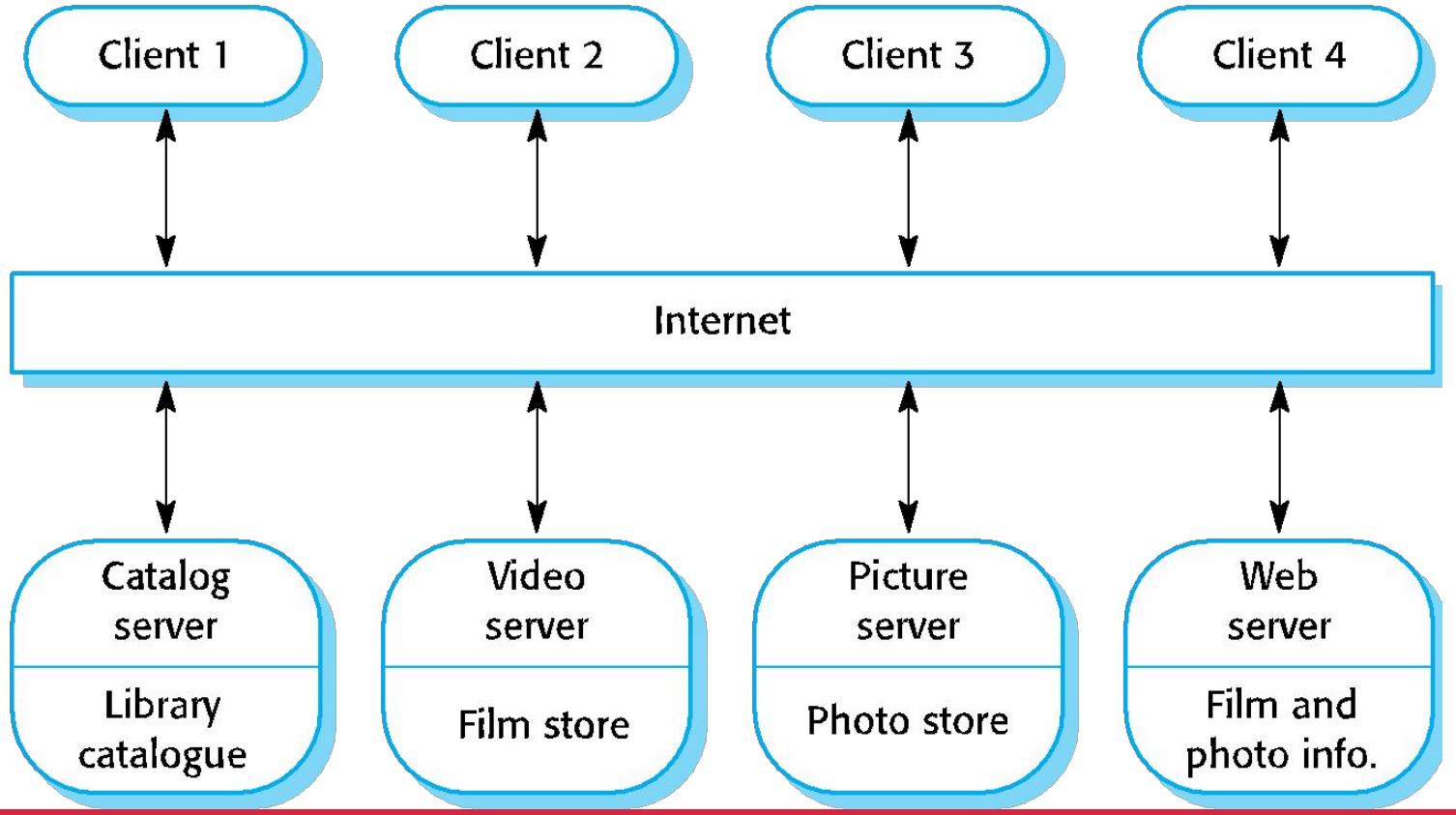
Ưu điểm: Các thành phần có thể độc lập—chúng không cần biết đến sự tồn tại của các thành phần khác. Các thay đổi được thực hiện bởi một thành phần có thể được lan truyền đến tất cả các thành phần. Tất cả dữ liệu có thể được quản lý nhất quán (ví dụ: sao lưu được thực hiện cùng một lúc) vì tất cả đều ở một nơi.

Nhược điểm: Kho lưu trữ là một điểm lỗi duy nhất, vì vậy các vấn đề trong kho lưu trữ ảnh hưởng đến toàn bộ hệ thống. Có thể có sự kém hiệu quả trong việc tổ chức tất cả giao tiếp thông qua kho lưu trữ. Việc phân phối kho lưu trữ trên nhiều máy tính có thể khó khăn.

Mẫu Kiến trúc Khách-Chủ

Mô hình hệ thống phân tán cho thấy cách dữ liệu và quá trình xử lý được phân phối trên một loạt các thành phần. Có thể được triển khai trên một máy tính duy nhất. Tập hợp các máy chủ độc lập cung cấp các dịch vụ cụ thể như in ấn, quản lý dữ liệu, v.v. Tập hợp các máy khách gọi các dịch vụ này. Mạng cho phép máy khách truy cập máy chủ.

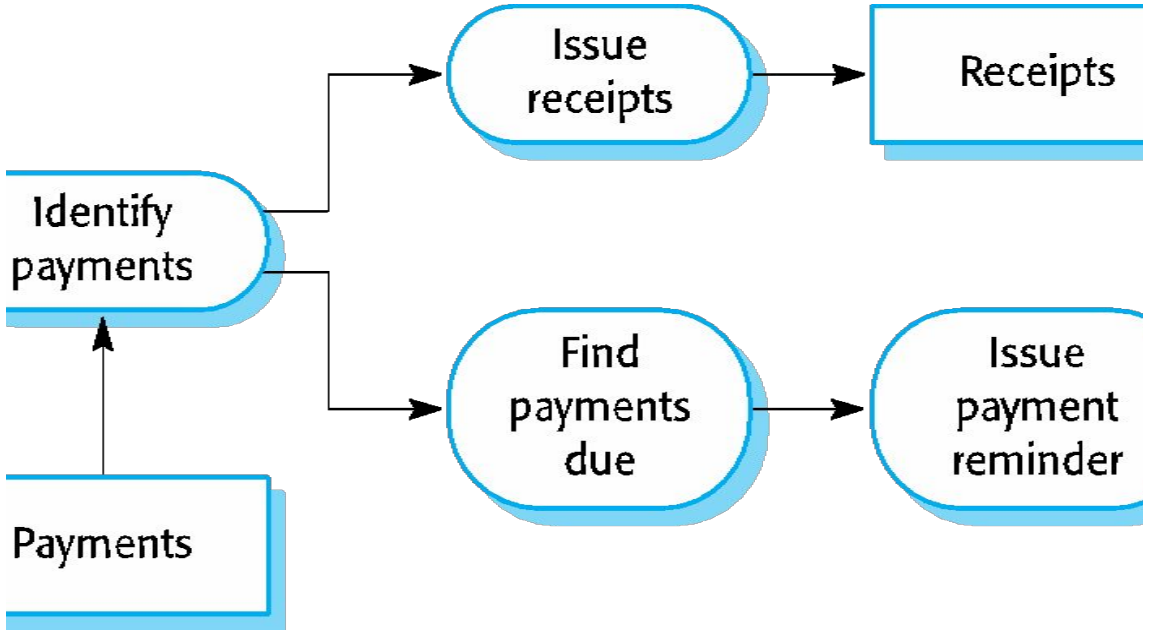
A client–server architecture for a film library



Mẫu Kiến trúc Ống và Bộ lọc (Pipe and Filter)

- Các phép biến đổi chức năng xử lý đầu vào của chúng để tạo ra đầu ra.
- Có thể được gọi là mô hình ống và bộ lọc (như trong UNIX shell).
- Các biến thể của phương pháp này rất phổ biến. Khi các phép biến đổi là tuần tự, đây là mô hình tuần tự hàng loạt được sử dụng rộng rãi trong các hệ thống xử lý dữ liệu.
- Không thực sự phù hợp cho các hệ thống tương tác.

Một ví dụ về kiến trúc ống và bộ lọc được sử dụng trong hệ thống thanh toán.



Mẫu Kiến trúc Ống và Bộ lọc (Pipe and Filter)

Mô tả: Quá trình xử lý dữ liệu trong một hệ thống được tổ chức sao cho mỗi thành phần xử lý (bộ lọc) là rời rạc và thực hiện một loại biến đổi dữ liệu. Dữ liệu truyền (như trong một đường ống) từ thành phần này sang thành phần khác để xử lý.

Ví dụ: về hệ thống ống và bộ lọc được sử dụng để xử lý hóa đơn.

Khi nào sử dụng: Thường được sử dụng trong các ứng dụng xử lý dữ liệu (cả dựa trên lô và giao dịch) trong đó đầu vào được xử lý trong các giai đoạn riêng biệt để tạo ra các đầu ra liên quan.

Ưu điểm: Dễ hiểu và hỗ trợ tái sử dụng biến đổi. Kiểu luồng công việc phù hợp với cấu trúc của nhiều quy trình kinh doanh. Việc phát triển bằng cách thêm các phép biến đổi là đơn giản. Có thể được triển khai dưới dạng hệ thống tuần tự hoặc đồng thời.

Nhược điểm: Định dạng để truyền dữ liệu phải được thống nhất giữa các phép biến đổi giao tiếp. Mỗi phép biến đổi phải phân tích cú pháp đầu vào của nó và phân tích cú pháp đầu ra của nó theo biểu mẫu đã thỏa thuận. Điều này làm tăng chi phí hệ thống và có thể có nghĩa là không thể sử dụng lại các phép biến đổi chức năng sử dụng các cấu trúc dữ liệu không tương thích.

Mẫu Kiến trúc Khách-Chủ

Mô tả: Trong kiến trúc máy khách – máy chủ, chức năng của hệ thống được tổ chức thành các dịch vụ, với mỗi dịch vụ được cung cấp từ một máy chủ riêng biệt. Máy khách là người dùng các dịch vụ này và truy cập các máy chủ để sử dụng chúng.

Ví dụ: thư viện phim và video/DVD được tổ chức như một hệ thống máy khách – máy chủ.

Khi nào sử dụng: Được sử dụng khi dữ liệu trong cơ sở dữ liệu dùng chung phải được truy cập từ nhiều vị trí khác nhau. Vì các máy chủ có thể được sao chép, nên cũng có thể được sử dụng khi tải trên hệ thống thay đổi.

Ưu điểm: Ưu điểm chính của mô hình này là các máy chủ có thể được phân phối trên một mạng lưới. Chức năng chung (ví dụ: dịch vụ in) có thể khả dụng cho tất cả các máy khách và không cần phải được triển khai bởi tất cả các dịch vụ.

Nhược điểm: Mỗi dịch vụ là một điểm lỗi duy nhất nên dễ bị tấn công từ chối dịch vụ hoặc lỗi máy chủ. Hiệu suất có thể không dự đoán được vì nó phụ thuộc vào mạng cũng như hệ thống. Có thể có vấn đề quản lý nếu các máy chủ thuộc sở hữu của các tổ chức khác nhau.

4 Các Mẫu Kiến trúc cho Hệ thống Phân tán

Khi thiết kế các ứng dụng phân tán, hãy chọn một mẫu hỗ trợ các yêu cầu phi chức năng quan trọng.

- **Kiến trúc Chủ-Tớ (Master-Slave Architecture)**: Được sử dụng trong các hệ thống thời gian thực yêu cầu thời gian phản hồi tương tác được đảm bảo. Một "chủ" điều khiển các quy trình "tớ" chuyên dụng cho các hành động cụ thể.
- **Kiến trúc Khách-Chủ Hai Tầng (Two-Tier Client-Server Architecture)**: Dạng đơn giản nhất với một máy chủ logic duy nhất và nhiều máy khách. Có thể là mô hình thin-client (trình bày trên máy khách) hoặc fat-client (một số xử lý ứng dụng trên máy khách).
- **Kiến trúc Khách-Chủ Nhiều Tầng (Multi-Tier Client-Server Architecture)**: Được sử dụng cho khối lượng giao dịch lớn, với xử lý ứng dụng trên một máy chủ tầng giữa.
- **Kiến trúc Thành phần Phân tán (Distributed Component Architecture)**: Kết hợp tài nguyên từ các hệ thống và cơ sở dữ liệu khác nhau, hoặc như một mô hình triển khai cho các hệ thống khách-chủ nhiều tầng. Yêu cầu phần mềm trung gian (middleware).
- **Kiến trúc Ngang hàng (Peer-to-Peer - P2P Architecture)**: Phân tán, không có máy khách hoặc máy chủ phân biệt. Hữu ích cho việc trao đổi thông tin giữa các máy tính riêng lẻ.

Kiến trúc ứng dụng

Các hệ thống ứng dụng được thiết kế để đáp ứng nhu cầu của tổ chức.

Vì các doanh nghiệp có nhiều điểm chung, các hệ thống ứng dụng của họ cũng có xu hướng có kiến trúc chung phản ánh các yêu cầu ứng dụng.

Kiến trúc ứng dụng chung là kiến trúc cho một loại hệ thống phần mềm có thể được định cấu hình và điều chỉnh để tạo ra một hệ thống đáp ứng các yêu cầu cụ thể.

Sử dụng kiến trúc ứng dụng

- Làm điểm khởi đầu cho thiết kế kiến trúc.
- Như một danh sách kiểm tra thiết kế.
- Như một cách tổ chức công việc của nhóm phát triển.
- Như một phương tiện đánh giá các thành phần để tái sử dụng.
- Như một từ vựng để nói về các loại ứng dụng.

Ví dụ các loại ứng dụng

Ứng dụng xử lý dữ liệu:
Các ứng dụng hướng dữ liệu xử lý dữ liệu theo lô mà không có sự can thiệp rõ ràng của người dùng trong quá trình xử lý.
Ứng dụng xử lý giao dịch:
Các ứng dụng tập trung vào dữ liệu xử lý các yêu cầu của người dùng và cập nhật thông tin trong cơ sở dữ liệu hệ thống.
Hệ thống xử lý sự kiện:
Các ứng dụng mà hành động của hệ thống phụ thuộc vào việc diễn giải các sự kiện từ môi trường của hệ thống.
Hệ thống xử lý ngôn ngữ:
Các ứng dụng mà ý định của người dùng được chỉ định bằng một ngôn ngữ chính thức được xử lý và diễn giải bởi hệ thống.

Những Điểm quan trọng

- Kiến trúc phần mềm là một mô tả về cách một hệ thống phần mềm được tổ chức.
- Các quyết định thiết kế kiến trúc bao gồm các quyết định về loại ứng dụng, sự phân phối của hệ thống, các kiểu kiến trúc sẽ được sử dụng.
- Các kiến trúc có thể được ghi lại từ nhiều góc độ hoặc dạng xem khác nhau như dạng xem khái niệm, dạng xem logic, dạng xem quy trình và dạng xem phát triển.
- Các mẫu kiến trúc là một phương tiện để tái sử dụng kiến thức về các kiến trúc hệ thống chung. Chúng mô tả kiến trúc, giải thích khi nào nó có thể được sử dụng và mô tả những ưu điểm và nhược điểm của nó.

Những Điểm quan trọng

- Các mô hình kiến trúc hệ thống ứng dụng giúp chúng ta hiểu và so sánh các ứng dụng, xác thực các thiết kế hệ thống ứng dụng và đánh giá các thành phần quy mô lớn để tái sử dụng.
- Hệ thống xử lý giao dịch là các hệ thống tương tác cho phép thông tin trong cơ sở dữ liệu được truy cập và sửa đổi từ xa bởi một số người dùng.
- Hệ thống xử lý ngôn ngữ được sử dụng để dịch văn bản từ ngôn ngữ này sang ngôn ngữ khác và để thực hiện các hướng dẫn được chỉ định trong ngôn ngữ đầu vào. Chúng bao gồm một trình biên dịch và một máy trừu tượng thực thi ngôn ngữ được tạo ra.

Giao thức Truyền thông tin

1. HTTP (Hypertext Transfer Protocol)
2. HTTPS (HTTP Secure)
3. TCP (Transmission Control Protocol)
4. UDP (User Datagram Protocol)
5. IP (Internet Protocol)
6. FTP (File Transfer Protocol)
7. SMTP (Simple Mail Transfer Protocol)
8. WebSocket
9. MQTT (Message Queuing Telemetry Transport)
10. gRPC

Lựa chọn ngôn ngữ lập trình

Ngôn Ngữ	Ưu Điểm	Nhược Điểm	Ứng Dụng Tiêu Biểu
C/C++	Tốc độ cao, kiểm soát phần cứng tối ưu	Phức tạp, dễ gây lỗi bộ nhớ	Hệ điều hành, game engines
Java	Đa nền tảng, thư viện phong phú	Tốn bộ nhớ, khởi động chậm	Ứng dụng doanh nghiệp, Android
Python	Cú pháp đơn giản, thư viện AI/Data mạnh	Chậm do thông dịch	Khoa học dữ liệu, scripting
Go	Xử lý đồng thời tốt, hiệu suất cao	Thiếu một số tính năng (generics)	Hệ thống mạng, cloud services
Rust	An toàn bộ nhớ, hiệu suất gần C++	Học curve dốc, phức tạp	Hệ thống nhúng, game engines
JavaScript	Linh hoạt, chạy mọi nơi trên web	Lỗi runtime do dynamic typing	Web frontend/backend
TypeScript	Thêm static type cho JavaScript	Tăng độ phức tạp khi triển khai	Ứng dụng web lớn

Lựa chọn ngôn ngữ lập trình

Việc lựa chọn ngôn ngữ lập trình nên được thực hiện ở giai đoạn đầu khi phân tích yêu cầu dự án. Quyết định này cần dựa trên các yếu tố như:

- Hiệu suất,
- Khả năng mở rộng,
- Môi trường phát triển,
- Kỹ năng đội ngũ,
- Tính lâu dài của ngôn ngữ.
- Mục tiêu dự án (tốc độ, bảo trì, quy mô)
- Ưu tiên (hiệu suất vs. thời gian phát triển).

- **Hiệu suất cao:** C/C++, Go, Rust phù hợp cho hệ thống yêu cầu tốc độ và tài nguyên tối ưu
- **Linh hoạt và dễ học:** Python, JavaScript phát triển nhanh nhưng đánh đổi hiệu suất
- **Cân bằng:** Java và TypeScript phù hợp cho dự án lớn cần ổn định và khả năng mở rộng

Tổng kết

Sau khi học nội dung 4, sinh viên được trang bị các kiến thức sau :

- Các khái niệm liên quan tới thiết kế kiến trúc phần mềm
- Giới thiệu về hai giai đoạn thiết kế kiến trúc phần mềm: thiết kế kiến trúc và thiết kế chi tiết
- Phân biệt tính móc nối (Coupling) và tính kết dính (Cohesion) trong thiết kế phần mềm
- Tầm quan trọng của thiết kế PM