**MỤC LỤC**

[LỜI MỞ ĐẦU 4](#_Toc79483136)

[CHƯƠNG 1: TỔNG QUAN VỀ THIẾT KẾ KIẾN TRÚC 5](#_Toc79483137)

[1.1. Tổng quan 5](#_Toc79483138)

[1.2 Khái niệm kiến trúc phần mềm 7](#_Toc79483139)

[1.3 Một số khái niệm cơ bản 12](#_Toc79483140)

[1.3.1. Kiến trúc 12](#_Toc79483141)

[1.3.2. Thành phần 12](#_Toc79483142)

[1.3.3. Kết nối 13](#_Toc79483143)

[1.3.4 Cấu hình 14](#_Toc79483144)

[1.4 Các mẫu kiến trúc phần mềm (Architecture Styles) 15](#_Toc79483145)

[1.5. Mẫu thiết kế (design pattern) 15](#_Toc79483146)

[1.6 Các thuộc tính chất lượng 16](#_Toc79483147)

[1.7 Hướng dẫn thiết kế kiến trúc phần mềm 18](#_Toc79483148)

[1.8 Tóm tắt 20](#_Toc79483149)

[CHƯƠNG 2: MÔ HÌNH HÓA KIẾN TRÚC 21](#_Toc79483150)

[2.1 Tổng quan 21](#_Toc79483151)

[2.2 Yêu cầu của mô hình hóa kiến trúc 23](#_Toc79483152)

[2.3 Kiểu kiến trúc và mô hình hóa 24](#_Toc79483153)

[2.4. Kỹ thuật mô hình 25](#_Toc79483154)

[2.5. Ngôn ngữ mô hình hóa thống nhất UML cho thiết kế kiến trúc 26](#_Toc79483155)

[2.5.1 Các biểu đồ cấu trúc 28](#_Toc79483156)

[2.5.2 Các biểu đồ hành vi 34](#_Toc79483157)

[2.6. Mô hình khung nhìn kiến trúc 42](#_Toc79483158)

[2.6.1 Khung nhìn kịch bản 43](#_Toc79483159)

[2.6.2 Khung nhìn logic hoặc khái niệm 44](#_Toc79483160)

[2.6.3 Khung nhìn phát triển hoặc Mô-đun 45](#_Toc79483161)

[2.6.4 Khung nhìn quy trình 45](#_Toc79483162)

[2.6.5 Khung nhìn vật lý 47](#_Toc79483163)

[2.6.6 Khung nhìn giao diện người dùng 48](#_Toc79483164)

[2.7. Ngôn ngữ mô tả kiến trúc (ADL) 49](#_Toc79483165)

[2.8. Tổng kết 51](#_Toc79483166)

[CHƯƠNG 3: CÁC THUỘC TÍNH CHẤT LƯỢNG THIẾT KẾ 52](#_Toc79483167)

[3.1. Tổng quan 52](#_Toc79483168)

[3.2. Kiến trúc và các yêu cầu 52](#_Toc79483169)

[3.3. Xem xét thuộc tính chất lượng (quality attribute - QA) 60](#_Toc79483170)

[3.4. Các kịch bản đặc tính chất lượng 53](#_Toc79483171)

[CHƯƠNG 4. CÁC KIỂU KIẾN TRÚC PHẦN MỀM 80](#_Toc79483173)

[4.1 Kiến trúc phần mềm dữ liệu tập trung 92](#_Toc79483174)

[4.1.1 Tổng quan 92](#_Toc79483175)

[4.1.2. Kiến trúc kho chứa (Repository Architecture, RA) 93](#_Toc79483176)

[4.1.3 Mẫu kiến trúc Blackboard 97](#_Toc79483177)

[4.2 Kiến trúc phần mềm tương tác 103](#_Toc79483178)

[4.2.1 Tổng quan 103](#_Toc79483179)

[4.2.2 Kiến trúc MVC 104](#_Toc79483180)

[4.3. Kiến trúc phân lớp 108](#_Toc79483181)

[4.4. Kiến trúc dựa trên thành phần 110](#_Toc79483182)

[4.4.1. Tổng quan 110](#_Toc79483183)

[4.4.2 Thành phần (cấu phần) là gì? 111](#_Toc79483184)

[4.4.3. Mô hình thành phần (component models) 116](#_Toc79483185)

[4.4.4 Nguyên tắc thiết kế dựa trên thành phần 119](#_Toc79483186)

[4.4.5. Miền áp dụng và ưu/nhược điểm của kiến trúc dựa trên thành phần 122](#_Toc79483187)

[4.4.6 Tóm tắt 123](#_Toc79483188)

[4.6.7. Case study 124](#_Toc79483189)

[4.5. Kiến trúc phân tán 127](#_Toc79483190)

[4.5.1 Tổng quan 127](#_Toc79483191)

[4.5.2. Máy khách – chủ (Client – Server) 128](#_Toc79483192)

[4.5.3. Kiến trúc khách chủ đa tầng (multi – tier client server) 132](#_Toc79483193)

[4.5.4. Kiến trúc Master-slave (chủ - tớ) 134](#_Toc79483194)

[4.5.5. Kiến trúc ngang hàng (peer – to – peer) 135](#_Toc79483195)

[4.5.6. Tóm tắt 138](#_Toc79483196)

[4.6. Kiến trúc hướng dịch vụ (SOA) 138](#_Toc79483197)

[4.6.1. Tổng quan 138](#_Toc79483198)

[4.6.2. Dịch vụ như các thành phần có thể tái sử dụng 146](#_Toc79483199)

[4.6.3 Triển khai SOA trong Dịch vụ Web 149](#_Toc79483200)

[4.6.4 Triển khai SOA cho Điện toán Dịch vụ Lưới (Grid Service Computing) 153](#_Toc79483201)

[CHƯƠNG 5: QUY TRÌNH THIẾT KẾ KIẾN TRÚC 155](#_Toc79483202)

[5.1. Tổng quan 155](#_Toc79483203)

[5.1.1 Quyết định các yêu cầu kiến trúc 156](#_Toc79483204)

[5.1.2. Xác định các yêu cầu kiến trúc 156](#_Toc79483205)

[5.1.3. Ưu tiên các yêu cầu kiến trúc 158](#_Toc79483206)

[5.2. Thiết kế kiến trúc 159](#_Toc79483207)

[5.2.1. Lựa chọn khung thiết kế 159](#_Toc79483208)

[5.2.2. Phân bổ các thành phần 163](#_Toc79483209)

[5.3. Thẩm định (validation) 166](#_Toc79483210)

[CHƯƠNG 6. TÀI LIỆU KIẾN TRÚC PHẦN MỀM 167](#_Toc79483211)

[6.1. Giới thiệu tổng quan về tài liệu kiến trúc 167](#_Toc79483212)

[6.2. Việc sử dụng và đối tượng cho tài liệu kiến trúc 168](#_Toc79483213)

[6.3. Ký hiệu cho Tài liệu Kiến trúc 168](#_Toc79483214)

[6.4. Khung nhìn 169](#_Toc79483215)

[6.5. Mẫu tài liệu kiến trúc phần mềm 173](#_Toc79483216)

[6.6. Case study 173](#_Toc79483217)

[CHƯƠNG 7: KIẾN TRÚC KHÔNG ĐỒNG NHẤT – HETEROGENEOUS 174](#_Toc79483218)

[7.1 Tổng quan 174](#_Toc79483219)

[7.2 Phương pháp luận về quyết định kiến ​​trúc 174](#_Toc79483220)

[7.3 Thuộc tính chất lượng 176](#_Toc79483221)

[7.4 Lựa chọn kiểu kiến ​​trúc 177](#_Toc79483222)

[7.5 Đánh giá thiết kế kiến ​​trúc 179](#_Toc79483223)

[7.6 Nghiên cứu điển hình: Nhà cung cấp máy tính trực tuyến 183](#_Toc79483224)

[7.6.1 Thiết kế kiến ​​trúc tổng thể của OCVS 184](#_Toc79483225)

[7.6.2 Thiết kế kiến ​​trúc của thành phần xử lý đơn hàng 188](#_Toc79483226)

[7.6.3 Thiết kế kiến ​​trúc của quản lý hàng tồn kho 191](#_Toc79483227)

[7.6.4 Thiết kế kiến trúc của thành phần sản xuất và vận chuyển 193](#_Toc79483228)

[7.7 Tóm tắt 196](#_Toc79483229)

[CHƯƠNG 8: KIẾN TRÚC VÀ MẪU THIẾT KẾ - DESIGN PATTERN 198](#_Toc79483230)

[8.1. Khái niệm mẫu thiết kế 198](#_Toc79483231)

[8.2. Định dạng mẫu thiết kế 200](#_Toc79483232)

[8.4. Sử dụng mẫu thiết kế 202](#_Toc79483236)

[8.4.1. Khi nào sử dụng mẫu thiết kế 202](#_Toc79483237)

[8.4.2. Sử dụng mẫu thiết kế như thế nào 203](#_Toc79483238)

# LỜI MỞ ĐẦU

Khi kích thước và độ phức tạp của các hệ thống phần mềm tăng lên, vấn đề thiết kế vượt ra ngoài các thuật toán và cấu trúc dữ liệu của tính toán: thiết kế và xác định cấu trúc hệ thống tổng thể nổi lên như một loại vấn đề mới. Các vấn đề về cấu trúc bao gồm cấu trúc tổng thể của hệ thống; các giao thức để giao tiếp, đồng bộ hóa và truy cập dữ liệu; phân công chức năng cho các thành phần thiết kế; phân phối vật chất; thành phần của các yếu tố thiết kế; mở rộng quy mô và hiệu suất; và lựa chọn giữa các phương án thiết kế.

# CHƯƠNG 1: TỔNG QUAN VỀ THIẾT KẾ KIẾN TRÚC

Mục tiêu của Chương này:

* Giới thiệu mối quan hệ giữa các yêu cầu phần mềm và kiến trúc
* Giới thiệu mối quan hệ giữa các mẫu kiến trúc và kiến trúc
* Giới thiệu các thành phần của kiến trúc phần mềm
* Mô tả các thuộc tính chất lượng và phân tích cân bằng giữa các thuộc tính

## **1.1. Tổng quan**

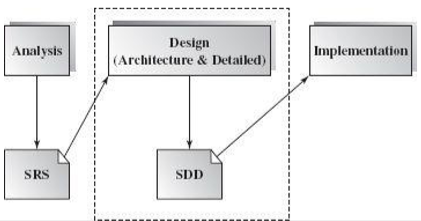
Mục tiêu của kiến trúc phần mềm là xây dựng một mô hình đáp ứng các yêu cầu của người dùng và dẫn đến việc thực thi thành công. Khi hệ thống phần mềm tiếp tục tăng quy mô, độ phức tạp và phân bố, những thiết kế phù hợp trở nên cực kỳ quan trọng trong sản xuất phần mềm. Bất kỳ phần mềm, bất kể miền ứng dụng của nó, nên có một thiết kế kiến trúc tổng thể hướng dẫn xây dựng và phát triển phần mềm. Sự thành công của một sản phẩm phần mềm hoặc hệ thống chủ yếu phụ thuộc vào sự thành công của thiết kế kiến trúc.

Thiết kế kiến trúc là gì? “Thiết kế kiến trúc xác định mối quan hệ giữa các thành phần chính của phần mềm, các mẫu thiết kế có thể được sử dụng đã đạt được các yêu cầu hệ thống và những hạn chế ảnh hưởng đến cách thức mà kiến trúc có thể được thực hiện” (Garlan và Shaw, 1996). Các đại diện thiết kế kiến trúc có được từ các đặc tả yêu cầu phần mềm và các mô hình phân tích.

Ai chịu trách nhiệm thiết kế kiến trúc? Kiến trúc sư phần mềm và nhà thiết kế tham gia vào quá trình này. Họ chuyển từ các yêu cầu hệ thống phần mềm thành thiết kế kiến trúc. Trong quá trình chuyển các yêu cầu, họ áp dụng các chiến lược thiết kế khác nhau để phân chia và chinh phục sự phức tạp của một miền ứng dụng và giải quyết kiến trúc phần mềm.

Tại sao kiến trúc phần mềm lại quan trọng? Có nhiều lý do để trả lời câu hỏi này. Một thiết kế nghèo nàn có thể dẫn đến một sản phẩm thiếu mà không đáp ứng các yêu cầu hệ thống, không thích hợp cho các thay đổi yêu cầu trong tương lai, không thể tái sử dụng, thể hiện hành vi không thể đoán trước được hoặc các hoạt động xấu. Nếu không có kế hoạch phù hợp trong giai đoạn thiết kế kiến trúc, sản xuất phần mềm có thể sẽ không hiệu quả về thời gian và chi phí. Ngược lại, một thiết kế phần mềm tốt làm giảm các rủi ro liên quan tới sản xuất phần mềm, giúp nhóm phát triển làm việc cùng nhau một cách có trình tự, làm cho hệ thống vẽ được để triển khai và thử nghiệm, và dẫn đến các sản phẩm phần mềm có các thuộc tính chất lượng cao hơn.

Khi nào thiết kế phần mềm được tiến hành? Thiết kế phần mềm là giai đoạn đầu của chu trình phát triển phần mềm (SDLC). Trong giai đoạn này, các nhà thiết kế phần mềm mô hình hóa hệ thống và đánh giá chất lượng của nó để có thể thực hiện các cải tiến trước khi phần mềm đi vào giai đoạn sản xuất. Như được hiển thị trong hình 1.1., SDLC bao gồm các giai đoạn sau: phân tích yêu cầu, thiết kế phần mềm (bao gồm thiết kế kiến trúc và thiết kế chi tiết), phát triển phần mềm và thực thi; kiểm tra và đảm bảo chất lượng, bảo trì và tiến hóa. Hình bên dưới mô tả phạm vi của thiết kế phần mềm. Các đặc tả yêu cầu phần mềm (SRS) cung cấp đầu vào cần thiết cho thiết kế. SRS là kết quả của việc phân tích yêu cầu; nó ghi lại các yêu cầu chức năng và phi chức năng phải được đáp ứng bởi hệ thống phần mềm



Hình 1.1. Vòng đời phát triển một phần mềm đơn giản

Kết quả của thiết kế kiến trúc phần mềm là gì? Nói một cách đơn giản, nó là một đại diện tổng thể của phần mềm sẽ được xây dựng. Theo IEEE Std 1016 – IEEE Recommended Practice for Software Design Descriptions (SDD), được thể hiện trong hình 1.1. mô tả một tổ chức cho mô tả thiết kế phần mềm. SDD đóng vai trò là bản thiết kế chi tiết cho giai đoạn thực hiện. Nó mô tả các thành phần của một hệ thống, các mô-đun tạo ra mỗi thành phần và thông tin chi tiết (chẳng hạn các thuộc tính dữ liệu, hoạt động và thuật toán) của mỗi mô-đun. SDD được sử dụng làm khuôn mẫu cho mô tả thiết kế phần mềm:

Dưới đây là một phác thảo mẫu của SDD dựa trên IEEE 1016:

* Mô tả phân rã (mô-đun, dữ liệu, quy trình)
* Mô tả sự phục thuộc và kết nối (giữa các mô – đun, dữ liệu và quy trình)
* Các thuộc tính
* Mô tả giao diện người dùng
* Thiết kế chi tiết (mô – đun và dữ liệu)

Lưu ý rằng thiết kế kiến trúc là thiết kế chiến lược (strategy design) trước khi thực hiện thiết kế chi tiết. Thiết kế chi tiết, còn được gọi là thiết kế chiến thuật (tactical design), liên quan đến các ràng buộc thiết kế cục bộ và các chi tiết bên trong của mỗi thành phần. Ví dụ, trong thiết kế kiến trúc của hệ thống điều khiển giao thông thành phố, nhà thiết kế có thể chỉ định một hàng đợi ưu tiên lưu trữ và gửi các yêu cầu đến. Trong thiết kế chi tiết, người thiết kế phải chọn cấu trúc dữ liệu bên trong từ các giải pháp thay thế. Ví dụ hàng đợi ưu tiên có thể được triển khai bằng cách sử dụng một danh sách liên kết đơn, danh sách liên kết đôi hoặc một mảng. Các nhà thiết kế sau đó phải ghi lại lý do của mình để chọn một cấu trúc dữ liệu nội bộ cụ thể. Trong thiết kế phần mềm quy mô lớn, kiến trúc sư phần mềm có thể thực hiện thiết kế hệ thống con trước khi thiết kế chi tiết.

Bây giờ, trong tài liệu sẽ trình bày chi tiết về các khái niệm thiết kế kiến trúc. Hãy xem việc xây nhà như một phép tương tự. Trước khi bắt đầu xây dựng, người xây dựng cần biết các yêu cầu từ khách hàng và kiến trúc sư phải thiết kế bản thiết kế. Các kiến trúc sư có nhiều tùy chọn để lựa chọn, chẳng hạn như mẫu (ví dụ nhà ống, nhà biệt thự, nhà cấp 4, v.v.), chức năng (ví dụ: kinh doanh hoặc nhà ở) và các đặc điểm của ngôi nhà (ví dụ: tầng hầm hoặc bể bơi, có vườn…). Tương tự, các thông số kỹ thuật của các thành phần phần mềm, trình kết nối, các ràng buộc (không gian, thời gian, ngân sách, v.v.) và các thuộc tính chất lượng mong muốn (chẳng hạn như tính khả dụng và hiệu suất) phải được giải quyết trong thiết kế phần mềm và điều này được gọi là “kiến trúc phần mềm” hoặc thiết kế mức cao.

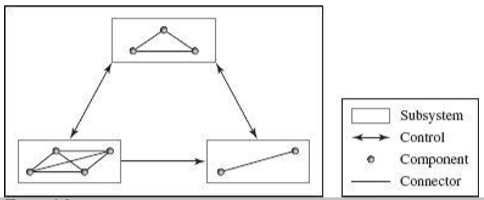
Trong thực tế, các nhà thiết kế chỉ định mẫu kiến trúc bằng cách tách các đặc điểm chung của các phần tử (elements) và các kết nối thành “họ kiến trúc - families of architecture”. Mỗi kiểu thể hiện một cấu trúc liên kết bố cục của các phần tử, các trình kết nối và tương tác giữa chúng. Mỗi kiểu cũng mô tả các ràng buộc và hành vi ngữ nghĩa của nó liên quan đến truyền dữ liệu và kiểm soát giữa các phần tử trong hệ thống, cũng như sự cân bằng các thuộc tính chất lượng.

Thuộc tính chất lượng phần mềm bao gồm các yêu cầu phi chức năng như hiệu suất, độ tin cậy, khả chuyển, khả năng sử dụng, bảo mật, khả năng kiểm thử, khả năng bảo trì, khả năng thích ứng, khả năng sửa đổi, và khả năng mở rộng. Các thuộc tính chất lượng có liên quan chặt chẽ tới mẫu kiến trúc trong đó mỗi kiểu kiến trúc hỗ trợ một số đặc điểm chất lượng. Một mẫu kiến trúc bao hàm sự cân bằng giữa nhiều thuộc tính chất lượng xung đột. Ví dụ, với hiệu suất hệ thống, luôn có sự cân bằng giữa thời gian/tài nguyên, và độ tin cậy và tính khả dụng của hệ thống.

## **1.2 Khái niệm kiến trúc phần mềm**

Kiến trúc phần mềm đóng một vài trò rất quan trọng trong vòng đời phát triển phần mềm. Các thiết kế kiến trúc cung cấp một một bản thiết kế (Blueprint) và hướng dẫn để phát triển một hệ thống phần mềm dựa trên đặc tả phân tích yêu cầu của nó. Thiết kế kiến trúc là hiện thân của những quyết định sớm nhất có tác động quyết định đến sự thành công cuối cùng của sản phẩm phần mềm. Thiết kế cho thấy các thành phần hệ thống được cấu trúc như thế nào và cách chúng hoạt động cùng nhau. Một thiết kế kiến trúc cũng phải bao gồm các yêu cầu chức năng và phi chức năng của phần mềm. Nó phục vụ như một kế hoạch đánh giá và thực hiện để phát triển phần mềm và cải tiến phần mềm.

Biểu đồ hộp và đường thẳng trong hình 1.2 cho thấy một thiết kế kiến trúc thường trông như thế nào. Lưu ý rằng nó không chứa tập hợp thông tin hoàn chỉnh được tìm thấy trong một thiết kế phát triển. Ví dụ, nó cung cấp đủ các hướng dẫn để các lập trình viên làm theo, cũng như không mô tả bất kỳ thuộc tính chất lượng nào. Trong hình 1.2, mỗi thành phần hay phần tử (còn gọi là “hệ thống con”) tượng trưng cho một trách nhiệm duy nhất như xử lý logic, điều khiển logic, thu thập dữ liệu, trình bày giao diện, dịch vụ. Việc phân chia các phần tử này dựa trên các chức năng, vị trí và các thời gian chạy của chúng. Các phần tử có thể ở dạng mô- đun, đối tượng, gói và phần tử đã được triển khai, nhiệm vụ, chức năng, truy trình, các chương trình phân tán, v.v. Cấu trúc liên kết của cấu trúc tĩnh tập trung vào cấu hình thành phần hệ thống như phân lớp, làm phẳng, kiểu sao, tập trung hoặc phân tán. Các trình kết nối thời gian chạy động có thể là tuần tự hàng loạt, đa luồng, lệnh gọi trực tiếp rõ ràng, lệnh gọi gián tiếp ngầm định (chẳng hạn như hàng đợi thông điệp hoặc thông báo sự kiện), giao tiếp đồng bộ hoặc không đồng bộ, trao đổi thông điệp ngang hàng (peer-to-peer) hoặc truyền phát thông điệp, hoặc phối hợp áp dụng khác và cơ chế hợp tác giữa các phần tử.



Hình 1.2. Các biểu đồ hộp và đường biểu diễn các hệ thống con.

Hình 1.2 minh họa ý tưởng về kiến trúc phần mềm, nhưng định nghĩa chính thức của nó là gì?

Thuật ngữ kiến trúc phần mềm (software architecture) lần đầu tiên được sử dụng [1] vào cuối những năm 1960 nhưng mãi đến những năm 1990 khái niệm này mới được xem xét, nghiên cứu rộng rãi trong giới hàn lâm và công nghiệp phần mềm. Kiến trúc phần mềm được hiểu như một khái niệm quan trọng bắt nguồn từ những nghiên cứu của Dijkstra vào năm 1968 và Parnas vào đầu những năm 1970. Các nghiên cứu này đã chỉ ra rằng kiến trúc của hệ thống phần mềm đóng một vai trò quan trọng và làm thế nào có được một kiến trúc hợp lý là vấn đề cốt lõi của quá trình phát triển phần mềm. Những năm 1990 đã chứng kiến nhiều nỗ lực nghiên cứu nhằm đề xuất các kiểu kiến trúc phần mềm, các ngôn ngữ mô tả kiến trúc, các mẫu thiết kế, các mô hình thành phần phần mềm, các phương pháp hình thức để mô hình hóa kiến trúc,…Động lực chính cho sự quan tâm mạnh mẽ này là mã nguồn của nhiều hệ phần mềm phổ biến đã lên tới hàng trăm nghìn hay hàng triệu dòng lệnh.

Kiến trúc hệ thống (system architecture) là một sự gắn kết các bộ phận trong một hệ thống với nhau gồm cấu trúc, giao diện, và các cơ chế để phối hợp hoạt động giữa các bộ phận. Khi xác định được kiến trúc phù hợp, chúng ta sẽ dễ dàng chuyển đổi, tìm được vị trí của một chức năng hay chỉ ra được nơi có thể thêm một chức năng mới phù hợp với kiến trúc chung. Một kiến trúc tốt [18] phải được chi tiết hóa đủ để có thể ánh xạ thành mã nguồn thực sự và cho phép thêm những chức năng mới hay những khái niệm mới mà không ảnh hưởng đến phần còn lại của hệ thống.

Kiến trúc một hệ phần mềm được hiểu là một quá trình xác định một giải pháp để đưa ra một cấu trúc thích hợp nhằm đáp ứng được các yêu cầu về mặt kỹ thuật và hoạt động của một tổ chức hay doanh nghiệp đồng thời có thể tối ưu hóa về mặt hiệu năng, an toàn và khả năng quản lý. Quá trình này liên quan đến một loạt các quyết định có thể có tác động thực sự đến chất lượng, hiệu năng, bảo trì và thành công chung của một ứng dụng.

Nhiều nhà nghiên cứu đã giải thích về kiến trúc phần mềm, và họ có các quan điểm khác nhau về cách trình bày tốt nhất về kiến trúc của hệ thống phần mềm. Không có cách giải thích nào là sai; mỗi kiến trúc có giá trị riêng. Định nghĩa của Bass L, *và cộng sự* nắm giữ được điểm cốt yếu mà kiến trúc phần mềm đòi hỏi:

*“Kiến trúc phần mềm của một chương trình hoặc hệ thống tính toán là cấu trúc hoặc các cấu trúc của hệ thống đó, gồm các thành phần của phần mềm, các thuộc tính có thể trông thấy được từ bên ngoài của các thành phần này, và các mối quan hệ giữa chúng*.” (Bass, Clements, Kazman)

Một định nghĩa khác về Kiến trúc phần mềm được đưa ra bởi: Philippe Kruchten, Grady Booch, Kurt Bittner, và Rich Reitman bắt nguồn và tinh chỉnh một định nghĩa về kiến ​​trúc dựa trên tác phẩm của Mary Shaw và David Garlan (Shaw and Garlan 1996). Định nghĩa của họ là:

*"Kiến trúc phần mềm bao gồm một loạt các quyết định quan trọng về việc tổ chức một hệ thống phần mềm bao gồm việc lựa chọn các yếu tố cấu trúc và giao diện của chúng bằng cách tạo ra hệ thống; hành vi được xác định trong hợp tác giữa các yếu tố đó; thành phần của các yếu tố cấu trúc và hành vi này vào các hệ thống con lớn hơn; và một mẫu kiến ​​trúc hướng dẫn tổ chức này. Kiến trúc phần mềm cũng bao hàm các chức năng, khả năng sử dụng, khả năng phục hồi, hiệu năng, tái sử dụng, tính dễ hiểu, các ràng buộc về kinh tế và công nghệ, sự cân bằng và mối quan tâm thẩm mỹ*. "

Hay

*“Kiến trúc phần mềm là một tập hợp các quyết định mà nếu làm không đúng sẽ là nguyên nhân làm cho dự án của bạn thất bại”* (Eoin Woods)

IEEE Std (Standard) 1471 định nghĩa kiến trúc hệ thống là “*tổ chức cơ bản của một hệ thống thể hiện trong các phần tử của nó, mối quan hệ của chúng với nhau và với môi trường và các nguyên tắc hướng dẫn thiết kế và phát triển của nó*” (Maier, Emery, Hilliard)

Theo Martin Fowler cũng đã chỉ ra một số chủ đề phổ biến khi giải thích về kiến trúc như sau: *việc phân rã hệ thống ở mức cao thành các bộ phận, những quyết định khó thay đổi, nhiều kiến trúc trong hệ thống, cái gì có thể thay đổi trong vòng đời của hệ thống…*

Trong các định nghĩa này, các phần tử kiến trúc có thể là một mô-đun, hệ thống con, đối tượng, hoặc phần mềm nhị phân (binary) như là một thành phần DLL, một thành phần JavaBean, EJB, CORBA, hoặc web, hoặc thậm chí là cả một hệ thống. Trong cuốn sách này, chúng tôi sử dụng “phần tử - element” để chỉ các đơn vị chung của kiến trúc phần mềm, và chúng tôi sử dụng “thành phần - component” làm từ đồng nghĩa của nó trong các cuộc thảo luận liên quan đến kiến trúc phần mềm.

Mục tiêu của kiến trúc là chỉ ra được các yêu cầu có ảnh hưởng như thế nào đến cấu trúc của ứng dụng. Kiến trúc tốt có thể giảm được rủi ro về nghiệp vụ trong khi xây dựng một giải pháp kỹ thuật và có thể xử lý được các sự cố xảy ra liên quan đến phần cứng hay phần mềm trong vòng đời của nó. Kiến trúc sư phần mềm cần phải xem xét ảnh hưởng tổng thể của những quyết định thiết kế, sự cân bằng vốn có của các thuộc tính chất lượng (như hiệu năng và bảo mật) và sự cân bằng cần thiết của yêu cầu người dùng, hệ thống và yêu cầu nghiệp vụ. Sau đây là một số hướng dẫn khi kiến trúc một hệ thống.

* Kiến trúc cần thể hiện được cấu trúc (structure) của hệ thống nhưng ẩn dấu các chi tiết cài đặt
* Kiến trúc phải hiện thực hóa được tất cả các ca sử dụng và kịch bản
* Kiến trúc phải thể hiện được sự quan tâm đến các yêu cầu của các bên liên quan
* Kiến trúc phải đáp ứng cả yêu cầu về chức năng lẫn yêu cầu phi chức năng.

Thiết kế phần mềm phụ thuộc vào các đặc tả yêu cầu phần mềm (Software Requirement Specification SRS) được tạo ra bằng cách phân tích trong bước đầu tiên của SDLC. Quy trình xác định yêu cầu bao gồm mô hình hóa miền thông tin, mô hình hóa chức năng, mô hình hóa dữ liệu, mô hình hóa hành vi, và mô hình hóa giao diện người dùng. Có hai khía cạnh của yêu cầu phần mềm: chức năng và phi chức năng. Một yêu cầu chức năng xác định chức năng của hệ thống phần mềm, trong khi một yêu cầu phi chức năng chỉ định chất lượng, các ràng buộc và hành vi của hệ thống.

Có rất nhiều cơ chế được sử dụng để xác định một yêu cầu phần mềm. Sơ đồ hình hộp và đường thẳng nổi tiếng trong Hình 1.2 cho thấy mô hình phân tích khái niệm của hệ thống, đây có thể là điểm khởi đầu cho thiết kế kiến ​​trúc phần mềm. Tuy nhiên, một sơ đồ hình hộp và đường thẳng không thể cung cấp đầy đủ ngữ nghĩa của thiết kế kiến ​​trúc phần mềm vì nó không cung cấp thông tin cần thiết cho việc phát triển phần mềm trong giai đoạn tiếp theo. Các ký hiệu mô tả khác cũng báo cáo kết quả phân tích yêu cầu; chúng bao gồm các đặc tả ca sử dụng của ngôn ngữ mô hình hóa (Unified Modelling Language – UML), biểu đồ luồng dữ liệu (DFD), các biểu đồ chuyển trạng thái (STD). Tất các ký hiệu và công cụ này có thể giúp các nhà thiết kế phần mềm hiểu rõ hơn về các yêu cầu phần mềm. Tuy nhiên, chúng là mô hình khái niệm để phân tích chứ không phải mô tả kiến trúc phần mềm.

Một đặc tả kiến trúc phần mềm hoàn chỉnh phải mô tả không chỉ các thành phần và sự kết nối giữa các thành phần mà còn bao gồm cả các ràng buộc và hành vị thời gian chạy để các nhà phát triển biết những gì và làm thế nào thiết kế được thực thi.

Sau đây, liệt kê nhiệm vụ của một kiến trúc sư phần mềm:

* Thực hiện phân vùng tĩnh và sự phân rã hệ thống thành các hệ thống con và giao tiếp giữa các hệ thống con. Một phần tử phần mềm có thể được cấu hình, phân phối, phát triển và triển khai và có thể được thay thế trong tương lai. Giao diện của mỗi phần tử đóng gói các chi tiết và cung cấp các kết nối lỏng lẻo với các phần tử hoặc hệ thống con khác.
* Thiết lập các mối quan hệ điều khiển động giữa các hệ thống con khác nhau về luồng dữ liệu, điều phối luồng điều khiển, hoặc gửi thông điệp (message dispatching).
* Xem xét và đánh giá các kiểu kiến trúc thay thế phù hợp với miền vấn đề hiện tại.
* Thực hiện phân tích cân bằng về các thuộc tính chất lượng và các yêu cầu phi chức năng trong việc quá trình lựa chọn kiểu kiến trúc. Việc lựa chọn loại phần tử và loại kết nối sẽ có tác động trực tiếp đến các thuộc tính hệ thống và các thuộc tính chất lượng của nó. Nhiều thuộc tính chất lượng phải được tính đến sớm trong giai đoạn thiết kế. Ví dụ, để tăng khả năng mở rộng, tính khả chuyển hoặc khả năng bảo trì của hệ thống phân tán, các thành phần phần mềm và dịch vụ Web có thể là lựa chọn tốt nhất trong các loại phần tử và kết nối lỏng lẻo giữa các phần tử này có thể thích hợp nhất. Các kiến trúc sư cần có các bên liên quan tham gia vào quá trình này.

Công việc quan trọng nhất của một kiến trúc sư phần mềm là ánh xạ đặc tả yêu cầu phần mềm (SRS) đến thiết kế kiến trúc phần mềm và đảm bảo rằng các yêu cầu chức năng và phi chức năng được đáp ứng. Nếu không thể đáp ứng tất cả các yêu cầu, nhà phân tích hệ thống và kiến trúc sư phần mềm có thể sử dụng thiết kế kiến trúc để trao đổi với các bên liên quan.

## **1.3 Một số khái niệm cơ bản**

Trong các phần trình bày ở trên, chúng ta đã bàn một số vấn đề liên quan đến khái niệm của kiến trúc phần mềm như tính thành phần, kết nối…Phần này nhằm trình bày chi tiết hơn một số định nghĩa về các thuật ngữ và những ý tưởng cơ bản về kiến trúc phần mềm để làm cơ sở cho trình bày các chương về sau.

### **1.3.1. Kiến trúc**

Theo Taylor [18] *kiến trúc một hệ thống phần mềm là một tập hợp các quyết định thiết kế chủ yếu được đưa ra cho hệ thống cần xây dựng.* Như vậy, kiến trúc phần mềm được xem là những định hướng cho phát triển và tiến hóa một hệ thống phần mềm. Khái niệm quyết định thiết kế (design decision) là trung tâm của kiến trúc phần mềm nhằm kết nối các khái niệm khác liên quan đến kiến trúc. Sau đây là một số khái niệm liên quan đến quyết định thiết kế.

* Các quyết định liên quan đến cấu trúc (structure) của hệ thống. Ví dụ, các thành phần kiến trúc cần phải tổ chức và hợp thành các gói phân cấp theo kiểu 3 tầng
* Các quyết định liên quan đến hành vi chức năng (functional behavior). Ví dụ, xử lý dữ liệu, lưu trữ và hiển thị sẽ được thực thi theo một thứ tự nghiêm ngặt.
* Các quyết định liên quan đến tương tác (interaction). Ví dụ, giao tiếp giữa các thành phần của hệ thống chỉ xảy ra bằng cách sử dụng thông báo, sự kiện hay giao thức RMI trong java.
* Các quyết định liên quan đến các tính chất phi chức năng (non-function properties) của hệ thống. Ví dụ khả năng phụ thuộc của hệ thống sẽ được đảm bảo bằng cách phục hồi các mô đun xử lý.
* Các quyết định liên quan đến cài đặt (implementation) hệ thống. Ví dụ, thành phần giao diện người sử dụng sẽ được xây dựng bằng cách sử dụng Java Swing hay JSP.

Rõ ràng các quyết định quan trọng khác nhau dẫn đến các kiểu kiến trúc khác nhau. Tuy nhiên, một số quyết định không được xem là chủ yếu và không ảnh hưởng đến kiến trúc của hệ thống như chi tiết của thuật toán hay cấu trúc dữ liệu

### **1.3.2. Thành phần**

Các quyết định tạo nên kiến trúc của một hệ thống phần mềm bao gồm việc kết hợp nhiều phần tử khác biệt nhưng có liên quan với nhau. Các phần tử này bao gồm những khía cạnh khác nhau:

* Xử lý: liên quan đến chức năng hay hành vi
* Trạng thái: liên quan đến thông tin hay dữ liệuAn architecture will inhibit or enable a system’s driving quality attributes
* Tương tác: liên quan đến cộng tác, phối hợp

Những phần tử đóng gói xử lý và dữ liệu bên trong kiến trúc hệ thống gọi là thành phần phần mềm (software component). Theo Taylor [18]:

*Thành phần phần mềm là một thực thể kiến trúc: (i) đóng gói một tập con chức năng và dữ liệu của hệ thống; (ii) chỉ có thể truy nhập vào thực thể thông qua một giao diện; (iii) xác định các phụ thuộc tùy theo ngữ cảnh mà nó yêu cầu.*

Như vậy, thành phần phần mềm là sự tích hợp của tính toán và trạng thái trong hệ thống. Tùy theo kiểu kiến trúc, quan điểm người thiết kế hay yêu cầu của hệ thống, thành phần có thể đơn giản như một thao tác, một phương thức hay phức tạp như một phần hệ thống. Đặc trưng quan trọng nhất của thành phần là chỉ được nhìn thấy từ bên ngoài qua giao diện. Thành phần phần mềm được cho là thể hiện đầy đủ các nguyên lý đóng gói (encapsulation, trừu tượng hóa (abstraction) và module hóa (modularity) của kỹ thuật phần mềm. Như vậy các thành phần hàm ý có khả năng kết hợp, sử dụng lại và tiến hóa.

Tuy nhiên, các khả năng này lại được thể hiện trong ngữ cảnh tạo ra thành phần như giao diện phần mềm, tính có sẵn của tài nguyên (dữ liệu…) thùng chứa thành phần, môi trường thực thi của chương trình, hệ điều hành, giao thức mạng, cấu hình phần cứng…

### **1.3.3. Kết nối**

Một đặc trưng cơ bản của hệ thống phần mềm là tương tác (interaction) giữa các thành phần trong hệ thống. Nhiều hệ thống hiện đại được xây dựng với nhiều thành phần phức tạp, phân tán qua mạng và cập nhật liên tục suốt vòng đời của nó. Do đó, đối với người phát triển việc đảm bảo tương tác giữa các thành phần thậm chí quan trọng hơn chính chức năng của bản thân các thành phần.

*Kết nối là một phần tử kiến trúc có nhiệm vụ thực thi tương tác giữa các thành phần*

Đối với hệ thống phần mềm trên máy để bàn truyền thống, các kết nối thường được thể hiện bởi lời gọi thủ tục hay truy nhập dữ liệu chung. Các kết nối này thưởng chỉ thể hiện dữ tương tác giữa các cặp thành phần. Gọi thủ tục được cài đặt trực tiếp trong các ngôn ngữ lập trình để giúp trao đổi đồng bộ dữ liệu và điều khiển giữa các cặp thành phần. Khi đó, thành phần triệu gọi chuyển luồng điều khiển cũng như dữ liệu dưới dạng tham số đến thành phần được và sau khi thực hiện yêu cầu, bên được gọi trả điều khiển cũng như kết quả tính toán cho bên gọi. Trong truy nhập dữ liệu, kiểu kết nối được thể hiện dưới dạng biến toàn cục hay bộ nhớ chung. Kiểu kết nối này cho phép nhiều thành phần tương tác bằng cách đọc hay ghi vào các phương tiện chung. Tương tác được phân bố không đồng bộ theo thời gian nghĩa là việc đọc không phục thuộc hay ràng buộc theo thời gian của việc ghi và ngược lại.

Tuy nhiên, trong các hệ thống phần mềm phức tạp trên mạng, các kết nối thường xảy ra cùng một lúc giữa một thành phần với nhiều thành phần dịch vụ khác nhau. Dựa vào cách thực thi dịch vụ tương tác, người ta thường phân các kết nối thành phần thành tám kiểu khác nhau [18]:

* Kết nối gọi thủ tục: ví dụ phương thức trong lập trình hướng đối tượng
* Kết nối sự kiện: ví dụ trong ứng dụng windows, các input GUI xem như sự kiện kích hoạt hệ thống
* Kết nối truy nhập dữ liệu: ví dụ các cơ chế truy vấn trong cơ sở dữ liệu SQL
* Kết nối liên kết: được sử dụng để ràng buộc các thành phần cùng giữ một trạng thái suốt trong quá trình tương tác với nhau.
* Kết nối luồng: được sử dụng để truyền một khối lượng lớn dữ liệu giữa các tiến trình tự chủ riêng. Ví dụ, các socket TCP/UDP hay các giao thức client – server như RMI, CORBA, FTP, SOAP.
* Kết nối trung gian: được sử dụng khi các thành phần biết sự có mặt của các thành phần khác nhưng không thể biết được nhu cầu, trạng thái của nó. Khi đó, bộ phận trung gian sẽ cung cấp dịch vụ phối hợp hay giải quyết tranh chấp...
* Kết nối thích nghi: cung cấp phương tiện để hỗ trợ tương tác giữa các thành phần không được thiết kế với tương tác. Ví dụ hệ phân tán có thể dựa vào gọi thủ tục từ xa (RPC: remote procedure call) cho mọi người tương tác.
* Kết nối theo tuyến: được sử dụng để chỉ ra các tuyến tương tác và thực thi định tuyến truyền thông cũng như phối hợp giữa các thành phần theo tuyến này.

Thông thường, các hệ phân tán sử dụng nhiều cách kết nối với nhau. Ví dụ, trong tính toán lưới, kết nối phân tán dữ liệu dựa trên hợp của bốn kiểu kết nối: gọi thủ tục, truy nhập dữ liệu, luồng và kết nối theo tuyến. Các kết nối này phân phối một khối lượng lớn dữ liệu giữa các thành phần trong môi trường này. Trong hệ P2P, kết nối phân tán gồm bốn kiểu: trung gian, truy nhập dữ liệu, luồng và kết nối theo tuyến. Trong các hệ phân tán dựa trên mô hình client – server, kết nối gồm bốn kiểu: gọi thủ tục, truy nhập dữ liệu, luồng và kết nối theo tuyến.

### **1.3.4 Cấu hình**

Các thành phần và kết nối thường được kết hợp với nhau theo cách nào đó trong cấu trúc để xác lập mục tiêu của hệ thống. Cách kết hợp này thể hiện cấu hình hay topo của hệ thống. Như vậy, cấu hình của kiến trúc là tập hợp các cách kết hợp giữa thành phần và kết nối của kiến trúc phần mềm.

Cấu hình có thể biểu diễn như một đồ thị trong đó các đỉnh biểu diễn thành phần và kết nối, các cạnh biểu diễn sự kết hợp.

## **1.4 Các mẫu kiến trúc phần mềm (Architecture Styles)**

Một mẫu kiến trúc (architecture pattern) tóm tắt các thuộc tính chung của một nhóm các thiết kế tương tự. Mẫu kiến trúc bao gồm một tập hợp các quy tắc, ràng buộc và các mẫu về cách cấu trúc một hệ thống thành một tập hợp các thành phần và sự kết nối giữa chúng. Nó điểu chỉnh các mô hình thiết kế cấu trúc tổng thể các loại thành phần cấu thành và sự tương tác trong thời gian chạy của chúng về điều khiển luồng và truyền dữ liệu. Các thành phần chính của mẫu kiến trúc bao gồm:

* Các thành phần thực hiện các chức năng theo yêu cầu của một hệ thống
* Trình kết nối cho phép truyền thông, phối hợp và sự hợp tác giữa các thành phần.
* Các ràng buộc xác định cách các thành phần có thể được tích hợp để tạo thành một hệ thống
* Các thuộc tính mô tả những lợi thế và hạn chế của kiến trúc đã chọn

Ví dụ, trong mẫu kiến trúc tập trung dữ liệu, kho lưu trữ dữ liệu đóng vai trò trung tâm và nó được truy cập thường xuyên bởi các phần tử sửa đổi dữ liệu khác. Trong mẫu kiến trúc luồng dữ liệu, dữ liệu đầu vào được biến đổi bởi một loạt các phần tử tính toán hoặc thao tác. Theo mẫu kiến trúc gọi và trả lại, các hàm và thủ tục là các phần tử được tổ chức theo hệ thống phân cấp điều khiển với một chương trình chính gọi một số chương trình con. Trong mẫu kiến trúc hướng đối tượng, các phần tử được biểu diễn dưới dạng các đối tượng đóng gói dữ liệu và hoạt động, và giao tiếp giữa chúng là bằng cách truyền thông điệp. Theo kiểu phân lớp, mỗi mô-đun hoặc gói hoàn thành các nhiệm vụ tiến triển trong khuôn khổ từ trừu tượng cấp cao hơn đến triển khai cấp thấp hơn. Tất cả các mẫu này sẽ được thảo luận chi tiết trong các chương sau.

## **1.5. Mẫu thiết kế (design pattern)**

Mẫu thiết kế là giải pháp lặp đi lặp lại đối với vấn đề nảy sinh trong vòng đời của ứng dụng phần mềm. Mục đích của mẫu thiết kế là giúp cải tiến chất lượng phần mềm (sử dụng lại, bảo trì, mở rộng…); giảm thời gian phát triển. Mục đích là giúp cải tiến chất lượng phần mềm (sử dụng lại, bảo trì, mở rộng) và giảm thời gian phát triển.

Mẫu thiết kế được phân thành các loại sau:

* Nhóm mẫu tạo dựng: Abstract Factory, Factory Method, Builder, Prototype và Singleton.
* Nhóm mẫu cấu trúc: Adapter, Bridge, composite, Decorator, Proxy, Fly Weight, và Facade
* Nhóm mẫu hành vi: Interpreter, Template Method, Chain of Responsibility, Command, Editor, Mediator, Memento, Observer, State, Strategy, và Visitor.

Chúng ta sẽ tìm hiểu kỹ hơn về các mẫu thiết kế ở chương 8

## **1.6 Các thuộc tính chất lượng**

Mỗi kiểu kiến trúc có ưu điểm, nhược điểm và tiềm ẩn nhiều rủi ro. Lựa chọn đúng kiểu kiến trúc để đáp ứng các yêu cầu chức năng và các thuộc tính chất lượng là rất quan trọng. Các thuộc tính chất lượng được xác định trong quá trình phân tích yêu cầu. Các thuộc tính chất lượng được phân thành ba nhóm sau:

1. Nhóm thuộc tính thực thi (không thể quan sát được trong thời gian chạy):
   1. Khả năng tương tác: khả năng truy cập và trao đổi dữ liệu giữa các thành phần bên trong và với các thành phần bên ngoài. Khả năng tương tác đòi hỏi sự phụ thuộc lỏng lẻo của cơ sở hạ tầng
   2. Khả năng bảo trì và mở rộng: khả năng sửa đổi hệ thống và mở rộng nó một cách thuận tiện
   3. Khả năng kiểm thử: mức độ mà hệ thống tạo điều kiện cho việc xây dựng các trường hợp kiểm thử. Khả năng kiểm thử thường yêu cầu một bộ tài liệu hoàn chỉnh đính kèm theo thiết kế hệ thống và triển khai hệ thống
   4. Tính khả chuyển: mức độ độc lập của hệ thống trên các nền tảng phần mềm và phần cứng. Các hệ thống được phát triển bằng ngôn ngữ lập trình mức cao thường có tính di động tốt. Một ví dụ điển hình là Java - hầu hết các chương trình Java chỉ cần được biên dịch một lần và có thể chạy ở khắp mọi nơi
   5. Khả năng mở rộng: khả năng của hệ thống thích ứng với sự gia tăng yêu cầu của người dùng. Khả năng mở rộng hệ thống gây bất lợi trong thiết kế hệ thống.
   6. Tính linh hoạt: dễ dàng sửa đổi hệ thống để phục vụ cho môi trường khác nhau hoặc các vấn đề mà hệ thống không được thiết kế ban đầu. Các hệ thống được phát triển bằng cách sử dụng kiến trúc dựa trên cấu phần hoặc kiến trúc hướng dịch vụ thường có thuộc tính này.
2. Nhóm thuộc tính thời gian chạy (quan sát tại thời gian chạy)
   1. Tính khả dụng (availability): khả năng của hệ thống luôn sẵn sang 24/7. Tính khả dụng có thể đạt được qua sao chép và thiết kế cẩn thận để đối phó với những lỗi phần cứng, phần mềm hoặc mạng
   2. Tính bảo mật (security): khả năng của hệ thống đối phó với các cuộc tấn công nguy hiểm từ bên ngoài hoặc bên trong hệ thống. Bảo mật có thể được cải thiện bằng cách cài đặt tường lửa, thiết lập quá trình xác thực, và ủy quyền cũng như sử dụng mã hóa
   3. Hiệu suất (performance): tăng hiệu quả của hệ thống liên quan đến thời gian phản hồi, thông lượng và sử dụng tài nguyên, các thuộc tính thường xung đột với nhau.
   4. Tính sử dụng (usability): mức độ hài lòng của người dùng khi sử dụng hệ thống. Khả năng sử dụng bao gồm các vấn đề về tính hoàn chỉnh, tính đúng đắn, khả năng tương thích, cũng như giao diện người dùng thân thiện, tài liệu đầy đủ và hỗ trợ kỹ thuật.
   5. Độ tin cậy (reliability): tần số lỗi, độ chính xác kết quả đầu ra, khả năng phục hồi sau lỗi, và khả năng dự đoán (**MTTF- Mean time to failure** (dùng cho chi tiết, thiết bị cần được thay thế và sửa chữa. Nghĩa là nó cho biết khoảng thời gian từ khi máy chạy tới lần hư hỏng tiếp theo))
   6. Khả năng bảo trì (maintainability – khả năng mở rộng, khả năng thích ứng, khả năng kiểm thử, tính tương thích và khả năng cấu hình): sự dễ dàng của việc thay đổi phần mềm
3. Nhóm thuộc tính nghiệp vụ (business):
   1. Time to market: thời gian đưa ra thị trường: thời gian từ phân tích yêu cầu đến ngày sản phẩm được phát hành
   2. Chi phí: các chi phí xây dựng, duy trì và vận hành hệ thống,
   3. Thời gian sống (Lifetime): khoảng thời gian mà sản phẩm “còn sống” trước khi “nghỉ hưu”.

Trong rất nhiều trường hợp, không có kiểu kiến trúc nào có thể đáp ứng đồng thời tất cả các thuộc tính chất lượng. Các kiến trúc sư phần mềm thường cần cân bằng giữa các thuộc tính. Các cặp cân bằng thuộc tính chất lượng điển hình bao gồm:

* Cân bằng giữa không gian và thời gian: Ví dụ, để tăng hiệu quả thời gian của một bảng băm đồng nghĩa với việc giảm hiệu quả không gian của nó
* Cân bằng giữa độ tin cậy và hiệu suất: Ví dụ: các chương trình Java cũng được bảo vệ tốt chống tràn bộ nhớ do các biện pháp bảo mật như kiểm tra ranh giới trên mảng. Các tính năng đáng tin cậy như vậy đi kèm với chi phí hiệu quả thời gian, so với ngôn ngữ C đơn giản hơn và nhanh hơn, cung cấp những con trỏ “nguy hiểm” nhưng hiệu quả.
* Cân bằng giữa khả năng mở rộng và hiệu suất: ví dụ một cách tiếp cận điển hình để tăng khả năng mở rộng của dịch vụ là tái tạo máy chủ. Ví dụ để đảm bảo ràng mỗi máy chủ có cùng một dữ liệu nhất quán về mặt logic), hiệu suất của toàn bộ dịch vụ sẽ bị ảnh hưởng.

Khi một mẫu kiến trúc không đáp ứng được tất cả các thuộc tính chất lượng mong muốn, kiến trúc sư phần mềm làm việc với nhà phân tích hệ thống và các bên liên quan để xác định các ưu tiên của các thuộc tính chất lượng. Bằng cách liệt kê các thiết kế kiến trúc thay thế và tính toán đánh giá trọng số của các thuộc tính chất lượng, kiến trúc sư phần mềm có thể chọn thiết kế tối ưu.

## **1.7 Hướng dẫn thiết kế kiến trúc phần mềm**

Trong phần sau, cuốn sách cung cấp một số quy tắc chung để giúp các nhà phát triển phần mềm hiểu rõ hơn về các yêu cầu, xác định các kiểu kiến trúc phù hợp để phân rã một hệ thống phức tạp thành các phần tử cấu thành của nó, chọn thành phần và loại kết nối thích hợp, đáp ứng các yêu cầu của các bên liên quan về các thuộc tính chất lượng và cung cấp các chiến thuật thực thi phù hợp để thực hiện hiệu quả.

- Nghĩ về việc cần làm trước khi nghĩ cách thực hiện. Các yêu cầu chức năng và phi chức năng cần được xác định, xác minh và xác nhận trước khi công việc thiết kế kiến trúc và chi tiết được thực hiện. Sử dụng thiết kế kiến trúc trừu tượng để giao tiếp với các bên liên quan giúp tránh phải sửa chữa lại thiết kế hệ thống trong các giai đoạn sau của chu trình phát triển phần mềm.

Một thiết kế kiến trúc thành công dựa trên phân tích yêu cầu lặp đi lặp lại vốn có. Lưu ý rằng các bên liên quan khác nhau của hệ thống phần mềm có mối quan tâm riêng của họ. Các kiến trúc sư phần mềm cần xác nhận những gì cần thiết và những gì có thể được cân nhắc lựa chọn. Ví dụ, các nhà đầu tư của một dự án thường quan tâm đến ngày phát hành hệ thống, ngân sách, khả năng sử dụng, v.v.; trong khi người dùng cuối của cùng một dự án quan tâm đến hiệu suất, độ tin cậy và các kịch bản sử dụng. Do đó, kiến trúc sư phần mềm phải quan tâm đến việc phân tích cân bằng các thuộc tính chất lượng, cũng như tính hoàn chỉnh và nhất quán của kiến trúc. Mặt khác, các nhà phát triển phần mềm tập trung vào việc triển khai và quan tâm đến việc liệu thiết kế phần mềm có đủ chi tiết để viết mã hay không. Các nhà quản lý dự án phần mềm có thể quan tâm đến sự phát triển và bảo trì kiến trúc phần mềm trong tương lai.

- Hãy nghĩ đến thiết kế trừu tượng trước khi nghĩ đến thiết kế cụ thể. Luôn bắt đầu với một thiết kế trừu tượng chỉ định giao diện của các thành phần và kiểu dữ liệu trừu tượng. Sử dụng nhiều cấp độ trừu tượng nếu cần thiết. Đưa ra các quyết định triển khai dựa trên các giao diện trừu tượng thay vì các giao diện cụ thể vì các giao diện đó ổn định hơn — chúng là các hợp đồng giữa nhà cung cấp dịch vụ và người yêu cầu dịch vụ, vì vậy chúng được xác định ở giai đoạn đầu của chu trình phát triển phần mềm.

- Hãy sớm nghĩ đến các yêu cầu phi chức năng trong quá trình thiết kế. Khi bạn ánh xạ các yêu cầu chức năng cho một thiết kế kiến ​​trúc, bạn cũng nên xem xét các yêu cầu phi chức năng. Giao tiếp với các bên liên quan và ghi lại sở thích của họ đối với các thuộc tính chất lượng. Nếu không thể tìm thấy một thiết kế đáp ứng tất cả các thuộc tính chất lượng, hãy cố gắng tìm sự cân bằng phù hợp của các thuộc tính chất lượng và xem xét các mẫu kiến ​​trúc không đồng nhất khi cần thiết.

- Hãy nghĩ đến khả năng tái sử dụng và khả năng mở rộng của phần mềm càng nhiều càng tốt. Đối với hầu hết các hệ thống phần mềm, có khả năng là các chức năng mới sẽ được thêm vào sau khi hệ thống được triển khai. Bạn cần xem xét cách sử dụng lại các thành phần phần mềm hiện có để tăng độ tin cậy và hiệu quả chi phí của hệ thống mới. Luôn cố gắng làm cho phần mềm có thể mở rộng trong tương lai.

- Cố gắng thúc đẩy sự gắn kết cao trong mỗi phần tử và sự liên kết lỏng lẻo giữa các phần tử. Một hệ thống con, thành phần hoặc mô-đun có tính thống nhất cao thực hiện một chức năng duy nhất. Ví dụ, trong thiết kế hướng đối tượng, nếu một lớp được chỉ định chịu hai trách nhiệm không liên quan, nó được coi là không mạch lạc. Bạn phải xem xét các yếu tố gắn kết trong giai đoạn đầu của quá trình thiết kế. Tính liên kết thấp của một hệ thống ngụ ý rằng thành phần chức năng không được thiết kế tốt; ví dụ, một chức năng đơn lẻ có thể nằm rải rác trên một số lượng lớn các thành phần khác nhau, làm cho nó rất khó bảo trì. Mỗi mẫu kiến ​​trúc cần thể hiện sự phân chia rõ ràng giữa các yếu tố để đảm bảo sự liên kết lỏng lẻo. Trong hầu hết các trường hợp, khớp nối lỏng lẻo có nghĩa là ít phụ thuộc lẫn nhau hơn giữa các thành phần, vì vậy sự thay đổi của một thành phần không có khả năng gây ra sự thay đổi gợn sóng của các thành phần khác. Thuộc tính khớp nối có thể được đo bằng số lượng chữ ký giao diện. Truyền thông điệp và truyền thông không đồng bộ là những ví dụ điển hình về sự kết hợp lỏng lẻo giữa người yêu cầu dịch vụ và nhà cung cấp dịch vụ. Ví dụ, một cuộc trò chuyện qua email có mối ràng buộc lỏng lẻo hơn nhiều so với cuộc trò chuyện qua điện thoại.

- Khoan dung tinh chỉnh thiết kế. Đừng bao giờ mong đợi thiết kế phần mềm hoàn toàn hoàn hảo trong một bước. Bạn có thể cần sử dụng tạo mẫu và lặp lại để tinh chỉnh thiết kế

- Tránh thiết kế mơ hồ và thiết kế quá chi tiết. Thiết kế mơ hồ thiếu các ràng buộc và thiết kế quá chi tiết hạn chế việc triển khai.

Các thiết kế kiến trúc được mô tả như thế nào? Ký hiệu UML là một trong nhiều giải pháp, ngoài tài liệu văn bản, có sẵn cho các nhà thiết kế. UML cung cấp các ký hiệu đồ họa có sẵn cho kiến trúc sư và nhà thiết kế trong gần như mọi giai đoạn của SDLC, ví dụ: sơ đồ ca sử dụng để ghi lại các yêu cầu hệ thống, sơ đồ lớp để mô tả cấu trúc logic của hệ thống, sơ đồ máy trạng thái và sơ đồ tương tác để chỉ định động các hành vi của một hệ thống. Mô hình quan điểm “4 + 1”, được phát triển bởi P. B. Kruchten, là một cách để thể hiện các quan điểm khác nhau của một hệ thống phần mềm, từ quan điểm của các bên liên quan khác nhau. Nó đặc biệt hữu ích trong việc mô tả một tập hợp đầy đủ các yêu cầu chức năng và phi chức năng. Một lựa chọn khác là sử dụng Ngôn ngữ mô tả kiến trúc (ADL) để chỉ định chính thức cấu trúc và ngữ nghĩa của kiến trúc phần mềm.

## **1.8 Tóm tắt**

Thiết kế kiến trúc phần mềm đã nổi lên như một phần quan trọng của phát triển phần mềm. Đặc tả kiến trúc phần mềm bao gồm các phần tử phần mềm, trình kết nối và sự cộng tác giữa các phần tử và các thuộc tính chất lượng mong muốn. Mẫu kiến trúc là một tập hợp các quy tắc, ràng buộc hoặc mẫu hướng dẫn cách cấu trúc hệ thống thành một tập hợp các phần tử và trình kết nối, cũng như cách chi phối các mẫu thiết kế cấu trúc tổng thể của các loại phần tử cấu thành và tương tác thời gian chạy của chúng. Một kiểu kiến trúc cụ thể có thể không đáp ứng tất cả các thuộc tính chất lượng mong muốn của hệ thống, trong trường hợp đó phải thực hiện cân bằng. Vì vậy, làm thế nào để cân bằng hợp lý các thuộc tính chất lượng là một vấn đề thiết kế quan trọng.

# CHƯƠNG 2: MÔ HÌNH HÓA KIẾN TRÚC

Chương này trình bày một số khái niệm cơ bản nhằm thể hiện kiến trúc từ những thành phần đơn giản đến các tính chất phức tạp hơn của hệ thống như các hành vi. Các ký hiệu mô hình kiến trúc có thể khá phong phú và nhập nhằng như ngôn ngữ tự nhiên đến ngôn ngữ rất hình thức và hạn chế ngữ nghĩa như ngôn ngữ mô tả kiến trúc ADL. Mô hình với ngôn ngữ UML có thể được xem là sự kết hợp giữa mô tả bằng biểu đồ với diễn đạt bằng ngôn ngữ tự nhiên và sẽ được sử dụng cho mô hình kiến trúc trong tài liệu này. Nội dung của chương này bao gồm:

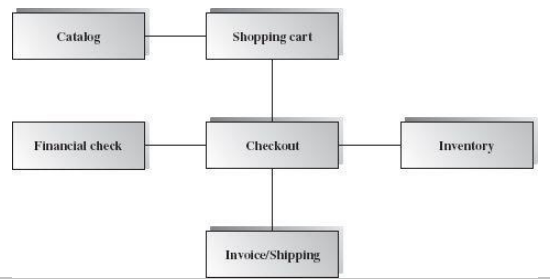
* Các khái niệm cơ bản về mô hình kiến trúc
* Các kỹ thuật mô hình
* Biểu diễn kiến trúc

## **2.1 Tổng quan**

Kiến trúc phần mềm xác định mức độ trừu tượng cao của hệ thống phần mềm bằng cách phân rã, xác định các thành phần, các kiểu kiến trúc và các thuộc tính chất lượng. Mọi kiến trúc phần mềm phải mô tả tập các thành phần của nó, kết nối và tương tác giữa các thành phần này. Nó cũng phải xác định cấu hình triển khai của tất cả các thành phần và kết nối. Ngoài ra thiết kế kiến trúc phần mềm phải phù hợp với các yêu cầu chức năng và phi chức năng của hệ thống.

Có rất nhiều cách để mô tả kiến trúc phần mềm. Biểu đồ dạng hộp và đường thường được sử dụng để mô tả các khái niệm và quy trình nghiệp vụ trong giai đoạn phân tích của vòng đời phát triển phần mềm. Các sơ đồ này đi kèm với các mô tả về các thành phần và kết nối, cũng như các mô tả khác cung cấp các diễn giải trực quan thông thường. Sơ đồ dạng hộp và đường sẽ được sử dụng xuyên suốt cuốn sách này cho các mục đích đặc tả.

Hình 2.1. trình bày sơ đồ hộp và đường cho một doanh nghiệp mua sắm trực tuyến nơi khách hàng duyệt qua danh mục và đặt các mặt hàng đã chọn của họ vào giỏ hàng. Sau khi khách hàng thanh toán hệ thống sẽ kiểm tra hồ sơ tín dụng của khách hàng, cập nhật hàng tồn kho và thông báo cho bộ phận vận chuyển để xử lý đơn hàng.



Hình 2.1. Biểu đồ hộp và đường

Các đường trong biểu đồ hộp và đường biểu thị mối quan hệ giữa các thành phần. Lưu ý rằng, không giống UML, ngữ nghĩa của các dòng có thể khác nhau - chúng có thể đề cập đến sự phụ thuộc, luồng điều khiển, luồng dữ liệu...Các đường có thể kết hợp với các mũi tên để chỉ hướng và trình tự quy trình.

Sơ đồ hộp và đường có thể được sử dụng như một sơ đồ khái niệm nghiệp vụ mô tả các khái niệm quy trình và miền ứng dụng của nó. Loại sơ đồ này giúp chúng ta hiểu các khái niệm nghiệp vụ và lấy được các sơ đồ mô hình hoá và thiết kế phần mềm khác như UML. UML là một trong những giải pháp hướng đối tượng được sử dụng trong mô hình hoá và thiết kế phần mềm.

Mô hình khung nhìn 4+1 là một cách khác để hiển thị các yêu cầu chức năng và phi chức năng của một dự án phần mềm. Có năm khung nhìn khác nhau trong mô hình: khung nhìn logic, quy trình, phát triển, vật lý và giao diện người dùng. Khung nhìn logic được sử dụng để xác định các mô - đun phần mềm và ranh giới của chúng, giao diện, môi trường bên ngoài, tình huống sử dụng...Khung nhìn quy trình giải quyết các yêu cầu phi chức năng như kiểu giao tiếp mô- đun và các vấn đề hiệu suất trong thời gian chạy. Khung nhìn phát triển tổ chức các đơn vị phần mềm theo những cách được xác định rõ ràng theo cấu trúc tệp hoặc thư mục thực tế. Khung nhìn vật lý xác định cơ sở hạ tầng triển khai về phần mềm, phần cứng, cấu hình mạng và cài đặt cho mục đích phân phối. Tất cả những khung nhìn này đều hoạt động và được xác thực bởi khung nhìn kịch bản. Khung nhìn giao diện người dùng cung cấp giao diện của hệ thống và nó cũng thể ảnh hưởng đến các khung nhìn khác.

Ngôn ngữ mô tả kiến trúc (ADL - Architecture Description Language) là một cách khác để mô tả kiến trúc phần mềm một cách chính thức.

Kiến trúc là một tập hợp các quyết định thiết kế chính của hệ thống. Một khi các quyết định thiết kế được đưa ra, chúng ta sẽ được thể hiện bởi một mô hình và quá trình tạo ra mô hình đó được gọi là mô hình hóa. Mô hình thể hiện những quyết định thiết kế kiến trúc với những mức độ nghiêm ngặt và hình thức hóa khác nhau. Trong nội dung của chương này sẽ bàn đến một số khái niệm được sử dụng để mô hình kiến trúc và xem xét cách lựa chọn các ký hiệu khác nhau cho mô hình.

## **2.2 Yêu cầu của mô hình hóa kiến trúc**

Một trong những quyết định then chốt nhất mà những bên liên quan đến phát triển kiến trúc phải chọn.

* Các quyết định và các khái niệm cần phải mô hình
* Mức độ chi tiết của mô hình những khái niệm
* Mức độ hình thức cho mô hình

Kiến trúc sư phải quyết định lựa chọn cái gì để mô hình và mức độ chi tiết thế nào tùy theo chi phí và lợi ích của mỗi bên đối tác. Nguyên tắc ưu tiên là cái gì quan trọng nhất sẽ được mô hình chi tiết và được hình thức hóa nghiêm ngặt. Việc chọn đặc trưng nào quan trọng phụ thuộc vào dự án đang tiến hành. Các hoạt động cơ bản liên quan đến mô hình hóa bao gồm:

* Xác định các khía cạnh liên quan đến phần mềm cần mô hình và phân loại các khía cạnh theo độ quan trọng
* Xác định mục đích để mô hình cho mỗi khía cạnh
* Lựa chọn các ký hiệu sẽ được sử dụng để mô hình các khía cạnh với mức chi tiết phù hợp
* Tạo ra một mô hình bằng cách sử dụng ngôn ngữ mô hình phù hợp

Một số khái niệm kiến trúc cơ bản:

* Thành phần: thành phần là khối kiến trúc đóng gói một tập chức năng hay dữ liệu hệ thống và bên ngoài chỉ có thể truy nhập qua một giao diện xác định
* Kết nối: kết nối là khối kiến trúc quy định tương tác giữa các thành phần
* Giao diện: giao diện là những điểm mà các thành phần và kết nối tương tác với thế giới bên ngoài như các thành và kết nối khác
* Cấu hình: cấu hình là tập các liên kết giữa các thành phần và kết nối của kiến trúc hệ phần mềm

Những khái niệm này tạo thành điểm khởi đầu cho mô hình hóa kiến trúc. Ở mức cơ bản nhất, mô hình những khái niệm này đòi hỏi một tập ký hiệu có thể biểu diễn một đồ thị các thành phần và kết nối. Mô hình ở mức cơ bản này ít có giá trị vì nó không để biểu diễn các dự án phức tạp. Do đó, mô hình này cần phải được mở rộng bằng nhiều khái niệm khác. Một số câu hỏi sau đây được xem là trọng tâm của mô hình hóa kiến trúc. Các chức năng giữa các thành phần được phân rã như thế nào? Các giao diện có bản chất và các kiểu gì? Ý nghĩa liên kết của các thành phần và kết nối như thế nào? Các tính chất của hệ thống thay đổi như thế nào qua thời gian?

Tùy theo bản chất và miền ứng dụng của hệ thống định phát triển, các phần tử cơ bản có thể biểu diễn theo các cách khác nhau. Ví dụ, với phần mềm ứng dụng máy tính để bàn có thể có 5000 đến 1000 thành phần và kết nối, thì dễ dàng đánh số và mô tả các thành phần này cũng như các kết nối giữa chúng. Tuy nhiên, các ứng dụng phức tạp lớn và phân tán trên mạng thì khó để mô hình hóa hơn. Ví dụ, khó có thể đánh số các thành phần trong hệ phân tán trên mạng vì số lượng quá nhiều và thường xuyên thay đổi. Với các hệ như vậy, cách tốt nhất là chỉ mô hình hóa những phần nào của hệ thống được biểu diễn bởi các ca sử dụng hoặc chỉ mô hình hóa kiểu kiến trúc chi phối các phần tử trong kiến trúc.

## **2.3 Kiểu kiến trúc và mô hình hóa**

Như trình bày ở chương 1, một kiểu kiến trúc là một loạt các quyết định thiết kế kiến trúc áp dụng trong một ngữ cảnh phát triển nào đó. Nó thể hiện các ràng buộc của những quyết định thiết kế đối với hệ thống dự định phát triển. Cùng với mô hình hóa các phần tử kiến trúc cơ bản, chúng ta cũng cần mô hình kiểu kiến trúc chi phối cách sử dụng các phần tử này. Giống như kiến trúc, các kiểu kiến trúc được tạo nên bởi các quyết định thiết kế và nó có một số lợi ích sau đây:

* Giảm thiểu được nhầm lẫn cái gì được phép và cái gì không được phép trong kiến trúc và do đố giảm thiểu được phác thảo và hủy bỏ kiến trúc sau này
* Giúp dễ dàng chỉ ra được quyết định trong kiến trúc có phù hợp với ràng buộc được đưa ra hay không và đồng thời hỗ trợ định hướng tiến hóa kiến trúc sau này
* Kiểu kiến trúc linh hoạt và hữu ích là mô hình kiến trúc với thành phần và kết nối cho các hệ thống lớn hay động. Nó là nơi duy nhất thể hiện được những quan tâm nhiều mặt và tính hợp lý của kiến trúc.
* Do kiểu được áp dụng cho nhiều dự án khác nhau nên các mô hình kiểu có thể sử dụng lại cho các dự án khác nhau sau này.

Những loại quyết định thiết kế tìm thấy trong một kiểu kiến trúc thông thường là trừu tượng và tổng quát hơn những quyết định trong một kiến trúc. Một số loại quyết định thiết kế sau đây có thể nằm trong mô hình kiểu:

* Kiểu có thể quy định các thành phần, kết nối hay giao diện sẽ sử dụng trong kiến trúc hay trong các tình huống đặc biệt
* Thành phần, kết nối và kiểu giao diện. Một số loại phần tử có thể được phép, phải hay cấm sử dụng trong kiến trúc. Nhiều cách tiếp cận mô hình với ngữ nghĩa khác nhau có thể kết hợp trong cùng kiểu hệ thống
* Ràng buộc về mặt tương tác: có những ràng buộc về mặt tương tác giữa các thành phần và kết nối với nhiều dạng khác nhau: ràng buộc thời gian (thành phần phải gọi init() trước bất kỳ phương thức nào khác); ràng buộc về cấu trúc (chỉ những thành phần trong tầng client mới được phép triệu gọi các thành phần trong tầng server; phải sử dụng các giao thức đã chỉ ra như FTP hay HTTP; các cách tiếp cận mô hình hỗ trợ các ràng buộc về logic như logic vị từ cấp 1, logic thời gian…
* Ràng buộc hành vi: ràng buộc có thể biểu diễn với những quy luật đơn giản hay đặc tả hành vi đầy đủ cho các thành phần theo biểu đồ máy trạng thái hữu hạn. Các cách tiếp cận mô hình hỗ trợ các ràng buộc hành vi có thể sử dụng logic hay mô hình như máy trạng thái hữu hạn.
* Ràng buộc đồng bộ: các phần tử nào thực hiện các chức năng một cách đồng thời và cách đồng bộ hóa truy nhập tài nguyên chung được thể hiện trong kiểu kiến trúc. Nhiều cách tiếp cận hỗ trợ mô hình đồng bộ sử dụng mô hình hành vi theo thời gian như biểu đồ tuần tự hay biểu đồ trạng thái.

## **2.4. Kỹ thuật mô hình**

Mỗi kiến trúc sư phần mềm đều có những ký hiệu và kỹ thuật để mô hình các khía cạnh khác nhau của kiến trúc. Các kỹ thuật này thay đổi theo những chiều khác nhau.

* Các kỹ thuật này có thể dùng để mô hình những đối tượng nào?
* Nó có thể hiện được ngữ nghĩa của kiến trúc và công cụ hỗ trợ không?
* Phương pháp luận hay tiến trình phát triển nào được sử dụng và nhằm mục đích gì? Điều này phụ thuộc vào các bên liên quan và kiến trúc sư hệ thống

Sử dụng các cách tiếp cận khác nhau để mô hình hóa kiến trúc hệ thống sẽ giúp chúng ta có một cái nhìn đa dạng, đầy đủ hơn về hệ thống cần phát triển. Trong phần tiếp theo sẽ trình bày các cách tiếp cận khác nhau để mô hình các kiến trúc phần mềm.

* Ngôn ngữ phân tích và thiết kế kiến trúc (AADL: Architecture Analysis & Design Language) là ngôn ngữ đầy đủ để thiết kế cả phần cứng và phần mềm của hệ thống. Nó hỗ trợ bộ xử lý, thiết bị và cổng cũng như tiến trình, luồng và dữ liệu
* Môi trường và ngôn ngữ dựa trên kiến trúc (ACME: the Architecture Based Language and Environment) là ngôn ngữ nhỏ và hơi đơn giản. Nó thể hiện các khái niệm hệ thống, thành phần, kết nối, cổng, vai trò, biểu diễn. Một hệ thống được cấu thành bởi các thành phần được liên kết bởi các kết nối, cổng là các điểm cuối của các kết nối. ACME có thể xem là tập con của AADL.
* Darwin: hỗ trợ phân cấp. Nghĩa là một thành phần là hợp của các thành phần sơ cấp. Darwin cũng hỗ trợ cấu trúc các chương trình song song và mô hình cấu hình mạng
* Wright: được xây dựng dựa trên các thành phần trừu tượng, các kết nối và cấu hình. Các cấu hình có thể chia thành các thể hiện (một kiểu đặc tả thành phần), gắn kết (mô tả cấu hình của hệ thống) và phân cấp (một thành phần có thể thể hiện các thành phần khác)
* Aesop (Aesop: The Software Architecture Design Environment Generator) là tập các công cụ được thiết kế để phát triển mô hình hệ thống. Nó dựa trên môi trường UNIX, có mở rộng kiểu ống và lọc để mô hình những đặc điểm này. Nhờ có nhân tổng quát, nó thích hợp cho mọi môi trường
* UML là ngôn ngữ mô hình hóa được sử dụng rộng rãi. Mặc dù UML không phải là ngôn ngữ mô tả kiến trúc nhưng nó lại phù hợp để mô hình hệ thống. Phần tiếp theo của chương này sẽ dành chủ yếu trình bày UML cho biểu diễn kiến trúc.
* TASM (TASM: Timed Abstract State Machine) thực sự không phải là ngôn ngữ mô tả kiến trúc nhưng nó có thể được sử dụng để mô hình những hệ khá phức tạp. Nó dựa trên máy trạng thái trừu tượng với mở rộng thời gian. Một mô hình hệ thống được cấu thành bởi tập biến theo dõi “đọc”, tập biến điều khiển “viết” và tập luật.

## **2.5. Ngôn ngữ mô hình hóa thống nhất UML cho thiết kế kiến trúc**

Ngôn ngữ mô hình hóa thống nhất UML (Unified Modeling Language) là bộ ký hiệu với biểu đồ quen thuộc nhất để tạo ra bản thiết kế phần mềm. UML đã kết hợp những khái niệm đã có trước đó như biểu đồ của Booch, OMT của Rumbaugh, OOSE của Jacobson và biểu đồ trạng thái của Harel. Những điểm mạnh chủ yếu của UML đã được thừa nhận rộng rãi với nhiều mô hình hóa cấu trúc, quan điểm và công cụ hỗ trợ. Nó cung cấp tính nhiều chiều cho kỹ thuật mô hình với mười bốn biểu đồ để mô hình hóa các khía cạnh tĩnh và động của phần mềm. UML là một công cụ ký hiệu thiết kế và phân tích hướng đối tượng mà điển hình là cung cấp nhiều sơ đồ phân tích hữu ích và thậm chí còn tạo khung mã cho phần mềm.

UML được sử dụng rộng rãi như một kim chỉ nam cho các tài liệu thiết kế và phân tích yêu cầu phần mềm, là cơ sở để phát triển phần mềm. Thông thường, UML có thể được sử dụng để mô hình hóa miền vấn đề; mô tả các yêu cầu của người dùng; xác định các yếu tố kiến trúc quan trọng trong quá trình thiết kế phần mềm chẳng hạn như các lớp và đối tượng, mô tả hành vi và tương tác giữa các yếu tố này và tổ chức cấu trúc phần mềm, chỉ định các ràng buộc của nó, mô tả các thuộc tính cần thiết…

UML cung cấp một số sơ đồ mô hình hóa có thể được nhóm thành hai loại: cấu trúc tĩnh và hành vi (động). Các biểu đồ mô tả cấu trúc tĩnh bao gồm các biểu đồ lớp, biểu đồ thành phần, biểu đồ triển khai…Các biểu đồ mô tả cấu trúc động bao gồm: biểu đồ trình tự, biểu đồ cộng tác, biểu đồ hoạt động, …

Có rất nhiều công cụ UML IDE (môi trường phát triển tương tác); một trong số chúng là mã nguồn mở. Các công cụ UML phổ biến nhất là Rational Rose, Boland Together và Microsoft Visio. Nhiều trong số phần mềm này có khả năng ánh xạ trực tiếp từ sơ đồ UML sang khung mã hóa bằng các ngôn ngữ lập trình phổ biến như C++, C# và Java.

Các phiên bản trước đây của UML 1.. \* tập trung nhiều vào thiết kế chi tiết cho các lớp, thuộc tính và phương thức. Uml 2.0 đã mở rộng UML để hỗ trợ tốt hơn các cấu trúc mức cao trong kiến trúc. Việc hiểu đầy đủ ý nghĩa của các biểu đồ UML được xem là then chốt để sử dụng cho việc mô tả các quyết định thiết kế. Dưới đây là bảng tóm tắt 13 biểu đồ UML 2.0 trong các danh mục cấu trúc tĩnh và hành vi:

1. Các biểu đồ cấu trúc (tĩnh)

|  |  |
| --- | --- |
| Biểu đồ | Mô tả |
| Lớp (class) | Tổng quan về các lớp dành cho mô hình hóa và thiết kế. Nó chỉ ra cách các lớp liên quan với nhau một cách tĩnh, nhưng không chỉ ra cách các lớp tương tác động với nhau. |
| Đối tượng (Object) | Các đối tượng và mối quan hệ của chúng trong thời gian chạy. Tổng quan về các trường hợp cụ thể của biểu đồ lớp tại một thời điểm cho một trường hợp cụ thể. Nó dựa trên sơ đồ lớp. |
| Composite structure (Cấu trúc gộp) | Mô tả cấu trúc bên trong của một thành phần bao gồm tất cả các lớp bên trong thành phần, giao diện của thành phần, v.v. |
| Component (thành phần) | Mô tả tất cả các thành phần trong hệ thống, mối quan hệ qua lại, tương tác của chúng và giao diện của hệ thống. Nó là một phác thảo về cấu trúc thành phần của các thành phần hoặc mô-đun. |
| Package (gói) | Mô tả cấu trúc và tổ chức gói. Nó bao gồm các lớp trong gói và các gói trong gói khác |
| Deployment (triển khai) | Mô tả phần cứng hệ thống, phần mềm và kết nối mạng cho máy tính phân tán. Nó bao gồm cấu hình máy chủ và kết nối mạng giữa các nút máy chủ trong cài đặt thế giới thực. |

2. Biểu đồ hành vi (động)

|  |  |
| --- | --- |
| Biểu đồ | Mô tả |
| Use case | Xuất phát từ các tình huống nghiên cứu ca sử dụng. Đây là tổng quan về các ca sử dụng, các tác nhân và các mối quan hệ giao tiếp của chúng để chứng minh cách hệ thống phản ứng với các yêu cầu từ người dùng bên ngoài. Nó được sử dụng để xác định các yêu cầu hệ thống. |
| Activity (hoạt động) | Phác thảo dữ liệu hoạt động và luồng điều khiển giữa các đối tượng liên quan. Hoạt động là một hành động cho một hoạt động hệ thống hoặc một quy trình nghiệp vụ, chẳng hạn như những hoạt động được nêu trong sơ đồ ca sử dụng. Nó cũng bao gồm các điểm quyết định và chủ đề của các quy trình hoạt động phức tạp. Nó mô tả cách các hoạt động được sắp xếp để đạt được chức năng cần thiết. |
| State-machine (máy trạng thái) | Mô tả vòng đời của các đối tượng bằng máy trạng thái hữu hạn. Biểu đồ bao gồm các trạng thái và sự chuyển đổi giữa các trạng thái. Quá trình chuyển đổi thường do các kích thích hoặc sự kiện bên ngoài gây ra. Chúng cũng có thể đại diện cho các chuyển động bên trong của đối tượng. |
| Sequence (trình tự) | Mô tả chuỗi thời gian của thông báo được chuyển giữa các đối tượng trong một dòng thời gian |
| Interaction overview (tổng quan tương tác) | Kết hợp các biểu đồ trình tự và hoạt động để cung cấp tổng quan về luồng điều khiển của hệ thống và quy trình nghiệp vụ. |
| Communication (giao tiếp) | Mô tả chuỗi thông báo truyền giữa các đối tượng trong hệ thống. Tương đương với biểu đồ tuần tự, ngoại trừ việc nó tập trung vào vai trò của đối tượng. Mỗi liên kết giao tiếp được liên kết với một số thứ tự trình tự cộng với các thông điệp đã chuyển. |
| Time sequence (trình tự thời gian) | Mô tả các thay đổi của thông báo trong trạng thái, điều kiện và sự kiện |

### **2.5.1 Các biểu đồ cấu trúc**

Các biểu đồ mô tả cấu trúc bao gồm các biểu đồ lớp và đối tượng; biểu đồ thành phần, cấu trúc và gói; và các biểu đồ triển khai. Chúng ta lần lượt thảo luận về từng vấn đề.

***2.5.1.1 Biểu đồ lớp***

Biểu đồ lớp cung cấp một cái nhìn tĩnh của hệ thống. Nó nắm bắt từ vựng của hệ thống được thiết kế. Đây là sơ đồ nền tảng của thiết kế hệ thống và cũng là sơ đồ UML được sử dụng thường xuyên nhất.

Biểu đồ lớp có thể được rút ra từ biểu đồ ca sử dụng hoặc từ phân tích văn bản của miền vấn đề đã cho. Sơ đồ lớp được tạo ra bởi các nhà phân tích và thiết kế hệ thống và sẽ được tinh chỉnh lặp đi lặp lại trong các giai đoạn tiếp theo trong vòng đời phát triển phần mềm.

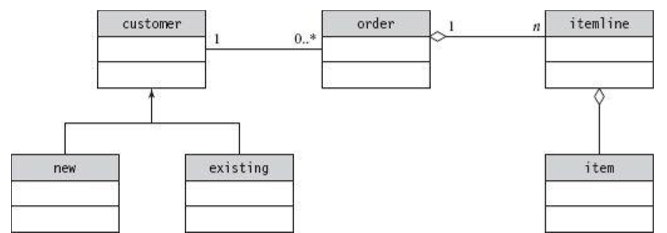
Biểu đồ lớp mô tả từng lớp riêng lẻ với kiểu, giao diện, thuộc tính và phương thức của nó. Khả năng truy cập (khả năng hiển thị) của từng thuộc tính và hoạt động cũng có thể được chỉ định. Các truy cập phổ biến bao gồm riêng tư, công khai, được bảo vệ và mặc định.

Một phần quan trọng của biểu đồ lớp là giao diện của mỗi lớp. Một giao diện lớp cung cấp các hợp đồng hành vi mà lớp đó phải hỗ trợ.

Có ba mối quan hệ chính giữa các lớp: kế thừa, cộng gộp và kết hợp. Các mối quan hệ này có thể được biểu diễn bằng đồ thị trong một biểu đồ lớp. Đối với mỗi mối quan hệ, tính đa dạng giữa các lớp cũng có thể được ký hiệu. Các kiểu đa dạng điển hình bao gồm ánh xạ một-một, một-nhiều và nhiều-nhiều. Trong các ký hiệu đa dạng của UML, 1 là viết tắt của một trường hợp, 0 là không có trường hợp nào, 0..1 là viết tắt của 0 hoặc một trường hợp, và 1 .. \* là viết tắt của ít nhất một trường hợp.

Hình 2.2 cho thấy một biểu đồ lớp cho một hệ thống xử lý đơn đặt hàng trực tuyến. Ở đây chúng ta thấy tất cả các loại mối quan hệ giữa các lớp như kế thừa (biểu diễn bằng mũi tên tam giác rỗng), tập hợp (biểu diễn bằng mũi tên hình thoi rỗng) và kết hợp (các đường không có mũi tên). Các chỉ số đa dạng cũng được hiển thị. Nói chung, biểu đồ mô tả cấu trúc logic của hệ thống đặt hàng bao gồm sáu lớp. Lớp Customer là lớp cơ sở của New và Existing. Một Customer có thể đặt không hoặc nhiều Order. Mỗi Order bao gồm nhiều ItemLine, lần lượt chứa các item.

Biểu đồ lớp có thể được tinh chỉnh theo thời gian trong vòng đời phát triển phần mềm. Biểu đồ đối tượng và biểu đồ cấu trúc thành phần có thể được bắt nguồn trực tiếp từ một sơ đồ lớp. Các biểu đồ hành vi động khác như biểu đồ trình tự và biểu đồ giao tiếp (cộng tác) cũng dựa trên biểu đồ lớp.

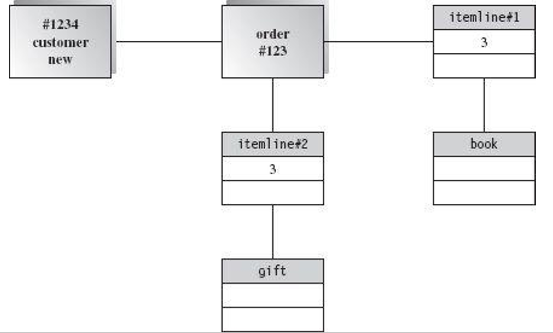


Hình 2.2 Biểu đồ lớp

***2.5.1.2 Biểu đồ đối tượng***

Đối tượng là thể hiện của các lớp. Biểu đồ đối tượng được sử dụng để mô tả một tập hợp con mẫu của các đối tượng trong hệ thống tại một thời điểm cụ thể. Biểu đồ này cho thấy một ảnh chụp nhanh về kết nối và tương tác của cá thể lớp. Nó có nguồn gốc từ biểu đồ lớp trước đó và là một ví dụ cụ thể về sơ đồ lớp trong thời gian chạy. Nhiều sơ đồ hành vi khác (sơ đồ tuần tự, sơ đồ giao tiếp và sơ đồ tương tác) có thể tham chiếu đến sơ đồ đối tượng.

Hình 2.3 cho thấy một sơ đồ đối tượng dựa trên sơ đồ lớp trong Hình 2.2. Mỗi hình hộp chữ nhật trong sơ đồ đại diện cho một đối tượng là một thể hiện của một số lớp. Sơ đồ cho chúng ta biết rằng khách hàng có số nhận dạng # 1234 đã đặt hàng hai mặt hàng: book và gift.



Hình 2.3. Biểu đồ đối tượng

***2.5.1.3 Biểu đồ cấu trúc tổng hợp***

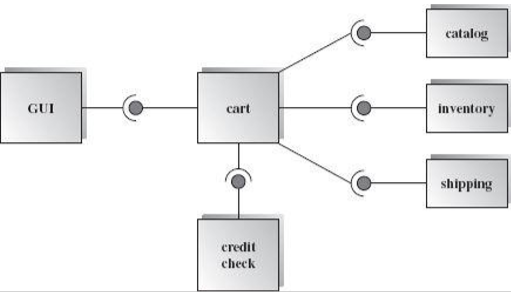
Biểu đồ cấu trúc tổng hợp được sử dụng để mô tả thành phần của các phần tử được kết nối với nhau hoặc sự cộng tác của các cá thể thời gian chạy. Có hai ký hiệu cơ bản trong một sơ đồ cấu trúc tổng hợp: cộng tác (được biểu diễn bằng cách sử dụng nhật thực nét đứt) và lớp có cấu trúc (được biểu diễn bằng hình hộp chữ nhật). Mỗi lớp cấu trúc có thể có một chú thích cho biết vai trò của nó trong sự cộng tác. Ví dụ, Hình 3.4 mô tả hai lớp tham gia vào một sự cộng tác OrderProcess. Lớp Khách hàng (Customer) đóng vai trò là “người mua” và Hệ thống xử lý đơn hàng (Order Processing System) đóng vai trò là “người bán”. Lưu ý rằng OrderProcess không phải là một lớp cũng không phải là một đối tượng, nó là một sự cộng tác.



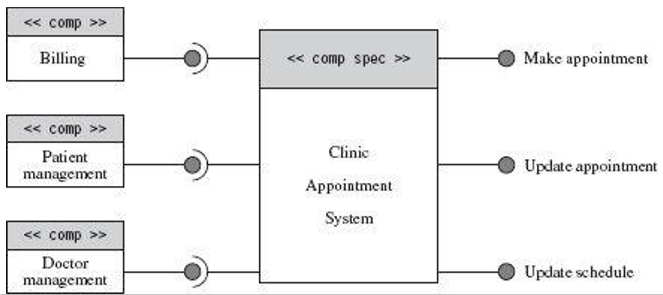
Hình 2.4. Cấu trúc tổng hợp

***2.3.1.4 Sơ đồ thành phần***

Một thành phần không phải là một lớp cũng không phải là một đối tượng. Thành phần là một khối xây dựng có thể triển khai, tái sử dụng được sử dụng trong thiết kế và phát triển phần mềm. Ví dụ: một thành phần JavaBean được triển khai trong một tệp jar, một thành phần EJB được triển khai trong một tệp. ear và thành phần NET được triển khai trong một tệp .dll. Mỗi thành phần có một giao diện để hiển thị các dịch vụ của nó và ẩn các triển khai của nó. Giao diện là hợp đồng giữa một thành phần có thể tái sử dụng và các máy khách của nó.



Hình 2.5 Biểu đồ thành phần



Hình 2.6. Một ví dụ khác của biểu đồ thành phần

UML 2.0 đã giới thiệu một ký hiệu mới cho các thành phần và kết nối của chúng. Hình dạng kẹo mút của một thành phần đại diện cho một giao diện được triển khai. Hình chiếc cốc đại diện cho giao diện bắt buộc và giao diện bắt buộc phải được cung cấp bởi một số thành phần khác. Trong sơ đồ thành phần, một số thành phần có thể tồn tại và có sẵn trong hệ thống hoặc bên ngoài. Các thành phần khác được thiết kế và phát triển bởi những người làm việc trong dự án.

Hình 2.5 cho thấy một sơ đồ thành phần cho một ứng dụng giỏ hàng. Thành phần giỏ hàng cung cấp các dịch vụ cho giao diện GUI front-end như ASP, JSP hoặc các trang web PHP. Bản thân thành phần giỏ hàng có thể cần các dịch vụ từ các thành phần khác: danh mục, hàng tồn kho, vận chuyển và kiểm tra tín dụng.

Sơ đồ thành phần trong Hình 3.6 cho thấy bốn thành phần: thành phần phòng khám (*Clinic Appointment System*), thành phần thanh toán (*Billing*), thành phần bệnh nhân (*Patient management*) và thành phần bác sĩ (*Doctor management*). Thành phần phòng khám cung cấp các dịch vụ sau (dưới dạng giao diện): đặt lịch hẹn theo bệnh nhân (*make appointment*), cập nhật lịch hẹn theo bệnh nhân (*update appointment*), cập nhật lịch khám theo lịch của bác sĩ (*update schedule)*. Thành phần phòng khám cũng cần các dịch vụ từ thành phần thanh toán, thành phần bệnh nhân và thành phần bác sĩ.

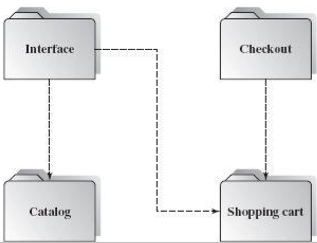
***2.5.1.5 Biểu đồ gói***

Một gói được đại diện bởi một thư mục theo thẻ cho biết vị trí của tất cả các lớp và gói con được bao gồm. Các gói đóng vai trò tương tự như một thư mục để nhóm các tệp trong hệ thống tệp; chúng cho phép tổ chức tất cả các lớp có liên quan chặt chẽ trong một “vùng chứa - container”.

Ví dụ, các không gian tên trong. NET và các gói trong Java cung cấp các cấu trúc được định hình tốt cho khả năng truy cập lớp và các mối tương quan của lớp. Chúng ta có thể tổ chức các lớp liên quan đến chức năng trong cùng một gói để các lớp này có thể truy cập lẫn nhau trong khả năng truy cập hoặc khả năng hiển thị mặc định. Chúng ta cũng có thể tổ chức các gói liên quan trong cùng một gói mẹ để xây dựng một cấu trúc phân cấp gói và lớp giống như vậy. Thư viện lớp NET và API Java. Một lý do khác để sử dụng tổ chức gói là chia sẻ không gian tên; theo cách này, tất cả các lớp trong cùng một gói có một tên duy nhất nhưng chúng có thể có cùng tên trong các gói khác nhau (không gian tên).

Sơ đồ gói cho thấy mối quan hệ phụ thuộc giữa các gói trong đó sự thay đổi của một gói có thể dẫn đến những thay đổi trong các gói khác. Sơ đồ gói cũng có thể chỉ định nội dung của một gói, tức là các lớp cấu thành một gói và các mối quan hệ của chúng. Việc sử dụng sơ đồ gói để biểu diễn cấu trúc hệ thống có thể giúp giảm độ phức tạp của sự phụ thuộc và đơn giản hóa mối quan hệ giữa các nhóm lớp.

Hình 2.7 cho thấy một sơ đồ gói đơn giản trong đó gói thanh toán, chứa tất cả các lớp liên quan đến thanh toán, phụ thuộc vào các lớp được nhóm trong gói giỏ hàng. Tương tự với gói giao diện người dùng có tất cả các lớp trình bày GUI để hiển thị danh mục và giỏ hàng. Sơ đồ gói cũng mô tả mối quan hệ phụ thuộc của các đơn vị gói. Sơ đồ này thường được sử dụng để thiết kế kiến ​​trúc phần mềm dựa trên thành phần.



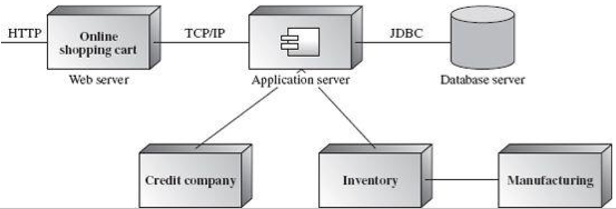
Hình 2.7 Package diagram

***2.5.1.6 Sơ đồ triển khai***

Sơ đồ triển khai mô tả cấu hình vật lý của hệ thống phần mềm được triển khai trên các nút máy chủ phần cứng và mạng giữa các nút (được định nghĩa là giao thức). Sơ đồ này được tạo ra trong giai đoạn sau của vòng đời phát triển phần mềm. Tất cả các thành phần trong hệ thống phải được triển khai trên các máy chủ để cung cấp dịch vụ thông qua giao thức mạng. Các sơ đồ thành phần là cơ sở cho các sơ đồ triển khai.

UML sử dụng một biểu tượng hình khối để đại diện cho một nút tài nguyên máy tính; tài nguyên đó có thể là một thiết bị phần cứng hoặc một hệ thống con phần mềm được triển khai. Ví dụ, máy chủ dữ liệu, máy chủ web và máy chủ ứng dụng có thể là các nút và được mô tả bằng các hình khối trong một sơ đồ triển khai. Liên kết giữa các nút là kết nối mạng được mô tả bởi giao thức mạng. Sơ đồ triển khai được sử dụng rộng rãi để mô hình hóa và thiết kế các hệ thống phần mềm phân tán.

Hình 2.8 cho thấy một sơ đồ triển khai trong đó một giỏ hàng được triển khai trong một máy chủ web, thành phần giao dịch kinh doanh được triển khai trong một máy chủ ứng dụng riêng biệt và cơ sở dữ liệu có sẵn trong một máy chủ dữ liệu. Các dịch vụ khác có sẵn từ ba thành phần được triển khai bởi các nhà cung cấp dịch vụ tương ứng.



Hình 2.8 Biểu đồ triển khai

### **2.5.2 Các biểu đồ hành vi**

Sơ đồ mô tả hành vi bao gồm sơ đồ ca sử dụng, sơ đồ hoạt động, máy trạng thái, sơ đồ tương tác, sơ đồ tuần tự, sơ đồ cộng tác và sơ đồ thời gian. Chúng ta lần lượt thảo luận về từng sơ đồ.

***2.5.2.1 Sơ đồ ca sử dụng***

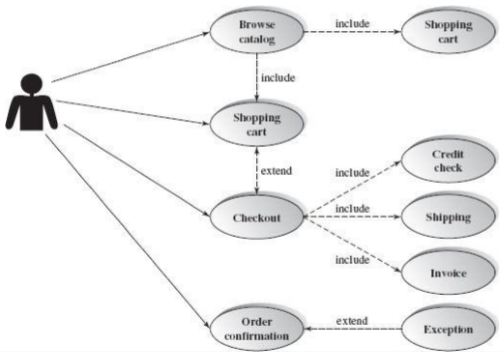
Biểu đồ ca sử dụng mô tả các yêu cầu của người dùng về chức năng hệ thống như một hợp đồng giữa người dùng (tác nhân) và hệ thống phần mềm. Sơ đồ này bao gồm các tác nhân, các trường hợp sử dụng và các liên kết giữa chúng. Ví dụ về một ca sử dụng được thể hiện trong Hình 2.9.

Tác nhân trong sơ đồ ca sử dụng là một “người dùng” bên ngoài; đây có thể là người dùng cuối, một ứng dụng, một thiết bị bên ngoài hoặc một hệ thống khác.

Mỗi ca sử dụng là một kịch bản hoạt động có ý nghĩa. Có nghĩa là, các ca sử dụng được định hướng theo kịch bản theo nghĩa là mỗi trường hợp là một chuỗi các bước làm việc được thực hiện bởi các lớp để cung cấp chức năng hệ thống cần thiết. Các bước chi tiết được quy định trong một ghi chú riêng bao gồm các điều kiện trước và sau của hành động trong trình tự.  
Liên kết kết nối (đơn giản) từ tác nhân đến ca sử dụng hiển thị hướng của các tác nhân trong việc sử dụng ca sử dụng. Liên kết <<include>>, một loại liên kết đặc biệt, từ trường hợp sử dụng này sang trường hợp sử dụng khác, chỉ ra rằng trường hợp sử dụng đầu tiên sử dụng lại hoặc bao gồm trường hợp sử dụng thứ hai và cần thiết để hoàn thành công việc cần thiết để đáp ứng yêu cầu. Liên kết <<include>> là một đường đứt nét với mũi tên trỏ đến trường hợp sử dụng được sử dụng. Một liên kết <<extend>>, một liên kết đặc biệt khác, hiển thị một ca sử dụng tùy chọn mới được tạo từ một ca sử dụng hiện có. Nó bao gồm các trường hợp thay thế có thể có hoặc không nhất thiết phải diễn ra. Một liên kết <<extend>> cũng là một đường thẳng đứt nét với mũi tên chỉ về phía trường hợp sử dụng mở rộng; các liên kết đặc biệt này được gắn nhãn tương ứng.

Một biểu đồ ca sử dụng hoàn chỉnh mô tả một tập hợp các tình huống; các kịch bản có thể có một tập hợp các trường hợp sử dụng cấp dưới hoặc cấp thấp hơn. Biểu đồ ca sử dụng được sử dụng trong giai đoạn đầu của vòng đời phát triển phần mềm, chẳng hạn như phân tích và thiết kế. Các sơ đồ này là một trong những sơ đồ UML được sử dụng thường xuyên nhất để phân tích và thiết kế hệ thống hướng đối tượng.

Các nhà phân tích hệ thống sử dụng sơ đồ ca sử dụng để nắm bắt và xác minh các yêu cầu của người dùng. Các kiến ​​trúc sư và nhà thiết kế sử dụng các sơ đồ này để lấy sơ đồ cấu trúc (ví dụ: sơ đồ lớp) và sơ đồ hành vi (ví dụ: sơ đồ trình tự và sơ đồ giao tiếp).



Hình 2.9 Use case diagram

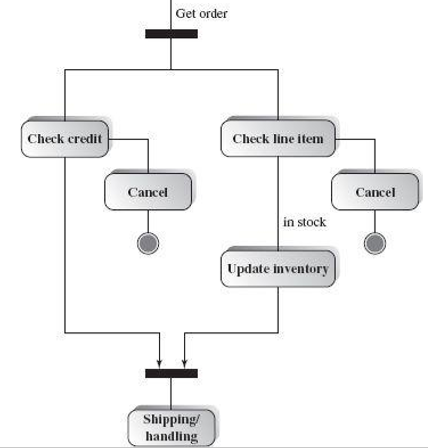
***2.5.2.2 Sơ đồ hoạt động***

Sơ đồ hoạt động được sử dụng để mô tả các quy trình nghiệp vụ phức tạp. Sơ đồ này thường liên quan đến quy trình làm việc phức tạp, ra quyết định, thực hiện đồng thời, xử lý ngoại lệ, kết thúc quy trình, v.v. Sơ đồ hoạt động là sơ đồ hướng quy trình làm việc mô tả các bước trong một quy trình duy nhất. Một sơ đồ hoạt động tương ứng với một quy trình nghiệp vụ. Có nhiều hoạt động trong một quy trình nghiệp vụ và sơ đồ này khám phá sự phụ thuộc của chúng trong quy trình.

Biểu đồ hoạt động UML sử dụng một hình chữ nhật tròn để biểu diễn một hoạt động. Mỗi sơ đồ hoạt động có một điểm bắt đầu và một hoặc nhiều điểm kết thúc. Một thời nhỏ đại diện cho một điểm quyết định trong sơ đồ. Biểu đồ hoạt động hỗ trợ xử lý song song bằng cách sử dụng một cặp thanh ngang màu đen để chỉ ra các hành động nối/rẽ nhánh tương ứng trong các lộ trình như vậy. Biểu đồ hoạt động UML cũng hỗ trợ giao tiếp giữa hai luồng đồng thời bằng cách gửi tín hiệu từ đường dẫn này sang đường dẫn khác. (Đây được gọi là một sự kiện và được ghi nhận là một cặp đa giác lồi.)

Biểu đồ hoạt động cung cấp một cái nhìn động chi tiết về một nhiệm vụ hoặc quy trình cụ thể trong một hệ thống để nhà phát triển phần mềm có thể dễ dàng nhận ra các yêu cầu thực hiện. Sơ đồ này là cơ sở cho một sơ đồ truyền thông và các sơ đồ tương tác động khác.

Hình 2.10 cho thấy một nhóm các hoạt động trong hệ thống xử lý đơn đặt hàng. Thanh màu đen đầu tiên chia (fork) hai hoạt động đồng thời có thể được thực hiện song song. Hoạt động vận chuyển/xếp dỡ (*shipping handling*) sẽ không bắt đầu cho đến khi cả hai hoàn thành và kết hợp với nhau (được biểu thị bằng thanh thứ hai ở dưới cùng).



Hình 2.10 Biểu đồ hoạt động

***2.5.2.3 Sơ đồ trạng thái máy***

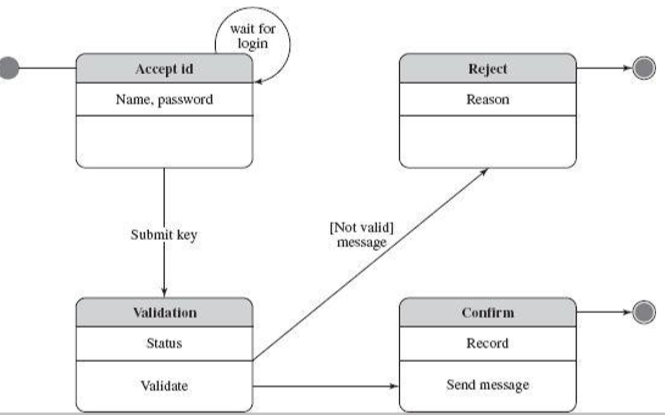
Biểu đồ trạng thái máy, được gọi là biểu đồ trạng thái trong UML 1.x, được sử dụng rộng rãi cho các hệ thống nhúng và phân tích và thiết kế phần mềm thiết bị. Nó là một sơ đồ hướng sự kiện, trong đó các phần tử của hệ thống thay đổi trạng thái của chúng để đáp ứng với các kích thích bên ngoài hoặc bên trong (chẳng hạn như các sự kiện thời gian hoặc các sự kiện hệ thống khác). Những sơ đồ này là lý tưởng để xác định hành vi bên trong của các đối tượng.

Trong sơ đồ máy trạng thái, trạng thái là một hình chữ nhật tròn với ba phần nhỏ: tên trạng thái, các biến trạng thái và các hoạt động của trạng thái. Trạng thái là một tình huống trong đó một đối tượng đáp ứng các điều kiện, thực hiện các hành động và chờ đợi một sự kiện mới. Khi một sự kiện mới diễn ra ở trạng thái hiện tại, máy sẽ thực hiện các hành động được chỉ định và sau đó sẽ chuyển sang trạng thái mới (trạng thái tiếp theo).

Một trạng thái hỗn hợp phức tạp có thể có một biểu đồ trạng thái cấp dưới. Các trạng thái con ở trạng thái hỗn hợp có thể được chuyển đổi từ trạng thái này sang trạng thái tiếp theo, tuần tự hoặc đồng thời.

Mỗi sơ đồ máy trạng thái có một điểm bắt đầu trong một vòng tròn màu đen đặc và có một hoặc nhiều điểm cuối, điểm cuối được biểu thị bằng các hình tròn. Các liên kết chuyển tiếp giữa các trạng thái là các đường liền nét với các đầu mũi tên để chỉ hướng. Biểu đồ trạng thái giúp nhà phát triển phần mềm hiểu cách hệ thống phản hồi và xử lý các sự kiện bên ngoài và điều kiện kích hoạt sự kiện tương ứng.

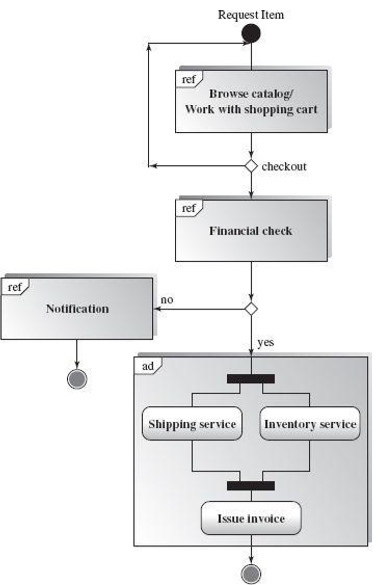
Hình 2.11 cho thấy một sơ đồ trạng thái máy mô tả một quá trình đăng nhập. Ban đầu, máy trạng thái thực hiện một vòng lặp bận để chờ đăng nhập của người dùng, và sau đó cặp tên người dùng / mật khẩu được xác minh. Nếu cặp khớp với hồ sơ hệ thống, đăng nhập được xác nhận; nếu không đăng nhập bị từ chối.



Hình 2.11 Biểu đồ máy trạng thái

***2.5.2.4 Sơ đồ tổng quan về tương tác***

Sơ đồ tổng quan về tương tác mô tả luồng điều khiển của các tương tác chứ không phải thông báo. Nó là một biến thể của sơ đồ hoạt động. Các nút trong sơ đồ tổng quan về tương tác đại diện cho một tham chiếu đến một sơ đồ hiện có (ref), một phần tử tương tác cơ bản [sơ đồ hoạt động (ad)] hoặc một sơ đồ trình tự (sd)].



Mỗi nút (hoặc khung) có thể là một sơ đồ tương tác như sơ đồ tuần tự, sơ đồ truyền thông, sơ đồ hoạt động hoặc sơ đồ tổng quan về tương tác lồng nhau. Một nút tham chiếu, được biểu thị bằng “ref” ở góc trên bên trái của khung, trỏ đến một sơ đồ hiện có, trong khi phần tử cơ bản hiển thị sơ đồ tương tác động của khung. Yếu tố cơ bản được biểu thị bằng nhãn "ad" cho sơ đồ hoạt động, nhãn "sd" cho sơ đồ tuần tự hoặc nhãn "cd" cho sơ đồ giao tiếp, v.v. Sơ đồ tổng quan về tương tác là sự trừu tượng hóa cấp cao của mô tả tổng quan về tương tác.

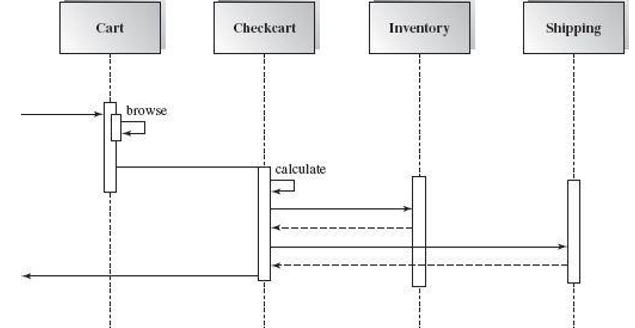
***2.5.2.5 Biểu đồ trình tự***

Biểu đồ trình tự là một trong những biểu đồ UML quan trọng nhất và được sử dụng rộng rãi để phân tích và thiết kế hệ thống phần mềm. Nó là một sơ đồ tương tác định hướng thời gian hiển thị trình tự thời gian của thông điệp giữa các đối tượng. Thông thường, một biểu đồ trình tự tương ứng với một ca sử dụng.

Mỗi đối tượng tham gia trong biểu đồ này có một mốc thời gian dọc cho vòng đời của nó. Time tiến lên cùng với dòng thời gian đi xuống. Mỗi dòng thời gian dọc có một số ô chữ nhật hẹp (gọi là kích hoạt) đại diện cho trạng thái kích hoạt đối tượng mà nó nhận hoặc gửi thông điệp. Mỗi hộp kích hoạt cũng có thể có một liên kết được định hướng tự đệ quy được trỏ về chính nó, chỉ ra rằng đối tượng chuyển thông điệp cho chính nó. Một kích hoạt cũng có thể phân nhánh hoặc phân nhánh nhiều đường sống riêng biệt cho các điều kiện kịch bản nếu lựa chọn; cuối cùng tất cả các dòng được chia sẻ kết hợp với nhau.

Truyền thông điệp giữa các đối tượng được biểu diễn bằng một liên kết mũi tên ngang từ nguồn đến đích. Một dòng thông báo chuyển đơn giản, được biểu diễn bằng một đường liền nét có đầu mũi tên, chuyển quyền điều khiển từ đối tượng này sang đối tượng khác. Một đối tượng có thể gửi một thông điệp đồng bộ đến một đối tượng khác bằng một dòng có đầu mũi tên đầy đủ. Một thông điệp đồng bộ có nghĩa là người gửi phải đợi phản hồi từ đối tượng đích trước khi nó có thể tiến lên trong dòng thời gian. Một đối tượng cũng có thể gửi một thông điệp không đồng bộ đến một đối tượng khác, được biểu thị bằng một dòng có nửa đầu mũi tên. Người gửi một thông điệp không đồng bộ có thể tiếp tục công việc của mình theo dòng thời gian mà không cần đợi thông điệp trả lại từ đối tượng đích.

Hình 2.13 cho thấy một sơ đồ trình tự đơn giản cho mua sắm trực tuyến. Chuỗi trao đổi thông điệp bắt đầu từ đối tượng giỏ hàng (Cart). Sau khi duyệt thông điệp, đối tượng giỏ hàng (Cart) sẽ gửi thông điệp đến đối tượng kiểm tra giỏ hàng (checkcart) để thanh toán. Checkcart thực hiện một phép tính (calculate - tức là thông qua cách tự gửi thông điệp) và sau đó gửi thông điệp đến kho. Thông báo từ checkcart đến kho là một thông báo đồng bộ, vì checkcart phải đợi thông báo phản hồi (được biểu diễn bằng một đường mũi tên chấm chấm). Sau đó checkcart liên hệ vận chuyển và cuối cùng gửi lại thông điệp cho tác nhân đã khởi tạo ca sử dụng.

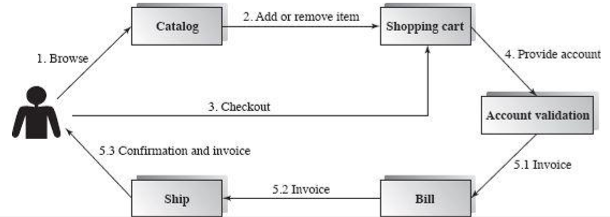


Hình 2.13. Sequence diagram

***2.5.2.6 Sơ đồ giao tiếp hoặc cộng tác***

Sơ đồ truyền thông UML, được gọi là sơ đồ cộng tác trong UML 1. x, là một sơ đồ hướng thông điệp mô tả tất cả các trình tự truyền thông điệp, điều khiển luồng, điều phối đối tượng, v.v., giữa các đối tượng tham gia vào các trường hợp sử dụng nhất định. Nó tóm tắt cách các đối tượng trong hệ thống nhận và gửi thông điệp. Nó là một phần mở rộng của sơ đồ đối tượng tĩnh trong đó các liên kết giữa các đối tượng thể hiện các mối quan hệ liên kết. Phía trên các liên kết trong một sơ đồ liên lạc là các thông điệp được đánh số, cho biết thứ tự mà chúng được gửi hoặc nhận. Các thông báo yêu cầu người nhận thực hiện một thao tác với các đối số được chỉ định. Mọi sơ đồ giao tiếp đều tương đương với một sơ đồ tuần tự, tức là một sơ đồ giao tiếp có thể được chuyển đổi thành một sơ đồ trình tự tương đương và ngược lại. Hai loại sơ đồ này lần lượt cung cấp một cái nhìn hướng thông điệp và hướng thời gian

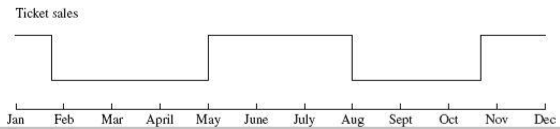
Hình 2.14 cho thấy một ví dụ về sơ đồ truyền thông. Nó tương đương với biểu đồ trình tự trong Hình 2.13 ngoại trừ các tên thông báo được đưa ra.



Hình 2.14. Communication diagram

***2.5.2.7 Sơ đồ thời gian***

Sơ đồ thời gian là một biểu đồ mới trong UML 2.0. Nó kết hợp biểu đồ trạng thái và chuỗi thời gian để hiển thị khung nhìn động về sự thay đổi trạng thái do các sự kiện bên ngoài gây ra theo thời gian. Nó thường được sử dụng trong các hệ thống quan trọng về thời gian như hệ điều hành thời gian thực, thiết kế hệ thống nhúng… Hình 2.15 cho thấy biểu đồ thời gian cho giá vé máy bay giảm giá theo mùa.



## **2.6. Mô hình khung nhìn kiến trúc**

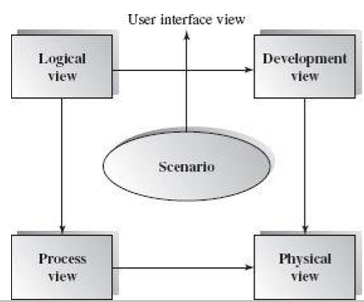
Mô hình là một mô tả hoàn chỉnh, đơn giản hóa của một hệ thống từ một khung nhìn hoặc góc nhìn cụ thể. Không có một khung nhìn duy nhất nào có thể trình bày tất cả các khía cạnh của phần mềm phức tạp cho các bên liên quan. Các mô hình khung nhìn cung cấp các đại diện một phần của kiến ​​trúc phần mềm cho các bên liên quan cụ thể như người dùng hệ thống, nhà phân tích/thiết kế, nhà phát triển/lập trình viên, nhà tích hợp hệ thống và kỹ sư hệ thống. Các nhà thiết kế phần mềm có thể tổ chức mô tả các quyết định về kiến ​​trúc của họ theo các khung nhìn khác nhau. Các bên liên quan có thể sử dụng khung nhìn để tìm những gì họ cần trong kiến ​​trúc phần mềm.

Mô hình 4 + 1 ban đầu được giới thiệu bởi Philippe Kruchten (Kruchten, 1995). Mô hình cung cấp bốn khung nhìn cơ bản: khung nhìn logic, khung nhìn quy trình, khung nhìn vật lý và khung nhìn phát triển. Khung nhìn logic mô tả các đối tượng và tương tác của chúng; khung nhìn quy trình mô tả các hoạt động của hệ thống, sự đồng thời và đồng bộ của chúng; khung nhìn vật lý mô tả việc ánh xạ phần mềm vào phần cứng, máy chủ và cấu hình mạng; và khung nhìn phát triển mô tả cấu trúc tĩnh của phần mềm trong một môi trường phát triển nhất định.

Cũng có một dạng khung nhìn khác được gọi là khung nhìn kịch bản; khung nhìn này mô tả các tình huống nắm bắt các khía cạnh quan trọng nhất của các yêu cầu chức năng, định hướng thiết kế hệ thống và xác minh hệ thống. Mô hình 4 + 1 là một mô hình nhiều khung nhìn nhằm giải quyết các khía cạnh và mối quan tâm khác nhau của hệ thống. Mô hình 4 + 1 chuẩn hóa các tài liệu thiết kế phần mềm và làm cho thiết kế dễ hiểu đối với tất cả các bên liên quan.

Chúng ta mở rộng mô hình 4 + 1 với một khung nhìn nữa, khung nhìn giao diện người dùng (UI). Giao diện người dùng, dành cho người dùng cuối của hệ thống phần mềm, mô tả giao diện người dùng đồ họa để xác minh và xác thực các yêu cầu giao diện người dùng; những điều này ảnh hưởng đáng kể đến khả năng sử dụng của hệ thống và các thuộc tính chất lượng khác.

Hình 2.16 cho thấy mô hình khung nhìn 4 + 1 được mở rộng với khung nhìn thứ năm của chúng ta, dạng xem giao diện người dùng. Khung nhìn kịch bản nhất quán với bốn dạng xem khác, trong khi khung nhìn giao diện người dùng tuân theo khung nhìn kịch bản và được hỗ trợ bởi bốn khung nhìn khác.



Hình 2.16. Mô hình khung nhìn 4+1

### **2.6.1 Khung nhìn kịch bản**

Khung nhìn kịch bản mô tả chức năng của hệ thống, tức là cách người dùng sử dụng hệ thống và cách hệ thống cung cấp dịch vụ cho người dùng. Chế độ khung nhìn này cung cấp nền tảng cho bốn chế độ khung nhìn khác và cho phép chúng hoạt động cùng nhau một cách liền mạch và chặt chẽ. Nó giúp các nhà thiết kế khám phá các yếu tố kiến trúc trong quá trình thiết kế và xác nhận thiết kế kiến trúc sau đó. Vì vậy, dạng khung nhìn kịch bản giúp làm cho kiến trúc phần mềm phù hợp với các yêu cầu chức năng và phi chức năng.

Biểu đồ ca sử dụng UML và các tài liệu bằng lời nói khác được sử dụng để mô tả dạng khung nhìn này. Các bên liên quan của chế độ khung nhìn này là người dùng cuối, kiến trúc sư, nhà phát triển và tất cả người dùng của 4 chế độ khung nhìn khác. Hình 3.9 trên trang trong nội dung trước mô tả sơ đồ ca sử dụng cho mua sắm trực tuyến.

Dạng khung nhìn kịch bản được sử dụng để thúc đẩy thiết kế kiến trúc trong các giai đoạn trước của phát triển phần mềm và cũng được sử dụng để xác thực phần mềm ở các giai đoạn sau của chu kỳ phát triển

### **2.6.2 Khung nhìn logic hoặc khái niệm**

Khung nhìn logic dựa trên các thực thể miền ứng dụng cần thiết để thực hiện các yêu cầu chức năng. Nó tập trung vào các yêu cầu chức năng, các khối xây dựng chính và các yếu tố trừu tượng chính của hệ thống. Chế độ khung nhìn logic là sự trừu tượng hóa các yêu cầu chức năng của hệ thống. Nó thường được sử dụng để lập mô hình hướng đối tượng (OO) mà từ đó các cấu trúc hệ thống tĩnh và động xuất hiện. Khung nhìn logic chỉ định sự phân rã hệ thống thành các thực thể khái niệm (chẳng hạn như các đối tượng) và các kết nối giữa chúng (chẳng hạn như các liên kết). Chế độ khung nhìn này giúp hiểu được tương tác giữa các thực thể trong miền không gian vấn đề của ứng dụng và sự thay đổi tiềm năng của chúng. Trong một hệ thống hướng đối tượng, các thành phần kiến ​​trúc có thể là các lớp và các đối tượng.

Khung nhìn logic thường được hỗ trợ bởi các biểu đồ tĩnh UML bao gồm biểu đồ lớp/đối tượng và biểu đồ động UML như biểu đồ tổng thể tương tác, biểu đồ trình tự, biểu đồ giao tiếp, biểu đồ trạng thái và biểu đồ hoạt động. Biểu đồ lớp được sử dụng để mô tả khung nhìn khái niệm hoặc logic. Biểu đồ lớp xác định các lớp và các thuộc tính, phương thức và liên kết của chúng với các lớp khác trong hệ thống. Biểu đồ lớp là tĩnh theo nghĩa là nó không mô tả bất kỳ tương tác nào của người dùng cũng như bất kỳ chuỗi tương tác mô-đun nào trong hệ thống.

Một sơ đồ khối cũng có thể được sử dụng để cung cấp một cái nhìn tổng thể về toàn bộ hệ thống. Biểu đồ trình tự cho thấy cách các đối tượng trong hệ thống tương tác. Sơ đồ giao tiếp hiển thị các đối tượng hệ thống và thông điệp truyền giữa chúng theo thứ tự thời gian.

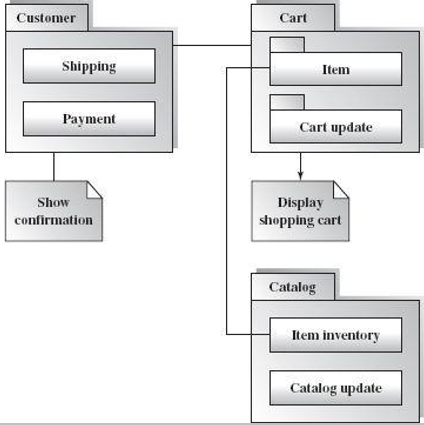
Tóm lại, khung nhìn logic chỉ ra tất cả các nhiệm vụ chính mà hệ thống phải hoàn thành, và trình bày các thành phần chính và mối quan hệ tĩnh của chúng. Các bên liên quan của khung nhìn logic là người dùng cuối, nhà phân tích và nhà thiết kế. Chúng ta có thể áp dụng phương pháp luận thiết kế hướng đối tượng trong khung nhìn logic vì bản thân khung nhìn là hướng đối tượng.

### **2.6.3 Khung nhìn phát triển hoặc Mô-đun**

Khung nhìn phát triển bắt nguồn từ khung nhìn logic và mô tả tổ chức tĩnh của các mô-đun hệ thống. Các mô-đun như không gian tên, thư viện lớp, hệ thống con hoặc gói là các khối xây dựng nhóm các lớp để phát triển và triển khai thêm. Khung nhìn này giải quyết vấn đề tổ chức và phân rã hệ thống con. Phần mềm được đóng gói và phân chia thành các đơn vị nhỏ như thư viện chương trình hoặc hệ thống con do nhiều nhóm nhà phát triển tạo ra. Mỗi gói có khả năng hiển thị và khả năng truy cập riêng như khả năng hiển thị của gói hoặc phạm vi mặc định (xem thảo luận về cấu trúc tĩnh trên phần trước).

Khung nhìn triển khai ánh xạ các phần tử thành phần phần mềm với các thư mục và tệp vật lý thực tế. dd Các bên liên quan của khung nhìn này có thể là lập trình viên và quản lý dự án phần mềm.

Hình 2.17 cho thấy một khung nhìn phát triển đơn giản bằng cách sử dụng một sơ đồ gói.



### **2.6.4 Khung nhìn quy trình**

Khung nhìn quy trình tập trung vào các khía cạnh động của hệ thống, tức là hành vi thời gian thực thi của nó. Khung nhìn này cũng xuất phát từ khung nhìn lôgic. Nó là một sự trừu tượng của các quy trình hoặc luồng xử lý đồng bộ hóa quy trình và đồng thời. Nó góp phần vào nhiều yêu cầu phi chức năng và các thuộc tính chất lượng như yêu cầu về khả năng mở rộng và hiệu suất.

Khung nhìn quy trình xem xét các quy trình của hệ thống và thông tin liên lạc giữa chúng. Một hệ thống phần mềm có thể được phân rã thành nhiều đơn vị thực thi thời gian chạy. Cách tổ chức tất cả các đơn vị thực thi trong thời gian chạy được trình bày trong khung nhìn này. Các thuộc tính chất lượng như hiệu suất, khả năng mở rộng, đồng thời, đồng bộ hóa, phân phối và thông lượng hệ thống đều được giải quyết trong khung nhìn quy trình. Khung nhìn này ánh xạ các chức năng, hoạt động và tương tác vào việc triển khai thời gian chạy với trọng tâm là các yêu cầu phi chức năng cũng như việc thực hiện các yêu cầu chức năng.

Khung nhìn quy trình xử lý các vấn đề đồng thời và đồng bộ hóa giữa các hệ thống con. Nó có thể được mô tả ở một số cấp độ trừu tượng, từ việc thực thi độc lập các mạng logic của các chương trình giao tiếp đến các tác vụ cơ bản chạy trong cùng một nút xử lý. Khung nhìn quy trình cũng phải giải quyết các yêu cầu phi chức năng như đa luồng và truyền thông đồng bộ / không đồng bộ về hiệu suất và tính khả dụng. Sơ đồ hoạt động UML và sơ đồ tổng quan về tương tác hỗ trợ khung nhìn này.

Hình 2.18 trình bày một sơ đồ hoạt động ghi lại khung nhìn quy trình. Lưu ý rằng sau bước kiểm tra tín dụng, hai quy trình được tạo ra để chạy đồng thời.

Các bên liên quan của khung nhìn này là các nhà phát triển và tích hợp. Nhiều kiểu kiến ​​trúc như đường ống và bộ lọc, nhiều tầng và các kiểu khác có thể được áp dụng trong khung nhìn quy trình.



Hình 2.18 Biểu đồ hoạt động trong khung nhìn quy trình

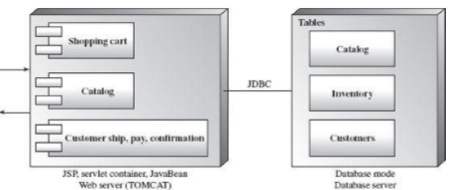
### **2.6.5 Khung nhìn vật lý**

Khung nhìn vật lý mô tả cài đặt, cấu hình và triển khai ứng dụng phần mềm. Nó liên quan đến việc làm thế nào để cung cấp hệ thống có thể triển khai. Khung nhìn vật lý cho thấy ánh xạ của phần mềm vào phần cứng. Nó đặc biệt quan tâm đến các hệ thống phân tán hoặc song song. Các thành phần là các thực thể phần cứng (bộ xử lý), và các liên kết là các đường giao tiếp; cùng với nhau, chúng chỉ định cách các phần tử khác nhau như giao thức truyền thông và máy chủ phần mềm trung gian được tìm thấy trong khung nhìn logic, quy trình và phát triển được ánh xạ vào các nút khác nhau trong môi trường thời gian chạy

Khung nhìn vật lý cũng tính đến các yêu cầu phi chức năng của hệ thống như tính khả dụng của hệ thống, độ tin cậy (khả năng chịu lỗi), hiệu suất thông lượng, hiệu suất khả năng mở rộng và bảo mật. Ví dụ: phần mềm có thể được phân phối trong các bố cục phần cứng và mạng khác nhau, điều này sẽ dẫn đến sự khác biệt đáng kể trong các thuộc tính chất lượng này.

Cấu trúc liên kết hệ thống về phần cứng, mạng và cơ sở hạ tầng khác đều là một phần của khung nhìn này. Khung nhìn này giải thích các yêu cầu phi chức năng và các thuộc tính chất lượng như hiệu suất, tính khả dụng, độ tin cậy và khả năng mở rộng. Khung nhìn vật lý phải giải quyết các kết nối mạng và giao thức truyền thông như các nút máy chủ và cấu hình môi trường phân tán nhiều tầng. Các sơ đồ triển khai UML và tài liệu khác thường được sử dụng để hỗ trợ khung nhìn này.

Các bên liên quan của khung nhìn này là người cài đặt hệ thống, quản trị viên hệ thống, kỹ sư hệ thống và người vận hành. Hình 2.19 trình bày một ví dụ về một sơ đồ triển khai cho thấy rằng hệ thống xử lý đơn hàng được triển khai trên hai máy chủ.



Hình 2.19 Biểu đồ triển khai trong khung nhìn vật lý

### **2.6.6 Khung nhìn giao diện người dùng**

Khung nhìn giao diện người dùng (UI) là một khung nhìn mở rộng cung cấp khung nhìn giao diện người dùng-máy tính rõ ràng và ẩn các chi tiết triển khai. Khung nhìn này có thể được cung cấp dưới dạng một loạt ảnh chụp nhanh màn hình hoặc một bản demo nguyên mẫu tương tác, động. Bất kỳ sửa đổi nào đối với khung nhìn này sẽ có tác động trực tiếp đến khung nhìn kịch bản. Ảnh chụp màn hình trong Hình 3.20 cho thấy giao diện người dùng GUI cho một giỏ hàng trực tuyến



Hình 3.20 Forms trong khung nhìn giao diện người dùng

Tóm lại, khung nhìn 4 + 1 là một kỹ thuật xác minh kiến trúc để nghiên cứu và lập hồ sơ thiết kế kiến trúc phần mềm. Mỗi khung nhìn cung cấp một cửa sổ vào các khía cạnh khác nhau của hệ thống. Khung nhìn 4 + 1 bao gồm tất cả các khía cạnh của kiến trúc phần mềm cho tất cả các bên liên quan. Các khung nhìn được kết nối với nhau; do đó, dựa trên khung nhìn kịch bản, chúng ta có thể bắt đầu với khung nhìn logic, chuyển sang khung nhìn phát triển hoặc quá trình, và cuối cùng chuyển đến khung nhìn vật lý. Khung nhìn giao diện người dùng cũng được thiết lập trong quá trình này.

## **2.7. Ngôn ngữ mô tả kiến trúc (ADL)**

ADL là một đặc tả ký hiệu cung cấp cú pháp và ngữ nghĩa để xác định kiến ​​trúc phần mềm. Nó cũng cung cấp cho các nhà thiết kế khả năng phân rã các thành phần, kết hợp các thành phần và xác định giao diện của các thành phần. ADL là một ngôn ngữ đặc tả chính thức với cú pháp và ngữ nghĩa được xác định rõ ràng được sử dụng để mô tả các thành phần kiến ​​trúc và các kết nối, giao diện và cấu hình của chúng.

Garlan và Shaw (1996) liệt kê các yêu cầu sau đối với ADL:

* Thành phần: “Có thể mô tả một hệ thống như một thành phần của các thành phần và kết nối độc lập.” Hệ thống lớn nên được xây dựng từ các yếu tố cấu thành và có thể xem xét từng yếu tố độc lập với hệ thống.
* Tính trừu tượng: “Có thể mô tả các thành phần và tương tác của chúng theo cách mô tả vai trò trừu tượng của chúng trong một hệ thống.” Không cần thiết phải xem xét các vấn đề thực hiện trong khi xác định kiến ​​trúc.
* Tái sử dụng: Khả năng tái sử dụng nên được tích hợp sẵn ở cấp độ thành phần và kết nối. Việc tạo ra các mẫu kiến ​​trúc cũng cần được hỗ trợ để tạo điều kiện sử dụng lại các mô tả kiến ​​trúc.
* Cấu hình: Mô tả kiến ​​trúc nên cho phép hiểu và sửa đổi kiến ​​trúc mà không cần kiểm tra từng thành phần và trình kết nối.
* Tính không đồng nhất: “Có thể kết hợp nhiều mô tả kiến ​​trúc không đồng nhất.
* Phân tích: Việc sử dụng ADL sẽ tạo điều kiện thuận lợi cho việc phân tích thiết kế kiến ​​trúc. Phân tích có thể bao gồm việc xem xét thông lượng, bế tắc, hành vi đầu vào/đầu ra và lập lịch trình.

Một số ADL đã được đề xuất trong vài năm qua. Chúng bao gồm UniCon và Wright, cả hai đều từ CMU; C2sadel từ UC Irvine và USC; Rapide từ Stanford; và Darwin từ Đại học Imperial, London. Acme là một ADL khác có sẵn trong cộng đồng nghiên cứu. UML cũng có thể cung cấp nhiều tạo tác cần thiết cho mô tả kiến trúc, nhưng không phải là một ADL đầy đủ.

Chúng ta hãy xem xét kỹ một trong những ADL được liệt kê trước đó. Acme cung cấp một cách chính thức và bán chính thức để mô tả một kiến trúc phần mềm như một cấu trúc tĩnh ở mức độ trừu tượng cao. ADL này cung cấp nhiều phần tử khối xây dựng để mô tả thiết kế kiến trúc. Ba trong số các yếu tố cơ bản này là các thành phần, đầu nối và hệ thống; các yếu tố bổ sung bao gồm các cổng, vai trò, biểu diễn và bản đồ lại. Các phần tử thành phần có thể là bất kỳ đơn vị tính toán hoặc lưu trữ dữ liệu nào. Trình kết nối đại diện cho các tương tác giữa các thành phần. Các phần tử kết nối thực hiện giao tiếp và tương tác giữa các thành phần trong hệ thống, chẳng hạn như truyền thông đồng bộ hoặc không đồng bộ.

Các thành phần và kết nối trong Acme đóng vai trò giống như hộp và đường trong sơ đồ khối (hộp và đường). Tuy nhiên, họ có một mục đích cụ thể hơn. Ví dụ, các thành phần có giao diện được xác định bởi một tập hợp các cổng. Mỗi cổng xác định một điểm tiếp xúc giữa thành phần và các máy khách của nó. Một thành phần có thể cung cấp nhiều giao diện bằng cách sử dụng các loại cổng khác nhau. Một cổng đơn giản có thể chỉ đại diện cho một chữ ký thủ tục duy nhất.

Các trình kết nối cũng có các giao diện được xác định bởi một tập hợp các vai trò. Mỗi vai trò đại diện cho một người tham gia tương tác của trình kết nối. Một trình kết nối nhị phân đơn giản có hai vai trò, chẳng hạn như vai trò người gọi và vai trò callee của đầu nối RPC, vai trò đọc và ghi của đường ống hoặc vai trò người gửi và người nhận của trình kết nối truyền thông điệp. Nhiều đầu nối vai trò có thể phục vụ nhiều vai trò; ví dụ: một trình kết nối phát sóng sự kiện có thể có một vai trò người phát hành sự kiện và nhiều vai trò người nhận sự kiện.

Ở đây, chúng ta trình bày một số ví dụ cho thấy Acme ADL và công cụ AcmeStudio hoạt động cùng nhau.

Như được hiển thị trong ảnh chụp màn hình sau, người dùng có thể tạo thiết kế kiến trúc một cách thuận tiện bằng công cụ AcmeStudio. Thiết kế đơn giản bao gồm hai thành phần, một máy khách và một máy chủ, được kết nối với nhau bằng một đầu nối. Mẫu kiến trúc này là mô hình máy khách / máy chủ. Chúng ta có thể xác định tên và thuộc tính của từng thành phần và đầu nối trong sơ đồ. Sơ đồ có thể được AcmeStudio dịch sang một thông số kỹ thuật bằng ngôn ngữ Acme. AcmeStudio có thể kiểm tra bất kỳ ràng buộc nào do các nhà thiết kế đặt trước trong đặc điểm kỹ thuật. AcmeStudio cũng có plug-ins cho nhiều kiểm tra khác, chẳng hạn như phân tích tính di động. Đặc tả Acme cũng có thể được sử dụng để xây dựng một khung mã trong các ngôn ngữ lập trình phổ biến như C ++.

## **2.8. Tổng kết**

Chương này đã trình bày một số khái niệm cơ bản về các biểu đồ UML liên quan đến biểu diễn kiến trúc phần mềm. Đặc biệt nội dung chú trọng đề cập đến mô hình với UML cho biểu diễn kiến trúc dựa trên các khung nhìn vì nó thể hiện các quan điểm khác nhau cần có khi tiến hành mô hình kiến trúc của hệ cần xây dựng.

# CHƯƠNG 3: CÁC THUỘC TÍNH CHẤT LƯỢNG THIẾT KẾ

Mục tiêu của chương này:

* Giới thiệu các thuộc tính chất lượng
* Các kịch bản thuộc tính chất lượng
* Các chiến lược để đạt được thuộc tính chất lượng

## **3.1. Tổng quan**

Trường hợp sử dụng (use case) trong biểu đồ Use case không thể hiện được các yêu cầu phi chức năng. Thuật ngữ yêu cầu phi chức năng thường được dùng để chỉ ra các ràng buộc của hệ thống như tính sửa đổi được, tính hiệu năng, … bên cạnh các yêu cầu chức năng. Điều này đưa vào một phân hoạch hay, rành mạch, nhưng ảo tưởng toàn bộ giữa yêu cầu chức năng và các yêu cầu khác. Trong các hệ thống phần mềm, thường phải đáp ứng các yêu cầu chức năng và yêu cầu phi chức năng. Thuật ngữ yêu cầu phi chức năng sẽ bị tránh trong cuốn sách này để thiên về thuật ngữ yêu cầu thuộc tính chất lượng. Một mô tả điển hình về yêu cầu thuộc tính chất lượng có thể là “hệ thống sẽ có hiệu năng cao”. Loại yêu cầu này không mang tính mô tả và đặt ra chỉ khái niệm chung mà là không thể được cho kiến trúc sư để thiết kế thành hệ thống hay đo trong hệ thống một khi được thực hiện. Điều này nói tới loại hiệu năng gì? Hiệu năng có nghĩa là thời gian đáp ứng, thông lượng, hay cái gì đó khác? Để kiến trúc sư hiểu đầy đủ cái gì được cần dưới dạng đặc tính chất lượng mà là yêu cầu của hệ thống, nhiều mô tả chi tiết hơn dưới dạng kích thích và đáp ứng đặc tính chất lượng là được cần sẽ được trình bày ở những nội dung tiếp theo của tài liệu này.

## **3.2. Kiến trúc và các yêu cầu**

Yêu cầu đối với hệ thống có nhiều dạng: yêu cầu văn bản, mô hình, hệ thống hiện có, ca sử dụng, câu chuyện người dùng, v.v. Bất kể nguồn nào, tất cả các yêu cầu đều bao gồm các danh mục sau:

1. Yêu cầu chức năng. Các yêu cầu này nêu rõ hệ thống phải làm gì và nó phải hoạt động như thế nào hoặc phản ứng với các kích thích thời gian chạy.
2. Yêu cầu thuộc tính chất lượng. Các yêu cầu này là tiêu chuẩn của các yêu cầu chức năng hoặc của sản phẩm tổng thể. Chứng nhận của một yêu cầu chức năng là một mục như chức năng phải được thực hiện nhanh như thế nào, hoặc khả năng phục hồi của nó đối với đầu vào sai. Chất lượng của sản phẩm tổng thể là một mục như thời gian triển khai sản phẩm hoặc giới hạn về chi phí hoạt động
3. Ràng buộc. Ràng buộc là một quyết định thiết kế không có bậc tự do. Đó là một quyết định thiết kế đã được đưa ra. Các ví dụ bao gồm yêu cầu sử dụng một ngôn ngữ lập trình nhất định hoặc sử dụng lại một mô-đun hiện có nhất định hoặc một định hướng quản lý để làm cho dịch vụ hệ thống của bạn được định hướng. Những lựa chọn này được cho là trong tầm nhìn của kiến trúc sư, nhưng các yếu tố bên ngoài (chẳng hạn như không thể đào tạo nhân viên bằng một ngôn ngữ mới, hoặc có thỏa thuận kinh doanh với nhà cung cấp phần mềm hoặc thúc đẩy các mục tiêu kinh doanh về khả năng tương tác dịch vụ) đã dẫn đến những có quyền quyết định các kết quả thiết kế này.

"Đáp ứng" của kiến trúc đối với từng loại yêu cầu này là gì?

1. Các yêu cầu chức năng được thỏa mãn bằng cách phân công một chuỗi trách nhiệm thích hợp trong suốt quá trình thiết kế.
2. Các yêu cầu thuộc tính chất lượng được thỏa mãn bởi các cấu trúc khác nhau được thiết kế trong kiến trúc, và các hành vi và tương tác của các yếu tố tạo nên các cấu trúc đó.
3. Các ràng buộc được thỏa mãn bằng cách chấp nhận quyết định thiết kế và điều chỉnh nó với các quyết định thiết kế bị ảnh hưởng khác

## **3.3. Xác định các yêu cầu thuộc tính chất lượng**

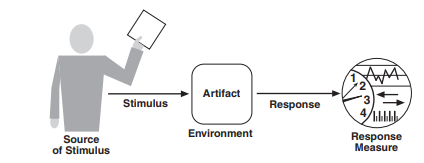
Các thuộc tính chất lượng đã được cộng đồng phần mềm quan tâm ít nhất là từ những năm 1970. Có rất nhiều cách phân loại và định nghĩa đã được xuất bản, và nhiều người trong số họ có cộng đồng nghiên cứu và hoạt động của riêng họ. Từ góc độ kiến trúc sư, có ba vấn đề với các cuộc thảo luận trước đây về các thuộc tính chất lượng hệ thống:

1. Các định nghĩa được cung cấp cho một thuộc tính không thể kiểm tra được. Sẽ là vô nghĩa khi nói rằng một hệ thống sẽ “có thể sửa đổi được”. Mọi hệ thống đều có thể sửa đổi được đối với một tập hợp các thay đổi và không thể sửa đổi đối với hệ thống khác. Về điểm này, các thuộc tính chất lượng khác cũng tương tự: một hệ thống có thể mạnh đối với một số lỗi và dễ hỏng đối với những hệ thống khác.
2. Thảo luận thường tập trung vào chất lượng mà mối quan tâm cụ thể thuộc về. Lỗi hệ thống do tấn công từ chối dịch vụ có phải là khía cạnh khả dụng, khía cạnh hiệu suất, khía cạnh bảo mật hay khía cạnh khả năng sử dụng? Tất cả bốn cộng đồng thuộc tính sẽ yêu cầu quyền sở hữu đối với lỗi hệ thống do tấn công từ chối dịch vụ. Tất cả, ở một mức độ nào đó, đều đúng. Nhưng điều này không giúp chúng tôi, với tư cách là kiến trúc sư, hiểu và tạo ra các giải pháp kiến trúc để quản lý các thuộc tính cần quan tâm
3. Mỗi cộng đồng thuộc tính đã phát triển vốn từ vựng của riêng mình. Cộng đồng hiệu suất có “sự kiện” đến với một hệ thống, cộng đồng bảo mật có “các cuộc tấn công” đến một hệ thống, cộng đồng khả dụng có “sự cố” của một hệ thống và cộng đồng khả năng sử dụng có “đầu vào của người dùng”. Tất cả những điều này thực sự có thể đề cập đến cùng một sự kiện, nhưng chúng được mô tả bằng các thuật ngữ khác nhau

**Kịch bản thuộc tính chất lượng**

Để hướng dẫn chọn lựa thiết kế và đo tính khớp của thiết kế, yêu cầu thuộc tính chất lượng phải được mô tả tương ứng với một số hoàn cảnh vận hành. Để làm điều này chúng ta sẽ dùng kịch bản đặc tính chất lượng (Barbacci et al., 2003) để định nghĩa đầy đủ hơn thuộc tính đặc tính chất lượng mà hệ thống phải có. Kịch bản thuộc tính chất lượng mô tả cho một số yêu cầu thuộc tính chất lượng dưới dạng kích thích, nguồn kích thích, điều kiện môi trường, phần tử thiết kế, đáp ứng, và đo lường đáp ứng. Đây là cơ sở cho khuôn khổ kịch bản thuộc tính chất lượng sáu phần để ghi lại, thương lượng, và phân tích yêu cầu thuộc tính chất lượng. Từng phần tử của khuôn khổ này được thảo luận dưới đây:

* Kích thích (Stimulus): Kích thích là điều kiện ảnh hưởng tới kiến trúc. Đây có thể là biến cố, yêu cầu của người dùng về dữ liệu, khởi đầu dịch vụ, hay thay đổi được đề nghị cho hệ thống.
* Nguồn của kích thích (Stimulus source): Đây là thực thể (người, tổ chức, công nghệ) là nguồn của kích thích được mô tả ở trên. Có thể có một hay nhiều nguồn.
* Điều kiện môi trường liên quan (Environment): Đây là những điều kiện hiện diện trong môi trường vận hành của hệ thống trong khi nhận kích thích. Điều kiện môi trường liên quan có thể là nhiều loại và sẽ tuỳ thuộc vào kích thích, nhưng các ví dụ có thể bao gồm “trong thực hiện khi chạy,” “trong phát triển khởi đầu,” “sau triển khai,” “trong đỉnh tải,” “trong khi bẩy trăm người dùng đăng nhập,” …..
* Phần tử kiến trúc (Artifacts): Đây là phần tử hay các phần tử của kiến trúc mà trực tiếp hay gián tiếp bị ảnh hưởng bởi kích thích. Trong thu thập yêu cầu sớm, khi yêu cầu thuộc tính chất lượng được phát triển khởi đầu, phần tử kiến trúc có lẽ không được biết tới. Tuy nhiên, sau khi thiết kế kiến trúc đã bắt đầu và được cải tiến liên tiếp, thông tin phần tử kiến trúc nên được thêm vào cho thông tin yêu cầu đặc tính chất lượng.
* Đáp ứng hệ thống (Response): Một mô tả về cách những người có liên quan hệ thống muốn kiến trúc / hệ thống đáp ứng với kích thích.
* Đo đáp lường ứng (Response Measure): Việc đo cách hệ thống đáp ứng. Loại đo đáp ứng được liệt kê sẽ tùy thuộc vào kích thích. Với kích thích thay đổi/sửa đổi, chúng ta có thể có biện pháp đáp ứng mà đo chi phí của thay đổi dưới dạng thời gian, nhân lực, chi phí, …. Với kích thích hiệu năng, chúng ta có thể đo đáp ứng dưới dạng thông lượng, thời gian đáp ứng, …



Hình 3.1. Các thành phần của kịch bản thuộc tính chất lượng

Chúng ta có thể nghĩ về các kịch bản đặc tính chất lượng như câu chuyện ngắn hay mô tả về cách hệ thống đáp ứng với kích thích đã cho dưới dạng thuộc tính đặc tính chất lượng mà hệ thống phải có. Các kịch bản đặc tính chất lượng khác với các kịch bản trường hợp sử dụng ở chỗ kịch bản đặc tính chất lượng hội tụ vào đáp ứng đặc tính chất lượng, trong khi trường hợp sử dụng hội tụ vào đáp ứng chức năng cho kích thích. Trong phần lớn các đặc tả yêu cầu, mọi điều kiến trúc sư có được là một từ, như an ninh, hiệu năng, tính sẵn có, …. Để hiểu rõ hơn các yêu cầu đặc tính chất lượng thô kiểu như những điều này trực tiếp từ những người có liên quan, chúng phải được đưa vào các kịch bản đặc tính chất lượng, việc này làm rõ ràng hơn nhu cầu để hướng dẫn chọn lựa thiết kế của kiến trúc sư. Điều này làm mạnh cho khẳng định rằng kiến trúc sư cần là một phần của quá trình yêu cầu để đảm bảo rằng thông tin được cần cho thiết kế được nắm bắt đầy đủ nhất có thể được.

Xét hệ thống SmartMarquee. SmartMarquee là sản phẩm có những nét nổi bật là một bảng hiển thị tấm phẳng, một bộ xử lí, và phần mềm tạo khả năng cho nó được lập cấu hình để hiển thị thông tin và đồ hoạ theo những cách khác nhau, bao gồm hiển thị trôi, ảnh tĩnh, văn bản, và video. Thị trường mục tiêu cho SmartMarquee là các môi trường khác nhau, như nhà hàng, sảnh tiếp tân khách sạn, mặt tiền cửa hàng, …, nơi thông tin quảng cáo có thể được hiển thị. SmartMarquee phải hỗ trợ cho đa dạng cấu hình, như bộ đồ treo trên tường, bộ đồ đứng tự do trên sàn, và bộ đồ dày dạn cho các ứng dụng bên ngoài trời. Thiết bị trong nhà và ứng dụng phải có thể chuyên biệt hoá được để đáp ứng cho các ham muốn thẩm mĩ đích xác và chức năng của người dùng. Phần mềm cho hệ thống này phải linh hoạt và khả chuyển để hỗ trợ cho các cấu hình phần cứng đa dạng. Phần mềm cũng phải hỗ trợ cho các cấu hình cục bộ động và ở xa của không gian quảng cáo. Với ví dụ này, một danh sách điển hình các yêu cầu mức cao cho SmartMarquee bao gồm:

1. Hệ thống sẽ cung cấp năng lực hiển thị văn bản và thông tin quảng cáo đồ hoạ để bao gồm:

1.1 Một tấm bảng với quảng cáo (đồ hoạ) kiểu biểu ngữ trượt

1.2. Một tấm bảng sẽ hiển thị quảng cáo bao gồm văn bản và đồ hoạ trộn lẫn, sẽ hiển thị từng quảng cáo trong một thời kì, rồi chuyển trang sang quảng cáo tiếp

1.3. Một tấm bảng sẽ hiển thị quảng cáo video 4. Một hệ thống hỗ trợ cho âm thanh và tuỳ chọn không âm thanh

2. Hệ thống sẽ là lập cấu hình được như bộ treo trên tường hay đứng tự do.

3. Hệ thống sẽ hỗ trợ cho thiết bị LCD bất kì kích cỡ nào.

4. Hệ thống sẽ cung cấp năng lực để lập cấu hình và tái cấu hình các quảng cáo vào lúc chạy.

5. Hệ thống sẽ hỗ trợ cho việc tái cấu hình quảng cáo tự động, theo kịch đoạn, dựa trên thời gian ban ngày.

6. Hệ thống sẽ cung cấp năng lực để nhận, xử lí, và hiển thị giá thị trường cổ phiếu theo luồng dữ liệu liên tục (trong ngôn ngữ định nghĩa dữ liệu thị trường - Market Data Definition Language (MDDL)) trong tấm biểu ngữ trượt nhất quán với yêu cầu của định nghĩa 1 về tấm biểu ngữ trượt.

7. Hệ thống sẽ hỗ trợ cho lập cấu hình cục bộ và từ xa thông qua ổ đĩa USB, mạng không dây/có dây, hay năng lực quay số điện thoại, tuỳ theo cái nào sẵn có.

8. SmartMarquee sẽ bao gồm một công cụ tương tác sẽ cho phép người quản trị định nghĩa và lập cấu hình các quảng cáo. Công cụ hỗ trợ sẽ cho phép người quản trị xuất chuyển cấu hình để triển khai SmartMarquee một cách cục bộ hay từ xa.

Vì SmartMarquee và ứng dụng phải được chuyên biệt hoá để đáp ứng đích xác mong muốn thẩm mĩ và chức năng của khách hàng, phần mềm hệ thống phải linh hoạt và khả chuyển để hỗ trợ cho đa dạng cấu hình phần cứng.

Rõ ràng có các yêu cầu về tính linh hoạt và tính khả chuyển, nhưng không có đủ thông tin trong mô tả này cho kiến trúc sư để thiết kế các tính năng này trong hệ thống. Từng yêu cầu trong những yêu cầu này có thể nói tới cùng khía cạnh của khả năng của hệ thống để triển khai phần mềm trên bất kì cấu hình phần cứng thiết bị SmartMarquee. Điều này vẫn không đủ thông tin. Sau rốt, nó là phần mềm! Cho đủ thời gian, nó có thể được sửa để vận hành trên bất kì phần cứng nào. Vấn đề không phải là liệu nó có thể được thay đổi không, mà thay vì thế là chi phí của thay đổi. Một khía cạnh then chốt của yêu cầu này đang bị thiếu là ở chỗ chúng ta không hiểu khả chuyển và linh hoạt nghĩa là gì trong hoàn cảnh này. Hệ thống linh hoạt và khả chuyển trong hoàn cảnh này nghĩa là phần mềm của SmartMarquee có thể được cài đặt trên bất kì cấu hình phần cứng thiết bị SmartMarquee với ít nỗ lực, điều dịch thành chi phí và lịch biểu (chi phí) được giảm đi. Điều không được biết là những người có liên quan sẵn lòng trả bao nhiêu cho tính khả chuyển/linh hoạt và khi nào họ muốn trả cho nó. Chẳng hạn, những người có liên quan có giả định chi phí thiết kế cao hơn so với chi phí phát triển và triển khai được giảm bớt, hay điều mấu chốt hơn là có được cái gì đó nhanh chóng và trả nhiều hơn trong chi phí phát triển và triển khai khi chúng ta phải chuyển lên nền mới? Việc của kiến trúc sư bây giờ là thúc ép những người có liên quan để có một câu chuyện đặc biệt mô tả tính linh hoạt và khả chuyển, nhưng làm sao tương tác này có thể được cấu trúc?

Kiến trúc sư có thể dùng khuôn khổ kịch bản đặc tính chất lượng sáu phần để hướng dẫn thảo luận với những người có liên quan để phát triển kịch bản đặc tính chất lượng để mô tả cho yêu cầu này. Điều được cần là một thể nghiệm đặc biệt của kích thích xuất hiện dưới các điều kiện môi trường đặc biệt, và rồi mô tả cách hệ thống nên đáp ứng với kích thích đó trong những điều kiện đó. Mặc dầu khó, kiến trúc sư vẫn phải ép những người có liên quan phải cụ thể nhất có thể được trong việc xác định đáp ứng cho kích thích này. Chẳng hạn, xem kịch bản sáu phần ở trên.

Một khi được nắm bắt, kịch bản đặc tính chất lượng có thể được phát biểu lại trong hình thức lời văn, nếu nuốn, như sau:

Khách hàng đặt một thiết bị SmartMarquee đặc biệt đáp ứng nhu cầu thẩm mĩ và vận hành của người đó. Sau khi nền thiết bị được lắp ráp ứng dụng SmartMarquee có thể được tải lên nền phần cứng trong ít hơn 30 phút mà không cần chuyên biệt phát triển phần mềm hay biên dịch lại, dùng một cấu hình chuẩn, thông thường với tất cả những chức năng ứng dụng cố hữu và thuộc tính sẵn có.

Mặc dầu một mô tả rõ ràng hơn nhiều về yêu cầu tính linh hoạt/khả chuyển, kịch bản này có thể được phân tích thêm, dẫn tới nhiều câu hỏi hơn mà có thể đã không được hỏi và còn lại không được nói ra mãi cho tới khi mã đã được viết và phần cứng được thiết kế. Vài câu hỏi tức khắc có thể bao gồm: Thuật ngữ nền thực sự ngụ ý gì? Nó có nói tới bộ xử lí hay chỉ là chỗ để? Điều này có nghĩa rằng các nền khác nhau có thể dùng các thiết bị hiển thị khác kích cỡ không? Các kĩ thuật viên biết gì? Họ có là những kĩ thuật viên máy tính có đào tạo hay không đủ tư cách?

Bảng 3.1 dưới đây là Ví dụ về kịch bản đặc tính chất lượng sáu phần cho SmartMarquee

|  |  |
| --- | --- |
| **Đặc tính chất lượng thô** | Tính linh hoạt và khả chuyển của SmartMarquee |
| **Kích thích** | Cài đặt ứng dụng SmartMarquee trên nền phần cứng thiết bị chuyên biệt mới |
| **Nguồn kích thích** | Khách hàng |
| **Điều kiện môi trường liên quan** | Cài đặt ứng dụng SmartMarquee trên nền phần cứng thiết bị chuyên biệt mới |
| **Các phần tử kiến trúc** | Ứng dụng SmartMarquee và phần cứng SmartMarquee |
| **Đáp ứng hệ thống** | Ứng dụng Smart Marquee được nạp vào nền phần cứng thiết bị chuyên biệt bởi kĩ thuật viên. |
| **Đo lường đáp ứng** | Cài đặt ứng dụng nên mất ít hơn 30 phút mà không phát triển hay biên dịch lại phần mềm chuyên biệt, dùng cấu hình thông thường, chuẩn và thủ tục cài đặt. Mọi chức năng ứng dụng và thuộc tính đều hiện diện trên nền chuyên biệt |

Khi các yêu cầu đặc tính chất lượng được phát triển, chúng sẽ dẫn tới nhiều câu hỏi hơn mà sẽ nảy sinh trong việc thu nhận nhiều thông tin về thuộc tính đặc tính chất lượng mà hệ thống phải có. Khi thiết kế tiến triển, các yêu cầu đặc tính chất lượng có thể cần được duyệt xét lại và cải tiến. Như chúng ta sẽ thấy về sau, tuyển tập các kịch bản đặc tính chất lượng sẽ được cải tiến qua quá trình thiết kế để trả lời các câu hỏi như thế này khi chúng nổi lên. Khuôn khổ kịch bản sáu phần giúp tăng cường suy nghĩ sâu hơn về các yêu cầu đặc tính chất lượng thực sự ngụ ý gì và buộc những người có liên quan nghĩ về cách họ muốn hệ thống của họ đáp ứng với kích thích đặc biệt thế nào. Yêu cầu đặc tính chất lượng có thể có nhiều khía cạnh. Chẳng hạn, hiệu năng có thể nói tới thời gian đáp ứng nhanh, thông lượng, tính lịch biểu, đáp ứng tất định, hay cái gì đó khác. Một hay nhiều kịch bản có thể được yêu cầu để nắm bắt thích hợp yêu cầu đặc tính chất lượng của hiệu năng. Được lấy như một nhóm, chúng sẽ mô tả cho yêu cầu đặc tính chất lượng đặc thù cho hệ thống. Tất cả các kịch bản đặc tính chất lượng đều mô tả cho thuộc tính đặc tính chất lượng mà hệ thống phải có. Một khi được phát triển đầy đủ, những từ chúng ta bắt đầu với ví dụ SmartMarquee, tính linh hoạt và tính khả chuyển, được mô tả qua kịch bản đặc tính chất lượng theo cách mà thực tế sẽ tạo điều kiện cho thiết kế. Chúng ta cũng sẽ thấy cách những kịch bản này có thể được dùng để đánh giá thiết kế kiến trúc và thậm chí còn kiểm thử hệ thống một khi đã được thực hiện. Kịch bản đặc tính chất lượng mô tả các thuộc tính hệ thống mơ hồ mà hệ thống phải có theo thuật ngữ định lượng nhiều hơn. Thông thường những mơ hồ này tồn tại trong tâm trí của người có liên quan, và kiến trúc sư phải làm việc với những người có liên quan để nhận diện và giải quyết mơ hồ này. Dùng khuôn khổ kịch bản đặc tính chất lượng sáu phần giúp cho các kiến trúc sư hỏi câu hỏi đúng để thương lượng và xác định các yêu cầu đặc tính chất lượng và làm rõ ràng nghĩa của chúng cho tất cả những người có liên quan Đặc tính chất lượng không bị hạn chế vào miền phần mềm đặc thù và có thể được dùng trong bất kì miền hệ thống dùng nhiều phần mềm nào, từ những ứng dụng nhúng chặt, nhỏ tới các hệ CNTT toàn cầu, lớn. Trong một ví dụ công nghệ thông tin, những người có liên quan hệ thống chỉ ra rằng hệ thống phải tin cậy. Thúc giục thêm những người có liên quan để khám phá ra nghĩa của tính tin cậy trong hoàn cảnh hệ thống, chúng ta khám phá ra rằng tính năng xử lí đơn hàng phải là trực tuyến và sẵn có 24/7, và không làm mất giao tác đặt hàng của khách hàng. Từ tính tin cậy không thâu tóm được chiều sâu thực của yêu cầu này, cho nên sẽ không thể đảm bảo được rằng hệ thống có thể được thiết kế thích hợp bằng những thuộc tính này. Dùng khuôn khổ sáu phần, chúng ta có thể dễ dàng chỉ ra thông tin thiếu trong yêu cầu này và hỏi câu hỏi đúng để tạo ra kịch bản đặc tính chất lượng hữu ích hơn (Bảng 3.2).

**Bảng 3.2 Ví dụ về Kịch bản Đặc tính chất lượng sáu phần cho máy phục vụ xử lí đơn hàng**

|  |  |
| --- | --- |
| **Đặc tính chất lượng thô** | Tính sẵn có máy phục vụ |
| **Kích thích** | Hệ thống xử lý đơn hàng chính trải qua hỏng phần cứng thảm hoạ |
| **Nguồn kích thích** | Phần cứng bộ xử lý nguồn phục vụ hệ thống xử lý đơn hàng |
| **Điều kiện môi trường liên quan** | Trong khi tải dùng đang ở mức đỉnh |
| **Các phần tử kiến trúc** | Trong khi tải dùng đang ở mức đỉnh |
| **Đáp ứng hệ thống** | Lỗi phần cứng được phát hiện, xử lí được chuyển từ nguồn phục vụ chính sang nguồn phục vụ phụ, và xử lí được cho chạy lại |
| **Đo lường đáp ứng** | Xử lí được cho chạy lại trong ít hơn 30 giây và không giao tác nào bị mất |

Trong tình huống này, chúng ta có thể thấy rằng một loại hỏng hóc phần cứng được thăm dò và đáp ứng mong muốn của hệ thống trong đối diện với hỏng hóc này được mô tả, và rằng các kịch bản khác cũng có thể được yêu cầu để mô tả đầy đủ thuộc tính tính tin cậy được yêu cầu của hệ thống. Lưu ý rằng trong kịch bản này biện pháp đáp ứng được mô tả dưới dạng tính sẵn có (“không giao tác nào bị mất”) nhưng cũng dưới dạng hiệu năng (“xử lí được quay trở lại trong không đầy 30 giây”). Điều này minh hoạ cho cách các yêu cầu đặc tính chất lượng thường bị xoắn xuýt và đa chiều. Ở đây tính sẵn có hệ thống là tuỳ thuộc vào việc không giao tác nào bị mất và khối lượng thời gian nó phục hồi khỏi lỗi. Thiết kế kiến trúc là mấu chốt để cân bằng những mối quan tâm tới đặc tính chất lượng trước khi thiết kế chi tiết, thực hiện, hay đầu tư vào nâng cấp cho hệ thống dùng nhiều phần mềm. Việc này tăng cường thêm quan điểm rằng lời dẫn đơn thuần, như tính sẵn có, hiệu năng, hay an ninh, là không thích hợp để mô tả các yêu cầu đặc tính chất lượng mơ hồ và phức tạp.

Ví dụ khác về cách các yêu cầu đặc tính chất lượng trở nên xoắn xuýt với nhau thông thường là trong yêu cầu tính dùng được. Tính dùng được thường xuyên là một đặc tính chất lượng quan trọng trong hệ thống CNTT, và trong nhiều trường hợp vấn đề thiết kế hệ thống được dán nhãn là vấn đề tính dùng được. Các vấn đề hệ thống thường được dán nhãn là vấn đề tính sử dụng được bởi vì người dùng cuối thường cảm nhận vấn đề hệ thống như khó khó khăn trong việc dùng hệ thống. Tính dùng được của hệ thống có thể bị tổn hại khi các thuộc tính chất lượng khác không được thoả mãn bên trong hệ thống. Chẳng hạn, nếu một hệ thống mà chậm, điều này sẽ tác động lên tính sử dụng được cảm nhận về hệ thống; tuy nhiên, vấn đề nền tảng thực là hiệu năng. Một ví dụ thông thường khác là tình huống mà hệ thống yêu cầu rằng bạn đưa vào mật khẩu nhiều lần để thu được truy nhập và dùng hệ thống. Lần nữa, người ta có thể tranh luận đây là vấn đề tính sử dụng. Tuy nhiên, vấn đền nền tảng thực có lẽ là mô hình an ninh yếu. Người ta thậm chí có thể tranh cãi rằng nếu dễ thêm vào chức năng, thế thì hệ thống sẽ có tính dùng được nhiều hơn, nhưng tính dùng được này được đạt tới bằng cách thiết kế hệ thống mà sẽ dễ dàng điều chỉnh những loại thay đổi nào đó. Đây là lí do tại sao mô tả yêu cầu đặc tính chất lượng là quan trọng thế bằng việc dùng kịch bản đặc tính chất lượng.

Tính dùng được là thuộc tính rất quan trọng cho nhiều hệ thống, và giao diện người dùng xứng đáng chú ý thiết kế hạng nhất. Tuy nhiên, thiết kế giao diện người dùng thường có thể bị phân tách và cô lập với thiết kế kiến trúc tổng thể. Giao diện người dùng trở thành một phần tử trong thiết kế tổng thể. Bởi vì kiến trúc của hệ thống được phơi ra cho người dùng cuối của hệ thống, tính dùng được là rất nhạy cảm với thuộc tính đặc tính chất lượng của hệ thống. Tuy nhiên, vấn đề với tính dùng lại thường là triệu chứng của cái gì đó sâu hơn mà sai với thiết kế của hệ thống. Điều này có thể được so sánh với người bị sốt. Người ta có thể bị sốt bởi vì cúm. Tuy nhiên, người bị sốt rét, bệnh mồ hóng, hàng hàng trăm bệnh khác cũng có thể bị sốt. Sốt là triệu chứng của vấn đề hệ thống và chỉ nên phục vụ như điểm bắt đầu cho phân tích sâu hơn về các thuộc tính hệ thống khác. Mặc dầu triệu chứng là hệt như trong từng trường hợp, nguyên nhân nền tảng và cách trị liệu là rất khác nhau

Năng lực của hệ thống đáp ứng cho mọi yêu cầu đặc tính chất lượng phụ thuộc tối cao vào tập các cấu trúc kiến trúc được kiến trúc sư lựa chọn. Nếu kiến trúc sư có hiểu biết không chính xác về dẫn lái kiến trúc, thì người đó có thể tạo ra thiết kế không thích hợp.

**Chiến thuật**

Viện Kĩ nghệ Phần mềm (SEI) gọi những chọn lựa thiết kế kiến trúc là chiến thuật (Bass et al., 2003), đây là khái niệm quan trọng mới nổi lên xứng đáng nhắc tới ở đây. SEI đã phát triển các chiến thuật cho các đặc tính chất lượng về tính sẵn có, tính hiệu năng, tính sửa đổi, tính an ninh, tính dùng được, và tính kiểm thử được. Chúng ta sẽ tìm hiểu kỹ hơn các chiến thuật đạt được các thuộc tính chất lượng ở nội dung tiếp theo.

## **3.4. Thuộc tính chất lượng (quality attribute - QA)**

Phần lớn cuộc đời của kiến trúc sư kiến trúc được dành để thiết kế hệ thống phần mềm đáp ứng một loạt các yêu cầu thuộc tính chất lượng.

Cũng như các chức năng của hệ thống không tự đứng vững nếu không xem xét kỹ lưỡng các thuộc tính chất lượng khác, các thuộc tính chất lượng cũng không tự đứng vững; chúng liên quan đến các chức năng của hệ thống. Nếu yêu cầu chức năng là “Khi người dùng nhấn nút màu xanh lục, hộp thoại Tùy chọn xuất hiện”, chú thích thuộc tính chất lưuongj về hiệu suất có thể mô tả hộp thoại sẽ xuất hiện nhanh như thế nào; chú thích về tính khả dụng của thuộc tính chất lượng có thể mô tả tần suất hoạt động của chức năng này và tốc độ sửa chữa nó; một chú thích QA về khả năng sử dụng có thể mô tả việc học hàm này dễ dàng như thế nào.

Các thuộc tính chất lượng được xác định trong quá trình phân tích yêu cầu. Các thuộc tính chất lượng có thể được phân loại thành ba nhóm sau:

1. Các thuộc tính triển khai (không thể quan sát được trong thời gian chạy)

* Khả năng tương tác: khả năng tiếp cận phổ quát và khả năng trao đổi dữ liệu giữa các thành phần bên trong và với thế giới bên ngoài. Khả năng tương tác đòi hỏi sự phụ thuộc lỏng lẻo của cơ sở hạ tầng.
* Khả năng bảo trì và khả năng mở rộng: khả năng sửa đổi hệ thống và mở rộng nó một cách thuận tiện.
* Khả năng kiểm thử: mức độ mà hệ thống tạo điều kiện cho việc thiết lập các trường hợp kiểm thử. Khả năng kiểm thử thường yêu cầu một bộ tài liệu đầy đủ kèm theo thiết kế và triển khai hệ thống.
* Tính khả chuyển: mức độ độc lập của hệ thống trên nền tảng phần mềm và phần cứng. Các hệ thống được phát triển bằng ngôn ngữ lập trình cấp cao thường có tính di động tốt. Một ví dụ điển hình là Java — hầu hết các chương trình Java chỉ cần được biên dịch một lần và có thể chạy ở mọi nơi.
* Khả năng mở rộng: khả năng của hệ thống thích ứng với sự gia tăng các yêu cầu của người dùng. Khả năng mở rộng gây bất lợi cho việc thiết kế hệ thống.
* Tính linh hoạt: dễ dàng sửa đổi hệ thống để phục vụ cho các môi trường hoặc các vấn đề khác nhau mà hệ thống không được thiết kế ban đầu. Các hệ thống được phát triển bằng cách sử dụng kiến ​​trúc dựa trên thành phần hoặc kiến ​​trúc hướng dịch vụ thường có thuộc tính này.

1. Thuộc tính thời gian chạy (có thể quan sát được trong thời gian chạy)

* Tính sẵn sàng: khả năng của hệ thống luôn sẵn sàng 24/7. Tính sẵn dùng có thể đạt được thông qua sao chép và thiết kế cẩn thận để đối phó với các lỗi phần cứng, phần mềm hoặc mạng.
* Bảo mật: khả năng của hệ thống đối phó với các cuộc tấn công nguy hiểm từ bên ngoài hoặc bên trong hệ thống. Bảo mật có thể được cải thiện bằng cách cài đặt tường lửa, thiết lập các quy trình xác thực và ủy quyền cũng như sử dụng mã hóa.
* Hiệu suất: tăng hiệu quả của hệ thống liên quan đến thời gian phản hồi, thông lượng và sử dụng tài nguyên, các thuộc tính thường xung đột với nhau.
* Khả năng sử dụng: mức độ hài lòng của con người khi sử dụng hệ thống. Khả năng sử dụng bao gồm các vấn đề về tính đầy đủ, tính đúng đắn, tính tương thích, cũng như giao diện người dùng thân thiện, tài liệu đầy đủ và hỗ trợ kỹ thuật.
* Độ tin cậy: tần suất lỗi, độ chính xác của kết quả đầu ra, Trung bình-Thời gian-Thất bại (MTTF), khả năng phục hồi sau lỗi và khả năng dự đoán lỗi.
* Khả năng bảo trì (khả năng mở rộng, khả năng thích ứng, khả năng phục vụ, khả năng kiểm tra, tính tương thích và khả năng cấu hình): sự dễ dàng của việc thay đổi hệ thống phần mềm

3. Thuộc tính nghiệp vụ:

* Thời gian đưa ra thị trường: khoảng thời gian từ khi phân tích yêu cầu đến ngày sản phẩm được tung ra thị trường.
* Chi phí: chi phí xây dựng, bảo trì và vận hành hệ thống.
* Thời gian tồn tại: khoảng thời gian mà sản phẩm “còn sống” trước khi nghỉ hưu

Trong nhiều trường hợp, không một mẫu kiến trúc nào có thể đáp ứng đồng thời tất cả các thuộc tính chất lượng. Các kiến trúc sư phần mềm thường cần cân bằng sự cân bằng giữa các thuộc tính. Các cặp cân bằng thuộc tính chất lượng điển hình bao gồm:

* Cân bằng giữa không gian và thời gian. Ví dụ, để tăng hiệu quả thời gian của bảng băm đồng nghĩa với việc giảm hiệu quả không gian của nó.
* Cân bằng giữa độ tin cậy và hiệu suất. Ví dụ, các chương trình Java được bảo vệ tốt chống tràn bộ đệm do các biện pháp bảo mật như kiểm tra ranh giới trên mảng. Các tính năng đáng tin cậy như vậy đi kèm với chi phí hiệu quả về thời gian, so với ngôn ngữ C đơn giản hơn và nhanh hơn, cung cấp các con trỏ “nguy hiểm,” nhưng hiệu quả.
* Cân bằng giữa khả năng mở rộng và hiệu suất. Ví dụ, một cách tiếp cận điển hình để tăng khả năng mở rộng của dịch vụ là tái tạo máy chủ. Để đảm bảo tính nhất quán của tất cả các máy chủ (ví dụ: để đảm bảo rằng mỗi máy chủ có cùng một dữ liệu nhất quán về mặt logic), hiệu suất của toàn bộ dịch vụ sẽ bị ảnh hưởng.

Khi một mẫu kiến trúc không thỏa mãn tất cả các thuộc tính chất lượng mong muốn, kiến trúc sư phần mềm làm việc với các nhà phân tích hệ thống và các bên liên quan để xác định mức độ ưu tiên của các thuộc tính chất lượng. Bằng cách liệt kê các thiết kế kiến trúc thay thế và tính toán đánh giá có trọng số của các thuộc tính chất lượng, kiến trúc sư phần mềm có thể chọn thiết kế tối ưu.

Để có ý nghĩa, các yêu cầu thuộc tính chất lượng phải cụ thể về cách một ứng dụng phải đạt được một nhu cầu nhất định. Một vấn đề phổ biến thường xuyên gặp phải trong các tài liệu kiến trúc là một tuyên bố chung chung như “Ứng dụng phải có khả năng mở rộng”.

Điều này là quá không chính xác và thực sự không có nhiều tác dụng đối với bất kỳ ai. Như được thảo luận ở phần sau của chương này, các yêu cầu về khả năng mở rộng rất nhiều và đa dạng, và mỗi yêu cầu liên quan đến các đặc điểm ứng dụng khác nhau. Vì vậy, ví dụ về ứng dụng dưới đây phải mở rộng quy mô để xử lý các kết nối người dùng đồng thời tăng lên không? Hoặc khối lượng dữ liệu tăng lên? Hay triển khai đến cơ sở người dùng lớn hơn? Hay tất cả những điều trên.

Việc xác định các biện pháp khả năng mở rộng nào trong số các biện pháp khả năng mở rộng này phải được hỗ trợ bởi hệ thống là rất quan trọng từ góc độ kiến trúc, vì các giải pháp là khác nhau. Do đó, điều quan trọng là xác định các yêu cầu thuộc tính chất lượng cụ thể, như là:

*“Phải có thể mở rộng quy mô triển khai từ 100 máy tính PC lên 10000 máy tính PC mà không tăng nỗ lực/chi phí cho việc cài đặt và cấu hình”*

Điều này là chính xác và có ý nghĩa. Là một kiến trúc sư, điều này chỉ cho tôi một con đường dẫn tới một tập hợp các giải pháp và công nghệ cụ thể tạo điều kiện thuận lợi cho việc cài đặt và triển khai không tốn công sức.

Tuy nhiên, lưu ý rằng nhiều thuộc tính chất lượng thực sư hơi khó xác thực và kiểm tra. Trong ví dụ này, không chắc trong quá trình thử nghiệm cho bản phát hành đầu tiên, một trường hợp thử nghiệm sẽ cài đặt và cấu hình ứng dụng trên 10000 máy tính để bàn.

Đây là lúc cảm giác và trải nghiệm chung xuất hiện. Giải pháp được thông qua rõ ràng phải hoạt động cho việc triển khai bàn đầu cho 100 người dùng. Dựa trên các cơ chế chính xác được sử dụng trong giải pháp (có thể là tải xuống từ Internet, phần mềm quản lý máy tính để bàn của công ty…) sau đó chúng tôi chỉ có thể phân tích nó trong khả năng tốt nhất của mình để đánh giá liệu có thể đáp ứng được yêu cầu cụ thể về khả năng mở rộng hay không? Nếu không có sai sót hoặc vấn đề rõ ràng, nó có thể an toàn để giải định rằng giải pháp sẽ mở rộng. Nhưng nó sẽ mở rộng đến 10.000? Như mọi khi với mọi phần mềm, chỉ có một cách để hoàn toàn chắc chắn 100% vì “tất cả chỉ là nói cho đến chạy hệ thống”

**3.4.1. Hiệu suất**

Mặc dù đối với nhiều ứng dụng CNTT, hiệu suất không phải là một vấn đề thực sự lớn, nhưng nó nhận được tất cả sự chú ý trong cộng đồng thuộc tính chất lượng. Tuy nhiên theo Ian Gorton thì lại cho rằng nó là một trong những thuộc tính chất lượng của một ứng dụng thường có thể được định lượng và xác thực một cách dễ dàng. Dù lý do là gì, khi hiệu suất là vấn đề, nó thực sự quan trọng.

**a. Thông lượng**

Thông lượng là thước đo lượng công việc mà một ứng dụng phải thực hiện trong đơn vị thời gian. Công việc thường được đo bằng giao dịch mỗi giây (tps) hoặc tin nhắn được xử lý mỗi giây (mps). Ví dụ: một ứng dụng ngân hàng trực tuyến có thể phải đảm bảo nó có thể thực hiện 1000 giao dịch mỗi giây từ khách hàng sử dụng dịch vụ ngân hàng trực tuyến. Hệ thống quản lý hàng tồn kho cho một nhà kho lớn có thể cần xử lý 50 tin nhắn mỗi giây từ các đối tác thương mại.

Điều quan trọng là phải hiểu chính xác ý nghĩa của yêu cầu thông lượng. Đó là thông lượng trung bình trong một khoảng thời gian nhất định (ví dụ: một ngày làm việc) hay thông lượng cao nhất? Đây là một sự khác biệt quan trọng.

Một ví dụ minh hoạ rõ về điều này là ứng dụng để đặt cược vào các sự kiện như đua ngựa. Trong hầu hết thời gian, ứng dụng này hoạt động rất ít và do đó có yêu cầu thông lượng trung bình thấp và dễ đạt được. Tuy nhiên, mỗi khi có một sự kiện đua xe, khoảng thời gian năm phút trước mỗi cuộc đua đều chứng kiến hàng trăm cược được đặt mỗi giây. Nếu ứng dụng không thể xử lý các cược này khi chúng được đặt, thì doanh nghiệp sẽ thua lỗ và người dùng không hài lòng. Do đó, đối với trường hợp này, ứng dụng phải được thiết kế để đáp ứng thông lượng đỉnh dự kiến, không phải trung bình. Trong thực tế, nếu ứng dụng chỉ hỗ trợ thông lượng trung bình có thể là một thảm hoạ.

**b. Thời gian phản hồi**

Đây là thước đo độ trễ mà ứng dụng thể hiện trong quá trình xử lý nghiệp vụ. Thời gian phản hồi thường xuyên nhất (nhưng không phải riêng) vì được kết hợp với thời gian ứng dụng cần để phản hồi một số thông tin đầu vào. Thời gian phản hồi nhanh cho phép người dùng làm việc hiệu quả hơn và do đó rất tốt cho công việc kinh doanh. Một ví dụ điển hình là ứng dụng điểm bán hàng hỗ trợ một cửa hàng lớn. Khi một mặt hàng được quét lúc thành toán, hệ thống sẽ nhận được phản hồi nhanh thông tin về giá bằng giây hoặc ít điều đó có nghĩa là khách hàng có thể được phục vụ nhanh chóng. Điều này làm cho khách hàng và cửa hàng hài lòng và điều đó tốt cho tất cả các bên liên quan.

Một lần nữa điều quan trọng là phải phân biệt giữa thời gian phản hồi được đảm bảo và thời gian phản hồi trung bình. Một số ứng dụng có thể cần tất cả các yêu cầu để được phục vụ trong một thời hạn nhất định. Đây là thời gian phản hồi được đảm bảo. Những người khác có thể chỉ định thời gian phản hồi trung bình, cho phép độ trễ lớn hơn khi ứng dụng cực kỳ bận. Nó cũng phổ biến trong trường hợp để chỉ định yêu cầu thời gian phản hồi giới hạn trên. Ví dụ: 95% tất cả các yêu cầu phải được xử lý trong vòng chưa đầy bốn giây và không có yêu cầu nào phải mất hơn 15 giây.

**c. Thời hạn**

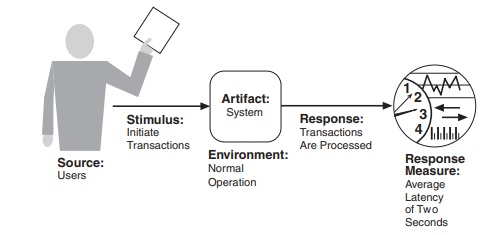
Mọi người có lẽ đã nghe nói về hệ thống dự báo thời tiết mất 36 giờ để đưa ra dự báo cho ngày hôm sau! Tuy nhiên, không chắc đây có phải là ngụy tạo hay không, nhưng đó là một ví dụ tuyệt vời về yêu cầu đáp ứng thời hạn hiệu suất. Thời hạn trong thế giới CNTT thường được liên kết với hàng loạt hệ thống. Hệ thống thanh toán an sinh xã hội phải hoàn thành kịp thời để gửi các khoản thanh toán của người yêu cầu vào tài khoản của họ vào một ngày nhất định. Nếu kết thúc muộn, những người yêu cầu sẽ không được thanh toán như họ mong đợi và điều này có thể gây ra sự gián đoạn và ảnh hưởng, không chỉ cho những người yêu cầu bồi thường. Nói chung, bất kỳ ứng dụng nào có khoảng thời gian giới hạn để hoàn thành sẽ có yêu cầu về thời hạn hiệu suất.

Ba thuộc tính hiệu suất này đều có thể được chỉ định rõ ràng và xác minh. Tuy nhiên, có một cạm bẫy phổ biến cần tránh. Nó nằm trong định nghĩa của một giao dịch, yêu cầu hoặc tin nhắn, tất cả đều được sử dụng rất thiếu chính xác ở phần trên. Về cơ bản, đây là định nghĩa về khối lượng công việc của một ứng dụng. Số lượng xử lý cần thiết cho một giao dịch nghiệp vụ nhất định là một thước đo ứng dụng cụ thể. Ngay cả trong một ứng dụng, có thể sẽ có nhiều loại yêu cầu hoặc giao dịch khác nhau, có thể thay đổi từ các yêu cầu đọc cơ sở dữ liệu nhanh đến các bản cập nhật phức tạp cho nhiều cơ sở dữ liệu phân tán.

**Kịch bản chung cho hiệu suất**

|  |  |
| --- | --- |
| **Phần kịch bản** | **Những giá trị khả thi** |
| Nguồn | Bên trong hoặc bên ngoài hệ thống |
| Kích thích | Xuất hiện một sự kiện định kỳ, rời rạc hoặc ngẫu nhiên |
| Phần tử kiến trúc | Hệ thống hoặc một hoặc nhiều thành phần trong hệ thống |
| Môi trường | Chế độ hoạt động: bình thường, khẩn cấp, tải cao điểm, quá tải |
| Đáp ứng | Xử lý sự kiện, thay đổi mức độ dịch vụ |
| Đo lường đáp ứng | Độ trễ, thời hạn, Thông lượng, chập chờn, tỷ lệ bỏ lỡ |

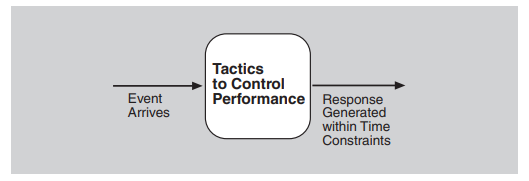
Hình 3.4 là một ví dụ cụ thể về kịch bản hiệu suất cụ thể. Nguoiwf dùng bắt đầu giao dịch theo các hoạt động bình thường. Hệ thống xử lý các giao dịch với độ trễ trung bình là hai giây.



Hình 3.2 Kịch bản hiệu suất cụ thể.

**Chiến thuật cho hiệu suất**

Mục tiêu của chiến thuật hiệu suất là tạo ra phản hồi cho một sự kiện đến hệ thống trong một số ràng buộc dựa trên thời gian. Sự kiện có thể là đơn lẻ hoặc một luồng và là trình kích hoạt để thực hiện tính toán. Các chiến thuật hiệu suất kiểm soát thời gian tạo phản hồi.



Hình 3.3 Mục tiêu của chiến thuật hiệu suất

Vào bất kỳ thời điểm nào trong khoảng thời gian sau khi sự kiện đến nhưng trước khi phản hồi của hệ thống chưa hoàn tất, hệ thống đang hoạt động để phản hồi sự kiện đó hoặc quá trình xử lý bị chặn vì một số lý do. Điều này dẫn đến hai yếu tố cơ bản đóng góp vào thời gian phản hồi: thời gian xử lý – processing time (khi hệ thống đang hoạt động để phản hồi) và thời gian bị chặn – blocked time (khi hệ thống không thể phản hồi)

- *Thời gian xử lý:* quá trình xử lý tiêu tốn tài nguyên, mất thời gian. Sự kiện được xử lý bằng cách thực thi một hoặc nhiều thành phần mà thời gian sử dụng là tài nguyên. Tài nguyên phần cứng bao gồm CPU, kho dữ liệu, băng thông mạng và bộ nhớ. Tài nguyên phần mềm bao gồm các thực thể được xác định bởi hệ thống theo thiết kế. Ví dụ, bộ đếm phải được quản lý và việc truy cập vào các phân quan trọng phải được thực hiện tuần tự.

Ví dụ: giả sử một thông báo được tạo bởi một thành phần. Sau đó nó sẽ được chuyển đi trong môi trường mạng tới một thành phần khác. Sau đó, nó được đặt trong một bộ đệm; biến đổi, xử lý theo một số thuật toán; chuyển đổi cho đầu ra; được cài đặt trong một bộ đệm đầu ra và được gửi tới một thành phần khác, một hệ thống khác hoặc một số tác nhân. Mỗi bước trong số này tiêu tốn tài nguyên, thời gian và góp phần vào độ trễ chung của quá trình xử lý sự kiện đó.

Các nguồn lực khác nhau hoạt động khác nhau khi việc sử dụng chúng đạt đến khả năng của chúng – nghĩa là khi chúng trở nên bão hoà. Ví dụ, khi CPU trở nên tải nặng hơn, hiệu suất thường giảm khá ổn định. Mặt khác, khi bạn bắt đầu hết bộ nhớ, tại một số điểm, việc hoán đổi trang trở nên quá tải và hiệu suất bị treo đột ngột.

- Thời gian bị chặn. Một phép tính có thể bị chặn vì tranh chấp một số tài nguyên cần thiết, vì tài nguyên không có sẵn hoặc vì việc tính toán phụ thuộc vào kết quả của các phép tính khác chưa có sẵn:

* Tranh chấp về nguồn lực. Nhiều tài nguyên chỉ có thể được sử dụng bởi một ứng dụng khách tại một thời điểm. Điều đó có nghĩa là các máy khách khác phải chờ truy cập vào các tài nguyên đó. Hình 3.3. cho thấy các sự kiện đến hệ thống. Các sự kiện này có thể trong một luồng hoặc trong nhiều luồng. Nhiều luồng cạnh tranh cho cùng một tài nguyên hoặc các sự kiện khác nhau trong cùng một luồng cạnh tranh cho cùng một tài nguyên góp phần gây ra độ trễ. Càng nhiều tranh chấp về tài nguyên, thì càng có khả năng xuất hiện độ trễ.
* Tính sẵn có của tài nguyên: ngay cả khi không có tranh chấp việc tính toán không thể tiến hành nếu không có sẵn tài nguyên. Tình trạng không khả dụng có thể do tài nguyên ngoại tuyến hoặc do thành phần bị lỗi hoặc vì một số lý do khác. Trong bất kỳ trường hợp nào, bạn phải xác định những nơi mà tài nguyên không có sẵn có thể gây ra một đóng góp đáng kể vào độ trễ tổng thể. Một số chiến thuật của chúng tôi nhằm đối phó với tình huống này
* Phụ thuộc vào tính toán khác. Một phép tính có thể phải đợi vì nó phải đồng bộ hoá với kết quả của một phép tính khác hoặc vì nó đang đợi kết quả của một phép tính mà nó khởi tạo. Nếu một thành phần gọi một thành phần khác và phải đợi thành phần đó phản hồi, thì thời gian có thể đáng kể nếu thành phần được gọi nằm ở đầu kia của mạng (trái ngược với việc nằm chung trên cùng một bộ xử lý)

Với nền tảng này, chúng ta chuyển sang các danh mục chiến thuật. Điều này giúp giảm nhu cầu về tài nguyên hoặc làm cho tài nguyeen có thể xử lý nhu cầu một cách hiệu quả hơn.

* Kiểm soát nhu cầu tài nguyên. Chiến thuật này hoạt động dựa trên nhu cầu để tạo ra nhu cầu nhỏ hơn về các nguồn lực sẽ phải phục vụ các sự kiện
* Quản lý tài nguyên. Chiến thuật này hoạt động ở khía cạnh phản ứng để làm cho các nguồn lực hiện có hoạt động hiệu quả hơn trong việc xử lý các yêu cầu đặt ra cho họ

***Kiểm soát nhu cầu tài nguyên***

Một cách để tăng hiệu suất là quản lý cẩn thận nhu cầu về tài nguyên. Điều này có thể được thực hiện bằng cách giảm số lượng sự kiện được xử lý bằng cách thực thi tốc độ lấy mẫu hoặc bằng cách giới hạn tốc độ hệ thống phản hồi các sự kiện. Ngoài ra, có một số kỹ thuật để đảm bảo rằng các tài nguyên mà bạn có thể áp dụng một cách cẩn thận:

* Quản lý tỷ lệ lấy mẫu: nếu có thể giảm tần suất lấy mẫu mà tại đó dòng dữ liệu môi trường được thu thập, thì nhu cầu có thể được giảm xuống, thường có một số mất độ tập trung của người tham gia. Lựa chọn thiết kế này được thực hiện để duy trì mức độ trễ có thể dự đoán được; bạn phải quyết định xem dòng dữ liệu có độ trung thực thấp hơn nhưng nhất quán có tốt hơn là mất gói dữ liệu
* Giới hạn phản hồi sự kiện: Khi các sự kiện rời rạc đến hệ thống quá nhanh để được xử lý, thì các sự kiện phải được xếp hàng đợi cho đến khi chúng có thể được xử lý. Trong trường hợp như vậy, có thể chọn chỉ xử lý các sự kiện với tỷ lệ tối đa đã đặt, do đó đảm bảo quá trình xử lý dễ đoán hơn khi các sự kiện thực sự được xử lý. Chiến thuật này có thể được kích hoạt bởi kích thước hàng đợi hoặc biện pháp sử dụng bộ xử lý vượt quá mức cảnh báo nào đó. Nếu bạn áp dụng chiến thuật này và không thể chấp nhận được việc mất bất kỳ sự kiện nào, thì bạn phải đảm bảo rằng hàng đợi của bạn đủ lớn để xử lý trường hợp xấu nhất. Mặt khác, nếu bạn chọn bỏ qua các sự kiện, thì bạn cần phải chọn một chính sách để xử lý tình huống này: bạn có ghi lại các sự kiện bị bỏ hay đơn giản là bỏ qua chúng? Bạn có thông báo cho các hệ thống, người dùng hoặc quản trị viên khác không?
* Ưu tiên các sự kiện: Nếu không phải tất cả các sự kiện đều quan trọng như nhau, bạn có thể áp đặt một sơ đồ ưu tiên xếp hạng các sự kiện theo mức độ quan trọng của việc phục vụ chúng. Nếu khôgn có đủ tài nguyên để phục vụ chúng khi chúng phát sinh, các sự kiện có mức độ ưu tiên thấp có thể bị bỏ qua. Việc bỏ qua các sự kiện tiêu tốn tài nguyên tối thiểu (bao gồm cả thời gian) và do đó làm tăng hiệu suất so với một hệ thống luuoon phục vụ tất cả các sự kiện. Ví dụ một hệ thống quản lý toà nhà có thể đưa ra nhiều loại báo động. Các báo động đe doạ tính mạng như cháy nên được ưu tiên cao hơn các báo động thông tin như phòng quá lạnh.
* Giảm chi phí: việc sử dụng các trung gian làm tăng tài nguyên tiêu thụ trong quá trình xử lý luồng sự kiện và do đó, việc loại bỏ chúng sẽ cải thiện độ trễ. Đây là sự cân bằng khả năng sửa đổi/ hiệu suất cổ điển. Tách các mối quan tâm, một cốt lõi của khả năng sửa đổi, cũng có thể làm tăng chi phí xử lý cần thiết để phục vụ một sự kiện nếu nó dẫn đến sự kiện được phục vụ bởi một chuỗi các thành phần thay vì một thành phần duy nhất. Chi phí chuyển đổi ngữ cảnh và giao tiếp giữa các thành phần cộng lại, đặc biệt là khi các thtafnh phần nằm trên các nút khác nhau trên mạng. Một chiéne lược để giảm chi phí tính toán là đồng định vị các tài nguyên. Điều nyaf có nghĩa là lưu trữ các thành phần trên cùng một bộ xử lý để tránh sự chậm trễ về thời gian của giao tiếp mạng; nó có thể có nghĩa là đặt các tài nguyên trên cùng một thành phần phần mềm thời gian chạy để tránh ngay cả chi phí của một cuộc gọi chương trình con. Một trường hợp đặc biệt để giảm chi phí tính toán là thực hiện dọn dẹp định kỳ các tài nguyên đã trở nên không hiệu quả. Ví dụ: bảng băm và bản đồ bộ nhớ ảo có thể yêu cầu tính toán lại và khởi động lại. Một chiến lược phổ biến khác là thực thi các máy chủ đơn luồng (để đơn giản và tránh gây tranh cãi) và chia nhỏ khối lượng công việc trên chúng.
* Thời gian thực hiện ràng buộc. Đặt giới hạn về lượng thời gian thực thi được sử dụng để phản hồi một sự kiện. Đối với các thuật toán lặp lại, phụ thuộc vào dữ liệu, giới hạn số lần lặp là một phương pháp để giới hạn thời gian thực hiện. Chi phí thường là một phép tính kém chính xác hơn. Nếu bạn áp dụng chiến thuật này, bạn sẽ cần phải đánh giá ảnh hưởng của nó đến độ chính xác và xem liệu kết quả có “đủ tốt” hay không? Chiến thuật quản lý tài nguyên này thường được kết hợp với chiến thuật quản lý tốc độ lấy mẫu.
* Tăng hiệu quả sử dụng tài nguyên. Cải thiện các thuật toán được sử dụng trong các khu vực quan trọng sẽ giảm độ trễ

***Quản lý tài nguyên***

Ngay cả khi không kiểm soát được nhu cầu về tài nguyên, thì việc quản lý các tài nguyên này vẫn có thể xảy ra. Đôi khi một tài nguyên này có thể được trao đổi cho một tài nguyên khác. Ví dụ, dữ liệu trung gian có thể được lưu trong bộ nhớ đệm hoặc có thể được tạo lại tuỳ thuộc vào thời gian và không gian sẵn có của tài nguyên. Chiến thuật này thường được áp dụng cho bộ xử lý những cũng hiệu quả khi áp dụng cho các tài nguyên khác như đĩa CD. Dưới đây là một số chiến thuật quản lý tài nguyên.

* Tăng tài nguyên. Bộ xử lý nhanh hơn, bộ xử lý bổ sung, bộ nhớ bổ sung và mạng nhanh hơn đều có khả năng giảm độ trễ. Chi phí thường được cân nhắc trong việc lựa chọn tài nguyên, nhưng tăng tài nguyên chắc chắn là một chiến thuật để giảm độ trễ và trong nhiều trường hợp là cách rẻ nhất để có được sự cải thiện ngay lập tức
* Giới thiệu đồng thời. Nếu các yêu cầu có thể xử lý song song, thời gian bị chặn có thể giảm xuống. Đồng thời có thể được giới thiệu bằng cách xử lý các luồng sự kiện khác nhau trên các luồng khác nhau hoặc bằng cách tạo ra các luồng bổ xung để xử lý các nhóm hoạt động khác nhau. Khi tính đồng thời đã được giới thiệu, các chính sách lập lịch có thể được sử dụng để đạt được các mục tiêu mà bạn mong muốn. Các chính sách lập lịch khác nhau có thể tối đa hoá tính công bằng (tất cả các yêu cầu đều có thời gian như nhau), thông lượng (thời gian ngắn nhất để hoàn thành trước) hoặc các mục tiêu khác.
* Duy trì nhiều bản sao của tính toán: Nhiều máy chủ trong mô hình máy khách – chủ là bản sao của tính toán Mục đích của các bản sao là để giảm bớt sự tranh cãi sẽ xảy ra nếu tất cả các phép tính diễn ra trên một máy chủ duy nhất. Bộ cân bằng tải là một phần mềm chỉ định công việc mới cho một trong các máy chủ trùng lặp có sẵn; các tiêu chí để gán khác nhau nhưng có thể đơn giản như chuyển đổi vòng tròn hoặc gán yêu cầu tiếp theo cho máy chủ ít bận nhất.
* Duy trì nhiều bản sao dữ liệu. Bộ nhớ đệm là một chiến thuật liên quan đến việc lưu giữ các bản sao dữ liệu (có thể là một tập con của dữ liệu kia) trên bộ nhớ với các tốc độ truy cập khác nhau. Các tốc độ truy cập khác nhau có thể là do có sẵn (bộ nhớ với bộ nhớ thứ cấp) hoặc có thể do nhu cầu giao tiếp mạng. Sao chép dữ liệu liên quan đến việc giữ cá bản sao dữ liệu riêng biệt để giảm bớt sự tranh cấp từ nhiều truy cập đồng thời. Vì dữ liệu được lưu trong bộ nhớ đệm hoặc sao chép thường là bản sao của dữ liệu hiện có, nên việc giữ cho các bản sao nhất quán và đồng bộ trở thành trách nhiệm mà hệ thống phải đảm nhận. Một trách nhiệm khác là chọn dữ liệu được lưu vào bộ nhớ đệm. Một số bộ nhớ đệm hoạt động bằng cách chỉ giữ bản sao của bất kỳ thứ gì được yêu cầu gần đây, nhưng cũng có thể dự đoán các yêu cầu trong tương lai của người dùng dựa trên các mẫu hành vi và bắt đầu tính toán hoặc tìm nạp để tuân thủ các yêu cầu đó trước khi người dùng thực hiện chúng
* Giới hạn kích thước hàng đợi. Điều này kiẻm soát số lượng tối đa các lượt đến được xếp hàng và do đó các nguồn lực được sử dụng để xử lý các lượt đến. Nếu bạn áp dụng chiến thuật này, bạn cần áp dụng chính sách về những gì sẽ xảy ra khi hàng đợi đầy và quyết định xem xét việc không phản hồi của sự kiện bị mất có được tranh chấp hay không. Chiến thuật này thường được kết hợp với chiến thuật phản hồi sự kiện giới hạn.
* Lên lịch cho các nguồn lực. Bất cứ khi nào có sự tranh giành tài nguyên, tài nguyên đó phải được lên lịch. Bộ xử lý được lên lịch, bộ đệm và mạng được lập lịch. Mục tiêu của bạn là hiểu các đặc điểm của việc sử dụng từng tài nguyên và chọn chiến lược lập lịch trình tương thích với tài nguyên đó.

Nguyên tắc lập lịch:

* + First in/first out: Hàng đợi FIFO coi tất cả các yêu cầu về tài nguyên là bằng nhau và lần lượt đáp ứng chúng. Một khả năng xảy ra với hàng đợi FIFO là một yêu cầu sẽ bị mắc kẹt sau một yêu cầu khác, mất nhiều thời gian để tạo phản hồi. Miễn là tất cả các yêu cầu thực sự bình đẳng thì đây không phải là vấn đề, nhưng nếu một số yêu cầu có mức độ ưu tiên cao hơn những yêu cầu khác thì đó là vấn đề.
  + Lập lịch ưu tiên cố định. Lập lịch ưu tiên cố định chỉ định mỗi nguồn yêu cầu tài nguyên một mức ưu tiên cụ thể và chỉ định tài nguyên theo thứ tự ưu tiên đó. Chiến lược này đảm bảo dịch vụ tốt hơn cho các yêu cầu có mức độ ưu tiên cao hơn. Nhưng nó thừa nhận khả năng có mức độ ưu tiên thấp hơn, nhưng quan trọng, yêu cầu mất một thời gian dài tùy ý để được phục vụ, bởi vì nó bị mắc kẹt đằng sau một loạt các yêu cầu có mức độ ưu tiên cao hơn. Ba chiến lược ưu tiên phổ biến là:
    - Tầm quan trọng về ngữ nghĩa. Mỗi luồng được chỉ định một mức độ ưu tiên tĩnh theo một số đặc tính miền của tác vụ tạo ra nó.
    - Đơn điệu thời hạn. Đơn điệu thời hạn-Deadline monotonic là phép gán mức độ ưu tiên tĩnh chỉ định mức độ ưu tiên cao hơn cho các luồng có thời hạn ngắn hơn. Chính sách lập lịch này được sử dụng khi các luồng có mức độ ưu tiên khác nhau với thời hạn thời gian thực phải được lên lịch.
    - Tỷ lệ đơn điệu. Tốc độ đơn điệu là phép gán mức độ ưu tiên tĩnh cho các luồng định kỳ chỉ định mức độ ưu tiên cao hơn cho các luồng có khoảng thời gian ngắn hơn. Chính sách lập lịch trình này là một trường hợp đặc biệt của tính đơn điệu thời hạn nhưng được biết đến nhiều hơn và có nhiều khả năng được hệ điều hành hỗ trợ hơn.
  + Lập lich ưu tiên động: Các chiến lược bao gồm:
    - Thi đấu vòng tròn. Round-robin là một chiến lược lập lịch trình sắp xếp các yêu cầu và sau đó, ở mọi khả năng phân công, chỉ định tài nguyên cho yêu cầu tiếp theo theo thứ tự đó. Một hình thức đặc biệt của vòng lặp là điều hành theo chu kỳ, trong đó các khả năng phân công là vào những khoảng thời gian cố định.
    - Sớm nhất-thời hạn-đầu tiên (Earliest-deadline-first) Thời hạn sớm nhất-trước hết chỉ định các ưu tiên dựa trên các yêu cầu đang chờ xử lý với thời hạn sớm nhất
    - Ít-ra-nhất (Least-slack-first) Chiến lược này chỉ định mức độ ưu tiên cao nhất cho công việc có ít "thời gian chờ" nhất, là sự khác biệt giữa thời gian thực hiện còn lại và thời gian đến hạn chót của công việc



Hình 3.4. Các chiến thuật hiệu suất

**3.4.2 Khả năng sử dụng**

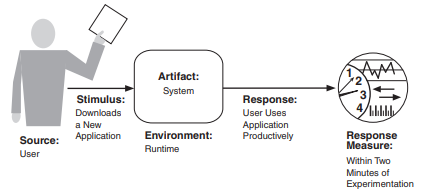
Khả năng sử dụng liên quan đến mức độ dễ dàng của người dùng để hoàn thành một tác vụ mong muốn và loại hỗ trợ người dùng mà hệ thống cung cấp. Trong những năm qua việc tập trung vào khả năng sử dụng đã cho thấy mình có một trong những cách rẻ nhất và dễ dàng nhất để cải thiện chất lượng hệ thống.

***Kịch bản khả năng sử dụng***

Các phần của kịch bản chung về khả năng sử dụng

|  |  |
| --- | --- |
| **Phần kịch bản** | **Những giá trị khả thi** |
| Nguồn kích thích | Người dùng cuối (quản trị viên hệ thống hoặc mạng) |
| Kích thích | Người sử dụng cuối mong muốn sử dụng hệ thống một cách hiệu quả, học cách sử dụng hệ thống, giảm thiểu tác động của lỗi, điều chỉnh hệ thống hoặc cấu hình hệ thống |
| Môi trường | Các hành động của người dùng liên quan đến khả năng sử dụng luôn xảy ra trong thời gian chạy hoặc tại thời điểm cấu hình hệ thống |
| Phần tử | Hệ thống hoặc phần cụ thể của hệ thống mà người dùng đang tương tác |
| Đáp ứng | Hệ thống phải cung cấp cho người dùng các tính năng cần thiết hoặc dự đoán nhu cầu của người dùng |
| Đo lường đáp ứng | Phản hồi được đo bằng thời gian thực hiện, số lỗi, số tác vụ đã hoàn thành, mức độ hài lòng của người dùng, tỷ lệ hoạt động thành công trên tổng số hoạt động hoặc lượng thời gian hoặc dữ liệu bị mất khi xảy ra lỗi |

Kịch bản thuộc tính khả năng sử dụng cụ thể được minh hoạ trong hình 3.5. dưới đây



Hình 3.5 Mẫu kịch bản khả năng sử dụng cụ thể.

***Các chiến thuật cho khả năng sử dụng***

Hãy nhớ lại rằng khả năng sử dụng liên quan đến mức độ dễ dàng của người dùng để hoàn thành một tác vụ mong muốn, cũng như loại hỗ trợ mà hệ thống cung cấp cho người dùng.

*Hỗ trợ người dùng*

Khi hệ thống đang thực thi, khả năng sử dụng được nâng cao bằng cách cung cấp cho người dùng phản hồi về những gì hệ thống đang làm và bằng cách cho phép người dùng đưa ra các phản hồi thích hợp. Ví dụ: các chiến thuật được mô tả tiếp theo — hủy bỏ, hoàn tác, tạm dừng / tiếp tục và tổng hợp — hỗ trợ người dùng sửa lỗi hoặc hiệu quả hơn.

Kiến trúc sư thiết kế một phản hồi cho sự chủ động của người dùng bằng cách liệt kê và phân bổ các trách nhiệm của hệ thống để phản hồi lại lệnh của người dùng. Dưới đây là một số ví dụ phổ biến về sáng kiến của người dùng:

- Hủy bỏ. Khi người dùng đưa ra lệnh hủy, hệ thống phải lắng nghe lệnh đó (do đó, có trách nhiệm phải có một bộ lắng nghe liên tục không bị chặn bởi các hành động của bất kỳ thứ gì đang bị hủy); lệnh bị hủy phải được chấm dứt; mọi tài nguyên đang được sử dụng bởi lệnh bị hủy phải được giải phóng; và các thành phần đang cộng tác với lệnh bị hủy phải được thông báo để họ cũng có thể thực hiện hành động thích hợp.

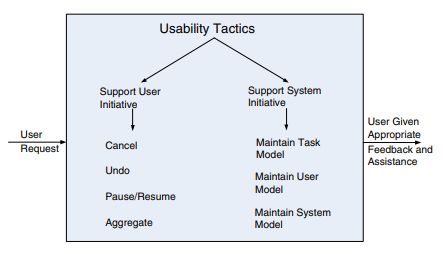
- Hoàn tác. Để hỗ trợ khả năng hoàn tác, hệ thống phải duy trì đủ lượng thông tin về trạng thái hệ thống để có thể khôi phục trạng thái trước đó theo yêu cầu của người dùng. Một bản ghi như vậy có thể ở dạng trạng thái “ảnh chụp nhanh” — ví dụ: các điểm kiểm tra — hoặc dưới dạng một tập hợp các hoạt động có thể đảo ngược. Không phải tất cả các thao tác đều có thể được đảo ngược một cách dễ dàng: ví dụ, việc thay đổi tất cả các lần xuất hiện của chữ “a” thành chữ “b” trong tài liệu không thể được đảo ngược bằng cách thay đổi tất cả các trường hợp của “b” thành “a”, bởi vì một số trường hợp đó của “b” có thể đã tồn tại trước khi thay đổi ban đầu. Trong trường hợp này, hệ thống phải duy trì một hồ sơ phức tạp hơn về sự thay đổi. Tất nhiên, một số thao tác, chẳng hạn như rung chuông, không thể hoàn tác

- Tạm dừng / tiếp tục. Khi người dùng đã bắt đầu một hoạt động lâu dài — chẳng hạn như tải xuống một tệp lớn hoặc một tập hợp tệp từ máy chủ — việc cung cấp khả năng tạm dừng và tiếp tục hoạt động thường hữu ích. Tạm dừng một cách hiệu quả một hoạt động đang chạy lâu dài đòi hỏi khả năng tạm thời giải phóng tài nguyên để chúng có thể được phân bổ lại cho các tác vụ khác.

- Tổng hợp. Khi người dùng đang thực hiện các thao tác lặp lại hoặc các thao tác ảnh hưởng đến một số lượng lớn các đối tượng theo cùng một cách, sẽ rất hữu ích khi cung cấp khả năng tổng hợp các đối tượng cấp thấp hơn vào một nhóm duy nhất, để thao tác có thể được áp dụng cho nhóm, do đó giải phóng người dùng khỏi sự vất vả (và có khả năng mắc lỗi) khi thực hiện lặp đi lặp lại cùng một thao tác. Ví dụ: tổng hợp tất cả các đối tượng trong một trang slide và thay đổi văn bản thành phông chữ 14pt

*Hỗ trợ hệ thống*

* Duy trì mô hình nhiệm vụ. Mô hình nhiệm vụ được sử dụng để xác định ngữ cảnh để hệ thống có thể có một số ý tưởng về những gì người dùng đang cố gắng và cung cấp hỗ trợ. Ví dụ: khi biết rằng các câu bắt đầu bằng chữ in hoa sẽ cho phép ứng dụng sửa một chữ thường ở vị trí đó.
* Duy trì mô hình người dùng. Mô hình này thể hiện rõ ràng kiến thức của người dùng về hệ thống, hành vi của người dùng về thời gian phản hồi dự kiến và các khía cạnh khác dành riêng cho người dùng hoặc một lớp người dùng. Ví dụ, việc duy trì mô hình người dùng cho phép hệ thống tăng tốc độ lựa chọn chuột để không phải tất cả tài liệu đều được chọn khi cần cuộn. Hoặc một mô hình có thể kiểm soát số lượng hỗ trợ và đề xuất được cung cấp tự động cho người dùng. Một trường hợp đặc biệt của chiến thuật này thường thấy trong tùy chỉnh giao diện người dùng, trong đó người dùng có thể sửa đổi rõ ràng mô hình người dùng của hệ thống.
* Duy trì mô hình hệ thống. Ở đây hệ thống duy trì một mô hình rõ ràng của chính nó. Điều này được sử dụng để xác định hành vi hệ thống mong đợi để có thể đưa ra phản hồi thích hợp cho người dùng. Biểu hiện phổ biến của mô hình hệ thống là thanh tiến trình dự đoán thời gian cần thiết để hoàn thành hoạt động hiện tại



Hình 3.6 Chiến thuật khả năng sử dụng

**3.4.3 Khả năng sửa đổi**

Thuộc tính chất lượng có thể sửa đổi là thước đo mức độ dễ dàng thay đổi một ứng dụng để đáp ứng cho các yêu cầu mới về chức năng và phi chức năng. Dự đoán khả năng sửa đổi yêu cầu ước tính nỗ lực và / hoặc chi phí để thực hiện thay đổi. Bạn chỉ biết chắc chắn chi phí của một thay đổi sau khi nó đã được thực hiện. Sau đó, bạn tìm ra ước tính của bạn tốt như thế nào.

Các biện pháp sửa đổi chỉ phù hợp trong bối cảnh của một giải pháp kiến trúc nhất định. Giải pháp này ít nhất phải được thể hiện về mặt cấu trúc như một tập hợp các thành phần, các mối quan hệ giữa các thành phần và mô tả cách các thành phần tương tác với môi trường. Sau đó, việc đánh giá khả năng sửa đổi yêu cầu kiến trúc sư xác nhận các kịch bản thay đổi có thể xảy ra để nắm bắt các yêu cầu có thể phát triển như thế nào. Đôi khi những điều này sẽ được biết đến với một mức độ chắc chắn. Trên thực tế, những thay đổi thậm chí có thể được chỉ định trong kế hoạch dự án cho các bản phát hành tiếp theo. Mặc dù vậy, phần lớn thời gian sẽ cần phải được gợi ý từ các bên liên quan đến ứng dụng và rút ra từ kinh nghiệm của kiến trúc sư.

Các tình huống thay đổi:

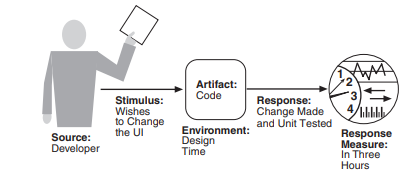
* Cung cấp quyền truy cập vào ứng dụng thông qua tường lửa ngoài quyền truy cập “đằng sau tường lửa” hiện có.
* Kết hợp các tính năng mới cho các ki-ốt trả phòng tự phục vụ.
* Nhà cung cấp phần mềm nhận dạng giọng nói COTS ngừng hoạt động và chúng tôi cần thay thế thành phần này.
* Ứng dụng cần được chuyển từ Linux sang nền tảng Microsoft Windows

Đối với mỗi kịch bản thay đổi, có thể đánh giá tác động của thay đổi dự kiến đối với kiến trúc. Tác động này hiếm khi dễ định lượng, vì thường không tồn tại giải pháp đang được đánh giá. Trong nhiều trường hợp, điều tốt nhất có thể đạt được là phân tích tác động thuyết phục của các thành phần trong kiến trúc sẽ cần sửa đổi hoặc minh họa cách giải pháp có thể thích ứng với việc sửa đổi mà không cần thay đổi.

Cuối cùng, dựa trên ước tính chi phí, quy mô hoặc nỗ lực cho các thành phần bị ảnh hưởng, một số định lượng hữu ích về chi phí thay đổi có thể được thực hiện. Các thay đổi được tách biệt đối với các thành phần đơn lẻ hoặc các hệ thống con được kết hợp lỏng lẻo có thể ít tốn kém hơn so với những thay đổi gây ra hiệu ứng gợn sóng trên toàn bộ kiến trúc. Nếu một thay đổi có khả năng xuất hiện khó khăn và phức tạp điều này có thể làm nổi bật một điểm yếu trong kiến trúc có thể biện minh cho việc xem xét thêm và thiết kế lại.

Kịch bản chung cho thuộc tính khả năng sửa đổi

|  |  |
| --- | --- |
| **Phần của kịch bản** | **Những giá trị khả thi** |
| Nguồn | Người dùng cuối, người phát triển, quản trị hệ thống |
| Kích thích | Chỉ thị để thêm / xóa / sửa đổi chức năng hoặc thay đổi thuộc tính chất lượng, công suất hoặc công nghệ |
| Phần tử | Mã, dữ liệu, giao diện, thành phần, tài nguyên, cấu hình, |
| Môi trường | Thời gian chạy, thời gian biên dịch, thời gian xây dựng, thời gian bắt đầu, thời gian thiết kế |
| Đáp ứng | Một hoặc những đáp ứng dưới đây:   * Thực hiện sửa đổi * Kiểm tra sửa đổi * Triển khai sửa đổi |
| Đo lường đáp ứng | Chi phí về những điều sau đây:   * Số lượng, kích thước, độ phức tạp của các hiện vật bị ảnh hưởng * Nỗ lực * Thời gian theo lịch * Chi phí * Mức độ ảnh hưởng của sửa đổi này đến các chức năng hoặc thuộc tính chất lượng khác * Các khiếm khuyết mới được đưa ra |



Hình 3.7. Kịch bản thuộc tính khả năng sửa đổi riêng

**Chiến thuật khả năng sửa đổi**

Các chiến thuật để kiểm soát khả năng sửa đổi có mục tiêu là kiểm soát mức độ phức tạp của việc thực hiện thay đổi, cũng như thời gian và chi phí để thực hiện thay đổi. Hình 3.10 cho thấy mối quan hệ này



Hình 3.8. Mục tiêu của chiến thuật cho khả năng sửa đổi



Hình 3.9 Các chiến thuật khả năng sửa đổi

***Giảm kích thước của một mô-đun***

* Phân chia mô-đun. Nếu mô-đun đang được sửa đổi bao gồm rất nhiều khả năng, chi phí sửa đổi có thể sẽ cao. Việc tinh chỉnh mô-đun thành một số mô-đun nhỏ hơn sẽ giảm chi phí trung bình cho những thay đổi trong tương lai.

***Tăng sự gắn kết***

Một số chiến thuật liên quan đến việc chuyển trách nhiệm từ mô-đun này sang mô-đun khác. Mục đích của việc chuyển một trách nhiệm từ mô-đun này sang mô-đun khác là để giảm khả năng xảy ra các tác dụng phụ ảnh hưởng đến các trách nhiệm khác trong mô-đun ban đầu.

* Tăng tính mạch lạc về ngữ nghĩa. Nếu các trách nhiệm A và B trong một mô-đun không phục vụ cùng một mục đích, chúng nên được đặt trong các mô-đun khác nhau. Điều này có thể liên quan đến việc tạo một mô-đun mới hoặc có thể liên quan đến việc chuyển một chức năng sang một mô-đun hiện có. Một phương pháp để xác định các trách nhiệm sẽ được di chuyển là đưa ra giả thuyết về những thay đổi có khả năng ảnh hưởng đến một mô-đun. Nếu một số trách nhiệm không bị ảnh hưởng bởi những thay đổi này, thì những trách nhiệm đó có lẽ nên được loại bỏ.

***Giảm khớp nối***

Bây giờ chúng ta chuyển sang chiến thuật làm giảm sự ghép nối giữa các mô-đun.

* Đóng gói. Đóng gói giới thiệu một giao diện rõ ràng cho một mô-đun. Giao diện này bao gồm giao diện lập trình ứng dụng (API) và các trách nhiệm liên quan của nó, chẳng hạn như “thực hiện chuyển đổi cú pháp trên một tham số đầu vào thành một đại diện bên trong”. Có lẽ chiến thuật khả năng sửa đổi phổ biến nhất, tính đóng gói làm giảm xác suất mà một thay đổi đối với một mô-đun truyền sang các mô-đun khác. Điểm mạnh của việc ghép nối trước đây dành cho mô-đun giờ chuyển sang giao diện cho mô-đun. Tuy nhiên, những điểm mạnh này bị giảm đi vì giao diện giới hạn các cách mà các trách nhiệm bên ngoài có thể tương tác với mô-đun (có thể thông qua một trình bao bọc). Các trách nhiệm bên ngoài giờ đây chỉ có thể tương tác trực tiếp với mô-đun thông qua giao diện tiếp xúc (tuy nhiên, các tương tác gián tiếp, chẳng hạn như sự phụ thuộc vào chất lượng dịch vụ, có thể sẽ không thay đổi). Các giao diện được thiết kế để tăng khả năng sửa đổi phải trừu tượng đối với các chi tiết của mô-đun có khả năng thay đổi — nghĩa là chúng phải ẩn các chi tiết đó.
* Sử dụng trung gian phá vỡ sự phụ thuộc. Do có sự phụ thuộc giữa trách nhiệm A và trách nhiệm B (ví dụ: thực hiện A trước tiên yêu cầu thực hiện B), sự phụ thuộc có thể bị phá vỡ bằng cách sử dụng một bên trung gian. Loại trung gian phụ thuộc vào loại phụ thuộc. Ví dụ: trung gian đăng ký xuất bản sẽ xóa kiến thức của nhà sản xuất dữ liệu về người tiêu dùng của họ. Một kho dữ liệu dùng chung cũng vậy, nó phân tách người đọc một phần dữ liệu với người viết dữ liệu đó. Trong kiến trúc hướng dịch vụ, trong đó các dịch vụ phát hiện ra nhau bằng cách tra cứu động, dịch vụ thư mục là một trung gian
* Hạn chế sự phụ thuộc là một chiến thuật hạn chế các mô-đun mà một mô-đun nhất định tương tác với hoặc phụ thuộc vào. Trên thực tế, chiến thuật này đạt được bằng cách hạn chế khả năng hiển thị của mô-đun (khi nhà phát triển không thể nhìn thấy giao diện, họ không thể sử dụng nó) và bằng cách ủy quyền (chỉ giới hạn quyền truy cập vào các mô-đun được ủy quyền). Chiến thuật này được áp dụng trong các kiến trúc phân lớp, trong đó một lớp chỉ được phép sử dụng các lớp thấp hơn (đôi khi chỉ lớp dưới tiếp theo) và trong việc sử dụng các trình bao bọc, nơi các thực thể bên ngoài chỉ có thể nhìn thấy (và do đó phụ thuộc vào) trình bao bọc chứ không chức năng nội bộ mà nó bao bọc
* Refactor là một chiến thuật được thực hiện khi hai mô-đun bị ảnh hưởng bởi cùng một thay đổi vì chúng là bản sao (ít nhất là một phần) của nhau. Tái cấu trúc mã là một thực hành chính của các dự án phát triển Agile, như một bước dọn dẹp để đảm bảo rằng các nhóm không tạo ra mã trùng lặp hoặc quá phức tạp; tuy nhiên, khái niệm này cũng áp dụng cho các yếu tố kiến trúc. Các trách nhiệm chung (và mã thực hiện chúng) được “tính theo yếu tố” của các mô-đun nơi chúng tồn tại và được chỉ định một ngôi nhà thích hợp của riêng chúng. Bằng cách đồng định vị các trách nhiệm chung — nghĩa là, làm cho chúng trở thành các mô-đun con của cùng một mô-đun mẹ — kiến trúc sư có thể giảm việc ghép nối.
* Các dịch vụ thông thường trừu tượng. Trong trường hợp hai mô-đun cung cấp các dịch vụ không giống nhau nhưng tương tự nhau, thì việc triển khai các dịch vụ chỉ một lần ở dạng tổng quát hơn (trừu tượng) có thể hiệu quả về chi phí. Sau đó, bất kỳ sửa đổi nào đối với dịch vụ (thông thường) sẽ chỉ cần thực hiện ở một nơi, giảm chi phí sửa đổi. Một cách phổ biến để giới thiệu một phần trừu tượng là tham số hóa mô tả (và triển khai) các hoạt động của mô-đun. Các tham số có thể đơn giản như các giá trị cho các biến chính hoặc phức tạp như các câu lệnh trong một ngôn ngữ chuyên dụng được diễn giải sau đó

***Trì hoãn ràng buộc***

Bởi vì công việc của con người hầu như luôn tốn kém hơn công việc của máy tính, việc để máy tính xử lý một thay đổi càng nhiều càng tốt hầu như sẽ luôn làm giảm chi phí thực hiện thay đổi đó. Nếu chúng tôi thiết kế các tạo tác với tính linh hoạt được tích hợp sẵn, thì việc thực hiện tính linh hoạt đó thường rẻ hơn so với viết tay một thay đổi cụ thể.

**3.4.4. Tính bảo mật**

Bảo mật là một chủ đề kỹ thuật phức tạp nhưng trong phần này chỉ nói được sơ qua. Ở cấp độ kiến trúc, bảo mật tập trung vào việc hiểu các yêu cầu bảo mật chính xác cho một ứng dụng và đưa ra các cơ chế để hỗ trợ chúng. Các yêu cầu phổ biến nhất liên quan đến bảo mật là:

* Xác thực: Các ứng dụng có thể xác minh danh tính của người dùng và các ứng dụng khác mà chúng giao tiếp.
* Phân quyền: Người dùng và ứng dụng được xác thực có quyền truy cập xác định vào các tài nguyên của hệ thống. Ví dụ: một số người dùng có thể có quyền truy cập chỉ đọc vào dữ liệu của ứng dụng, trong khi những người khác có quyền đọc ghi.
* Mã hóa: Các tin nhắn được gửi đến / từ ứng dụng được mã hóa.
* Tính toàn vẹn: Điều này đảm bảo nội dung của một tin nhắn không bị thay đổi trong quá trình truyền tải.
* Không từ chối: Người gửi tin nhắn có bằng chứng về việc gửi và người nhận được đảm bảo về danh tính của người gửi. Điều này có nghĩa là sau đó không ai có thể từ chối sự tham gia của họ vào việc trao đổi tin nhắn.

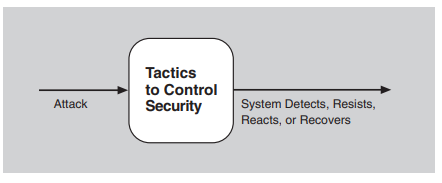
Có những công nghệ nổi tiếng và được sử dụng rộng rãi hỗ trợ các yếu tố bảo mật ứng dụng này. Lớp cổng bảo mật (SSL) và cơ sở hạ tầng khóa công khai (PKI) thường được sử dụng trong các ứng dụng Internet để cung cấp xác thực, mã hóa và không từ chối. Xác thực và ủy quyền được hỗ trợ trong các công nghệ Java bằng cách sử dụng Dịch vụ xác thực và ủy quyền Java (JAAS). Hệ điều hành và cơ sở dữ liệu cung cấp bảo mật dựa trên đăng nhập để xác thực và ủy quyền

Kịch bản chung cho thuộc tính bảo mật

|  |  |
| --- | --- |
| **Phần kịch bản** | **Các giá trị khả thi** |
| Nguồn | Con người hoặc một hệ thống có thể được xác định trước đây (chính xác hoặc không chính xác) hoặc hiện tại có thể chưa được xác định. Kẻ tấn công con người có thể từ bên ngoài tổ chức hoặc từ bên trong tổ chức |
| Kích thích | Cố gắng trái phép được thực hiện để hiển thị dữ liệu, thay đổi hoặc xoá dữ liệu, truy cập các dịch vụ hệ thống, thay đổi hoạt động của hệ thống hoặc giảm tính khả dụng |
| Phần tủ | Dịch vụ hệ thống, dữ liệu trong hệ thống, một thành phần hoặc tài nguyên của hệ thống, dữ liệu được tạo ra hoặc sử dụng bởi hệ thống |
| Môi trường | Hệ thống trực tuyến hoặc ngoại tuyến; được kết nối với hoặc ngắt kết nối khỏi mạng; hoặc đằng sau một bức tường lửa hoặc mở vào một mạng; hoạt động hoàn toàn, hoạt động một phần, hoặc không hoạt động. |
| Đáp ứng | Các giao dịch được thực hiện theo cách thức sao cho:   * Dữ liệu hoặc dịch vụ được bảo vệ khỏi truy cập trái phép. * Dữ liệu hoặc dịch vụ không bị thao túng nếu không được phép. * Các bên tham gia giao dịch được xác định với sự đảm bảo. * Các bên tham gia giao dịch không thể thoái thác sự tham gia của họ. * Dữ liệu, tài nguyên và các dịch vụ hệ thống sẽ có sẵn để sử dụng hợp pháp.   Hệ thống theo dõi các hoạt động bên trong nó bằng cách:   * Ghi lại quyền truy cập hoặc sửa đổi * Ghi lại các nỗ lực truy cập dữ liệu, tài nguyên hoặc dịch vụ * Thông báo cho các thực thể thích hợp (người hoặc hệ thống) khi một cuộc tấn công rõ ràng đang xảy ra |
| Đo lường đáp ứng | Một hoặc nhiều điều sau đây:   * Bao nhiêu hệ thống bị xâm phạm khi một thành phần hoặc giá trị dữ liệu cụ thể bị xâm phạm * Thời gian trôi qua trước khi phát hiện một cuộc tấn công * Có bao nhiêu cuộc tấn công đã được chống lại * Mất bao lâu để phục hồi sau một cuộc tấn công thành công * Lượng dữ liệu là bao nhiêu dễ bị tấn công cụ thể |

**Các chiến thuật cho thuộc tính bảo mật**

Một phương pháp để suy nghĩ về cách đạt được bảo mật trong một hệ thống là suy nghĩ về bảo mật vật lý. Các cài đặt an toàn có quyền truy cập hạn chế (ví dụ: bằng cách sử dụng các điểm kiểm tra an ninh), có phương tiện phát hiện những kẻ xâm nhập (ví dụ: bằng cách yêu cầu khách truy cập hợp pháp đeo huy hiệu), có các cơ chế ngăn chặn như vũ trang bảo vệ, có các cơ chế phản ứng như khóa cửa tự động và có các cơ chế phục hồi như sao lưu ngoài địa điểm. Những điều này dẫn đến bốn loại chiến thuật của chúng tôi: phát hiện, chống lại, phản ứng và phục hồi. Hình 3.7 cho thấy các danh mục này là mục tiêu của các chiến thuật bảo mật.



Hình 3.10 Mục tiêu của chiến thuật bảo mật

Các chiến thuật bảo mật được chia thành 3 loại sau:

* Phát hiện cuộc tấn công:
  + Phát hiện xâm nhập
  + Phát hiện từ chối dịch vụ là sự so sánh mẫu hoặc chữ ký của lưu lượng mạng đi vào hệ thống với các cấu hình lịch sử của các cuộc tấn công từ chối dịch vụ đã biết
  + Xác minh tính toàn vẹn của tin nhắn: chiến thuật này sử dụng các kỹ thuật như tổng kiểm tra hoặc giá trị băm để xác định tính toàn vẹn của thông báo, tệp tài nguyên, tệp triển khai và tệp cấu hình
  + Phát hiện độ trễ tín nhắn nhằm phát hiện các cuộc tấn công trung gian tiềm ẩn, trong đó một bên độc hại đang chặn (và có thể sửa đổi) tin nhắn. Bằng cách kiểm tra thời gian gửi tin nhắn, có thể phát hiện hành vi thực hiện thời gian đáng ngờ, trong đó thời gian gửi tin nhắn rất thay đổi.
* Chống lại các cuộc tấn công

Có một số phương tiện nổi tiếng để chống lại cuộc tấn công:

* + Xác định các tác nhân. Xác định “tác nhân” thực sự là xác định nguồn của bất kỳ đầu vào bên ngoài nào vào hệ thống. Người dùng thường được xác định thông qua ID người dùng. Các hệ thống khác có thể được “nhận dạng” thông qua mã truy cập, địa chỉ IP, giao thức, cổng, v.v.
  + Xác thực tác nhân. Xác thực có nghĩa là đảm bảo rằng một tác nhân (người dùng hoặc một máy tính từ xa) thực sự là ai hoặc mục đích của nó là gì. Mật khẩu, mật khẩu dùng một lần, chứng chỉ kỹ thuật số và nhận dạng sinh trắc học cung cấp phương tiện xác thực
  + Ủy quyền cho các tác nhân. Ủy quyền có nghĩa là đảm bảo rằng một tác nhân được xác thực có quyền truy cập và sửa đổi dữ liệu hoặc dịch vụ. Cơ chế này thường được kích hoạt bằng cách cung cấp một số cơ chế kiểm soát truy cập trong hệ thống. Kiểm soát truy cập có thể được thực hiện bởi một tác nhân hoặc bởi một lớp tác nhân. Các loại tác nhân có thể được xác định theo nhóm tác nhân, theo vai trò của diễn viên hoặc theo danh sách các cá nhân.
  + Giới hạn truy cập. Giới hạn quyền truy cập liên quan đến việc kiểm soát những gì và ai có thể truy cập vào phần nào của hệ thống. Điều này có thể bao gồm việc hạn chế quyền truy cập vào các tài nguyên như bộ xử lý, bộ nhớ và kết nối mạng, có thể đạt được bằng cách sử dụng quản lý quy trình, bảo vệ bộ nhớ, chặn máy chủ lưu trữ, đóng cổng hoặc từ chối giao thức. Ví dụ: tường lửa là một điểm truy cập duy nhất vào mạng nội bộ của tổ chức. Khu phi quân sự (DMZ) là một mạng con giữa Internet và mạng nội bộ, được bảo vệ bởi hai bức tường lửa: một đối diện với Internet và một là mạng nội bộ. DMZ được sử dụng khi một tổ chức muốn cho phép người dùng bên ngoài truy cập vào các dịch vụ phải được cung cấp công khai bên ngoài mạng nội bộ. Bằng cách này có thể giảm thiểu số lượng cổng đang mở trong tường lửa nội bộ. Chiến thuật này cũng giới hạn quyền truy cập cho các tác nhân (bằng cách xác định, xác thực và ủy quyền cho họ)
  + Hạn chế tiếp xúc. Hạn chế tiếp xúc đề cập đến cuối cùng và gián tiếp giảm xác suất tấn công thành công hoặc hạn chế lượng sát thương tiềm ẩn. Điều này có thể đạt được bằng cách che giấu sự thật về một hệ thống cần được bảo vệ (“bảo mật bằng cách che giấu”) hoặc bằng cách phân chia và phân phối các tài nguyên quan trọng để việc khai thác một điểm yếu không thể làm tổn hại hoàn toàn bất kỳ tài nguyên nào (“đừng đặt tất cả trứng của bạn vào một giỏ ”). Ví dụ, một quyết định thiết kế để ẩn hệ thống có bao nhiêu điểm vào là một cách để hạn chế sự tiếp xúc. Quyết định phân phối máy chủ giữa một số trung tâm dữ liệu phân tán về mặt địa lý cũng là một cách để hạn chế sự tiếp xúc.
  + Mã hóa dữ liệu. Dữ liệu cần được bảo vệ khỏi sự truy cập trái phép. Tính bảo mật thường đạt được bằng cách áp dụng một số hình thức mã hóa cho dữ liệu và giao tiếp. Mã hóa cung cấp khả năng bảo vệ bổ sung để dữ liệu được duy trì liên tục ngoài dữ liệu có sẵn từ cấp phép. Mặt khác, các liên kết giao tiếp có thể không có các kiểm soát ủy quyền. Trong những trường hợp như vậy, mã hóa là biện pháp bảo vệ duy nhất để truyền dữ liệu qua các liên kết giao tiếp có thể truy cập công khai. Liên kết có thể được thực hiện bởi mạng riêng ảo (Virtual private network VPN) hoặc bởi Lớp cổng bảo mật (Secure Sockets Layer - SSL) để Mã hóa dữ liệu. Dữ liệu cần được bảo vệ khỏi sự truy cập trái phép. Tính bảo mật thường đạt được bằng cách áp dụng một số hình thức mã hóa cho dữ liệu và giao tiếp. Mã hóa cung cấp khả năng bảo vệ bổ sung để dữ liệu được duy trì liên tục ngoài dữ liệu có sẵn từ cấp phép. Mặt khác, các liên kết giao tiếp có thể không có các kiểm soát ủy quyền. Trong những trường hợp như vậy, mã hóa là biện pháp bảo vệ duy nhất để truyền dữ liệu qua các liên kết giao tiếp có thể truy cập công khai. Liên kết có thể được triển khai bởi mạng riêng ảo (VPN) hoặc bởi Lớp cổng bảo mật (SSL) cho một liên kết dựa trên web. Mã hoá có thể đối xứng (cả hai bên sử dụng cùng một khoá) hoặc không đối xứng (kháo công khai và khoá riêng)
  + Các thực thể riêng biệt. Tách các thực thể khác nhau trong hệ thống có thể được thực hiện thông qua việc phân tách vật lý trên các máy chủ khác nhau được gắn với các mạng khác nhau; việc sử dụng máy ảo hoặc "khe hở không khí – air gap", nghĩa là không có kết nối giữa các phần khác nhau của hệ thống. Cuối cùng, dữ liệu nhạy cảm thường được tách biệt khỏi dữ liệu nhạy cảm để giảm khả năng tấn công từ những người có quyền truy cập vào dữ liệu nhạy cảm
  + Thay đổi cài đặt mặc định. Nhiều hệ thống có cài đặt mặc định được chỉ định khi hệ thống được phân phối. Việc buộc người dùng thay đổi các cài đặt đó sẽ ngăn những kẻ tấn công truy cập vào hệ thống thông qua các cài đặt thường được công bố công khai
* Phản ứng với các cuộc tấn công

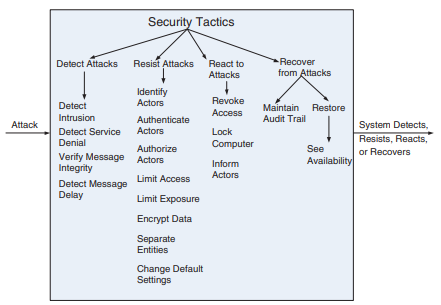
Một số chiến thuật nhằm đối phó với một cuộc tấn công tiềm tàng

* + Thu hồi truy cập. Nếu hệ thống hoặc quản trị viên hệ thống tin rằng một cuộc tấn công đang diễn ra, thì quyền truy cập có thể bị giới hạn nghiêm trọng đối với các tài nguyên nhạy cảm, ngay cả đối với những người dùng và sử dụng hợp pháp thông thường. Ví dụ: nếu màn hình của bạn đã bị vi-rút xâm nhập, quyền truy cập của bạn vào một số tài nguyên nhất định có thể bị hạn chế cho đến khi vi-rút được loại bỏ khỏi hệ thống của bạn.
  + Khóa máy tính. Các lần đăng nhập không thành công lặp đi lặp lại có thể cho thấy một cuộc tấn công tiềm ẩn. Nhiều hệ thống giới hạn quyền truy cập từ một máy tính cụ thể nếu có nhiều lần truy cập không thành công vào tài khoản từ máy tính đó. Người dùng hợp pháp có thể mắc lỗi khi cố gắng đăng nhập. Do đó, quyền truy cập hạn chế có thể chỉ trong một khoảng thời gian nhất định.
  + Thông báo cho các tác nhân. Các cuộc tấn công đang diễn ra có thể yêu cầu hành động của người điều hành, nhân viên khác hoặc hệ thống hợp tác. Nhân viên hoặc hệ thống như vậy — tập hợp các tác nhân có liên quan — phải được thông báo khi hệ thống phát hiện thấy một cuộc tấn công
* Phục hồi sau các cuộc tấn công

Khi một hệ thống đã phát hiện và cố gắng chống lại một cuộc tấn công, nó cần phải khôi phục. Một phần của phục hồi là khôi phục các dịch vụ. Ví dụ, các máy chủ bổ sung hoặc kết nối mạng có thể được giữ lại cho mục đích như vậy. Vì một cuộc tấn công thành công có thể được coi là một loại thất bại, bộ chiến thuật sẵn sàng (từ Chương 5) để giải quyết việc khôi phục sau một thất bại cũng có thể được áp dụng cho khía cạnh bảo mật này.

Ngoài các chiến thuật tính khả dụng cho phép khôi phục dịch vụ, chúng tôi cần duy trì dấu vết kiểm tra. Chúng tôi kiểm tra — nghĩa là, lưu giữ hồ sơ về các hành động của người dùng và hệ thống cũng như ảnh hưởng của chúng — để giúp theo dõi các hành động của và xác định kẻ tấn công. Chúng tôi có thể phân tích các dấu vết kiểm tra để cố gắng truy tố những kẻ tấn công hoặc để tạo ra các biện pháp phòng thủ tốt hơn trong tương lai.

Dưới đây là một tập hợp các chiến thuật bảo mật được đưa ra trong hình 3.8



Hình 3.11 Các chiến thuật bảo mật

**3.4.5. Tính khả dụng**

Tính khả dụng liên quan đến độ tin cậy của ứng dụng. Nếu một ứng dụng không có sẵn để sử dụng khi cần thiết thì ứng dụng đó không có khả năng đáp ứng các yêu cầu chức năng của nó. Tính khả dụng tương đối dễ xác định và đo lương. Về đặc điểm kỹ thuật, nhiều ứng dụng CNTT phải có sẵn ít nhất trong giờ làm việc bình thường. Hầu hết các trang internet đều mong muốn có 100% tính khả dụng, vì không có giờ làm việc thường xuyên trực tuyến. Đối với một hệ thống trực tiếp, tính khả dụng có thể được đo bằng tỷ lệ thời gian cần thiết mà nó có thể sử dụng được.

Lỗi trong các ứng dụng khiến chúng không khả dụng. Lỗi ảnh hưởng đến độ tin cậy của ứng dụng, thường được đo bằng thời gian trung bình giữa các lần gặp lỗi. Khoảng thời gian kéo dài bất kỳ khoảng thời gian không khả dụng nào được xác định bằng khoảng thời gian cần thiết để phát hiện lỗi và khởi động lại hệ thống. Do đó, các ứng dụng yêu cầu tính sẵn dùng cao sẽ giảm thiểu hoặc tốt nhất là loại bỏ các điểm lỗi đơn lẻ và thiết lập các cơ chế tự động phát hiện lỗi và khởi động lại các thành phần bị lỗi.

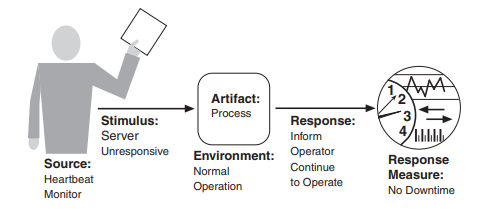
Nhân rộng các thành phần là một chiến lược đã được thử nghiệm và thử nghiệm để có tính khả dụng cao. Khi một thành phần được sao chép không thành công, ứng dụng có thể tiếp tục thực thi bằng cách sử dụng các bản sao vẫn đang hoạt động. Điều này có thể dẫn đến hiệu suất bị giảm sút trong khi thành phần bị lỗi bị hỏng, nhưng tính khả dụng không bị ảnh hưởng

Khả năng phục hồi liên quan chặt chẽ đến tính khả dụng. Ứng dụng có thể khôi phục được nếu ứng dụng có khả năng thiết lập lại mức hiệu suất cần thiết và khôi phục dữ liệu bị ảnh hưởng sau khi ứng dụng hoặc hệ thống bị lỗi. Hệ thống cơ sở dữ liệu là ví dụ cổ điển về hệ thống có thể phục hồi. Khi một máy chủ cơ sở dữ liệu bị lỗi, nó sẽ không khả dụng cho đến khi nó được khôi phục. Điều này có nghĩa là khởi động lại ứng dụng máy chủ và giải quyết mọi giao dịch đang diễn ra khi sự cố xảy ra. Các vấn đề thú vị đối với các ứng dụng có thể khôi phục là cách phát hiện lỗi và bắt đầu khôi phục (tốt nhất là tự động) và thời gian khôi phục trước khi thiết lập lại dịch vụ đầy đủ. Trong quá trình khôi phục, ứng dụng không khả dụng và do đó thời gian khôi phục trung bình là một số liệu quan trọng cần xem xét.

**Kịch bản chung của tính khả dụng**

|  |  |
| --- | --- |
| **Phần kịch bản** | **Những giá trị khả thi** |
| Nguồn | Bên trong/bên ngoài: con người, phần cứng, phần mềm, hạ tầng vật lý, môi trường vật lý |
| Kích thích | Lỗi: thiếu sót, sự cố, thời gian không chính xác, phản hồi không chính xác |
| Phần tử kiến trúc | Bộ xử lý, kênh giao tiếp, lưu trữ liên tục, quy trình |
| Môi trường | Hoạt động bình thường, khởi động, tắt máy, chế độ sửa chữa, hoạt động xuống cấp, hoạt động quá tải |
| Đáp ứng | Ngăn lỗi trở thành lỗi  Phát hiện lỗi:   * Ghi lại lỗi * Thông báo cho các thực thể thích hợp (con người hoặc hệ thống)   Khôi phục sau lỗi:   * Tắt nguồn sự kiện gây ra lỗi * Tạm thời không sử dụng được khi đang sửa chữa |
| Đo lường đáp ứng | Khoảng thời gian hoặc khoảng thời gian khi hệ thống phải khả dụng. Phần trăm khả dụng (ví dụ 99,999%)  Đã đến lúc phát hiện lỗi  Đã đến lúc sửa chữa lỗi  Thời gian hoặc khoảng thời gian mà hệ thống có thể ở chế độ xuống cấp  Tỷ lệ (ví dụ: 99%) hoặc tỷ lệ (ví dụ: lên đến 100 mỗi giây) của một số loại lỗi mà hệ thống ngăn chặn hoặc xử lý mà không bị lỗi |

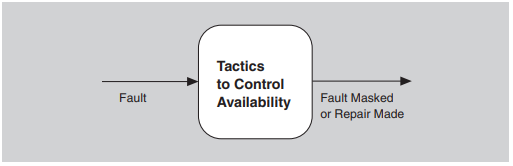
Hình dưới đây chỉ ra kịch bản riêng của tính sẵn dùng



Hình 3.12 Kịch bản riêng của tính sẵn dùng

**Chiến thuật cho tính sẵn dùng**

Lỗi xảy ra khi hệ thống không còn cung cấp dịch vụ phù hợp với đặc điểm kỹ thuật của nó nữa; các tác nhân của hệ thống có thể quan sát được lỗi này. Một lỗi (hoặc sự kết hợp của các lỗi) có khả năng gây ra hỏng hóc. Do đó, các chiến thuật sẵn sàng được thiết kế để cho phép một hệ thống chịu đựng các lỗi hệ thống để một dịch vụ do hệ thống cung cấp vẫn tuân thủ đặc điểm kỹ thuật của nó. Các chiến thuật mà chúng tôi thảo luận trong phần này sẽ giữ cho các lỗi không trở thành lỗi hoặc ít nhất là ràng buộc các tác động của lỗi và giúp cho việc sửa chữa có thể xảy ra. Hình 3.6 minh họa cách tiếp cận này trong

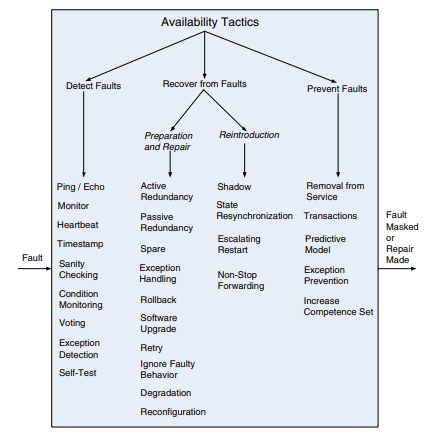


Hình 3.13. Mục tiêu của chiến thuật tính sẵn dùng

Các chiến thuật cho tính sẵn dùng được chia thành 3 loại:

* Phát hiện lỗi
* Khôi phục lỗi
* Ngăn cản lỗi

Về cơ bản, các chiến thuật phát hiện phụ thuộc vào việc phát hiện các dấu hiệu của sự sống từ các thành phần khác nhau. Các chiến thuật khôi phục là một số kết hợp của việc thử lại một hoạt động hoặc duy trì dữ liệu hoặc tính toán dư thừa. Các chiến thuật phòng ngừa phụ thuộc vào việc loại bỏ các yếu tố khỏi dịch vụ hoặc sử dụng các cơ chế để hạn chế phạm vi lỗi



Hình 3.14. Các chiến thuật tính sẵn dùng (khả dụng)

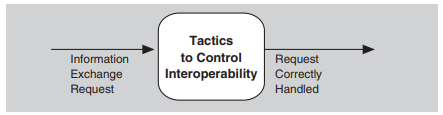
**3.4.3. Khả năng tương tác**

Khả năng tương tác liên quan đến sự dễ dàng mà một ứng dụng có thể được kết hợp một cách hữu ích vào một ngữ cảnh ứng dụng rộng lớn hơn. Giá trị của một ứng dụng hoặc thành phần thường có thể tăng lên rất nhiều nếu chức năng hoặc dữ liệu của nó có thể được sử dụng theo những cách mà người thiết kế không lường trước được. Các chiến lược phổ biến nhất để cung cấp tích hợp là thông qua tích hợp dữ liệu hoặc cung cấp giao diện lập trình ứng dụng (API)

Kịch bản chung của khả năng tương tác

|  |  |
| --- | --- |
| **Phần kịch bản** | **Những giá trị khả thi** |
| Nguồn | Một hệ thống khởi tạo một yêu cầu để tương tác với một hệ thống khác. |
| Kích thích | Yêu cầu trao đổi thông tin giữa (các) hệ thống. |
| Môi trường | Các hệ thống muốn tương tác với nhau. |
| Phần tử | (Các) hệ thống muốn tương tác được phát hiện trong thời gian chạy hoặc được biết trước thời gian chạy. |
| Đáp ứng | Một hoặc nhiều điều sau đây:   * Yêu cầu bị từ chối (một cách thích hợp) và các thực thể thích hợp (con người hoặc hệ thống) được thông báo. * Yêu cầu được chấp nhận (một cách thích hợp) và thông tin được trao đổi thành công. * Yêu cầu được ghi lại bởi một hoặc nhiều hệ thống liên quan. |
| Đo lường đáp ứng | Một hoặc nhiều điều sau đây:   * Tỷ lệ trao đổi thông tin được xử lý chính xác * Tỷ lệ trao đổi thông tin bị từ chối chính xác |

Chiến thuật cho thuộc tính khả năng tích hợp



Hình 3.15 Mục tiêu của chiến thuật khả năng tích hợp

***Định vị***

Chỉ có một chiến thuật trong danh mục này: khám phá dịch vụ. Nó được sử dụng khi các hệ thống tương tác phải được phát hiện trong thời gian chạy.

- Khám phá dịch vụ. Xác định vị trí của một dịch vụ thông qua việc tìm kiếm một dịch vụ thư mục đã biết. ("Dịch vụ", chúng tôi chỉ đơn giản có nghĩa là một tập hợp các khả năng có thể truy cập thông qua một số loại giao diện.) Có thể có nhiều cấp chuyển hướng trong quá trình định vị này — nghĩa là, một vị trí đã biết chỉ đến một vị trí khác mà lần lượt có thể đã tìm kiếm dịch vụ. Dịch vụ có thể được định vị theo loại dịch vụ, theo tên, theo vị trí hoặc theo một số thuộc tính khác

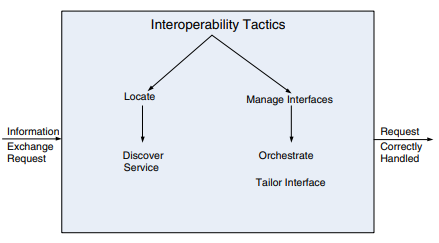
***Quản lý giao diện***

Quản lý giao diện bao gồm hai chiến thuật: giao diện phối hợp và điều chỉnh.

- Điều phối. Điều phối là một chiến thuật sử dụng một cơ chế điều khiển để điều phối và quản lý cũng như trình tự các lệnh gọi của các dịch vụ cụ thể (có thể không biết về nhau). Điều phối được sử dụng khi các hệ thống tương tác phải tương tác theo kiểu phức tạp để hoàn thành một nhiệm vụ phức tạp; dàn nhạc "kịch bản" tương tác. Các công cụ quy trình làm việc là một ví dụ về việc sử dụng chiến thuật điều phối. Mẫu thiết kế bộ hòa giải có thể phục vụ chức năng này để điều phối đơn giản. Điều phối phức tạp có thể được chỉ định bằng một ngôn ngữ như BPEL.

- Giao diện chỉnh sửa. Giao diện chỉnh sửa là một chiến thuật thêm hoặc bớt các khả năng cho một giao diện. Có thể thêm các khả năng như dịch, thêm bộ đệm hoặc làm mịn dữ liệu. Khả năng cũng có thể bị loại bỏ. Một ví dụ về việc loại bỏ các khả năng là ẩn các chức năng cụ thể khỏi những người dùng không đáng tin cậy.

Dịch vụ bus doanh nghiệp làm nền tảng cho nhiều kiến trúc hướng dịch vụ kết hợp cả hai chiến thuật giao diện quản lý.



Hình 3.16. Các chiến thuật của thuộc tính khả năng tích hợp

# CHƯƠNG 4. CÁC KIỂU KIẾN TRÚC PHẦN MỀM

Mục tiêu của chương này:

* Giới thiệu về các mẫu kiến trúc phần mềm
* Chỉ ra đặc điểm của từng mẫu kiến trúc
* Ưu/nhược điểm của từng mẫu kiến trúc
* Thảo luận về các miền ứng dụng của kiến trúc phần mềm
* Đánh giá lợi ích và hạn chế của mẫu kiến trúc phần mềm
* Case study cho các mẫu kiến trúc

## **4.1 Kiến trúc phần mềm dữ liệu tập trung**

### **4.1.1 Tổng quan**

Kiến trúc phần mềm lấy dữ liệu làm trung tâm được đặc trưng bởi một kho lưu trữ dữ liệu tập trung được chia sẻ bởi tất cả các thành phần phần mềm xung quanh. Hệ thống phần mềm được phân tách thành hai phân vùng chính: kho dữ liệu và thành phần phần mềm độc lập - agents. Các kết nối giữa mô-đun dữ liệu và các thành phần phần mềm được thực hiện bằng cách gọi phương thức rõ ràng hoặc bằng cách gọi phương thức ngầm định. Trong kiến ​​trúc phần mềm thuần túy lấy dữ liệu làm trung tâm, các thành phần phần mềm không giao tiếp trực tiếp với nhau; thay vào đó, tất cả các giao tiếp được thực hiện thông qua kho dữ liệu. Mô-đun dữ liệu được chia sẻ cung cấp tất cả các cơ chế để các thành phần phần mềm truy cập vào nó, chẳng hạn như chèn, xóa, cập nhật và truy xuất.

Có hai loại kiến ​​trúc lấy dữ liệu làm trung tâm: kho lưu trữ và blackboard. Chúng được phân biệt bởi chiến lược kiểm soát luồng. Kho lưu trữ dữ liệu trong kiến ​​trúc kho lưu trữ là thụ động và các máy khách của kho dữ liệu đang hoạt động; nghĩa là, các máy khách (thành phần phần mềm hoặc tác nhân) kiểm soát luồng logic. Khách hàng có thể truy cập kho lưu trữ tương tác hoặc theo yêu cầu giao dịch hàng loạt. Kiểu kho lưu trữ được sử dụng rộng rãi trong các hệ quản trị cơ sở dữ liệu, hệ thống thông tin thư viện, kho lưu trữ giao diện (IR) trong CORBA, cơ quan đăng ký UDDI cho các dịch vụ web, trình biên dịch và môi trường hệ thống công cụ được sử dụng để thiết kế và phát triển các phần mềm (CASE – Computer Aided Software Engineering). Một công cụ CASE nổi tiếng là Rational Rose. Nó hỗ trợ trình soạn thảo đồ họa để vẽ sơ đồ UML, tạo nhiều mã lập trình khác nhau và cung cấp chức năng thiết kế ngược để tạo sơ đồ đồ họa từ mã. Tất cả Môi trường phát triển tương tác (IDE) và các bộ công cụ phát triển phần mềm tương tự là những ví dụ điển hình về các miền ứng dụng cho kiến ​​trúc kho lưu trữ. Nó cũng được sử dụng rộng rãi trong các hệ thống quản lý thông tin phức tạp mà vấn đề quan trọng nhất là quản lý dữ liệu đáng tin cậy.

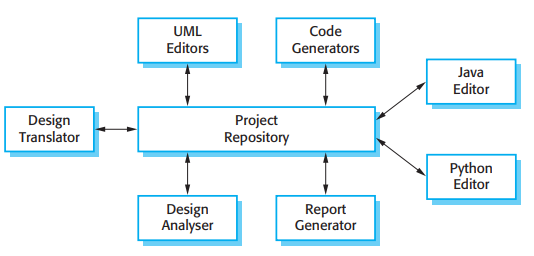
Kho lưu trữ dữ liệu trong tùy chọn kiến ​​trúc blackboard đang hoạt động và các máy khách của nó là thụ động; do đó, luồng logic được xác định bởi trạng thái dữ liệu hiện tại trong kho dữ liệu. Khách hàng của blackboard được gọi là nguồn tri thức, người nghe hoặc người đăng ký. Một thay đổi dữ liệu mới có thể kích hoạt các sự kiện để các nguồn tri thức thực hiện các hành động để phản hồi các sự kiện này. Những hành động này có thể tạo ra dữ liệu mới, từ đó có thể thay đổi luồng logic; điều này có thể xảy ra liên tục cho đến khi đạt được mục tiêu. Nhiều ứng dụng được thiết kế theo kiến ​​trúc blackboard bao gồm hệ thống AI dựa trên tri thức, hệ thống nhận dạng giọng nói và hình ảnh, hệ thống bảo mật, hệ thống quản lý tài nguyên nghiệp vụ, v.v.

### **4.1.2. Kiến trúc kho chứa (Repository Architecture, RA)**

Kiến trúc kho chứa, mô tả cách một tập hợp các thành phần tương tác có thể chia sẻ dữ liệu. Phần lớn các hệ thống sử dụng lượng lớn dữ liệu được tổ chức xung quanh một cơ sở dữ liệu hoặc kho lưu trữ dùng chung. Do đó, mô hình này phù hợp với các ứng dụng trong đó dữ liệu được tạo ra bởi một thành phần và được sử dụng bởi thành phần khác. Ví dụ về loại hệ thống này bao gồm hệ thống chỉ huy và điều khiển, hệ thống thông tin quản lý, hệ thống CAD và môi trường phát triển tương tác cho phần mềm

**Mô tả:** Quản lý dữ liệu trong kiến trúc RA được giao cho một kho trung tâm của hệ thống. Mọi cấu phần của hệ thống đều có thể truy cập đến kho này. Các cấu phần không giao tiếp trực tiếp với nhau mà buộc phải thông qua kho.

Hình 4.1 là một minh họa về tình huống mà một kho lưu trữ có thể được sử dụng. Biểu đồ này cho thấy một IDE bao gồm các công cụ khác nhau để hỗ trợ phát triển theo hướng mô hình. Kho lưu trữ trong trường hợp này có thể là một môi trường được kiểm soát phiên bản theo dõi các thay đổi đối với phần mềm và cho phép khôi phục về các phiên bản trước đó. Người sử dụng các công cụ UML editors có thể vẽ sơ đồ thiết kế UML như sơ đồ lớp, sơ đồ cộng tác hoặc sơ đồ tuần tự theo phương pháp Booch, phương pháp Rumbaugh hoặc phương pháp Jacobson và lưu trữ bản thiết kế trong kho dữ liệu. Các sơ đồ UML này sau đó có thể được chuyển đổi từ định dạng này sang định dạng khác. Mã khung Java hoặc Python cũng có thể được tạo dựa trên các sơ đồ UML này.



Hình 4.1. Kiến trúc kho lưu trữ cho một IDE

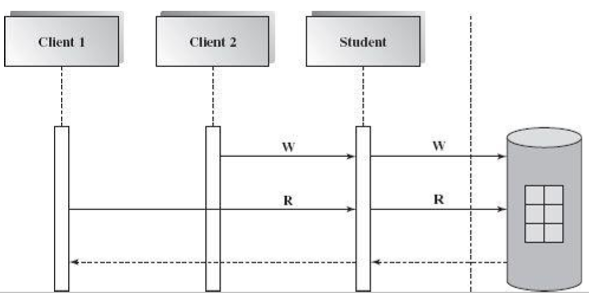
Tổ chức các công cụ xung quanh một kho lưu trữ là một cách hiệu quả để chia sẻ một lượng lớn dữ liệu. Không cần phải truyền dữ liệu một cách rõ ràng từ thành phần này sang thành phần khác. Tuy nhiên, các thành phần phải hoạt động xung quanh một mô hình dữ liệu kho lưu trữ đã thỏa thuận. Không thể tránh khỏi, đây là sự thỏa hiệp giữa các nhu cầu cụ thể của từng công cụ và có thể khó hoặc không thể tích hợp các thành phần mới nếu mô hình dữ liệu của chúng không phù hợp với lược đồ đã thỏa thuận. Trong thực tế, có thể khó phân phối kho lưu trữ trên một số máy. Mặc dù có thể phân phối một kho lưu trữ tập trung hợp lý, nhưng có thể có vấn đề về sự dư thừa và không nhất quán của dữ liệu



Hình 4.2 Biểu đồ lớp

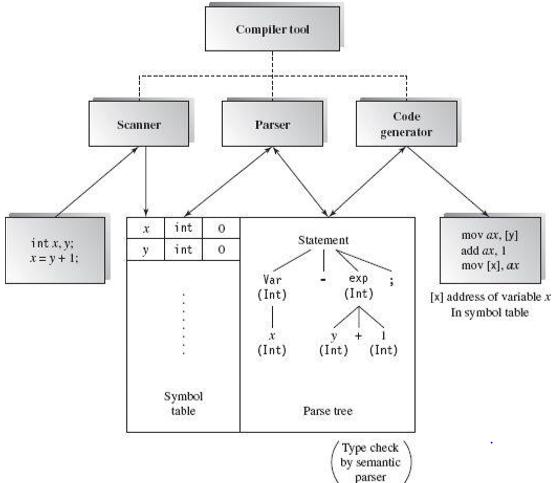
Hình 4.2 cho thấy một sơ đồ lớp cho một kho dữ liệu đơn giản để quản lý **Student**. Nó mô tả các mối quan hệ tĩnh giữa các lớp dữ liệu và các bảng cơ sở dữ liệu sao lưu của chúng, và giữa các lớp dữ liệu và các lớp thu thập của chúng. Sơ đồ này trình bày một cái nhìn theo hướng lập trình của kiến ​​trúc thiết kế kho lưu trữ. Tập hợp Students (Vector hoặc ArrayList, hoặc bất kỳ kiểu tập hợp phù hợp nào khác) có mối quan hệ thu nạp với lớp Student. Người dùng có thể thêm vào hoặc xóa một student khỏi Students cũng sẽ ảnh hưởng đến bảng Students trong cơ sở dữ liệu thông qua công nghệ kết nối cơ sở dữ liệu.

Hình 4.3. Mô tả một khung nhìn động của kiến ​​trúc kho này. Nó chỉ ra rằng một đối tượng thể hiện có thể được chia sẻ và truy cập bởi nhiều máy khách bằng cách đọc và ghi để tìm kiếm, cập nhật, chèn và xóa. Các máy khách có thể truy cập cùng một dữ liệu với giao diện dòng lệnh, giao diện GUI, giao diện chương trình, giao diện Gọi thủ tục từ xa (Remote Procedure Call interfaces - RPC) hoặc Gọi phương thức từ xa hướng đối tượng (object-oriented Remote Method Invocation - RMI).



Hình 4.3. Sơ đồ trình tự của kiến trúc kho lưu trữ

Xây dựng trình biên dịch là một ví dụ điển hình khác về thiết kế kiến ​​trúc kho lưu trữ. Mỗi hệ thống biên dịch đều có bảng từ khóa dành riêng, bảng ký hiệu định danh, bảng hằng số được tạo sau khi phân tích từ vựng, cây cú pháp và ngữ nghĩa được tạo ra bởi phân tích cú pháp và ngữ nghĩa. Cấu trúc dữ liệu của bảng này trong bộ nhớ được chia sẻ bởi tất cả các giai đoạn của quá trình biên dịch. Mỗi giai đoạn sẽ tạo ra dữ liệu mới hoặc cập nhật dữ liệu hiện có trong kho dữ liệu. Điều khiển luồng được kiểm soát bởi một chương trình lấy mã nguồn làm đầu vào của nó, sau đó đi qua từng giai đoạn từng bước và cuối cùng tạo ra mã nhị phân đích có thể thực thi hoặc có thể diễn giải được, chẳng hạn như mã bytecode của Java. Nói cách khác, tất cả các agents trong một hệ thống kho lưu trữ không nhất thiết phải hoàn toàn độc lập. Có một thứ tự hợp lý trong việc thực hiện tất cả các giai đoạn biên dịch. Vẫn có thể có một số thông tin liên lạc giữa các agents riêng biệt. Ví dụ, phân tích từ vựng có thể tìm thấy một số ký tự không được chấp nhận, do đó việc biên dịch phải được bỏ và các lỗi biên dịch phải được báo cáo. Chúng ta hãy xem xét kỹ hơn ví dụ biên dịch đơn giản được hiển thị trong Hình 4.4.



Hình 4.4. Compiler system – hệ thống trình biên dịch

**Tình huống vận dụng**: RA được vận dụng khi thiết kế các hệ thống phát sinh những tập dữ liệu lớn đòi hỏi phải có kho chứa trong khoảng thời gian dài. RA còn được vận dụng trong các hệ thống hướng dữ liệu, trong đó mỗi khi sản sinh dữ liệu trong kho chứa một công cụ hoặc một thao tác nào đó của hệ thống sẽ được kích hoạt.

**Ưu điểm:** Các thành phần được hoạt động độc lập với nhau. Mỗi thành phần trong hệ thống không cần quan tâm đến sự tồn tại của cấu phần khác. Do dữ liệu được tập trung tại một kho nên sự thay đổi (dữ liệu) do một thành phần sinh ra sẽ được truyền bá đến mọi thphần khác trong hệ thống. Mọi dữ liệu trong hệ thống được đảm bảo toàn vẹn (thí dụ, hệ thống có thể thực hiện sao lưu tại bất kỳ thời điểm nào). Khả năng mở rộng hệ thống và khả năng tái sử dụng cao thể hiện thông qua việc thêm các thành phần phần mềm mới dễ dàng vì chúng không có giao tiếp với nhau.

**Nhược điểm:** Mọi khó khăn nảy sinh đều rơi vào nguyên nhân tập trung dữ liệu. Truy nhập dữ liệu từ xa, hoặc phân tán trên nhiều máy tính có thể không hiệu quả. Mọi lỗi dữ liệu đều ảnh hưởng đến mọi cấu phần hệ thống. Độ tin cậy và tính khả dụng của kho dữ liệu là những vấn đề quan trọng. Kho lưu trữ tập trung dễ bị thất bại so với kho lưu trữ phân tán có nhân bản dữ liệu. Chi phí di chuyển dữ liệu trên mạng nếu dữ liệu được phân phối.

Trong ví dụ minh họa trong Hình 5.1 kho lưu trữ là thụ động và việc kiểm soát là trách nhiệm của các thành phần sử dụng kho lưu trữ. Một cách tiếp cận thay thế, đã được bắt nguồn cho các hệ thống AI, sử dụng mô hình **‘Blackboard - bảng đen’** để kích hoạt các thành phần khi có dữ liệu cụ thể. Điều này thích hợp khi hình thức của dữ liệu kho lưu trữ có cấu trúc kém hơn. Quyết định về việc kích hoạt công cụ nào chỉ có thể được thực hiện khi dữ liệu đã được phân tích. Mô hình này được giới thiệu bởi Nii (1986). Bosch (2000) bao gồm một cuộc thảo luận tốt về cách mà phong cách này liên quan đến các thuộc tính chất lượng hệ thống.

### **4.1.3 Mẫu kiến trúc Blackboard**

Kiến trúc blackboard được phát triển cho các ứng dụng nhận dạng giọng nói vào những năm 1970. Các ứng dụng khác cho kiến ​​trúc này là hệ thống nhận dạng mẫu hình ảnh và phát sóng thời tiết. Ví dụ điển hình của kiến ​​trúc này là hệ thống chuyên gia nhận dạng giọng nói Hearsay-II và hệ thống phân tích cấu trúc phân tử CRYSTALIS.

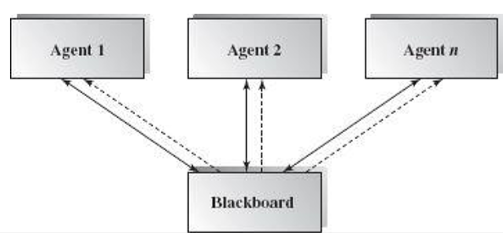
Từ blackboard bắt nguồn từ việc dạy và học trên lớp. Giáo viên và học sinh có thể chia sẻ dữ liệu trong việc giải quyết các vấn đề trong lớp học qua blackboard. Học sinh và giáo viên đóng vai trò là tác nhân góp phần giải quyết vấn đề. Tất cả đều có thể làm việc song song và độc lập, cố gắng tìm ra giải pháp tốt nhất.

Ý tưởng về kiến ​​trúc blackboard tương tự như blackboard trong lớp học được sử dụng để giải quyết các vấn đề mà không có kết quả xác định. Nó là một kiến ​​trúc hướng dữ liệu và một phần hướng dữ liệu. Toàn bộ hệ thống được phân tách thành hai phân vùng chính. Một phân vùng, được gọi là bảng đen, được sử dụng để lưu trữ dữ liệu (giả thuyết và dữ kiện), trong khi phân vùng còn lại, được gọi là nguồn tri thức, lưu trữ tri thức cụ thể của từng miền. Cũng có thể có một phân vùng thứ ba, được gọi là bộ điều khiển, được sử dụng để khởi tạo bảng đen và các nguồn tri thức và có vai trò khởi động và kiểm soát giám sát tổng thể.

Các kết nối giữa hệ thống con blackboard và các nguồn tri thức về cơ bản là các lệnh gọi ngầm từ blackboard tới các nguồn tri thức cụ thể, được đăng ký trước với blackboard. Thay đổi dữ liệu trong blackboard kích hoạt một hoặc nhiều nguồn kiến ​​thức phù hợp để tiếp tục xử lý. Sự thay đổi dữ liệu có thể được gây ra bởi thông tin suy luận mới hoặc kết quả giả thuyết bởi một số nguồn kiến ​​thức. Kết nối này có thể được thực hiện ở chế độ publish/subscribe.

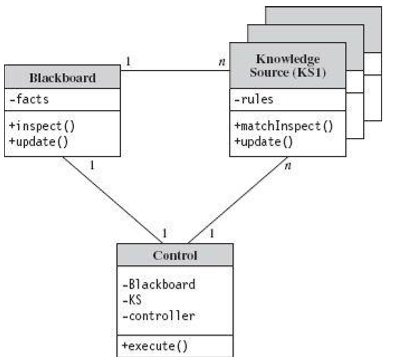
Hình 4.5 minh họa một sơ đồ khối của kiến ​​trúc blackboard. Các đường liền nét biểu thị các liên kết dữ liệu, trong khi các đường đứt nét biểu thị điều khiển logic luồng, được kiểm soát bởi bất kỳ thay đổi dữ liệu nào trong kho dữ liệu. Tức là, dữ liệu trong kho lưu trữ blackboard định hướng luồng tính toán.

Nhiều nguồn kiến ​​thức dành riêng cho miền cộng tác với nhau để giải quyết một vấn đề phức tạp như nhận dạng mẫu hoặc xác thực trong bảo mật thông tin. Mỗi nguồn kiến ​​thức tương đối độc lập với các nguồn kiến ​​thức khác. Chúng không cần phải tương tác với nhau, điều này rất giống với một hệ thống kho lưu trữ. Chúng chỉ cần tương tác và phản hồi với hệ thống con blackboard. Mỗi nguồn hoạt động trên một khía cạnh cụ thể của vấn đề và đóng góp một phần giải pháp cho giải pháp cuối cùng.



Hình 4.5. Kiến trúc Blackboard

Hình 4.6. cho thấy sơ đồ lớp UML của kiến ​​trúc phần mềm blackboard dựa trên quy tắc. Như chúng ta có thể thấy, một blackboard có thể có nhiều nguồn tri thức được liên kết với nó, làm việc trên dữ liệu đã cho và dữ liệu suy luận có sẵn trong hệ thống con blackboard. Mỗi nguồn kiến ​​thức giúp giải quyết các vấn đề trong lĩnh vực chuyên môn của nó. Kiến thức có thể được lưu trữ trong các định dạng biểu diễn tri thức khác nhau tùy thuộc vào chiến lược lập luận. Ví dụ: một nguồn tri thức lưu trữ tất cả các quy tắc liên quan và cung cấp các cơ chế kích hoạt để blackboard kích hoạt trong hệ thống chuyên gia dựa trên quy tắc. Tất nhiên, các nguồn kiến ​​thức phải tự đăng ký với blackboard để nếu có bất kỳ thay đổi nào diễn ra trên blackboard, chúng sẽ được thông báo để kích hoạt các hành động trong nguồn kiến ​​thức tương ứng, từ đó có thể suy ra các dữ kiện mới và cập nhật blackboard. Mỗi nguồn kiến ​​thức riêng lẻ có thể có chiến lược giải quyết vấn đề riêng và sử dụng kiến ​​thức chuyên môn của riêng mình để đóng góp vào một giải pháp từng phần dẫn đến giải pháp cuối cùng. Lớp blackboard giữ trạng thái dữ liệu hiện tại và giải pháp vấn đề cuối cùng sẽ được đặt trong blackboard để bộ điều khiển nhận và sử dụng để tạo báo cáo cuối cùng. Chiến lược dựa trên quy tắc là một trong nhiều thuật toán lập luận được sử dụng ngày nay. Có nhiều chiến lược giải quyết vấn đề khác có thể được áp dụng bao gồm lý thuyết tập hợp mờ, xác suất và thống kê, mạng nơ-ron, khai thác dữ liệu và tìm kiếm theo phương pháp heuristic.



Hình 4.6 Sơ đồ lớp của kiến trúc blackboard

Vì kiến trúc blackboard về cơ bản là một hệ thống tự kích hoạt, hệ thống con bộ điều khiển trong kiến trúc chỉ hoạt động ở phần đầu của quá trình khởi tạo blackboard và tất cả các nguồn tri thức; nó cũng định kỳ kiểm tra trạng thái hiện tại của blackboard để xác định xem có nên chấm dứt quá trình xử lý hay không nếu giải pháp là chấp nhận được hoặc đủ tối ưu.

Hãy cùng xem xét một hệ thống dựa trên kiến thức nhận dạng động vật nổi tiếng (KBS). Kiến thức được thể hiện dưới dạng các quy tắc sản xuất với các phần điều kiện và hành động. Đối với mỗi quy tắc, nếu điều kiện là đúng thì hành động sẽ được thực hiện. Hành động là đưa dữ liệu kết luận mới vào kho dữ liệu, đó là blackboard.

Dưới đây là một tập các luật:

R1: NẾU động vật cho sữa thì động vật là động vật có vú

R2: NẾU động vật ăn thịt thì động vật là động vật ăn thịt

R3: NẾU động vật là động vật có vú và động vật là động vật ăn thịt và động vật có màu hung và động vật có vằn đen thì động vật là hổ

Tập hợp các sự kiện:

F1: động vật ăn thịt

F2: động vật cho sữa

F3: con vật có sọc đen

F4: động vật có màu xám

Mục đích là để nhận ra con vật. Vấn đề có thể yêu cầu bạn chấp thuận một tuyên bố hoặc nhận được sự công nhận tốt nhất trong danh mục động vật.

Tất cả các dữ kiện được đặt trong blackboard và tất cả kiến thức được đặt trong các nguồn kiến thức bởi bộ điều khiển. Có thể có nhiều mô hình lý luận khác nhau như lý luận tiến hoặc lý luận lùi.

Suy luận chuyển tiếp bắt đầu với trạng thái ban đầu của dữ liệu và tiến tới mục tiêu. Nó có thể đạt được mục tiêu đã định hoặc có thể không đạt được mục tiêu. Nếu mục tiêu không được đưa ra, lý luận sẽ đến lúc không thể rút ra thêm dữ kiện mới, cho thấy rằng kết quả tốt nhất đã được xác định. Lý luận lạc hậu hoạt động theo hướng ngược lại.

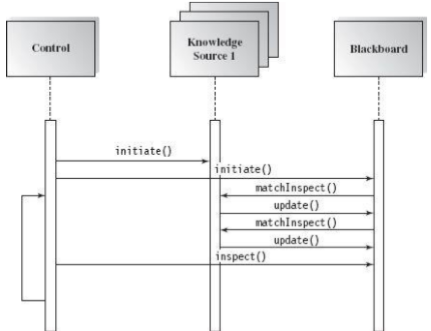
Hãy làm theo trình tự lập luận thuận. F1 phù hợp với điều kiện của R2, và "động vật là động vật ăn thịt" được bắt nguồn và lưu trữ trong blackboard. Nếu thuật toán kiểm tra dữ liệu mới nhất được tạo trước và dữ liệu mới không khớp với bất kỳ quy tắc nào, thì thuật toán sẽ kiểm tra F2 phù hợp với điều kiện của R1. Sau đó, "động vật là động vật có vú" được bắt nguồn và lưu trữ trong blackboard. Sau thời điểm này, không có sự kiện nào

Hãy làm theo trình tự lập luận thuận. F1 phù hợp với điều kiện của R2, và "động vật là động vật ăn thịt" được bắt nguồn và lưu trữ trong blackboard. Nếu thuật toán kiểm tra dữ liệu mới nhất được tạo trước và dữ liệu mới không khớp với bất kỳ quy tắc nào, thì thuật toán sẽ kiểm tra F2 phù hợp với điều kiện của R1. Sau đó, "động vật là động vật có vú" được bắt nguồn và lưu trữ trong blackboard. Sau thời điểm này, không có sự kiện nào phù hợp với bất kỳ quy tắc nào và nếu dữ liệu kết hợp được sử dụng, R3 sẽ được khớp và “động vật là hổ” được lấy ra và đưa trở lại kho dữ liệu. Mặc dù đây là một ví dụ tương đối đơn giản, nhưng nó cho chúng ta biết kiến ​​thức được sử dụng như thế nào trong lập luận. Trong một hệ thống blackboard thực tế hơn, có nhiều tài nguyên kiến ​​thức, mỗi tài nguyên có các biểu diễn kiến ​​thức và thuật toán suy luận riêng.

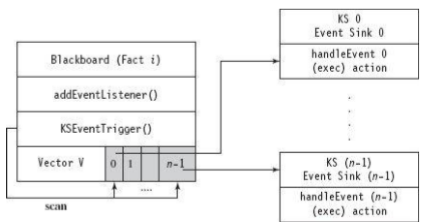
Biểu đồ trình tự trong Hình 4.7 minh họa các tương tác động trong hệ thống blackboard. Bộ điều khiển khởi tạo blackboard và tất cả các nguồn tri thức trước, sau đó bắt đầu xử lý bằng cách kiểm tra blackboard và các nguồn tri thức và kích hoạt luồng điều khiển. Mọi thay đổi trên blackboard có thể đáp ứng các điều kiện của quy tắc và các hành động mới sẽ được thực hiện đối với blackboard và blackboard được cập nhật. Nó sẽ tiếp tục cho đến khi bộ điều khiển quyết định dừng quá trình xử lý sau khi đưa ra kết luận tối ưu có thể chấp nhận được hoặc đơn giản là bỏ quá trình xử lý do lỗi.

Sơ đồ trong Hình 4.8 minh họa mối quan hệ xuất bản/đăng ký giữa blackboard và các nguồn tri thức trong hệ thống blackboard.

Tất cả các nguồn kiến ​​thức đều đăng ký hoặc đăng ký các sự kiện và giả thuyết cụ thể được lưu trữ trong kho lưu trữ blackboard bằng phương thức addEventListener() của blackboard. Các đăng ký được lưu trữ trong vectơ v. Nếu bất kỳ dữ liệu nào được thêm vào hoặc thay đổi, blackboard sẽ thông báo cho các nguồn tri thức đã đăng ký bằng eventTrigger() để các nguồn tri thức sẽ thực hiện các hành động để phản hồi hoặc xử lý các sự kiện.



Hình 4.7 Sơ đồ trình tự của kiến trúc blackboard



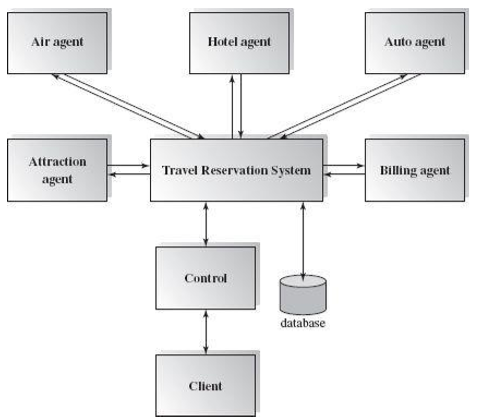
Hình 4.8 Xuất bản/đăng ký mối quan hệ giữa blackboard và nguồn kiến thức

Một ứng dụng ví dụ đơn giản từ cuộc sống hàng ngày minh họa các nguyên tắc cơ bản của kiến ​​trúc blackboard. Hãy nghĩ đến một hệ thống tư vấn du lịch thông minh có thể hỗ trợ bạn lập kế hoạch du lịch hiệu quả về mặt thời gian và chi phí. Khách hàng chỉ cần điền vào một biểu mẫu và hệ thống sẽ trả lời với nhiều gói tùy chọn để lựa chọn.

Kiến trúc blackboard rất phù hợp vì việc lập kế hoạch du lịch đòi hỏi nhiều đại lý gần kề như hãng hàng không, đặt phòng khách sạn, thuê ô tô và các đại lý thu hút khách. Nó cũng liên quan đến một kho dữ liệu về ngân sách, thời gian có sẵn và các dữ kiện khác sẽ được chia sẻ bởi tất cả các đại lý. Hệ thống tự tạo ra tất cả các kế hoạch mà không cần nhiều tương tác bổ sung.

Sau khi khách hàng gửi yêu cầu, hệ thống sẽ lưu trữ tất cả các dữ liệu này vào kho dữ liệu và gửi yêu cầu đến các đại lý hàng không. Sau khi dữ liệu đặt phòng hàng không trở lại và được lưu trong kho dữ liệu, thay đổi sẽ kích hoạt các đại lý bán vé khách sạn, cho thuê ô tô và điểm du lịch làm việc cùng nhau để đưa ra một kế hoạch tối ưu trong phạm vi ngân sách và thời gian biểu nhất định. Sau khi khách hàng chọn một trong các gói, hệ thống sẽ kích hoạt quy trình thanh toán để hoàn tất gói.

Hình 4.9 cho thấy kiến ​​trúc blackboard cho hệ thống tư vấn du lịch này. Có thể có nhiều đại lý du lịch hàng không, hệ thống đặt phòng khách sạn, công ty cho thuê xe hơi, hoặc hệ thống đặt chỗ điểm tham quan để đăng ký hoặc đăng ký thông qua hệ thống lập kế hoạch du lịch này. Khi hệ thống nhận được yêu cầu của khách hàng, nó sẽ công bố yêu cầu đó cho tất cả các tác nhân liên quan và soạn các phương án kế hoạch để khách hàng lựa chọn. Hệ thống cũng lưu trữ tất cả các dữ liệu cần thiết trong cơ sở dữ liệu. Sau khi hệ thống nhận được xác nhận từ khách hàng, nó sẽ gọi hệ thống thanh toán tài chính để xác minh lý lịch tín dụng và xuất hóa đơn. Dữ liệu trong kho dữ liệu đóng một vai trò tích cực trong hệ thống này. Nó không đòi hỏi nhiều sự tương tác của người dùng sau khi hệ thống nhận được yêu cầu của khách hàng vì dữ liệu yêu cầu sẽ chỉ đạo tính toán và kích hoạt tất cả các nguồn kiến ​​thức liên quan để giải quyết vấn đề.



Hình 4.9 Kiến trúc blackboard cho một hệ thống tư vấn du lịch

**Miền áp dụng của kiến trúc blackboard:**

* Thích hợp để giải quyết các vấn đề mở và phức tạp như các vấn đề về trí tuệ nhân tạo (AI) mà không tồn tại các giải pháp đặt trước.
* Vấn đề bao gồm nhiều lĩnh vực, và mỗi vấn đề liên quan đến các loại kiến thức chuyên môn hoàn toàn khác nhau và các vấn đề giải quyết các mô hình đòi hỏi sự hợp tác.
* Giải pháp một phần hoặc gần đúng có thể chấp nhận được đối với các vấn đề.
* Tìm kiếm vét cạn là không thể và không thực tế vì nó có thể mất mãi mãi vì kiến thức sẵn có và thậm chí cả dữ liệu và giả thuyết có thể không đầy đủ hoặc không chính xác.

**Những lợi ích:**

* Khả năng mở rộng: dễ dàng bổ sung hoặc cập nhật nguồn kiến ​​thức.
* Đồng thời: tất cả các nguồn tri thức có thể hoạt động song song vì chúng độc lập với nhau.
* Hỗ trợ thử nghiệm cho các giả thuyết.
* Khả năng tái sử dụng của các tác nhân nguồn tri thức.

**Hạn chế:**

* Do sự phụ thuộc chặt chẽ giữa blackboard và nguồn tri thức, sự thay đổi cấu trúc của blackboard có thể có tác động đáng kể đến tất cả các tác nhân của nó.
* Vì chỉ mong đợi các giải pháp một phần hoặc gần đúng, có thể khó quyết định khi nào chấm dứt suy luận.
* Đồng bộ hóa nhiều tác nhân là một vấn đề. Vì nhiều tác nhân đang làm việc và cập nhật dữ liệu được chia sẻ trong blackboard đồng thời, nên ưu tiên hoặc mức độ ưu tiên thực thi của nhiều tác nhân phải được phối hợp.
* Gỡ lỗi và kiểm tra hệ thống là một thách thức vì nó không tuân theo một thuật toán xác định

## **4.2 Kiến trúc phần mềm tương tác**

### **4.2.1 Tổng quan**

Ngày càng có nhiều ứng dụng phần mềm liên quan đến các tương tác đầu vào/đầu ra của người dùng và giao diện người dùng cụ thể với hệ thống phần mềm. Trong chương này, chúng ta tập trung vào kiến ​​trúc phần mềm hỗ trợ tốt nhất cho tương tác của người dùng trong ứng dụng. Chúng ta cũng sẽ thảo luận về các kiến ​​trúc phần mềm để thiết kế giao diện người dùng.

Kiến trúc phần mềm hướng tương tác phân tách hệ thống thành ba phân vùng chính: mô-đun dữ liệu (data), mô-đun điều khiển (control) và mô-đun trình diễn (view). Mỗi mô-đun có trách nhiệm riêng. Mô-đun dữ liệu cung cấp tính trừu tượng hóa dữ liệu và tất cả logic nghiệp vụ cốt lõi về xử lý dữ liệu. Mô-đun trình diễn chịu trách nhiệm trình bày đầu ra dữ liệu hình ảnh hoặc âm thanh và cũng có thể cung cấp giao diện đầu vào của người dùng khi cần thiết. Mô-đun điều khiển xác định luồng điều khiển liên quan đến việc lựa chọn giao diện hiển thị, giao tiếp giữa các mô-đun, điều phối công việc và một số hành động khởi tạo dữ liệu và cấu hình hệ thống. Điểm mấu chốt của kiến ​​trúc này là nó tách biệt các tương tác của người dùng khỏi trừu tượng hóa dữ liệu và xử lý dữ liệu nghiệp vụ. Vì có thể có nhiều bản trình diễn ở các định dạng khác nhau, nhiều giao diện (view) có thể được hỗ trợ cho cùng một tập dữ liệu. Ngay cả đối với một bản trình diễn cụ thể, các giao diện hoặc dạng xem có thể cần phải thay đổi thường xuyên, vì vậy việc kết hợp lỏng lẻo giữa phần trừu tượng hóa dữ liệu và bản trình diễn của nó là hữu ích và được kiểu này hỗ trợ. Các kết nối khớp nối lỏng lẻo có thể được thực hiện theo nhiều cách khác nhau như gọi phương thức rõ ràng hoặc gọi phương thức đăng ký/thông báo ngầm. Mô-đun điều khiển đóng vai trò trung tâm làm trung gian cho mô-đun dữ liệu và xem các mô-đun trình diễn. Tất cả ba mô-đun có thể được kết nối đầy đủ. Presentation – Abstraction - Control (PAC) và Model – View - Controller (MVC).

Hai mô hình này tương tự nhau theo nghĩa là chúng đề xuất ba sự phân rã thành phần. Mô-đun Presentation của PAC giống như mô-đun View của MVC; mô-đun Abstraction của PAC giống như mô-đun Model của MVC; mô-đun Control của PAC giống như mô-đun Controller của MVC. Cả MVC và PAC đều được sử dụng cho các ứng dụng tương tác như ứng dụng trực tuyến web và ứng dụng phân tán với nhiều tác vụ và tương tác của người dùng. Chúng khác nhau về quy trình kiểm soát và tổ chức. PAC là một kiến ​​trúc phân cấp dựa trên tác nhân, trong khi MVC không có cấu trúc phân cấp rõ ràng và cả ba mô-đun được kết nối với nhau.

Apache Strut là một khuôn khổ dựa trên mẫu nổi tiếng cho kiến ​​trúc MVC được sử dụng rộng rãi trong thiết kế ứng dụng web.

### **4.2.2 Kiến trúc MVC**

Mô hình kiến ​​trúc MVC đã tồn tại từ lâu trong kỹ thuật phần mềm. Hầu hết các nhà phát triển Web đều quen thuộc với kiến ​​trúc MVC vì nó được áp dụng rộng rãi cho thiết kế ứng dụng tương tác trên trang web máy chủ như mua sắm trực tuyến, khảo sát, đăng ký sinh viên và nhiều hệ thống dịch vụ tương tác khác. Kiến trúc MVC được sử dụng đặc biệt trong các ứng dụng mà giao diện người dùng dễ bị thay đổi dữ liệu.

Kiến trúc này tách phần trình bày và tương tác khỏi dữ liệu hệ thống. Kiến trúc này lần đầu tiên được giới thiệu trong Smalltalk-80. Theo Glenn Krasner và Stephen Pope (1988), hệ thống được cấu trúc thành ba thành phần logic Model, View và Controller tương tác với nhau.

**Model**: cấu phần mô hình quản trị dữ liệu hệ thống (bao gồm cả các thao tác trên dữ liệu)

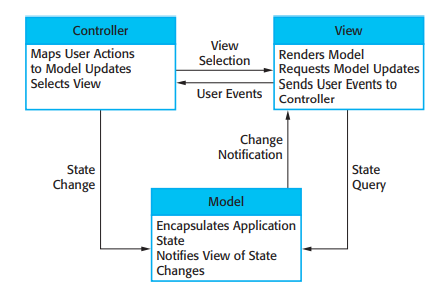
**View (hiển thị)**: xác định và quản lý các dữ liệu được trình bày cho người dùng. Thành phần này đảm nhận việc hiển thị thông tin, tương tác với người dùng, nơi chứa tất cả các đối tượng GUI như textbox, button, images….

Ví dụ: View trong ASP.NET MVC là HTML, CSS và một số cú pháp đặc biệt (cú pháp Razor) giúp dễ dàng giao tiếp với mô hình và bộ điều khiển.

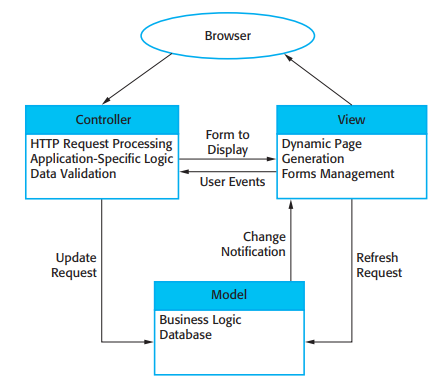
**Controller** - Bộ điều khiển: quản lý tương tác của người dùng. Thông thường, Controller nhận yêu cầu sau đó chuyển những tương tác này đến View hoặc Model.

Các mô hình đồ họa của kiến trúc liên kết với mẫu MVC được thể hiện trong Hình 4.10 và 4.11. Chúng trình bày kiến trúc từ các khung nhìn khác nhau — Hình 4.10 là một khung nhìn khái niệm và Hình 4.11 cho thấy một kiến trúc thời gian chạy có thể có khi mô hình này được sử dụng để quản lý tương tác trong một hệ thống dựa trên web.

Hình 4.10 mô tả mẫu Model-View-Controller nổi tiếng. Mô hình này là cơ sở của quản lý tương tác trong nhiều hệ thống dựa trên web.

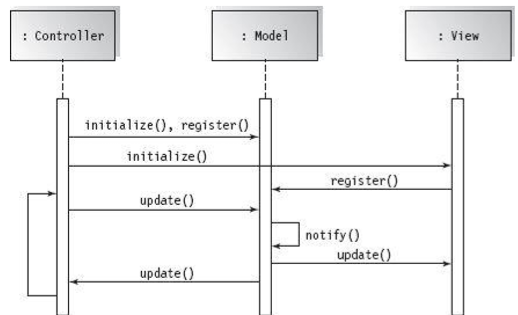


Hình 4.10. Tổ chức của mô hình MVC



Hình 4.11. Kiến trúc ứng dụng Web sử dụng mẫu kiến trúc MVC

Hình 4.12 mô tả một biểu đồ trình tự cho một kiến trúc MVC chung. Sau khi máy khách khởi động ứng dụng MVC, Controller khởi tạo Model và View, đồng thời gắn chính nó và View vào Model (điều này được gọi là đăng ký với Model). Sau đó, Controller lấy một yêu cầu của người dùng trực tiếp từ một dòng lệnh hoặc thông qua giao diện View và chuyển tiếp yêu cầu tới Model để cập nhật dữ liệu trong Model. Các thay đổi trong Model sẽ kích hoạt Model thông báo cho tất cả người nghe được đính kèm hoặc đã đăng ký về tất cả các thay đổi và các giao diện trong View cũng sẽ được cập nhật ngay lập tức



Hình 4.12. Sơ đồ trình tự cho kiến trúc MVC

Ví dụ sau minh họa một cách triển khai đơn giản của kiến trúc MVC trong đó chỉ có một lớp Java trong mỗi mô-đun trong ba mô-đun của kiến trúc MVC. Lớp MyBean JavaBean đóng vai trò của Mô hình dữ liệu; MyServlet Lớp Servlet đóng vai trò của Controller; và from-Servlet JSP đóng vai trò của View hiển thị.

Hình 4.13. cho thấy sơ đồ kiến trúc của ứng dụng web này. MyServlet Servlet đặt một giá trị mục và lưu trữ mục này trong một JavaBean có tên là myBean. Sau đó, nó chuyển quyền điều khiển đến một trang JSP có tên fromServlet.jsp, trang này lấy mục từ myBean và hiển thị nó trên một trang web.



Hình 4.13. Kiến trúc MVC trên nền tảng Java Web

**Các miền ứng dụng của kiến trúc MVC:**  được sử dụng khi có nhiều cách để view và tương tác với dữ liệu. Các ứng dụng có sự phân chia rõ ràng giữa các mô-đun điều khiển, view và data để các chuyên gia khác nhau có thể được chỉ định làm việc đồng thời trên các khía cạnh khác nhau của các ứng dụng đó.

**Ưu điểm**

* Cho phép dữ liệu (model) thay đổi độc lập với trình diễn (view) của nó và ngược lại.
* Hỗ trợ trình bày cùng một dữ liệu theo những cách khác nhau với những thay đổi được thực hiện trong một biểu diễn được hiển thị trong tất cả chúng.
* Thể hiện tính chuyên nghiệp trong lập trình, phân tích thiết kế.
* Loose coupling: (gắn kết lỏng lẻo): bản chất của MVC framework là có sự ghép nối thấp giữa các models, views, controllers
* Do được chia thành các thành phần độc lập nên giúp phát triển ứng dụng nhanh, đơn giản, dễ nâng cấp và bảo trì, …
* Nhiều bộ công cụ Framework của nhà cung cấp MVC có sẵn
* Rất hiệu quả cho sự phát triển nếu các chuyên gia phát triển đồ họa, lập trình và cơ sở dữ liệu đang làm việc trong một nhóm trong một dự án được thiết kế.

**Nhược điểm**

* Có thể liên quan đến chứa các mã trình bổ xung và mã trình phức tạp ngay cả trong trường hợp mô hình dữ liệu và các tương tác đơn giản.
* Đối với những dự án nhỏ việc áp dụng mô hình MVC gây cồng kềnh, tốn thời gian trong quá trình phát triển
* Tốn thời thời gian trung chuyển dữ liệu của các thành phần.
* Không phù hợp với các ứng dụng hướng về tác nhân như các ứng dụng di động và robot tương tác.
* Nhiều cặp bộ controller và view dựa trên cùng một model làm cho bất kỳ thay đổi mô hình dữ liệu nào trở nên đắt đỏ.
* Code navigability (khả năng điều hướng mã nguồn): các framework điều hướng có thể phức tạp vì nó giới thiệu các gián tiếp mới và yêu cầu người dùng phải thích ứng với các tiêu chí phân rã của MVC.
* Nhất quán giữa các thành phần: phân tách một tính năng thành 3 thành phần gây ra sự tán xạ. Do đó, yêu cầu các nhà phát triển phải duy trì một tính nhất quán giữa các thành phần.

## **4.3. Kiến trúc phân lớp**

Các khái niệm về sự tách biệt và độc lập là cơ bản trong thiết kế kiến trúc vì chúng cho phép bản địa hóa những thay đổi. Mẫu MVC, phân tách các phần tử của hệ thống, cho phép chúng thay đổi độc lập. Ví dụ: thêm một chế độ xem mới hoặc thay đổi một chế độ xem hiện có có thể được thực hiện mà không có bất kỳ thay đổi nào đối với dữ liệu cơ bản trong mô hình. Mô hình kiến trúc phân lớp là một cách khác để đạt được sự tách biệt và độc lập. Mô hình phân lớp sẽ tổ chức chức năng hệ thống thành các lớp riêng biệt và mỗi lớp chỉ dựa vào các tiện ích và dịch vụ được cung cấp bởi lớp ngay bên dưới nó.

Cách tiếp cận phân lớp này hỗ trợ sự phát triển gia tăng của các hệ thống. Khi một lớp được phát triển, một số dịch vụ do lớp đó cung cấp có thể được cung cấp cho người dùng. Kiến trúc cũng có thể thay đổi và di động. Miễn là giao diện của nó không thay đổi, một lớp có thể được thay thế bằng một lớp khác tương đương. Hơn nữa, khi giao diện lớp thay đổi hoặc các cơ sở mới được thêm vào một lớp, chỉ lớp liền kề bị ảnh hưởng. Khi các hệ thống phân lớp bản địa hóa sự phụ thuộc của máy trong các lớp bên trong, điều này giúp dễ dàng cung cấp các triển khai đa nền tảng của một hệ thống ứng dụng. Chỉ các lớp bên trong, phụ thuộc vào máy cần được triển khai lại để tính đến các cơ sở của một hệ điều hành hoặc cơ sở dữ liệu khác.

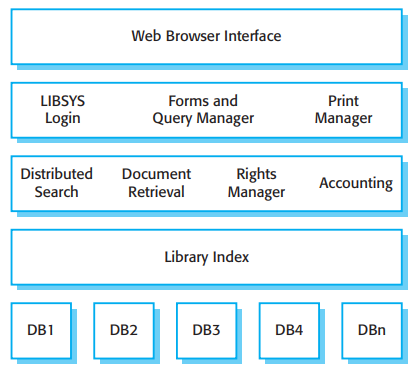
Mô tả: Kiến trúc phân tầng tổ chức hệ thống theo các tầng, mỗi tầng đảm nhận các nhóm chức năng tương ứng. Tầng dưới phục vụ cho tầng trên. Tầng dưới cùng chính là các dịch vụ cơ sở.

Hình 4.15 là một ví dụ về kiến trúc phân lớp với bốn lớp. Lớp thấp nhất bao gồm phần mềm hỗ trợ hệ thống — thường là hỗ trợ cơ sở dữ liệu và hệ điều hành. Lớp tiếp theo là lớp ứng dụng bao gồm các thành phần liên quan đến chức năng ứng dụng và các thành phần tiện ích được sử dụng bởi các thành phần ứng dụng khác. Lớp thứ ba liên quan đến quản lý giao diện người dùng và cung cấp xác thực và ủy quyền người dùng, với lớp trên cùng cung cấp các tiện ích giao diện người dùng. Tất nhiên, số lượng các lớp là tùy ý. Bất kỳ lớp nào trong Hình 4.15 đều có thể được chia thành hai hoặc nhiều lớp.



Hình 4.15 Kiến trúc phân lớp chung

Hình 4.16 là một ví dụ về cách mẫu kiến trúc phân lớp này có thể được áp dụng cho một hệ thống thư viện được gọi là LIBSYS, cho phép truy cập điện tử có kiểm soát vào tài liệu bản quyền từ một nhóm thư viện đại học. Điều này có kiến trúc năm lớp, với lớp dưới cùng là các cơ sở dữ liệu riêng lẻ trong mỗi thư viện



Hình 4.16. Kiến trúc của hệ thống quản lý thư viện LIBSYS

**Tình huống vận dụng:** LA được vận dụng trong các tình huống sau: (1) Khi xây dựng các tiện ích mới nằm ở tầng trên của hệ thống hiện có; (2) Khi một bước phát triển được phân chia cho nhiều nhóm, mỗi nhóm triển khai một chức năng; (3) Khi đòi hỏi bảo mật theo nhiều mức

**Ưu/nhược điểm của kiến trúc phân lớp**

**Ưu điểm:** Kiến trúc LA cho phép thay thế trọn một tầng khi cần cập nhật lại các khuôn dạng hiển thị dữ liệu. Có thể xây dựng các khả năng dự phòng dư thừa nhằm nâng cao tính phụ thuộc cho hệ thống, thí dụ, khi cần tổ chức bảo mật, cấp và xác thực bản quyền, ngăn chặn người sử dụng xem các dữ liệu nhạy cảm, ta có thể bổ sung thêm cơ chế bảo mật, phân quyền vào tầng hiển thị hoặc thêm hẳn một tầng mới đặt dưới tầng hiển thị nhằm xác thực quyền truy nhập của người sử dụng

**Nhược điểm**: Trong thực tế việc tổ chức phân định rạch ròi giữa các tầng thường là khó và tầng cao thường đôi khi cần tương tác với các tầng thấp không nằm trực tiếp dưới nó. Kiến trúc LA có thể gây ảnh hưởng đến năng suất vì mọi yêu cầu được chuyển đến tầng khác đều phải được kiểm soát và xử lý...

## **4.4. Kiến trúc dựa trên thành phần**

### **4.4.1. Tổng quan**

Kiến trúc phần mềm dựa trên thành phần chia một vấn đề thành các bài toán con, mỗi bài toán được liên kết với các phân vùng thành phần. Giao diện của các thành phần đóng vai trò quan trọng trong thiết kế dựa trên thành phần. Động lực chính đằng sau thiết kế dựa trên thành phần là khả năng tái sử dụng thành phần: một thành phần đóng gói chức năng và hành vi của phần tử phần mềm thành một đơn vị nhị phân có thể tái sử dụng và tự triển khai. Các thiết kế có thể tận dụng các thành phần thương mại bán sẵn (commercial off-the-shelf - COTS) có thể tái sử dụng cho hệ thống hiện tại và chúng có thể tạo ra các thành phần có thể tái sử dụng trong tương lai. Điều này làm tăng độ tin cậy của hệ thống tổng thể vì độ tin cậy của từng thành phần riêng lẻ nâng cao độ tin cậy của toàn bộ hệ thống thông qua việc tái sử dụng.

Có nhiều khuôn khổ thành phần tiêu chuẩn như COM/DCOM, JavaBean, EJB, CORBA, .NET, dịch vụ web và dịch vụ lưới. Các công nghệ thành phần mục tiêu này được áp dụng rộng rãi trong các thiết kế ứng dụng GUI cục bộ trên máy tính để bàn như các thành phần JavaBean đồ họa, các thành phần MS ActiveX và các thành phần COM có thể được sử dụng lại bằng cách kéo và thả. Nhiều thành phần là vô hình, đặc biệt là những thành phần được phân phối trong các ứng dụng doanh nghiệp và ứng dụng web Internet như Enterprise JavaBean (EJB), .NET và các thành phần CORBA. Sự kết hợp của các công nghệ thành phần và định hướng dịch vụ đang được chú ý nhiều hơn ngày nay; chúng bao gồm các dịch vụ web và lưới.

Một thành phần là một gói phần mềm có thể triển khai có thể cung cấp dịch vụ cho khách hàng của nó; bản thân nó cũng có thể yêu cầu các dịch vụ từ các thành phần khác. Một thành phần vẫn độc lập và có thể thay thế miễn là giao diện của nó không thay đổi.

Một thiết kế hướng thành phần thể hiện mức độ trừu tượng cao hơn so với thiết kế hướng đối tượng tương đương; ban đầu cần định nghĩa các thành phần và kết nối giữa chúng thay vì các lớp và kết nối giữa các lớp. Một thành phần là một khái niệm cấp cao hơn, thường kết hợp nhiều hơn một lớp. Do đó, trong thiết kế hướng thành phần, trước tiên chúng ta xác định tất cả các thành phần và giao diện của chúng thay vì xác định các lớp và mối quan hệ của chúng.

Kiến trúc và thiết kế phần mềm hướng thành phần có nhiều ưu điểm hơn so với các phần mềm hướng đối tượng truyền thống của chúng. Chúng bao gồm: giảm thời gian tạo ra hệ thống, giảm chi phí phát triển bằng cách tái sử dụng các thành phần hiện có và tăng độ tin cậy khi sử dụng lại các thành phần hiện có.

Thách thức là thiết kế các thành phần theo cách nào để làm cho nó có thể thích ứng với các thành phần hiện có để tái sử dụng chúng.

### **4.4.2 Thành phần (cấu phần) là gì?**

Có một sự thống nhất chung trong phát triển phần mềm hướng cấu phần rằng một thành là một đơn vị phần mềm độc lập có thể được cấu tạo với các thành phần khác để tạo ra một hệ thống phần mềm. Tuy nhiên, ngoài ra, mọi người đã đề xuất các định nghĩa khác nhau về một thành phần phần mềm. Council và Heineman (2001) định nghĩa một thành phần là:

*“Một thành phần phần mềm tuân theo một mô hình thành phần tiêu chuẩn và có thể được triển khai và cấu thành một cách độc lập mà không cần sửa đổi theo một tiêu chuẩn thành phần.”*

Định nghĩa này về cơ bản dựa trên các tiêu chuẩn để một đơn vị phần mềm phù hợp với các tiêu chuẩn này là một thành phần. Thành phần là một tập hợp mô-đun (cố kết), có thể triển khai - deployable (portable), có thể thay thế - replaceable (plug-and-play) và có thể tái sử dụng gồm các chức năng được xác định rõ, đóng gói việc triển khai và xuất nó dưới dạng giao diện cấp cao hơn.

Tuy nhiên, Szyperski (2002) không đề cập đến các tiêu chuẩn trong định nghĩa của mình về một thành phần mà thay vào đó tập trung vào các đặc điểm chính của các thành phần: *“Một thành phần phần mềm là một đơn vị cấu thành với các giao diện được chỉ định theo hợp đồng và chỉ phụ thuộc ngữ cảnh rõ ràng. Một thành phần phần mềm có thể được triển khai độc lập và chịu sự chi phối của các bên thứ ba”.*

Từ định nghĩa này, một cấu phần là một chương trình hay một tập hợp các chương trình có thể được biên dịch và thực thi. Đó là khép kín, do đó, nó cung cấp chức năng mạch lạc. Nó là tự triển khai để nó có thể được cài đặt và thực hiện trong một môi trường người dùng cuối. Nó có thể được lắp ráp với các thành phần khác để có thể được tái sử dụng như một đơn vị trong các ngữ cảnh khác nhau. Sự tích hợp được thực hiện thông qua giao diện của một cấu phần, có nghĩa là việc thực hiện nội bộ của một cấu phần thường được ẩn đối với người sử dụng.

Điểm chung của các định nghĩa trên là chúng đồng ý rằng các thành phần là độc lập và chúng là đơn vị cơ bản của thành phần trong một hệ thống. Theo quan điểm của Ian Sommerville, một định nghĩa tốt hơn về một thành phần có thể được rút ra bằng cách kết hợp các đề xuất này. Bảng dưới đây cho thấy những gì tôi coi là đặc điểm thiết yếu của một thành phần được sử dụng trong hệ thống phát triển phần mềm cấu phần.

|  |  |
| --- | --- |
| **Các đặc điểm của cấu phần** | **Mô tả** |
| Chuẩn hóa (Standardized) | Tiêu chuẩn hóa thành phần có nghĩa là một thành phần được sử dụng trong quy trình CBSE phải phù hợp với mô hình thành phần tiêu chuẩn. Mô hình này có thể xác định các giao diện thành phần, siêu dữ liệu thành phần, tài liệu, thành phần và triển khai. |
| Độc lập (Independent) | Một thành phần phải độc lập — có thể soạn và triển khai nó mà không cần phải sử dụng các thành phần cụ thể khác. Trong các tình huống mà thành phần cần các dịch vụ được cung cấp bên ngoài, các dịch vụ này phải được trình bày rõ ràng trong đặc tả giao diện ‘yêu cầu’. |
| Composable (có thể dùng một lần) | Để một thành phần có thể kết hợp, tất cả các tương tác bên ngoài phải diễn ra thông qua các giao diện được xác định công khai. Ngoài ra, nó phải cung cấp quyền truy cập bên ngoài vào thông tin về chính nó, chẳng hạn như các phương pháp và thuộc tính của nó. |
| Có thể triển khai (Deployable) | Để có thể triển khai, một thành phần phải được khép kín. Nó phải có thể hoạt động như một thực thể độc lập trên nền tảng thành phần cung cấp việc triển khai mô hình thành phần. Điều này thường có nghĩa là thành phần là nhị phân và không phải được biên dịch trước khi nó được triển khai. Nếu một thành phần được triển khai như một dịch vụ, nó không phải được triển khai bởi người dùng của một thành phần. Đúng hơn, nó được triển khai bởi nhà cung cấp dịch vụ. |
| Được ghi lại (Document) | Các thành phần phải được ghi lại đầy đủ để người dùng tiềm năng có thể quyết định xem các thành phần có đáp ứng nhu cầu của họ hay không. Cú pháp và lý tưởng nhất là ngữ nghĩa của tất cả các giao diện thành phần phải được chỉ định. |

Một cách hữu ích để nghĩ về một thành phần là nhà cung cấp một hoặc nhiều dịch vụ. Khi một hệ thống cần một dịch vụ, nó sẽ gọi một thành phần để cung cấp dịch vụ đó mà không cần quan tâm đến nơi mà thành phần đó đang thực thi hoặc ngôn ngữ lập trình được sử dụng để phát triển thành phần đó. Ví dụ, một thành phần trong hệ thống thư viện có thể cung cấp dịch vụ tìm kiếm cho phép người dùng tìm kiếm các danh mục thư viện khác nhau. Một thành phần chuyển đổi từ định dạng đồ họa này sang định dạng đồ họa khác (ví dụ: TIFF sang JPEG) cung cấp dịch vụ chuyển đổi dữ liệu, v.v.

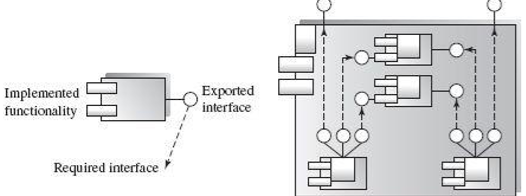
Việc xem một thành phần như một nhà cung cấp dịch vụ nhấn mạnh hai đặc điểm quan trọng của một thành phần có thể tái sử dụng:

1. Thành phần là một thực thể thực thi độc lập được xác định bởi các giao diện của nó. Bạn không cần bất kỳ kiến thức nào về mã nguồn của nó để sử dụng nó. Nó có thể được tham chiếu như một dịch vụ bên ngoài hoặc được bao gồm trực tiếp trong một chương trình.

2. Các dịch vụ được cung cấp bởi một thành phần được cung cấp thông qua một giao diện và tất cả các tương tác đều thông qua giao diện đó. Giao diện thành phần được thể hiện dưới dạng các hoạt động được tham số hóa và trạng thái bên trong của nó không bao giờ được tiết lộ.

 Trong mô hình chung trong Hình 4.17, một thành phần được biểu diễn bằng một hộp, bên trong là phần thực hiện của nó. Các hộp tối trên đường biên đại diện cho các phần tử giao diện được xuất và các "phích cắm" nhô ra thể hiện ngữ cảnh rõ ràng bắt buộc (thường được cắm vào giao diện của thành phần khác). Sơ đồ bên phải cho thấy cách một thành phần (lớn hơn) được thực hiện bằng cách kết nối các thành phần khác với nhau. Thành phần lớn hơn này có thể được kết nối với thành phần khác. Theo cách này, một “hệ thống” được xây dựng từ “mạng lưới các thành phần” bản thân nó cũng là một thành phần.

Một thành phần được biểu diễn trong ký hiệu UML 2.0 được thể hiện trong Hình 4.18.



Hình 4.17. Khái niệm chung về một thành phần và thành phần lồng nhau

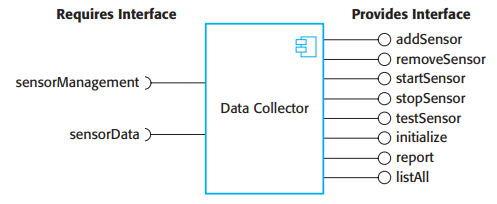


Hình 4.18. Ký hiệu thành phần trong UML

Các thành phần có hai giao diện liên quan, như trong Hình 7.1. Các giao diện này phản ánh các dịch vụ mà thành phần cung cấp và các dịch vụ mà thành phần yêu cầu để hoạt động chính xác:

* Giao diện ‘cung cấp’ xác định các dịch vụ được cung cấp bởi thành phần. Về cơ bản, giao diện này là API thành phần. Nó xác định các phương thức có thể được gọi bởi người dùng thành phần. Trong sơ đồ thành phần UML, giao diện ‘cung cấp’ cho một thành phần được biểu thị bằng một vòng tròn ở cuối dòng từ biểu tượng thành phần.
* Giao diện ‘request’ chỉ định những dịch vụ nào phải được cung cấp bởi các thành phần khác trong hệ thống nếu một thành phần hoạt động chính xác. Nếu những thứ này không có sẵn, thì thành phần sẽ không hoạt động. Điều này không ảnh hưởng đến tính độc lập hoặc khả năng triển khai của một thành phần bởi vì giao diện ‘yêu cầu’ không xác định cách thức cung cấp các dịch vụ này. Trong UML, biểu tượng cho giao diện ‘yêu cầu’ là một hình bán nguyệt ở cuối dòng từ biểu tượng thành phần. Lưu ý rằng các biểu tượng giao diện ‘cung cấp’ và ‘yêu cầu’ có thể khớp với nhau giống như một quả bóng và ổ cắm.

Để minh họa các giao diện này, Hình 4.19 cho thấy một mô hình của một thành phần đã được thiết kế để thu thập và đối chiếu thông tin từ một loạt các cảm biến. Nó chạy tự động để thu thập dữ liệu trong một khoảng thời gian và, theo yêu cầu, cung cấp dữ liệu được đối chiếu cho một thành phần đang gọi. Giao diện ‘cung cấp’ bao gồm các phương pháp thêm, xóa, bắt đầu, dừng và kiểm tra cảm biến. Phương thức báo cáo trả về dữ liệu cảm biến đã được thu thập và phương thức listAll cung cấp thông tin về các cảm biến được đính kèm. Mặc dù tôi không hiển thị điều này ở đây, nhưng các phương pháp này có các tham số liên quan chỉ định số nhận dạng cảm biến, vị trí, v.v.



Hình 4.19. Mô hình của thành phần thu thập dữ liệu

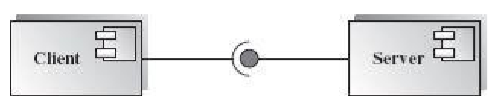
Giao diện ‘yêu cầu’ được sử dụng để kết nối thành phần với các cảm biến. Nó giả định rằng các cảm biến có giao diện dữ liệu, được truy cập thông qua sensorData và giao diện quản lý, được truy cập thông qua sensorManagement. Giao diện này được thiết kế để kết nối với các loại cảm biến khác nhau, do đó nó không bao gồm các hoạt động cảm biến cụ thể như Kiểm tra, cung cấp, đọc, v.v. Thay vào đó, các lệnh được sử dụng bởi một loại cảm biến cụ thể được nhúng trong một chuỗi, là tham số để các hoạt động trong giao diện 'yêu cầu'. Các thành phần bộ điều hợp phân tích cú pháp chuỗi này và dịch các lệnh được nhúng vào giao diện điều khiển cụ thể của từng loại cảm biến. Tôi thảo luận về việc sử dụng bộ điều hợp ở phần sau của chương này, nơi tôi trình bày cách thành phần thu thập dữ liệu được liên kết với cảm biến.

Một sự khác biệt quan trọng giữa một thành phần như một dịch vụ bên ngoài và một thành phần là một phần tử chương trình là các dịch vụ là các thực thể hoàn toàn độc lập. Chúng không có giao diện ‘yêu cầu’. Các chương trình khác nhau có thể sử dụng các dịch vụ này mà không cần thực hiện bất kỳ hỗ trợ bổ sung nào theo yêu cầu của dịch vụ.

 Một thành phần UML có thể được ánh xạ tới bất kỳ thành phần công nghệ đích nào. Một thành phần EJB được đóng gói trong tệp lưu trữ .JAR. Một thành phần web Java được đóng gói trong một tệp lưu trữ WAR. Một thành phần được phân phối từ xa được truy cập bằng lệnh "tra cứu" cho tên triển khai của thành phần. Thành phần .NET được triển khai dưới dạng tệp thư viện liên kết động .DLL được truy cập bằng chỉ thị “sử dụng” trong mã máy khách .NET. Một thành phần JavaBean được triển khai dưới dạng tệp JAR và được truy cập bằng lệnh “import <package>. \*” Trong mã máy khách Java.

**Tại sao phải sử dụng các thành phần**? Khả năng tái sử dụng, năng suất, khả năng tổng hợp, khả năng thích ứng, khả năng mở rộng, độ tin cậy và nhiều tính năng khác làm cho thiết kế và phát triển theo hướng thành phần rất hữu ích, đặc biệt là trong các khuôn khổ doanh nghiệp. Bởi vì một thành phần được phát triển để có thể tái sử dụng, nó phải có chức năng rõ ràng và độc lập. Các thành phần phải được thiết kế và phát triển để dễ dàng sử dụng lại. Do đó, các thành phần phải rất đáng tin cậy. Hơn thế, vì việc viết mã chi tiết được giảm thiểu đáng kể bằng cách áp dụng kiến ​​trúc thành phần, phương pháp này làm tăng năng suất trong quá trình phát triển phần mềm. Thay vì phát triển bằng lập trình, có thể xây dựng một thành phần mới hoặc phần mềm mới theo thành phần thành phần với các bộ phát triển. Nó phải dễ dàng thay thế hoặc điều chỉnh bất kỳ thành phần hiện có. Cũng có thể mở rộng khả năng phục vụ bằng cách thay thế các thành phần để tăng chức năng trong hệ thống.

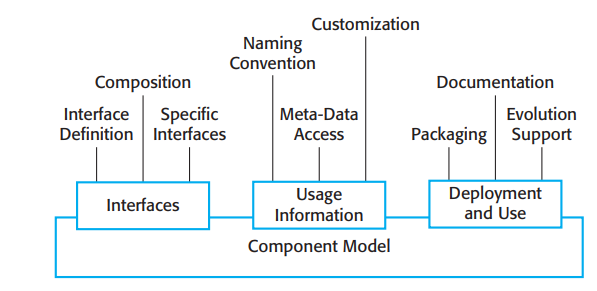
Sơ đồ trong Hình 4.20 cho thấy một cấu thành thành phần đơn giản trong đó thành phần khách yêu cầu (nhận) các dịch vụ từ một thành phần máy chủ thông qua cổng được cung cấp của máy chủ.



Hình 4.20. Component composition

### **4.4.3. Mô hình thành phần (component models)**

Mô hình thành phần là một định nghĩa của các tiêu chuẩn cho việc triển khai, tài liệu và triển khai thành phần. Các tiêu chuẩn này dành cho các nhà phát triển thành phần để đảm bảo rằng các thành phần có thể tương tác với nhau. Chúng cũng dành cho các nhà cung cấp cơ sở hạ tầng thực thi thành phần, những người cung cấp phần mềm trung gian để hỗ trợ hoạt động của thành phần. Nhiều mô hình thành phần đã được đề xuất, nhưng các mô hình quan trọng nhất hiện nay là mô hình WebServices, mô hình Sun’s Enterprise Java Beans (EJB) và mô hình .NET của Microsoft (Lau và Wang, 2007)



Hình 4.21 Các thành phần cơ bản của mô hình thành phần

Các yếu tố cơ bản của một mô hình thành phần lý tưởng được Weinreich và Sametinger (2001) thảo luận. Tôi tóm tắt các phần tử mô hình này trong Hình 4.21. Sơ đồ này cho thấy rằng các phần tử của một mô hình thành phần xác định các giao diện thành phần, thông tin mà bạn cần để sử dụng thành phần đó trong một chương trình và cách một thành phần sẽ được triển khai:

**1. Giao diện:** Các thành phần được xác định bằng cách chỉ định các giao diện của chúng. Mô hình thành phần chỉ định cách các giao diện nên được định nghĩa và các phần tử, chẳng hạn như tên hoạt động, tham số và ngoại lệ, cần được đưa vào định nghĩa giao diện. Mô hình cũng nên chỉ định ngôn ngữ được sử dụng để xác định các giao diện thành phần. EJB là Java cụ thể nên Java được sử dụng làm ngôn ngữ định nghĩa giao diện; trong .NET, các giao diện được định nghĩa bằng Ngôn ngữ Trung gian Chung (Common Intermediate Language - CIL). Một số mô hình thành phần yêu cầu các giao diện cụ thể phải được xác định bởi một thành phần. Chúng được sử dụng để tạo thành phần với cơ sở hạ tầng mô hình thành phần, cung cấp các dịch vụ tiêu chuẩn hóa như bảo mật và quản lý giao dịch

**2. Cách sử dụng** Để các thành phần được phân phối và truy cập từ xa, chúng cần phải có một tên hoặc xử lý duy nhất được liên kết với chúng. Tên này phải là duy nhất trên toàn cầu — ví dụ: trong EJB, tên phân cấp được tạo với tên gốc dựa trên tên miền Internet. Các dịch vụ có một URI (Uniform Resource Identifier - Định danh tài nguyên đồng nhất) duy nhất.

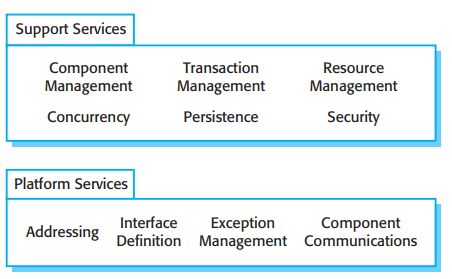
Siêu dữ liệu thành phần (component meta - data) là dữ liệu về chính thành phần đó, chẳng hạn như thông tin về các giao diện và thuộc tính của nó. Siêu dữ liệu rất quan trọng vì nó cho phép người dùng thành phần tìm hiểu những dịch vụ được cung cấp và yêu cầu. Việc triển khai mô hình thành phần thường bao gồm các cách cụ thể (chẳng hạn như sử dụng giao diện phản chiếu trong Java) để truy cập siêu dữ liệu thành phần này

Các thành phần là các thực thể chung và khi được triển khai, chúng phải được cấu hình để phù hợp với một hệ thống ứng dụng. Ví dụ: bạn có thể định cấu hình thành phần Bộ thu thập dữ liệu (Hình 4.19) bằng cách xác định số lượng cảm biến tối đa trong một mảng cảm biến. Do đó, mô hình thành phần có thể chỉ định cách các thành phần nhị phân có thể được tùy chỉnh cho một môi trường triển khai cụ thể.

**3. Triển khai** Mô hình thành phần bao gồm đặc tả về cách các thành phần nên được đóng gói để triển khai như các thực thể độc lập, có thể thực thi. Bởi vì các thành phần là các thực thể độc lập, chúng phải được đóng gói với tất cả phần mềm hỗ trợ không được cung cấp bởi cơ sở hạ tầng thành phần hoặc không được xác định trong giao diện ‘yêu cầu’. Thông tin triển khai bao gồm thông tin về nội dung của một gói và tổ chức nhị phân của nó.

Không thể tránh khỏi, khi các yêu cầu mới xuất hiện, các thành phần sẽ phải được thay đổi hoặc thay thế. Do đó, mô hình thành phần có thể bao gồm các quy tắc điều chỉnh khi nào và cách thức thay thế thành phần được phép. Cuối cùng, mô hình thành phần có thể xác định tài liệu thành phần cần được tạo ra. Điều này được sử dụng để tìm thành phần và quyết định xem nó có phù hợp không

Đối với các thành phần được triển khai dưới dạng đơn vị chương trình chứ không phải dịch vụ bên ngoài, mô hình thành phần đặt ra các dịch vụ được cung cấp bởi phần mềm trung gian hỗ trợ các thành phần thực thi. Weinreich và Sametinger (2001) sử dụng sự tương tự của một hệ điều hành để giải thích các mô hình thành phần. Hệ điều hành cung cấp một tập hợp các dịch vụ chung có thể được sử dụng bởi các ứng dụng. Việc triển khai mô hình thành phần cung cấp các dịch vụ được chia sẻ có thể so sánh được cho các thành phần. Hình 7.5 cho thấy một số dịch vụ có thể được cung cấp bởi việc triển khai một mô hình thành phần.



Hình 4.22. Các dịch vụ phần mềm trung gian được xác định trong một mô hình thành phần

Các dịch vụ được cung cấp bởi triển khai mô hình thành phần được chia thành hai loại:

1. Dịch vụ nền tảng, cho phép các thành phần giao tiếp và tương tác trong môi trường phân tán. Đây là những dịch vụ cơ bản phải có trong tất cả các hệ thống dựa trên thành phần.

2. Dịch vụ hỗ trợ, là những dịch vụ phổ biến có khả năng được yêu cầu bởi nhiều thành phần khác nhau. Ví dụ, nhiều thành phần yêu cầu xác thực để đảm bảo rằng người dùng của các dịch vụ thành phần được ủy quyền. Việc cung cấp một bộ tiêu chuẩn các dịch vụ phần mềm trung gian để tất cả các thành phần sử dụng. Điều này làm giảm chi phí phát triển thành phần và có thể tránh được sự không tương thích tiềm ẩn của thành phần.

Phần mềm trung gian thực hiện các dịch vụ thành phần và cung cấp giao diện cho các dịch vụ này. Để sử dụng các dịch vụ được cung cấp bởi cơ sở hạ tầng mô hình thành phần, bạn có thể coi các thành phần như được triển khai trong một ‘vùng chứa’. Vùng chứa là một triển khai các dịch vụ hỗ trợ cộng với định nghĩa về các giao diện mà một thành phần phải cung cấp để tích hợp nó với vùng chứa. Bao gồm thành phần trong vùng chứa có nghĩa là thành phần đó có thể truy cập các dịch vụ hỗ trợ và vùng chứa có thể truy cập các giao diện thành phần. hi được sử dụng, bản thân các giao diện thành phần không được truy cập trực tiếp bởi các thành phần khác, thay vào đó, chúng được truy cập thông qua giao diện vùng chứa gọi mã để truy cập giao diện của thành phần được nhúng.

Vùng chứa rất lớn và phức tạp, khi bạn triển khai một thành phần trong vùng chứa, bạn sẽ có quyền truy cập vào tất cả các dịch vụ phần mềm trung gian. Tuy nhiên, các thành phần đơn giản có thể không cần tất cả các tiện ích do phần mềm trung gian hỗ trợ cung cấp. Do đó, cách tiếp cận trong các dịch vụ web đối với việc cung cấp dịch vụ thông thường khá khác nhau. Đối với các dịch vụ web, các tiêu chuẩn đã được xác định cho các dịch vụ thông thường như quản lý giao dịch và bảo mật và các tiêu chuẩn này đã được thực hiện dưới dạng thư viện chương trình. Nếu bạn đang triển khai một thành phần dịch vụ, bạn chỉ sử dụng các dịch vụ phổ biến mà bạn cần.

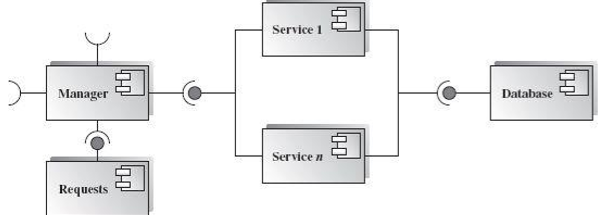
### **4.4.4 Nguyên tắc thiết kế dựa trên thành phần**

***4.4.4.1. Kết nối của các thành phần***

Hệ thống phần mềm được phân rã thành các đơn vị thành phần có thể tái sử dụng, gắn kết và đóng gói. Mỗi thành phần có giao diện riêng chỉ định các cổng yêu cầu và các cổng được cung cấp; mỗi thành phần ẩn việc triển khai chi tiết của nó. Nói cách khác, mỗi thành phần có thể được xem như một khối xây dựng hộp đen nhóm các chức năng và dữ liệu một cách gắn kết như một mô-đun (gói).

Các trình kết nối kết nối các thành phần, xác định và điều khiển tương tác của chúng. Tương tác thành phần có thể ở dạng lệnh gọi phương thức, lệnh gọi không đồng bộ như trình nghe (event listener) và đăng ký sự kiện (registrations), phát sóng (broadcasting), tương tác theo hướng thông điệp (message-driven interactions), truyền thông luồng dữ liệu (data stream communications) và các tương tác cụ thể của giao thức khác. Kiểu tương tác được xác định bởi các giao diện của các thành phần.

Hình 4.23 cho thấy một kiến ​​trúc phần mềm dựa trên thành phần. Thành phần trình quản lý được hỗ trợ dịch vụ từ một thành phần (yêu cầu) và một hệ thống con thành phần dịch vụ kết hợp (Dịch vụ 1 và Dịch vụ n), và thành phần sau nhận được dịch vụ truy cập dữ liệu từ cơ sở dữ liệu. Toàn bộ hệ thống phần mềm là một tập hợp các thành phần được kết nối bởi các giao diện của chúng. Mỗi thành phần có thể được thay thế hoặc cập nhật mà không có bất kỳ thay đổi nào của phần khác của hệ thống.



Hình 4.23. Kiến trúc dựa trên thành phần UML

Có ba loại kết nối:

* Kết nối Ball and cup (b2c hoặc c2b): kết nối hai thành phần bên trong hoặc hai thành phần bên ngoài
* Phân quyền Ball to ball (b2b) hoặc cup to cup (c2c): kết nối cổng ball (cup) bên ngoài với cổng ball (cup) bên trong
* Kết nối luồng I/O

Tất cả các kết nối trên có thể là lệnh gọi phương thức trực tiếp, lệnh gọi dựa trên sự kiện ngầm (gián tiếp) hoặc thông báo dựa trên thông điệp. Kết nối luồng thường được sử dụng cho các hoạt động kết nối mạng, đọc tệp và ghi tệp.

Kiến trúc phần mềm hướng thành phần là một phong cách kiến ​​trúc có thể được áp dụng cho cả thiết kế hệ thống và một phần hệ thống. Thiết kế phần mềm dựa trên thành phần là thiết kế theo hợp đồng. Hợp đồng chỉ định giao diện giữa thành phần và các máy khách của nó.

Để hỗ trợ khả năng tái sử dụng, một trong những tính năng quan trọng nhất của các thành phần, chúng ta có thể thay thế bất kỳ thành phần nào trong hệ thống nếu thành phần mới cung cấp không ít hơn những gì thành phần cũ cung cấp và không yêu cầu nhiều hơn những gì thành phần cũ yêu cầu.

Trong quá trình thiết kế, chúng tôi không chỉ xem xét việc tái sử dụng hoặc thích ứng các thành phần hiện có mà còn xem xét việc xây dựng các thành phần mới có thể tái sử dụng.

Các thành phần là các khối xây dựng giống như các lớp trong mô hình hướng đối tượng. Biểu đồ lớp là bản thiết kế trong thiết kế hướng đối tượng; các sơ đồ khác đề cập đến và tồn tại trong ngữ cảnh của cấu trúc lớp tổng thể này.

Trong cách tiếp cận thiết kế dựa trên thành phần, trước tiên cần phải xác định tất cả các thành phần (thay vì xác định các lớp trước). Trong hầu hết các trường hợp, một số lần lặp được yêu cầu để xác định tất cả các thành phần. Chúng ta cần tinh chỉnh từng bước thiết kế dựa trên thành phần để đáp ứng tất cả các yêu cầu, sử dụng tối đa các thành phần có sẵn và làm cho tất cả các thành phần mới cần thiết để tái sử dụng trong tương lai. Sau khi tất cả các thành phần được xác định và giao diện của chúng được thiết kế, chúng ta có thể tiếp tục thiết kế các lớp phù hợp với từng thành phần.

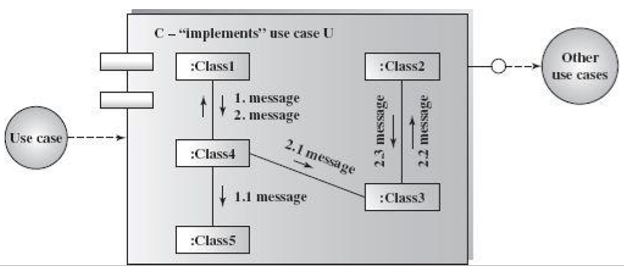
Chúng ta có thể thấy rằng thiết kế phần mềm dựa trên thành phần bắt đầu từ cấp độ trừu tượng hơn so với thiết kế OO

***4.4.4.2 Hướng dẫn thiết kế cấp thành phần***

Mô hình ca sử dụng và mô hình hóa khái niệm nghiệp vụ của dự án có sẵn trước giai đoạn thiết kế. Biểu đồ ca sử dụng mô tả các tương tác của người dùng với hệ thống chỉ ra tất cả các hoạt động cần thiết (giao diện) của thành phần. Chúng tôi có thể ánh xạ các sơ đồ ca sử dụng đến các giao diện dịch vụ được cung cấp của đặc tả thành phần đầu tiên.

Sơ đồ khái niệm nghiệp vụ mô tả các mối quan hệ của các thực thể quy trình nghiệp vụ trong miền. Chúng tôi có thể trích xuất các thực thể quy trình nghiệp vụ có thể tồn tại độc lập mà không có bất kỳ sự phụ thuộc liên quan nào vào các thực thể khác. Chúng ta có thể nhận ra và khám phá những thực thể độc lập này như những thành phần mới. Thành phần hiện tại cần các dịch vụ được cung cấp từ các thành phần mới được phát hiện này. Bằng cách này, chúng ta cũng có thể ánh xạ các sơ đồ khái niệm nghiệp vụ đến các giao diện dịch vụ cần thiết của đặc điểm kỹ thuật thành phần sơ lược.

Cộng tác là bất kỳ phần chức năng nào có thể nhận dạng được mô tả một dịch vụ có ý nghĩa liên quan đến một số khái niệm nói chung. Một sự hợp tác có thể được coi là việc thực hiện một ca sử dụng. Do đó, đối với mỗi ca sử dụng U, sẽ có một sơ đồ cộng tác “được gói gọn” trong một thành phần C (một hoặc nhiều biểu đồ trình tự sẽ được vẽ để thực hiện ca sử dụng thông qua các kịch bản). Hình 4.25 minh họa việc triển khai thành phần của ca sử dụng U (xảy ra ở giữa hai ca sử dụng khác).



Hình 4.25. Triển khai thành phần ca sử dụng

Ở bước thiết kế đầu tiên, chúng ta lấy được một thiết kế thành phần đầu tiên xác định số lượng thành phần. Chúng tôi biết những dịch vụ mà mỗi thành phần có thể cung cấp ở giai đoạn này. Trong bước tiếp theo, chúng ta cần tìm hiểu những dịch vụ mà mỗi thành phần yêu cầu để cung cấp các dịch vụ của riêng mình. Một số dịch vụ bắt buộc có thể có sẵn nhưng một số dịch vụ khác có thể không. Chúng tôi cần tinh chỉnh thiết kế, điều này có thể dẫn đến các thông số kỹ thuật thành phần mới và các cổng giao diện bổ sung cho các thành phần hiện có.

Sau hai bước này, chúng tôi nhận được một đặc tả thành phần mới với giao diện dịch vụ được cung cấp (từ bước 1) và các giao diện dịch vụ bắt buộc của nó (từ bước 2). Chúng tôi cũng nhận được các thành phần mới khác, sẽ cần được tinh chỉnh. Chúng ta cần lặp lại các bước này cho đến khi không còn thành phần mới nào nữa.

UML là một công cụ thiết kế tốt cho thiết kế phần mềm dựa trên thành phần. Để trình bày trình tự thời gian của hành vi hướng sự kiện, không chỉ cần có các sơ đồ thành phần và lớp mà còn cần các sơ đồ động như sơ đồ cộng tác trình tự. Một lớp có thể được sử dụng trong nhiều thành phần và một thành phần có thể cần nhiều lớp cộng tác. Một thành phần có thể được sử dụng lại nhiều lần trong hệ thống phần mềm và một hệ thống phần mềm cần nhiều thành phần hoạt động cùng nhau.

Chúng ta không cần phải giải quyết bất kỳ công nghệ thành phần mục tiêu cụ thể nào trong giai đoạn thiết kế vì kiến ​​trúc phần mềm thành phần có thể được ánh xạ tới một công nghệ mục tiêu sau đó trong giai đoạn triển khai.

### **4.4.5. Miền áp dụng và ưu/nhược điểm của kiến trúc dựa trên thành phần**

**Các miền áp dụng của kiến trúc dựa trên thành phần:**

* Các ứng dụng mà hợp đồng giao diện giữa các hệ thống con rõ ràng
* Các ứng dụng yêu cầu khớp nối lỏng lẻo giữa các thành phần và nơi có nhiều thành phần có thể tái sử dụng
* Thích hợp cho tổ chức hệ thống thư viện lớp (thư viện lớp .NET và API Java được xây dựng trong kiến trúc thành phần)

**Ưu/nhược điểm của kiến trúc thành phần**

**Ưu điểm**

* Khả năng tái sử dụng của các thành phần.
* Bảo trì và phát triển hệ thống; dễ dàng thay đổi và cập nhật việc thực hiện mà không ảnh hưởng đến phần còn lại của hệ thống.
* Tính độc lập và khả năng kết nối linh hoạt của các thành phần.
* Phát triển độc lập các thành phần của các nhóm khác nhau song song.
* Năng suất cho việc phát triển phần mềm hiện tại và tương lai.
* Nhiều công cụ thiết kế OO cũng có thể được sử dụng để phát triển phần mềm dựa trên thành phần.

**Hạn chế:**

* Có thể khó tìm thấy các thành phần sẵn có phù hợp để sử dụng lại.
* Sự thích nghi của các thành phần là một vấn đề.
* Rất ít công cụ thiết kế chuyên dụng hướng thành phần có sẵn.

### **4.4.6 Tóm tắt**

Kiến trúc phần mềm dựa trên thành phần là một phần mở rộng của kiến ​​trúc hướng đối tượng. Đơn vị cơ bản của nó là một thành phần trong định hướng thành phần thay vì các lớp trong hướng đối tượng.

Thành phần là một mức khái niệm trừu tượng cao hơn so với lớp. Một thành phần có thể bao gồm nhiều lớp liên quan được lưu trữ và triển khai trong cùng một tổ hợp nhị phân. Giao diện của một thành phần đóng vai trò hợp đồng giữa việc thực hiện và các khách hàng của nó; do đó cách tiếp cận còn được gọi là kiến ​​trúc phần mềm hướng hợp đồng. Bước đầu tiên trong thiết kế phần mềm dựa trên thành phần là xác định các thành phần đầu tiên và chỉ định tất cả các kết nối cần thiết giữa các thành phần này. Các kết nối được thực hiện thông qua giao diện của các thành phần.

Biểu đồ thành phần UML rất hữu ích để chỉ định cấu hình cấu trúc liên kết của các thành phần được kết nối. Biểu đồ UML động cũng cần thiết để mô tả các hành vi động của tập hợp các thành phần được kết nối.

Một kiến ​​trúc phần mềm dựa trên thành phần trừu tượng cuối cùng được ánh xạ tới một kiến ​​trúc thành phần công nghệ mục tiêu sẽ được thực hiện bởi công nghệ đó.

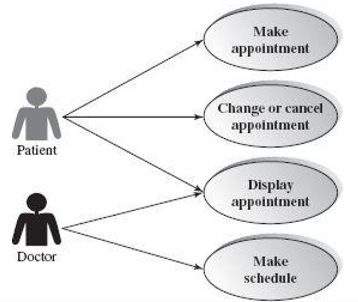
Thiết kế phần mềm dựa trên thành phần có nhiều lợi thế hơn thiết kế hướng đối tượng về các thuộc tính chất lượng của khả năng tái sử dụng, năng suất, độ tin cậy, khả năng mở rộng, tính di động và các thuộc tính khác. Các thách thức đối với việc áp dụng kiến ​​trúc dựa trên thành phần bao gồm sự thích ứng giữa các thành phần, xác định đơn vị thành phần và tìm kiếm các thành phần hiện có phù hợp.

### **4.6.7. Case study**

Bây giờ chúng ta hãy xem xét một miền vấn đề để xem cách áp dụng kiến ​​trúc phần mềm dựa trên thành phần trong thiết kế của nó. Phần mềm được sử dụng để quản lý các cuộc hẹn khám bệnh. Yêu cầu được mô tả như sau.

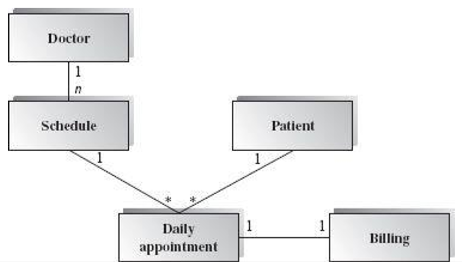
*Một phòng khám có hàng chục bác sĩ gia đình. Mỗi bác sĩ có một lịch trình hàng ngày với các khoảng thời gian 15 phút. Bệnh nhân có thể đặt lịch hẹn với bất kỳ bác sĩ nào. Bệnh nhân có thể thay đổi hoặc hủy bỏ cuộc hẹn. Bệnh nhân mới được chào đón. Tất cả các cuộc hẹn bao gồm tên bệnh nhân, tên bác sĩ, dữ liệu, khung thời gian, lý do và chính sách bảo hiểm. Bệnh nhân phải thanh toán bất kỳ khoản đồng thanh toán nào cho lần khám tại văn phòng vào thời điểm đăng ký và văn phòng kế toán sẽ lập hóa đơn cho phần còn lại của chi phí.*

Giả sử rằng chúng ta biết các yêu cầu hệ thống từ quá trình phân tích. Các yêu cầu hệ thống đi kèm với các tài liệu văn bản, sơ đồ UML như sơ đồ ca sử dụng và sơ đồ khái niệm nghiệp vụ. Biểu đồ lớp phân tích cũng có thể có sẵn. Biểu đồ ca sử dụng được hiển thị trong Hình 7.6 cung cấp mô tả một phần của kịch bản. Người đọc có thể trau chuốt thêm.



Hình 4.26. Biểu đồ UC cho hệ thống đặt hẹn phòng khám

Sơ đồ khái niệm quy trình nghiệp vụ thể hiện trong Hình 4.26 mô tả các tình huống có thể xảy ra. Mỗi bác sĩ có nhiều lịch trình hàng ngày; bất kỳ bác sĩ nào cũng có thể đánh dấu bất kỳ vị trí nào không còn trống trong lịch trình hàng ngày để bệnh nhân có thể chọn các vị trí có sẵn từ lịch trình khi đặt lịch hẹn. Lịch trình hàng ngày của bác sĩ bao gồm nhiều cuộc hẹn với các cá nhân khác nhau. Mỗi cuộc hẹn phải kết nối với hệ thống thanh toán. Một bệnh nhân có thể có nhiều nhất một cuộc hẹn hàng ngày. Sơ đồ này cho thấy các thực thể miền và các khái niệm nghiệp vụ.

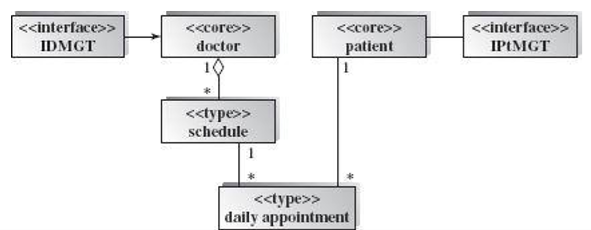


Hình 4.27. Biểu đồ khái niệm nghiệp vụ

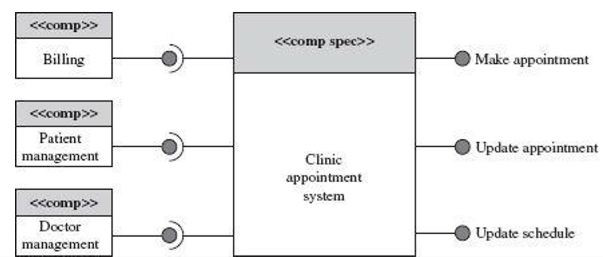
Thiết kế dựa trên thành phần tạo ra một giải pháp dự thảo để phát triển phần mềm dựa trên phân tích yêu cầu. Nó liên quan đến các lần lặp lại sàng lọc từ trên xuống. Ví dụ, kiến trúc thành phần cắt đầu tiên hiển thị tất cả các thành phần cấp cao nhất; sau đó mỗi lần lặp lại tinh chỉnh tiếp theo sẽ chia nhỏ một thành phần lớn và phức tạp thành một số thành phần con cho đến khi tất cả các phần tử thành phần phù hợp với một số thành phần có thể tái sử dụng hoặc phù hợp để được tạo lại.

Giả sử không có sẵn sơ đồ lớp. Chúng tôi tinh chỉnh sơ đồ khái niệm nghiệp vụ với quy trình sau.

Đầu tiên, xác định các thực thể độc lập, những thực thể có thể tồn tại một mình mà không phụ thuộc vào các thực thể khác (Cheesman và Daniels, 2001). Bác sĩ và bệnh nhân là những thực thể như vậy. Lịch trình thuộc về bác sĩ và bắt nguồn từ các cuộc hẹn; các cuộc hẹn hàng ngày được tạo ra từ lịch trình bệnh nhân-bác sĩ. Kí hiệu UML cho kiểu giao diện hoặc kiểu lõi là “<< >>”. Loại <<core>> có thể tồn tại độc lập và mỗi loại lõi có giao diện quản lý đối với loại lõi. Các loại khác là các loại có nguồn gốc không thể tồn tại độc lập. Chúng tôi tập trung vào các loại cốt lõi và hoàn thành các loại quản lý giao diện trước.



Hình 4.28. Xác định các thành phần

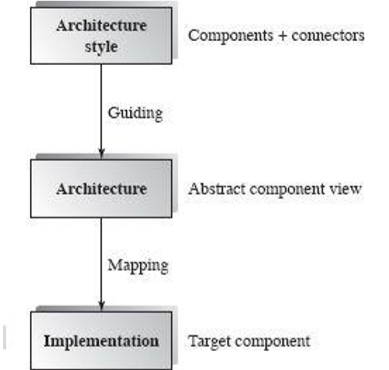


Hình 4.29 Lắt cắt đầu tiên của kiến trúc phần mềm dựa trên thành phần

Hình 4.29 mô tả phần đầu tiên của kiến ​​trúc phần mềm dựa trên thành phần cho miền vấn đề quy trình nghiệp vụ này. Có hai cấp thành phần: các cuộc hẹn tại phòng khám ở trên cùng và các thành phần quản lý thanh toán, bệnh nhân và bác sĩ ở cấp tiếp theo.

Mỗi thực thể kiểu lõi được chuyển đổi thành một phần tử thành phần cung cấp dịch vụ cho thành phần kiểm soát gốc. Mỗi ca sử dụng được chuyển đổi thành một nhà cung cấp giao diện để người dùng tương tác. Thanh toán có thể là một hệ thống khác, sau đó có thể được thiết kế theo cách tương tự: chia nhỏ thành nhiều thành phần con và thiết lập các giao diện tương ứng. Thành phần bệnh nhân và thành phần bác sĩ được sử dụng để quản lý dữ liệu liên tục. Bản thân thành phần hệ thống đặt lịch khám tại phòng khám có rất nhiều logic nghiệp vụ để quản lý. Nếu nó quá phức tạp, chúng tôi có thể tinh chỉnh nó theo các hướng dẫn tương tự như một hệ thống con riêng biệt.

Chúng ta có thể tiến hành thiết kế dựa trên thành phần từ trên xuống như hình 4.30. Chúng ta cũng có thể thiết kế kiến ​​trúc từ dưới lên nếu sơ đồ lớp có sẵn. Chúng ta có thể nhóm các lớp liên quan thành một thành phần và tìm ra giao diện của thành phần này. Đôi khi một thiết kế phần mềm dựa trên thành phần cần một thành phần keo để tích hợp nhiều thành phần vào một thành phần duy nhất để chúng có thể hoạt động cùng nhau.



Hình 4.30. Vòng đời

## **4.5. Kiến trúc phân tán**

### **4.5.1 Tổng quan**

Hệ thống phân tán là một tập hợp các thiết bị tính toán và lưu trữ được kết nối thông qua một mạng truyền thông. Trong loại hệ thống này, dữ liệu, phần mềm và người dùng được phân phối. Các hệ thống con hoặc các thành phần trong một hệ thống phân tán giao tiếp với nhau bằng một số phương thức bao gồm truyền thông điệp, gọi thủ tục từ xa và gọi phương thức từ xa. Hai yếu tố quan trọng của việc thiết kế một hệ thống phân tán là: cấu trúc liên kết mạng, cách mà các thực thể được tổ chức để tạo thành một mạng kết nối; và phương thức giao tiếp, phương thức mà các thành phần giao tiếp với nhau.

Nhiều hệ thống trong thế giới thực được phân phối tự nhiên. Các hệ thống này được sử dụng rộng rãi trong các môi trường doanh nghiệp lớn như hệ thống cơ sở dữ liệu cho phép truy cập dữ liệu từ xa, hệ thống B2B với các nhà cung cấp và khách hàng phân tán của nó, hoặc hệ thống SAP để quản lý các tài nguyên phân tán. Hệ thống phân tán ở khắp mọi nơi.

Một hệ thống phân tán có thể được mô hình hóa bởi kiến ​​trúc máy khách-máy chủ và điều này tạo cơ sở cho kiến ​​trúc nhiều tầng. Các lựa chọn thay thế là kiến ​​trúc môi giới như CORBA và kiến ​​trúc hướng dịch vụ (SOA) như dịch vụ web và dịch vụ lưới. Các đặc điểm chính của kiến ​​trúc phân tán là tính minh bạch của vị trí dịch vụ, độ tin cậy và tính khả dụng của dịch vụ. Ngoài ra, có một số khuôn khổ công nghệ để hỗ trợ kiến ​​trúc phân tán, bao gồm các dịch vụ web .NET, J2EE, CORBA, .NET, dịch vụ web AXIS Java và dịch vụ lưới GloBus.

Các nhà phát triển hệ thống phân tán phải tổ chức các thiết kế hệ thống của họ để tìm ra sự cân bằng giữa hiệu suất, độ tin cậy, bảo mật và khả năng quản lý của hệ thống. Không có mô hình chung nào tổ chức hệ thống phù hợp với mọi thuộc tính chất lượng nên nhiều kiểu kiến trúc phân tán đã xuất hiện, chúng ta nên chọn mẫu kiến trúc hỗ trợ các yêu cầu phi chức năng trong hệ thống của mình.

1. Kiến trúc chủ tớ (Master - slave) được sử dụng trong các hệ thống thời gian thực, trong đó yêu cầu thời gian phản hồi tương tác được đảm bảo
2. Kiến trúc 2 tầng Client – Server được sử dụng cho hệ thống máy chủ - máy khách đơn giản và trong tình huống cần tập trung hệ thống vì lý do bảo mật. Trong những trường hợp như vậy, giao tiếp giữa máy khách và máy chủ thường được mã hóa.
3. Kiến trúc đa máy khách – máy chủ được sử dụng khi máy chủ xử lý khối lượng giao dịch lớn.
4. Kiến trúc thành phần phân tán được sử dụng khi tài nguyên từ các hệ thống và cơ sở dữ liệu khác nhau cần được kết hợp hoặc làm mô hình triển khai cho hệ thống máy khách – máy chủ nhiều tầng.
5. Kiến trúc ngang hàng peer - to – peer được sử dụng khi các máy khách trao đổi thông tin được lưu trữ cục bộ và vai trò của máy chủ là giới thiệu các máy khách với nhau. Nó cũng có thể được sử dụng khi có thể phải thực hiện một số lượng lớn các phép tính độc lập.

Tiếp theo, chúng ta thảo luận về các chi tiết kỹ thuật của các kiểu kiến ​​trúc phân tán khác nhau bao gồm máy khách-máy chủ, nhiều tầng, nhà môi giới và kiến ​​trúc hướng dịch vụ.

### **4.5.2. Máy khách – chủ (Client – Server)**

Mô hình máy khách-máy chủ là kiến trúc hệ thống phân tán phổ biến nhất. Nó dựa trên hai quy trình giao tiếp, thường chạy trên các bộ xử lý khác nhau, và do đó phân tách một hệ thống thành hai hệ thống con chính: máy khách và máy chủ. Để máy chủ và máy khách có thể gửi yêu cầu và phản hồi lại yêu cầu thì cần có sự kết nối bằng môi trường mạng. Khách hàng có thể phải biết tên của các máy chủ có sẵn và các dịch vụ mà họ cung cấp. Tuy nhiên, máy chủ không cần biết danh tính của các máy khách hoặc có bao nhiêu máy khách đang truy cập dịch vụ của họ. Khách hàng truy cập các dịch vụ do máy chủ cung cấp thông qua các lệnh gọi thủ tục từ xa bằng giao thức trả lời yêu cầu như giao thức http được sử dụng trong WWW. Tiến trình đầu tiên, máy khách, đưa ra yêu cầu tới tiến trình thứ hai, máy chủ. Tiến trình máy chủ nhận yêu cầu (cung cấp dữ liệu từ cơ sở dữ liệu, in tài liệu), thực hiện và gửi trả lời cho máy khách.

**Ưu điểm:**

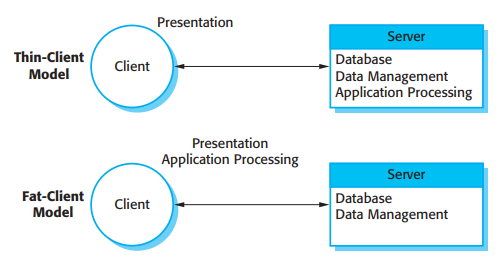
* Ưu điểm quan trọng nhất của mô hình máy khách-máy chủ là nó là một kiến trúc phân tán. Các máy chủ được phân phối trên mạng. Việc sử dụng hiệu quả có thể được thực hiện bởi các hệ thống nối mạng với nhiều bộ xử lý phân tán. Có thể dễ dàng thêm một máy chủ mới và tích hợp nó với phần còn lại của hệ thống hoặc nâng cấp máy chủ một cách minh bạch mà không ảnh hưởng đến các phần khác của hệ thống.
* Chức năng chung (ví dụ: dịch vụ in) có thể sẵn cho tất các các clients và không cần thực hiện bởi các dịch vụ.
* Khả năng tái sử dụng của các thành phần máy chủ

**Nhược điểm:**

* Thiếu cơ sở hạ tầng không đồng nhất để đối phó với những thay đổi yêu cầu
* Sự phức tạp về bảo mật
* Tính khả dụng và độ tin cậy của máy chủ
* Khả năng kiểm tra và khả năng mở rộng
* Fat client với cách trình bày và logic nghiệp vụ cùng nhau
* Định dạng truyền dữ liệu phải được thống nhất giữa các phép chuyển đổi giao tiếp. Mỗi phép chuyển đổi giao tiếp phải phân tích cú pháp đầu vào của nó và tác đầu ra của nó thành dạng 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 cấu trúc dữ liệu không tương thích.

Kiến trúc máy khách – máy chủ hai tầng là dạng kiến trúc máy khách – máy chủ đơn giản nhất. Hệ thống được triển khai như một máy chủ logic duy nhất cộng với một số lượng không xác định các máy khách sử dụng máy chủ đó.

Hình 4.31 hiển thị một kiến trúc máy khách-máy chủ hai tầng mẫu. Tầng front-end tập trung vào sự tương tác của người dùng và tầng back-end tập trung vào logic nghiệp vụ và quản lý cơ sở dữ liệu.



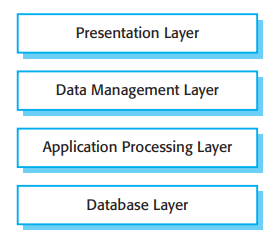
Hình 4.31. Kiến trúc thin and fat – client (2 tầng client – server)

Việc tách máy khách khỏi máy chủ dữ liệu giúp cho việc giải phóng máy khách khỏi truy vấn và quản lý dữ liệu (chẳng hạn như phát triển SQL) để nó hỗ trợ các phát triển song song cho các cấp tương ứng.

1. Mô hình thin-client, trong đó lớp trình bày được thực hiện trên máy khách và tất cả các lớp khác (quản lý dữ liệu, xử lý ứng dụng và cơ sở dữ liệu) được thực hiện trên máy chủ. Phần mềm máy khách có thể là một chương trình được viết đặc biệt trên máy khách để xử lý bản trình bày. Tuy nhiên, thường xuyên hơn, trình duyệt web trên máy khách được sử dụng để trình bày dữ liệu.
2. Mô hình fat-client, trong đó một số hoặc tất cả quá trình xử lý ứng dụng được thực hiện trên máy khách. Các chức năng quản lý dữ liệu và cơ sở dữ liệu được thực hiện trên máy chủ.

*Mô hình ứng dụng phân lớp (Hình 4.3.2) trong đó các lớp khác nhau trong hệ thống có thể thực thi trên các máy tính khác nhau. Ở đây hệ thống máy khách – chủ phụ thuộc vào việc có sự tách biệt rõ ràng giữa việc trình bày thông tin và các tính toán tạo ra và xử lý thông tin đó. Do đó, bạn nên thiết kế kiến trúc của các hệ thống máy khách – máy chủ phân tán để chúng được cấu trúc thành nhiều lớp logic, với các giao diện rõ ràng giữa các lớp này. Điều này cho phép mỗi lớp được phân bố trên một máy tính khác nhau. Hình 4.32 minh họa mô hình này, cho thấy một ứng dụng được cấu trúc thành bốn lớp:*

* *Lớp trình bày (presentation layer) liên quan tới trình bày thông tin cho người dùng và quản lý tất các các tương tác của người dùng;*
* *Lớp quản lý dữ liệu quản lý dữ liệu (data management layer) được chuyển đến và đi từ máy khách. Lớp này có thể thực hiện kiểm tra dữ liệu, tạo trang web, …*
* *Một lớp xử lý ứng dụng liên quan đến việc triển khai logic của ứng dụng và do đó cung cấp chức năng cần thiết cho người dùng cuối*
* *Lớp cơ sở dữ liệu lưu trữ dữ liệu và cung cấp các dịch vụ quản lý dịch vụ, …*

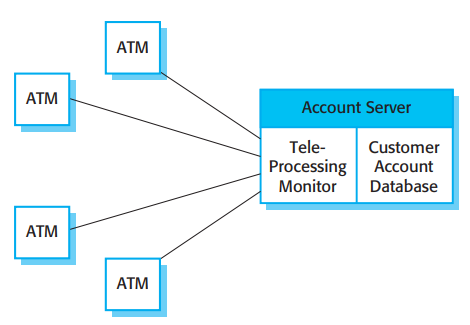


Hình 4.32. Mô hình kiến trúc phân lớp trong ứng dụng máy khách – máy chủ.

Ưu điểm của mô hình thin-client là dễ dàng quản lý các client. Đây là một vấn đề lớn nếu có một số lượng lớn khách hàng, vì có thể khó và tốn kém để cài đặt phần mềm mới cho tất cả họ. Nếu sử dụng trình duyệt web làm ứng dụng khách, không cần cài đặt bất kỳ phần mềm nào. Tuy nhiên, nhược điểm của phương pháp thin-client là nó có thể đặt một tải xử lý nặng lên cả máy chủ và mạng. Máy chủ chịu trách nhiệm về tất cả các tính toán và điều này có thể dẫn đến việc tạo ra lưu lượng mạng đáng kể giữa máy khách và máy chủ. Do đó, việc triển khai hệ thống sử dụng mô hình này có thể yêu cầu đầu tư thêm vào mạng và công suất máy chủ. Tuy nhiên, các trình duyệt có thể thực hiện một số xử lý cục bộ bằng cách thực thi các tập lệnh (ví dụ: Javascript) trong trang web được trình duyệt truy cập.

Mô hình fat-client sử dụng sức mạnh xử lý có sẵn trên máy tính đang chạy phần mềm máy khách và phân phối một số hoặc tất cả quá trình xử lý ứng dụng và bản trình bày cho máy khách. Máy chủ thực chất là một máy chủ giao dịch quản lý tất cả các giao dịch cơ sở dữ liệu. Quản lý dữ liệu đơn giản vì không cần quản lý sự tương tác giữa máy khách và hệ thống xử lý ứng dụng. Tất nhiên, vấn đề với mô hình fat-client là nó yêu cầu quản lý hệ thống bổ sung để triển khai và duy trì phần mềm trên máy tính khách.

Một ví dụ về tình huống trong đó kiến trúc fat-client được sử dụng là trong hệ thống ATM ngân hàng, hệ thống này chuyển tiền mặt và các dịch vụ ngân hàng khác cho người dùng. ATM là máy tính của khách hàng và máy chủ thường là một máy tính lớn chạy cơ sở dữ liệu tài khoản khách hàng. Máy tính lớn là một máy mạnh được thiết kế để xử lý giao dịch. Do đó, nó có thể xử lý khối lượng lớn các giao dịch được tạo ra bởi các máy ATM, hệ thống rút tiền khác và ngân hàng trực tuyến. Phần mềm trong máy rút tiền thực hiện rất nhiều xử lý liên quan đến khách hàng liên quan đến một giao dịch.



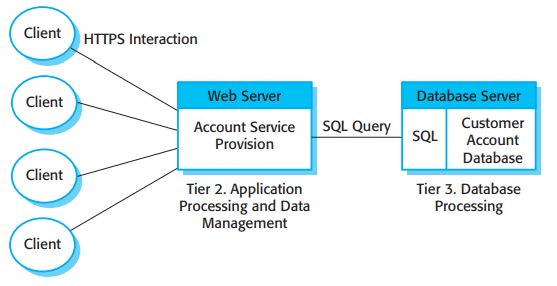
Hình 4.33. Kiến trúc fat – client cho một hệ thống ATM

Hình 4.33 cho thấy một phiên bản đơn giản của tổ chức hệ thống ATM. Lưu ý rằng các máy ATM không kết nối trực tiếp với cơ sở dữ liệu khách hàng, mà là kết nối với một màn hình xử lý từ xa. Màn hình điều khiển từ xa (TP) là một hệ thống phần mềm trung gian tổ chức giao tiếp với các khách hàng từ xa và tuần tự hóa các giao dịch của khách hàng để cơ sở dữ liệu xử lý. Điều này đảm bảo rằng các giao dịch là độc lập và không can thiệp vào nhau. Sử dụng các giao dịch nối tiếp có nghĩa là hệ thống có thể khôi phục từ các lỗi mà không làm hỏng dữ liệu hệ thống

Trong khi mô hình fat-client phân phối quá trình xử lý hiệu quả hơn so với mô hình đơn lẻ, việc quản lý hệ thống phức tạp hơn. Chức năng ứng dụng được trải rộng trên nhiều máy tính. Khi phần mềm ứng dụng phải được thay đổi, điều này liên quan đến việc cài đặt lại trên mọi máy khách. Đây có thể là một chi phí lớn nếu có hàng trăm khách hàng trong hệ thống. Hệ thống có thể phải được thiết kế để hỗ trợ nâng cấp phần mềm từ xa và có thể cần phải tắt tất cả các dịch vụ hệ thống cho đến khi phần mềm máy khách được thay thế

### **4.5.3. Kiến trúc khách chủ đa tầng (multi – tier client server)**

Vấn đề cơ bản với phương pháp tiếp cận máy chủ - máy khách hai tầng là các lớp logic trong hệ thống — trình bày, xử lý ứng dụng, quản lý dữ liệu và cơ sở dữ liệu - phải được ánh xạ vào hai hệ thống máy tính: máy khách và máy chủ. Điều này có thể dẫn đến các vấn đề về khả năng mở rộng và hiệu suất nếu mô hình khách hàng mỏng được chọn hoặc các vấn đề về quản lý hệ thống nếu mô hình fat-client được sử dụng. Để tránh một số vấn đề này, có thể sử dụng kiến trúc ‘máy khách – máy chủ nhiều tầng’. Trong kiến trúc này, các lớp khác nhau của hệ thống, cụ thể là trình bày, quản lý dữ liệu, xử lý ứng dụng và cơ sở dữ liệu, là các quy trình riêng biệt có thể thực thi trên các bộ xử lý khác nhau



Hình 4.34. Kiến trúc 3 tier cho hệ thống ngân hàng trực tuyến

Hệ thống ngân hàng Internet (Hình 4.34) là một ví dụ về kiến trúc máy khách-máy chủ nhiều tầng, trong đó có ba tầng trong hệ thống. Cơ sở dữ liệu khách hàng của ngân hàng (thường được lưu trữ trên máy tính lớn như đã thảo luận ở trên) cung cấp các dịch vụ cơ sở dữ liệu. Máy chủ web cung cấp các dịch vụ quản lý dữ liệu như tạo trang web và một số dịch vụ ứng dụng. Các dịch vụ ứng dụng như phương tiện chuyển tiền mặt, tạo bảng sao kê, thanh toán hóa đơn, v.v. được triển khai trong máy chủ web và dưới dạng các tập lệnh được thực thi bởi máy khách. Máy tính riêng của người dùng có trình duyệt Internet là máy khách. Hệ thống này có thể mở rộng vì tương đối dễ dàng thêm máy chủ (mở rộng quy mô) khi số lượng khách hàng tăng lên.

Trong trường hợp này, việc sử dụng kiến trúc ba tầng cho phép tối ưu hóa việc truyền thông tin giữa máy chủ web và máy chủ cơ sở dữ liệu. Giao tiếp giữa các hệ thống này có thể sử dụng các giao thức trao đổi dữ liệu cấp thấp, nhanh chóng.

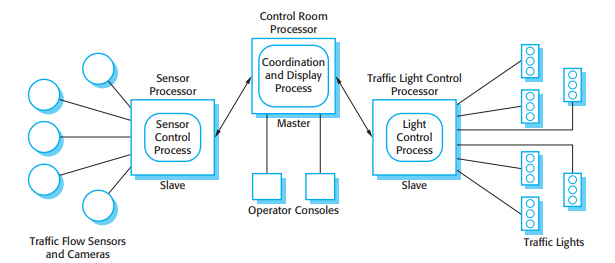
Phần mềm trung gian hiệu quả hỗ trợ truy vấn cơ sở dữ liệu trong SQL (Ngôn ngữ truy vấn có cấu trúc) được sử dụng để xử lý truy xuất thông tin từ cơ sở dữ liệu.

Lợi thế của đa tầng so với kiến ​​trúc hai tầng là tăng cường khả năng tái sử dụng và khả năng mở rộng bằng cách thêm tầng giữa. Mọi thay đổi liên quan đến nghiệp vụ chỉ được thực hiện đối với cấp giữa. Ví dụ: tầng giữa trong kiến ​​trúc ba tầng có thể có thiết kế và triển khai di động và không độc quyền. Tầng giữa cũng có thể cung cấp hỗ trợ đa luồng cho khả năng mở rộng. Kiến trúc nhiều tầng cũng làm giảm lưu lượng trên mạng. Nhược điểm chính của nó là khả năng kiểm tra do thiếu các công cụ kiểm tra nói chung. Thêm nhiều máy chủ trong hệ thống làm cho độ tin cậy và tính khả dụng của máy chủ càng trở nên quan trọng hơn.

### **4.5.4. Kiến trúc Master-slave (chủ - tớ)**

Kiến trúc chủ - tớ cho các hệ thống phân tán thường được sử dụng trong các hệ thống thời gian thực, nơi có thể có các bộ xử lý riêng biệt liên quan đến việc thu thập dữ liệu từ môi trường của hệ thống, xử lý dữ liệu, tính toán và quản lý thiết bị truyền động - là các thiết bị được điều khiển bởi hệ thống phần mềm, hoạt động để thay đổi môi trường của hệ thống. Ví dụ, một bộ truyền động có thể điều khiển một van và thay đổi trạng thái của nó từ 'mở' sang 'đóng'. Quy trình 'chính' thường chịu trách nhiệm tính toán, điều phối và giao tiếp và nó kiểm soát các quy trình 'phụ'. Quy trình ‘nô lệ’ dành riêng cho các hành động cụ thể, chẳng hạn như thu thập dữ liệu từ một loạt các cảm biến.

Hình 4.35 minh họa mô hình kiến trúc này. Nó là một mô hình của hệ thống kiểm soát giao thông trong một thành phố và có ba quy trình logic chạy trên các bộ xử lý riêng biệt. Quy trình tổng thể là quy trình phòng điều khiển, giao tiếp với các quy trình phụ riêng biệt chịu trách nhiệm thu thập dữ liệu giao thông và quản lý hoạt động của đèn giao thông



Hình 4.35. Hệ thống quản lý đèn giao thông với kiến trúc chủ - tớ

Một tập hợp các cảm biến phân tán thu thập thông tin về luồng giao thông. Quá trình điều khiển cảm biến thăm dò các cảm biến theo định kỳ để nắm bắt thông tin về luồng giao thông và đối chiếu thông tin này để xử lý thêm. Bản thân bộ xử lý cảm biến được thăm dò định kỳ để biết thông tin bởi quy trình chính có liên quan đến việc hiển thị trạng thái lưu lượng cho người vận hành, tính toán chuỗi đèn giao thông và chấp nhận lệnh của người vận hành để sửa đổi các chuỗi này. Hệ thống phòng điều khiển sẽ gửi các lệnh tới quy trình điều khiển đèn giao thông để chuyển các lệnh này thành tín hiệu để điều khiển phần cứng đèn giao thông. Hệ thống phòng điều khiển chính được tổ chức như một hệ thống máy khách - máy chủ, với các quy trình máy khách chạy trên bảng điều khiển của người vận hành.

Bạn sử dụng mô hình chủ-tớ này của hệ thống phân tán trong các tình huống mà bạn có thể dự đoán quá trình xử lý phân tán được yêu cầu và nơi quá trình xử lý có thể được bản địa hóa dễ dàng cho các bộ xử lý phụ. Tình huống này phổ biến trong các hệ thống thời gian thực, nơi điều quan trọng là phải đáp ứng thời hạn xử lý. Bộ xử lý nô lệ có thể được sử dụng cho các hoạt động tính toán chuyên sâu, chẳng hạn như xử lý tín hiệu và quản lý thiết bị do hệ thống điều khiển.

### **4.5.5. Kiến trúc ngang hàng (peer – to – peer)**

Mô hình tính toán máy khách-máy chủ đã thảo luận trong các phần trước của chương này phân biệt rõ ràng giữa máy chủ, là nhà cung cấp dịch vụ và máy khách, là máy sử dụng dịch vụ. Mô hình này thường dẫn đến sự phân bổ tải không đồng đều trên hệ thống, nơi các máy chủ hoạt động nhiều hơn các máy khách. Điều này có thể dẫn đến việc các tổ chức chi tiêu rất nhiều vào dung lượng máy chủ trong khi dung lượng xử lý không được sử dụng trên hàng trăm hoặc hàng nghìn PC được sử dụng để truy cập vào các máy chủ hệ thống.

Hệ thống ngang hàng (p2p) là hệ thống phi tập trung, trong đó các phép tính có thể được thực hiện bởi bất kỳ nút nào trên mạng. Về nguyên tắc, không có sự phân biệt nào được thực hiện giữa máy khách và máy chủ. Trong các ứng dụng ngang hàng, hệ thống tổng thể được thiết kế để tận dụng sức mạnh tính toán và khả năng lưu trữ có sẵn trên một mạng máy tính khổng lồ tiềm năng. Các tiêu chuẩn và giao thức cho phép truyền thông qua các nút được nhúng trong chính ứng dụng và mỗi nút phải chạy một bản sao của ứng dụng đó.

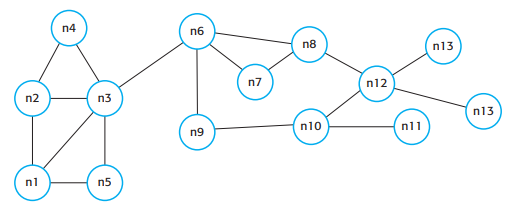
Công nghệ ngang hàng chủ yếu được sử dụng cho các hệ thống cá nhân hơn là doanh nghiệp (Oram, 2001). Ví dụ: các hệ thống chia sẻ tệp dựa trên giao thức Gnutella và BitTorrent được sử dụng để trao đổi tệp trên PC của người dùng. Các hệ thống nhắn tin tức thời như ICQ và Jabber cung cấp thông tin liên lạc trực tiếp giữa người dùng mà không cần máy chủ trung gian. SETI @ home là một dự án lâu dài nhằm xử lý dữ liệu từ kính thiên văn vô tuyến trên máy tính gia đình để tìm kiếm dấu hiệu về sự sống ngoài Trái đất. Freenet là một cơ sở dữ liệu phi tập trung được thiết kế để giúp việc xuất bản thông tin ẩn danh dễ dàng hơn và gây khó khăn cho các cơ quan chức năng trong việc ngăn chặn thông tin này. Dịch vụ điện thoại thoại qua IP (VOIP), chẳng hạn như Skype, dựa trên giao tiếp ngang hàng giữa các bên liên quan đến cuộc gọi điện thoại hoặc hội nghị.

Tuy nhiên, các hệ thống ngang hàng cũng đang được các doanh nghiệp sử dụng để khai thác sức mạnh trong mạng PC của họ (McDougall, 2000). Intel và Boeing đều đã triển khai các hệ thống p2p cho các ứng dụng tính toán chuyên sâu. Điều này tận dụng khả năng xử lý không sử dụng trên các máy tính cục bộ. Thay vì mua phần cứng hiệu suất cao đắt tiền, các tính toán kỹ thuật có thể được chạy qua đêm khi máy tính để bàn không được sử dụng. Các doanh nghiệp cũng sử dụng rộng rãi các hệ thống p2p thương mại, chẳng hạn như hệ thống nhắn tin và VOIP  
Sử dụng mô hình kiến trúc ngang hàng cho một hệ thống trong hai trường hợp:

1. Trường hợp hệ thống chuyên sâu về tính toán và có thể tách quá trình xử lý cần thiết thành một số lượng lớn các phép tính độc lập. Ví dụ: một hệ thống ngang hàng hỗ trợ tính toán khám phá thuốc phân phối các phép tính tìm kiếm phương pháp điều trị ung thư tiềm năng bằng cách phân tích một số lượng lớn các phân tử để xem liệu chúng có các đặc điểm cần thiết để ngăn chặn sự phát triển của ung thư hay không. Mỗi phân tử có thể được xem xét riêng biệt nên không cần các đồng đẳng trong hệ thống giao tiếp

2. Trường hợp hệ thống chủ yếu liên quan đến việc trao đổi thông tin giữa các máy tính riêng lẻ trên mạng và thông tin này không cần được lưu trữ hoặc quản lý tập trung. Ví dụ về các ứng dụng như vậy bao gồm hệ thống chia sẻ tệp cho phép đồng nghiệp trao đổi tệp cục bộ như tệp nhạc và video và hệ thống điện thoại hỗ trợ giao tiếp thoại và video giữa các máy tính

Về nguyên tắc, mọi nút trong mạng p2p có thể nhận biết được mọi nút khác. Các nút có thể kết nối và trao đổi dữ liệu trực tiếp với bất kỳ nút nào khác trong mạng. Tất nhiên, trong thực tế, điều này là không thể, vì vậy các nút được tổ chức thành các 'địa phương' với một số nút hoạt động như cầu nối với các địa phương nút khác. Hình 4.36 cho thấy kiến trúc p2p phi tập trung này.



Hình 4.36 Kiến trúc p2p phi tập trung

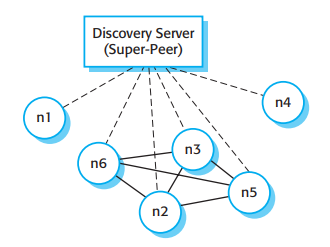
Trong một kiến trúc phi tập trung, các nút trong mạng không chỉ đơn giản là các phần tử chức năng mà còn là các thiết bị chuyển mạch truyền thông có thể định tuyến dữ liệu và điều khiển tín hiệu từ nút này sang nút khác. Ví dụ, giả sử rằng Hình 8.4 đại diện cho một hệ thống quản lý tài liệu, phi tập trung. Hệ thống này được sử dụng bởi một nhóm các nhà nghiên cứu để chia sẻ tài liệu và mỗi thành viên của nhóm sẽ duy trì kho tài liệu của riêng mình. Tuy nhiên, khi một tài liệu được truy xuất, nút truy xuất tài liệu đó cũng làm cho nó có sẵn cho nút khác

Nếu ai đó cần tài liệu được lưu trữ ở đâu đó trên mạng, họ sẽ đưa ra lệnh tìm kiếm, lệnh này sẽ được gửi đến các nút trong ‘địa phương’ của họ. Các nút này kiểm tra xem chúng có tài liệu hay không và nếu có, hãy trả lại cho người yêu cầu. Nếu họ không có nó, họ định tuyến tìm kiếm đến các nút khác. Do đó, nếu n1 thực hiện tìm kiếm tài liệu được lưu trữ tại n10, thì tìm kiếm này được chuyển qua các nút n3, n6 và n9 đến n10. Cuối cùng khi tài liệu được phát hiện, nút giữ tài liệu sau đó sẽ gửi trực tiếp đến nút yêu cầu bằng cách tạo kết nối ngang hàng

Kiến trúc phi tập trung này có ưu điểm là nó có tính dự phòng cao và do đó có khả năng chịu lỗi và chịu được các nút ngắt kết nối khỏi mạng. Tuy nhiên, nhược điểm ở đây là nhiều nút khác nhau có thể xử lý cùng một tìm kiếm và cũng có chi phí đáng kể trong giao tiếp ngang hàng được sao chép.

Một mô hình kiến trúc p2p thay thế, khác với kiến trúc p2p thuần túy, là một kiến trúc bán phân cấp, trong đó, trong mạng, một hoặc nhiều nút hoạt động như máy chủ để tạo điều kiện cho giao tiếp nút. Điều này làm giảm lưu lượng giữa các nút. Hình 4.37 minh họa mô hình này.

Trong kiến trúc bán tập trung, vai trò của máy chủ (đôi khi được gọi là superpeer) là giúp thiết lập liên hệ giữa các đồng nghiệp trong mạng hoặc điều phối kết quả của một phép tính. Ví dụ, nếu Hình 8.5 đại diện cho một hệ thống nhắn tin tức thời, thì các nút mạng giao tiếp với máy chủ (được biểu thị bằng các đường đứt nét) để tìm ra các nút khác có sẵn. Một khi các nút này được phát hiện, thông tin liên lạc trực tiếp có thể được thiết lập và kết nối đến máy chủ là không cần thiết. Do đó các nút n2, n3, n5 và n6 đang giao tiếp trực tiếp



Hình 4.37. Kiến trúc p2p bán tập trung

Trong một hệ thống p2p tính toán, nơi tính toán chuyên sâu của bộ xử lý được phân phối trên một số lượng lớn các nút, một số nút là siêu tốc là điều bình thường. Vai trò của họ là phân phối công việc cho các nút khác và đối chiếu và kiểm tra kết quả tính toán.

Kiến trúc ngang hàng cho phép sử dụng hiệu quả dung lượng trên toàn mạng. Tuy nhiên, những lo ngại chính đã hạn chế việc sử dụng chúng là các vấn đề về bảo mật và sự tin cậy. Giao tiếp ngang hàng liên quan đến việc mở máy tính của bạn để tương tác trực tiếp với các hệ thống khác và điều này có nghĩa là các hệ thống này, có khả năng truy cập bất kỳ tài nguyên nào của bạn. Để chống lại điều này, bạn cần tổ chức hệ thống của mình để các tài nguyên này được bảo vệ. Nếu điều này được thực hiện không chính xác, thì hệ thống của bạn có thể không an toàn

Vấn đề cũng có thể xảy ra khi các thiết bị ngang hàng trên mạng cố tình cư xử theo cách độc hại. Ví dụ, đã có trường hợp các công ty âm nhạc cho rằng bản quyền của họ đang bị lạm dụng đã cố tình đưa ra những lời ‘đầu độc’. Khi một thiết bị ngang hàng khác tải xuống thứ mà họ nghĩ là một bản nhạc, thì tệp thực tế được phân phối là phần mềm độc hại, có thể là phiên bản nhạc bị hỏng có chủ ý hoặc cảnh báo cho người dùng vi phạm bản quyền.

### **4.5.6. Tóm tắt**

Phần này thảo luận về kiến ​​trúc phần mềm cho các hệ thống máy tính phân tán, những hệ thống mà dữ liệu, người dùng và các phần tử xử lý đều được phân phối trên các trang web từ xa được kết nối bởi mạng.

Kiến trúc nhiều tầng phân phối và tách biệt dữ liệu và các nhiệm vụ xử lý qua các tầng khác nhau để mỗi tầng có trách nhiệm riêng. Nó làm giảm lưu lượng thông điệp trên mạng và tăng độ tin cậy của hệ thống.

Kiến trúc máy khách-máy chủ được sử dụng rộng rãi trong lĩnh vực kinh doanh và công nghiệp hiện tại. Máy chủ web, máy chủ dữ liệu và máy chủ ứng dụng là tất cả các ví dụ về các cấp máy chủ.

Một nhà môi giới, trong kiến ​​trúc nhà môi giới, có trách nhiệm môi giới các thông điệp giữa các thành phần từ xa hoặc các nhà môi giới khác để một hệ thống doanh nghiệp phức tạp có thể liên quan đến nhiều nhà môi giới, khách hàng và máy chủ.

## **4.6. Kiến trúc hướng dịch vụ (SOA)**

### **4.6.1. Tổng quan**

Sự phát triển của Web trong những năm 1990 đã cách mạng hóa việc trao đổi thông tin tổ chức. Các máy khách có thể truy cập thông tin trên các máy chủ từ xa bên ngoài tổ chức của chúng. Tuy nhiên, việc truy cập chỉ thông qua trình duyệt web và truy cập trực tiếp vào thông tin bằng các chương trình khác là không thực tế. Điều này có nghĩa là các kết nối cơ hội giữa các máy chủ, ví dụ, một chương trình đã truy vấn một số danh mục từ các nhà cung cấp khác nhau, là không thể.

Để giải quyết vấn đề này, khái niệm về một dịch vụ web đã được đề xuất. Sử dụng dịch vụ web, các tổ chức muốn cung cấp thông tin của họ cho các chương trình khác có thể truy cập bằng cách xác định (defining) và xuất bản (publishing) giao diện dịch vụ web. Giao diện này xác định dữ liệu có sẵn và cách nó có thể được truy cập. Nói chung, một dịch vụ web là một đại diện tiêu chuẩn cho một số tính toán hoặc thông tin tài nguyên có thể được sử dụng bởi các chương trình khác. Đây có thể là các nguồn thông tin, chẳng hạn như danh mục các phần; tài nguyên máy tính, chẳng hạn như một bộ xử lý chuyên dụng; hoặc tài nguyên lưu trữ. Ví dụ, một dịch vụ lưu trữ có thể được triển khai để lưu trữ vĩnh viễn và đáng tin cậy dữ liệu tổ chức mà theo luật, phải duy trì trong nhiều năm.

Dịch vụ web là một ví dụ của khái niệm chung hơn về dịch vụ, được định nghĩa (Lovelock và cộng sự, 1996) là:

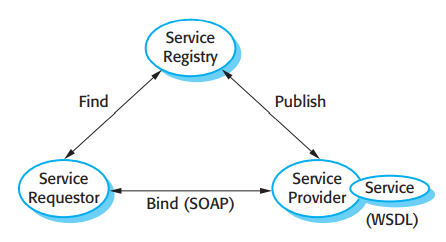
*“Một hành động hoặc màn trình diễn do một bên cung cấp cho bên khác. Mặc dù quy trình có thể gắn liền với một sản phẩm vật chất, nhưng việc thực hiện về cơ bản là vô hình và thường không dẫn đến quyền sở hữu bất kỳ yếu tố sản xuất nào”.*

Do đó, bản chất của dịch vụ là việc cung cấp dịch vụ độc lập với ứng dụng sử dụng dịch vụ (Turner et al., 2003). Các nhà cung cấp dịch vụ có thể phát triển các dịch vụ chuyên biệt và cung cấp những dịch vụ này cho nhiều người sử dụng dịch vụ từ các tổ chức khác nhau.

Kiến trúc hướng dịch vụ (SOA) là một cách phát triển hệ thống phân tán trong đó các thành phần hệ thống là các dịch vụ độc lập, thực thi trên các máy tính phân tán theo địa lý. Các giao thức dựa trên XML tiêu chuẩn, chẳng hạn như SOAP và WSDL, đã được thiết kế để hỗ trợ giao tiếp dịch vụ và trao đổi thông tin. Do đó, các dịch vụ độc lập với nền tảng và ngôn ngữ triển khai. Hệ thống phần mềm có thể được xây dựng bằng cách tổng hợp các dịch vụ cục bộ và dịch vụ bên ngoài từ các nhà cung cấp khác nhau, với sự tương tác liền mạch giữa các dịch vụ trong hệ thống

Kiến trúc hướng dịch vụ (SOA) bắt đầu với quy trình của doanh nghiệp. Trong bối cảnh này, dịch vụ là một chức năng nghiệp vụ được xác định rõ ràng, khép kín, độc lập với các dịch vụ khác, và được xuất bản và sẵn sàng sử dụng thông qua giao diện lập trình chuẩn. Phần mềm quản lý các quy trình nghiệp vụ thông qua SOA với các giao diện tiêu chuẩn, được xác định rõ ràng, có thể xây dựng, nâng cao và mở rộng cơ sở hạ tầng hiện có của chúng một cách linh hoạt hơn. Dịch vụ SOA có thể được tái sử dụng rộng rãi trong một miền hoặc dòng sản phẩm nhất định, ngay cả giữa các hệ thống cũ. Sự kết hợp lỏng lẻo của định hướng dịch vụ mang lại sự linh hoạt tuyệt vời cho các doanh nghiệp trong việc sử dụng tất cả các nguồn dịch vụ sẵn có bất kể các hạn chế về nền tảng và công nghệ.

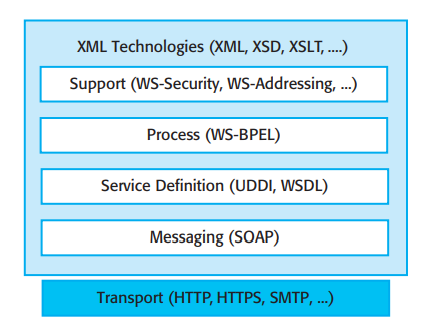
Hình 4.38 tóm tắt ý tưởng về SOA. Các nhà cung cấp dịch vụ thiết kế và triển khai các dịch vụ và chỉ định giao diện cho các dịch vụ này. Họ cũng xuất bản thông tin về các dịch vụ này trong một sổ đăng ký có thể truy cập được. Người yêu cầu dịch vụ (đôi khi được gọi là khách hàng dịch vụ) muốn sử dụng một dịch vụ sẽ khám phá thông số kỹ thuật của dịch vụ đó và tìm nhà cung cấp dịch vụ. Sau đó, họ có thể liên kết ứng dụng của mình với dịch vụ cụ thể đó và giao tiếp với nó, sử dụng các giao thức dịch vụ tiêu chuẩn



Hình 4.38. Kiến trúc hướng dịch vụ

Ngay từ đầu, đã có một quá trình tiêu chuẩn hóa tích cực cho SOA, làm việc cùng với các phát triển kỹ thuật. Tất cả các công ty phần cứng và phần mềm lớn đều cam kết tuân theo các tiêu chuẩn này. Do đó, SOA không gặp phải tình trạng không tương thích thường phát sinh với các cải tiến kỹ thuật, nơi các nhà cung cấp khác nhau duy trì phiên bản công nghệ độc quyền của họ.

Hình 4.39 cho thấy các tiêu chuẩn chính đã được thiết lập để hỗ trợ các dịch vụ web. Do sự tiêu chuẩn hóa ban đầu này, các vấn đề, chẳng hạn như nhiều mô hình thành phần không tương thích trong CBSE, được thảo luận trong chương Kiến trúc hướng thành phần, đã không phát sinh trong phát triển hệ thống hướng dịch vụ



Hình 4.39. Các chuẩn dịch vụ Web

Các giao thức dịch vụ web bao gồm tất cả các khía cạnh của SOA, từ các cơ chế cơ bản để trao đổi thông tin dịch vụ (SOAP) đến các tiêu chuẩn ngôn ngữ lập trình (WS-BPEL). Tất cả các tiêu chuẩn này đều dựa trên XML, một ký hiệu có thể đọc được của con người và máy móc cho phép định nghĩa dữ liệu có cấu trúc trong đó văn bản được gắn thẻ bằng một số nhận dạng có ý nghĩa. XML có một loạt các công nghệ hỗ trợ, chẳng hạn như XSD để định nghĩa lược đồ, được sử dụng để mở rộng và thao tác các mô tả XML. Erl (2004) cung cấp một bản tóm tắt tốt về các công nghệ XML và vai trò của chúng trong các dịch vụ web.

Tóm lại, các tiêu chuẩn chính cho SOA web như sau:

* SOAP: Đây là một tiêu chuẩn trao đổi thông điệp hỗ trợ giao tiếp giữa các dịch vụ. Nó xác định các thành phần thiết yếu và tùy chọn của thông điệp được truyền giữa các dịch vụ.
* WSDL: Ngôn ngữ Định nghĩa Dịch vụ Web (WSDL) là một tiêu chuẩn để định nghĩa giao diện dịch vụ. Nó quy định cách thức xác định các hoạt động dịch vụ (tên hoạt động, tham số và kiểu của chúng) và các ràng buộc dịch vụ.
* WS-BPEL: Đây là một tiêu chuẩn cho một ngôn ngữ dòng công việc được sử dụng để xác định các chương trình quy trình liên quan đến một số dịch vụ khác nhau.

Một chuẩn khám phá dịch vụ (UDDI) cũng được đề xuất nhưng điều này chưa được áp dụng rộng rãi. Tiêu chuẩ UDDI (Universal Description, Discovery, and Integration) xác định các thành phần của đặc tả dịch vụ, có thể được sử dụng để khám phá sự tồn tại của một dịch vụ. Chúng bao gồm thông tin về nhà cung cấp dịch vụ, các dịch vụ được cung cấp, vị trí của mô tả WSDL của giao diện dịch vụ và thông tin về các mối quan hệ kinh doanh. Mục đích của tiêu chuẩn này sẽ cho phép các công ty thiết lập các cơ quan đăng ký với các mô tả UDDI xác định các dịch vụ mà họ cung cấp.

Một số công ty, chẳng hạn như Microsoft, đã thiết lập các cơ quan đăng ký UDDI vào những năm đầu của thế kỷ 21 nhưng những công ty này hiện đã đóng cửa. Những cải tiến trong công nghệ công cụ tìm kiếm đã khiến chúng trở nên thừa thãi. Khám phá dịch vụ bằng cách sử dụng công cụ tìm kiếm tiêu chuẩn để tìm kiếm tiêu chuẩn để tìm kiếm các mô tả WSDL được nhận xét phù hợp hiện là phương pháp được ưa thích để khám phá các dịch vụ bên ngoài.

Các tiêu chuẩn SOA chính được hỗ trợ tập trung vào các khía cạnh chuyên biệt hơn của SOA. Có một số lượng lớn các tiêu chuẩn hỗ trợ cho chúng vì chúng nằm hỗ trợ SOA trong các loại ứng dụng doanh nghiệp khác nhau. Một ví dụ về các tiêu chuẩn này bao gồm:

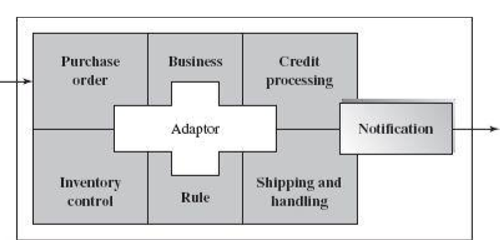
1. WS - Reliable Message: một tiêu chuẩn để trao đổi tin nhắn đảm bảo tin nhắn sẽ được gửi một lần mà một lần duy nhất
2. WS - Security: một tập hợp các tiêu chuẩn hỗ trợ bảo mật dịch vụ Web bao gồm các thẻ tiêu chuẩn nêu rõ định nghĩa về các chính sách và tiêu chuẩn bảo mật bao gồm chữ ký số.
3. WS - Addressing xác định cách thông tin địa chỉ sẽ được thể hiện trong một bản tin SOAP
4. Giao dịch WS xác định cách thức điều phối các giao dịch trên các dịch vụ phân tán

Các tiêu chuẩn dịch vụ Web là một chủ đề lớn và không có thời gian để thảo luận chi tiết về chúng ở đây. Trong cuốn sách của Erl (2004;2005) đã giới thiệu về các tiêu chuẩn này, chúng ta có thể tham khảo.

Các tiêu chuẩn dịch vụ web hiện tại đã bị chỉ trích là các tiêu chuẩn 'nặng ký' quá chung chung và không hiệu quả. Việc triển khai các tiêu chuẩn này đòi hỏi một lượng xử lý đáng kể để tạo, truyền và diễn giải các thông điệp XML liên quan. Vì lý do này, một số tổ chức, chẳng hạn như Amazon, sử dụng cách tiếp cận đơn giản hơn, hiệu quả hơn để truyền thông dịch vụ bằng cách sử dụng cái gọi là dịch vụ RESTful (Richardson và Ruby, 2007). Phương pháp RESTful hỗ trợ tương tác dịch vụ hiệu quả nhưng không hỗ trợ các tính năng cấp doanh nghiệp như WS- Reliability và WS-Transaction. Pautasso và cộng sự (2008) so sánh cách tiếp cận RESTful với các dịch vụ web được chuẩn hóa

Việc xây dựng ứng dụng dựa trên các dịch vụ cho phép các công ty và tổ chức khác hợp tác và sử dụng các chức năng nghiệp vụ của nhau. Do đó, các hệ thống liên quan đến việc trao đổi thông tin rộng rãi qua các ranh giới của công ty, chẳng hạn như hệ thống chuỗi cung ứng nơi một công ty đặt hàng từ một công ty khác, có thể dễ dàng được tự động hóa. Các ứng dụng dựa trên dịch vụ có thể được xây dựng bằng cách liên kết các dịch vụ từ các nhà cung cấp khác nhau bằng cách sử dụng ngôn ngữ lập trình chuẩn hoặc ngôn ngữ quy trình làm việc chuyên dụng

Hình 4.40 trình bày một ứng dụng dịch vụ web kinh doanh mua hàng sử dụng lại một số dịch vụ web có sẵn như đơn đặt hàng, xử lý tín dụng, kiểm soát hàng tồn kho, vận chuyển và cũng như xử lý logic nghiệp vụ của riêng nó. Chúng ta có thể xây dựng ứng dụng này theo chương trình bằng cách nhúng tất cả các dịch vụ web có sẵn này để giảm đáng kể chi phí và thời gian phát triển phần mềm. Việc triển khai nội bộ của các dịch vụ web đó (ví dụ: xử lý tín dụng và kiểm soát hàng tồn kho) có thể được thay đổi theo thời gian miễn là chúng bám vào giao diện công khai của chúng. Một tính năng tích cực khác là khả năng tái sử dụng của tất cả các dịch vụ web. Một số dịch vụ có thể được sử dụng lại bởi các ứng dụng khác mà chúng không được thiết kế ban đầu. Ví dụ: dịch vụ xử lý tín dụng có thể được bán cho một số công ty khác (ví dụ: một nhà tổ chức đám cưới) nếu họ muốn hỗ trợ thanh toán trực tuyến.

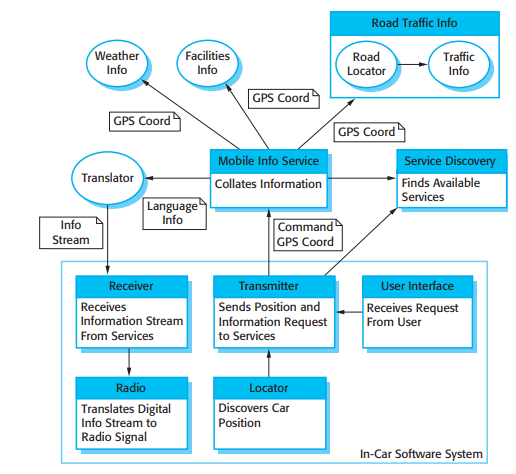


Hình 4.40 Thành phần dịch vụ

SOA là các kiến trúc được kết hợp lỏng lẻo nơi các ràng buộc dịch vụ có thể thay đổi trong quá trình thực thi. Điều này có nghĩa là một phiên bản dịch vụ khác, nhưng tương đương có thể được thực thi vào những thời điểm khác nhau. Một số hệ thống sẽ chỉ được xây dựng bằng cách sử dụng các dịch vụ web và những hệ thống khác sẽ kết hợp các dịch vụ web với các thành phần được phát triển cục bộ. Để minh họa cách các ứng dụng sử dụng hỗn hợp các dịch vụ và thành phần có thể được tổ chức, hãy xem xét tình huống sau:

Hệ thống thông tin trong ô tô cung cấp cho người lái xe thông tin về thời tiết, điều kiện giao thông đường bộ, thông tin địa phương, v.v. Điều này được liên kết với radio trên ô tô để thông tin được phân phối dưới dạng tín hiệu trên một kênh radio cụ thể. Xe được trang bị bộ thu tín hiệu GPS để phát hiện vị trí của nó và dựa trên vị trí đó, hệ thống truy cập một loạt các dịch vụ thông tin. Sau đó, thông tin có thể được cung cấp bằng ngôn ngữ được chỉ định của người lái xe.

Hình 4.41 minh họa một tổ chức khả thi cho một hệ thống như vậy. Phần mềm trên xe hơi bao gồm năm mô-đun. Các chức năng này xử lý thông tin liên lạc với người lái xe, với bộ thu GPS báo cáo vị trí của ô tô và với radio trên ô tô. Mô-đun Máy phát và Máy thu xử lý tất cả các giao tiếp với các dịch vụ bên ngoài.



Hình 4.41. Hệ thống thông tin trong xe hơi dựa trên dịch vụ

Chiếc xe giao tiếp với một dịch vụ thông tin di động (Mobile info Service) bên ngoài tổng hợp thông tin từ một loạt các dịch vụ khác, cung cấp thông tin về thông tin thời tiết giao thông và các phương tiện địa phương. Các nhà cung cấp khác nhau ở những nơi khác nhau cung cấp các dịch vụ này và hệ thống trên xe sử dụng dịch vụ khám phá để xác định các dịch vụ thông tin thích hợp và liên kết với chúng. Services Discovery cũng được dịch vụ thông tin di động (Mobile Info Services) sử dụng để liên kết với các dịch vụ thời tiết, giao thông và cơ sở vật chất thích hợp. Các dịch vụ trao đổi thông báo SOAP bao gồm thông tin vị trí GPS được các dịch vụ sử dụng để chọn thông tin thích hợp. Thông tin tổng hợp sau đó sẽ được gửi đến ô tô thông qua một dịch vụ dịch thông tin đó thành ngôn ngữ ưa thích của người lái xe

Ví dụ này minh họa một trong những ưu điểm chính của phương pháp hướng dịch vụ. Không cần thiết phải quyết định khi nào hệ thống được lập trình hoặc triển khai nhà cung cấp dịch vụ nào nên được sử dụng hoặc những dịch vụ cụ thể nào nên được truy cập.

Khi xe di chuyển xung quanh, phần mềm trên xe sử dụng dịch vụ khám phá dịch vụ để tìm dịch vụ thông tin thích hợp nhất và liên kết với dịch vụ đó. Do sử dụng dịch vụ dịch thuật, dịch vụ này có thể di chuyển xuyên biên giới và do đó cung cấp thông tin địa phương cho những người không nói ngôn ngữ địa phương.

Cách tiếp cận hướng dịch vụ đối với kỹ thuật phần mềm là một mô hình kỹ thuật phần mềm mới, là một sự phát triển quan trọng như kỹ thuật phần mềm hướng đối tượng. Sự thay đổi mô hình này sẽ được đẩy nhanh nhờ sự phát triển của ‘điện toán đám mây’ (Carr, 2009), nơi các dịch vụ được cung cấp trên cơ sở hạ tầng điện toán tiện ích do các nhà cung cấp lớn như Google và Amazon lưu trữ. Điều này đã và sẽ tiếp tục có những ảnh hưởng sâu sắc đến các sản phẩm hệ thống và quy trình kinh doanh.

Newcomer và Lomow (2005), trong cuốn sách của họ về SOA, tóm tắt tiềm năng của các phương pháp tiếp cận theo hướng dịch vụ:

*“Được thúc đẩy bởi sự hội tụ của các công nghệ quan trọng và sự áp dụng phổ biến của các dịch vụ Web, doanh nghiệp định hướng dịch vụ hứa hẹn sẽ cải thiện đáng kể sự nhanh nhạy của doanh nghiệp, tăng tốc thời gian đưa ra thị trường cho các sản phẩm và dịch vụ mới, giảm chi phí CNTT và cải thiện hiệu quả hoạt động.”*

**Ưu điểm của SOA:**

* Kết nối lỏng lẻo: Kết nối lỏng lẻo là thuộc tính quan trọng của kiến ​​trúc hướng dịch vụ. Mỗi thành phần dịch vụ là độc lập do tính năng dịch vụ không trạng thái. Việc triển khai một dịch vụ sẽ không ảnh hưởng đến ứng dụng của nó miễn là giao diện kết nối không bị thay đổi. Điều này làm cho phần mềm SOA dễ dàng phát triển và cập nhật hơn nhiều. Do đó cũng làm tăng linh hoạt và triển khai hệ thống
* Khả năng tương tác: Về mặt kỹ thuật, bất kỳ khách hàng hoặc dịch vụ nào đều có thể truy cập các dịch vụ khác bất kể nền tảng, công nghệ, nhà cung cấp hoặc triển khai ngôn ngữ của họ.
* Khả năng tái sử dụng: Bất kỳ dịch vụ nào cũng có thể được sử dụng lại bởi bất kỳ dịch vụ nào khác. Bởi vì khách hàng của một dịch vụ chỉ cần biết các giao diện công khai của nó, việc cấu thành và tích hợp dịch vụ trở nên dễ dàng hơn nhiều. Điều này làm cho việc phát triển ứng dụng kinh doanh dựa trên SOA hiệu quả hơn nhiều về thời gian và chi phí.
* Khả năng mở rộng: Các dịch vụ kết hợp lỏng lẻo rất dễ mở rộng. Các tính năng dịch vụ chi tiết thô, hướng tài liệu và không đồng bộ nâng cao thuộc tính khả năng mở rộng.
* Dễ dàng và nhanh chóng tạo ra các tiến trình nghiệp vụ từ các service đã có
* Thích ứng với những thay đổi trong tương lai
* Hỗ trợ đa thiết bị và đa nền tảng.

**Nhược điểm của SOA:**

* Khó khăn trong xác định, phân tích và thiết kế dịch vụ
* Khó khăn trong tích hợp các dịch vụ
* Băng thông mạng phải đảm bảo, do các dịch vụ giao tiếp và triệu gọi nhau thông qua hình thức gửi thông điệp XML

### **4.6.2. Dịch vụ như các thành phần có thể tái sử dụng**

Trong 4.5, đã giới thiệu về kỹ thuật phần mềm dựa trên thành phần (CBSE), trong đó hệ thống phần mềm được xây dựng bằng cách tạo ra các thành phần phần mềm dựa trên mô hình thành phần chuẩn. Dịch vụ là sự phát triển tự nhiên của các thành phần phần mềm mà về bản chất, mô hình thành phần là một tập hợp các tiêu chuẩn liên quan đến các dịch vụ web. Do đó, một dịch vụ có thể được định nghĩa như sau:

Một thành phần phần mềm được kết hợp lỏng lẻo, có thể tái sử dụng, đóng gói chức năng rời rạc, có thể được phân phối và truy cập theo chương trình. Dịch vụ web là một dịch vụ được truy cập bằng cách sử dụng Internet tiêu chuẩn và các giao thức dựa trên XML.

Sự khác biệt quan trọng giữa một dịch vụ và một thành phần phần mềm, như được định nghĩa trong CBSE, là các dịch vụ phải độc lập và kết hợp chặt chẽ với nhau; có nghĩa là, chúng phải luôn hoạt động theo cùng một cách, bất kể môi trường thực thi của chúng. Giao diện của chúng là giao diện ‘cung cấp’ cho phép truy cập vào chức năng dịch vụ. Các dịch vụ nhằm mục đích độc lập và có thể sử dụng được trong các ngữ cảnh khác nhau. Do đó, trong CBSE, chúng không có giao diện ‘yêu cầu’, xác định các thành phần hệ thống khác phải có mặt

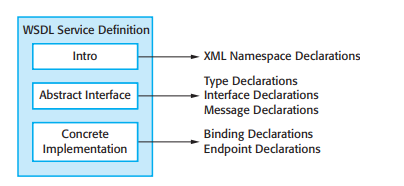
Các dịch vụ giao tiếp bằng cách trao đổi thông điệp, được thể hiện bằng XML và những thông điệp này được phân phối bằng cách sử dụng các giao thức truyền tải Internet tiêu chuẩn như HTTP và TCP / IP. Một dịch vụ xác định những gì nó cần từ một dịch vụ khác bằng cách đặt ra các yêu cầu của nó trong một tin nhắn và gửi đến dịch vụ đó. Dịch vụ nhận phân tích cú pháp tin nhắn, thực hiện tính toán và sau khi hoàn thành, gửi trả lời, dưới dạng tin nhắn, đến dịch vụ yêu cầu. Sau đó, dịch vụ này sẽ phân tích cú pháp trả lời để trích xuất thông tin cần thiết. Không giống như các thành phần phần mềm, các dịch vụ không sử dụng các lệnh gọi phương thức hoặc thủ tục từ xa để truy cập chức năng được liên kết với các dịch vụ khác

Khi bạn có ý định sử dụng một dịch vụ web, bạn cần biết vị trí của dịch vụ đó (URI của nó) và các chi tiết về giao diện của nó. Chúng được mô tả trong mô tả dịch vụ được thể hiện bằng ngôn ngữ dựa trên XML được gọi là WSDL. Đặc tả WSDL xác định ba điều về một dịch vụ web: dịch vụ này làm gì, cách nó giao tiếp và tìm nó ở đâu:

* Phần ‘cái gì’ của tài liệu WSDL, được gọi là giao diện, chỉ định những hoạt động mà dịch vụ hỗ trợ và xác định định dạng của thông báo được gửi và nhận bởi dịch vụ.
* Phần ‘cách thức’ của tài liệu WSDL, được gọi là ràng buộc, ánh xạ giao diện trừu tượng tới một tập hợp các giao thức cụ thể. Ràng buộc chỉ định các chi tiết kỹ thuật về cách giao tiếp với dịch vụ web.
* Phần ‘ở đâu’ của tài liệu WSDL mô tả vị trí triển khai dịch vụ web cụ thể (điểm cuối của nó).

Mô hình khái niệm WSDL (Hình 4.42) cho thấy các yếu tố của mô tả dịch vụ. Mỗi điều này được thể hiện bằng XML và có thể được cung cấp trong các tệp riêng biệt. Những phần này là:

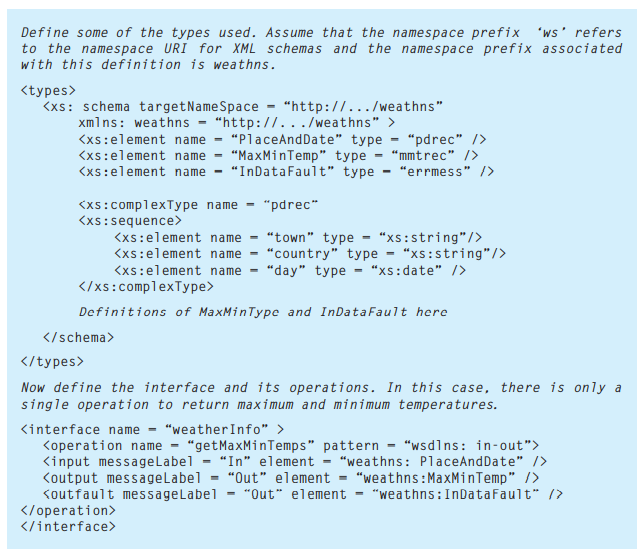
* Phần giới thiệu thường xác định không gian tên XML được sử dụng và có thể bao gồm phần tài liệu cung cấp thông tin bổ sung về dịch vụ.
* Mô tả tùy chọn về các kiểu được sử dụng trong các tin nhắn được dịch vụ trao đổi.
* Mô tả về giao diện dịch vụ; nghĩa là các hoạt động mà dịch vụ cung cấp cho các dịch vụ hoặc người dùng khác.
* Mô tả các thông báo đầu vào và đầu ra được xử lý bởi dịch vụ.
* Mô tả ràng buộc được sử dụng bởi dịch vụ (tức là giao thức nhắn tin sẽ được sử dụng để gửi và nhận tin nhắn). Mặc định là SOAP nhưng các ràng buộc khác cũng có thể được chỉ định. Liên kết đặt ra cách các thông điệp đầu vào và đầu ra được liên kết với dịch vụ sẽ được đóng gói thành một thông báo và chỉ định các giao thức truyền thông được sử dụng. Ràng buộc cũng có thể chỉ định cách đưa thông tin hỗ trợ, chẳng hạn như thông tin xác thực bảo mật hoặc số nhận dạng giao dịch.
* Thông số kỹ thuật điểm cuối là vị trí thực của dịch vụ, được thể hiện dưới dạng Mã định danh tài nguyên đồng nhất (URI) —địa chỉ của tài nguyên có thể được truy cập qua Internet



Hình 4.42 Tổ chức của một đặc tả WSDL

Các mô tả dịch vụ hoàn chỉnh, được viết bằng XML, dài, chi tiết và khó đọc. Chúng thường bao gồm các định nghĩa về không gian tên XML, là các định nghĩa cho tên. Định danh vùng tên có thể đứng trước bất kỳ số nhận dạng nào được sử dụng trong mô tả XML, giúp phân biệt giữa các số nhận dạng có cùng tên đã được xác định trong các phần khác nhau của mô tả XML. Bạn không cần phải hiểu chi tiết về không gian tên để hiểu các ví dụ ở đây. Bạn chỉ cần biết rằng các tên có thể được bắt đầu bằng mã định danh không gian tên và không gian tên: cặp tên phải là duy nhất

Các thông số kỹ thuật của WSDL hiện nay hiếm khi được viết bằng tay và hầu hết thông tin trong một thông số kỹ thuật có thể được tạo tự động. Bạn không cần phải biết chi tiết của một thông số kỹ thuật để hiểu các nguyên tắc của WSDL, vì vậy tôi tập trung vào mô tả của giao diện trừu tượng ở đây. Đây là một phần của đặc tả WSDL tương đương với giao diện ‘cung cấp’ của một thành phần phần mềm. Hình 4.43 cho thấy một phần của giao diện cho một dịch vụ đơn giản, với một ngày tháng và một địa điểm, được chỉ định là một thị trấn trong một quốc gia, trả về nhiệt độ tối đa và tối thiểu được ghi lại ở nơi đó vào ngày đó. Thông báo đầu vào cũng chỉ định xem các nhiệt độ này sẽ được trả lại bằng độ C hay độ F.



Hình 4.43 Một phần của mô tả WSDL cho một dịch vụ web

Phần thứ hai của mô tả cho thấy cách giao diện dịch vụ được xác định. Trong ví dụ này, dịch vụ weatherInfo có một hoạt động duy nhất, mặc dù không có hạn chế về số lượng hoạt động có thể được xác định. Hoạt động weatherInfo có một mẫu in out liên quan có nghĩa là nó nhận một thông báo đầu vào và tạo ra một thông báo đầu ra. Thông số kỹ thuật WSDL 2.0 cho phép một số mẫu trao đổi tin nhắn khác nhau như chỉ trong, vào-ra, chỉ-ra, vào-tuỳ-chọn-ra, ra-vào, v.v. Các thông báo đầu vào và đầu ra, đề cập đến các định nghĩa được đưa ra trước đó trong phần loại, sau đó được xác định.

Vấn đề chính với WSDL là định nghĩa của giao diện dịch vụ không bao gồm bất kỳ thông tin nào về ngữ nghĩa của dịch vụ hoặc các đặc điểm phi chức năng của nó, chẳng hạn như hiệu suất và độ tin cậy. Nó chỉ đơn giản là một mô tả về chữ ký dịch vụ (tức là các hoạt động và các tham số của chúng). Lập trình viên dự định sử dụng dịch vụ phải tìm ra dịch vụ thực sự làm gì và ý nghĩa của các trường khác nhau trong thông báo đầu vào và đầu ra. Hiệu suất và độ tin cậy phải được khám phá bằng cách thử nghiệm với dịch vụ. Tên và tài liệu có ý nghĩa giúp hiểu được chức năng được cung cấp nhưng người đọc vẫn có thể hiểu sai dịch vụ.

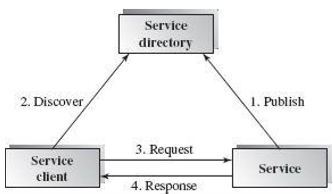
### **4.6.3 Triển khai SOA trong Dịch vụ Web**

Dịch vụ web (như trong Hình 4.45) là một dịch vụ giao tiếp với các dịch vụ hoặc máy khách khác thông qua các giao thức và công nghệ tiêu chuẩn như SOAP, XML và HTTP. Dịch vụ web là một dịch vụ hướng thông điệp có thể gửi các thông điệp hướng tài liệu cũng như các thông điệp RPC (Remote Procedure Call interfaces). Bởi vì một thông báo dựa trên XML được cấu trúc một cách hợp lý, nó làm cho một kiến trúc dịch vụ web có thể truy cập được và linh hoạt trên toàn cầu.

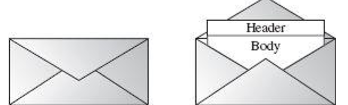
Hai tiêu chuẩn đặc tả chính cho kiến trúc dịch vụ web là Giao thức truy cập đối tượng đơn giản (SOAP) và Ngôn ngữ mô tả dịch vụ web (WSDL). SOAP (như trong Hình 4.46) cung cấp một định dạng trao đổi thông điệp chung giữa các máy khách và dịch vụ, và cả giữa các dịch vụ. Phong bì chỉ định một không gian tên XML và một kiểu mã hóa. Không gian tên XML chỉ định các tên có thể được sử dụng trong thông báo SOAP. Không gian tên XML phục vụ các mục đích tương tự như không gian tên .NET hoặc gói Java — chúng cho phép tránh xung đột tên.

Tiêu đề SOAP có thể bao gồm bảo mật, vị trí nguồn và đích và các phần tử khác có thể được sử dụng bởi một dịch vụ trung gian. Phần thân chứa phần chính của thông báo SOAP; nghĩa là phần dành cho người nhận cuối cùng của thông điệp SOAP.

Thông báo SOAP là một tài liệu dựa trên XML, độc lập với bất kỳ nền tảng nào và do đó có thể được vận chuyển bởi nhiều giao thức, chẳng hạn như HTTP hoặc SMTP.  
Sau đây là thông báo SOAP yêu cầu báo giá cổ phiếu. Phong bì SOAP <SOAP-ENV: Envelope> chứa phần tử <SOAP-ENV: Header> tùy chọn và các phần tử <SOAP-ENV: Body> bắt buộc. Rõ ràng, thông báo SOAP chỉ định một câu trích dẫn yêu cầu giá của “Cuộc đời và thời đại của tôi”.

’

Hình 4.45. Mô hình làm việc dịch vụ



Hình 4.46. SOAP envelope

<SOAP-ENV: Envelope xmlns:SOAP-ENV=

<http://schemas.xmlsoap.org/soap/envelope/>

SOAP-ENV:encodingStyle=

“http://schemas.xmlsoap.org/soap/encoding/”>

<SOAP-ENV:Header>

<from>a\_client</from>

<to>a\_target</to>

</SOAP-ENV:Header>

<SOAP-ENV:Body>

<m:GetQuote xmlns:m=“myURI”>

<name>My Life and Times</name>

</m: GetQuote>

</SOAP-ENV:Body> </SOAP-Envelope>

**Web Service Description Language (WSDL)**

Tài liệu WSDL là tài liệu giao diện mô tả giao diện của một dịch vụ web. Để truy cập một dịch vụ web, khách hàng phải biết điểm cuối của dịch vụ web một cách trực tiếp hoặc gián tiếp thông qua kho đăng ký dịch vụ web nơi dịch vụ Web đã đăng ký. Thông tin giao diện trong tài liệu WSDL giúp xây dựng các cuộc gọi có lập trình đến dịch vụ web.

Tài liệu WSDL mô tả một dịch vụ web như một tập hợp các “cổng” hoặc “điểm cuối” và chữ ký của các hành động được thực hiện bởi dịch vụ web. Các hành động được biểu thị bằng "phép toán", trong khi dữ liệu đối số được biểu thị bằng "thông báo". Tập hợp các hoạt động liên quan được gọi là “loại cổng”. Một ràng buộc chỉ định giao thức mạng và thông số kỹ thuật định dạng thông báo cho một loại cổng cụ thể. Một ràng buộc liên kết một cổng với một địa chỉ IP. Trước tiên, một ứng dụng khách dịch vụ web định vị tài liệu WSDL, tìm địa chỉ mạng và ràng buộc cho mỗi cổng, sau đó gọi các hoạt động của dịch vụ web này.

Đây là tài liệu WSDL cho dịch vụ web tìm kiếm giá chứng khoán trực tuyến, trong đó getStock là hoạt động của dịch vụ web này được khai báo trong WSDL.

<operation name=“getStock” …

Thông báo đầu vào là một ký hiệu và thông báo đầu ra là giá cho hoạt động này

<complexType>

<all>

<element name=“symbol” type=“string”/>

<element name=“price” type=“string”/>

</all>

Thẻ liên kết chỉ định kiểu tài liệu trong yêu cầu SOAP.

<soap:binding transport= “transport=http://schemas.xmlsoap.org/soap/http” . . .

Ngôn ngữ thực thi quy trình nghiệp vụ (Business Process Execution Language - BPEL) là ngôn ngữ mô tả dựa trên XML để tổ chức các dịch vụ web hiện có hoạt động cùng nhau. BPEL mô tả các dịch vụ web tham gia vào một quy trình, quy trình làm việc của các dịch vụ web này và tương tác giữa chúng. Ví dụ: mục nhập BPEL sau đây mô tả một trong các dịch vụ trong quy trình đặt hàng: Nó nhận đơn đặt hàng bằng thao tác gọi là buyOrder với loại cổng buyOrder chỉ định thông báo đầu vào và đầu ra của nó từ liên kết mua hàng của đối tác dịch vụ web tham gia. Sau khi nhận được đơn đặt hàng, BPEL tạo điều khiển luồng thành hai luồng đồng thời: makeInvoice và SchedShipping.

<sequence>

<receive partnerLink=“purchase”

portType=“lns:purchaseOrderPT”

operation=“PurchaseOrder”

variable=“PO”>

</receive>

<flow>

<invoke partnerLink=“invoicelnk”

portType=“lns:invoicePT”

operation=“makeInvoice”

/>

<invoke partnerLink=“shippinglnk”

portType=“lns:shipping” operation="scheduleShipping"

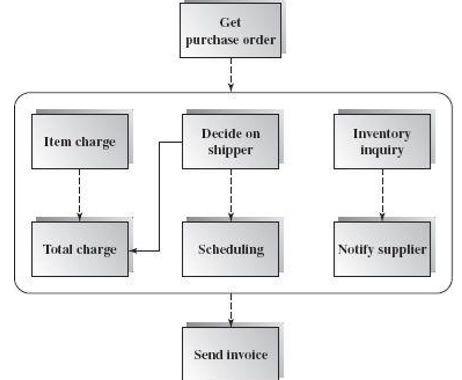
/>

</flow>

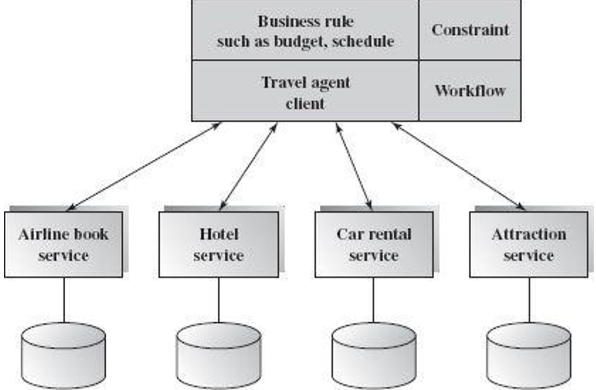
</sequence>

Hình 4.47 trình bày một đơn đặt hàng đơn giản khác làm ví dụ sử dụng thành phần BPEL. Có ba luồng song song dọc trong hộp giữa, cho biết rằng cả ba luồng được thực hiện song song. Dòng mũi tên liền nét cho biết quá trình thực thi tuần tự của dịch vụ web được kết nối từ trên xuống dưới. Đường mũi tên giữa các luồng chỉ ra các ràng buộc bởi các luồng khác trong quá trình thực thi đồng thời. Ví dụ, tổng phí không thể được xác định cho đến khi phí vận chuyển và xếp dỡ được xác định. Sơ đồ cung cấp một bức tranh rõ ràng về cách BPEL sử dụng các dịch vụ web để soạn một ứng dụng quy trình kinh doanh.

Hình 4.48 trình bày một ví dụ khác về hệ thống đại lý du lịch trực tuyến. Nó bao gồm bốn dịch vụ web hiện có: đặt vé máy bay, cho thuê xe hơi, đặt phòng khách sạn và đặt chỗ thu hút du khách. Với các quy tắc kinh doanh và chiến lược kinh doanh được tích hợp sẵn như ra quyết định và các ràng buộc về ngân sách và lịch trình, hệ thống sẽ có thể đưa ra các quyết định tối ưu cho phép khách hàng lựa chọn kết hợp các hãng hàng không và khách sạn. BPEL cũng có thể được sử dụng để xây dựng quy trình làm việc giữa các dịch vụ web này. Bản thân hệ thống này có thể cung cấp một dịch vụ web mới cho khách hàng sử dụng.



Hình 4.47 Các thành phần dịch vụ Web của BPEL

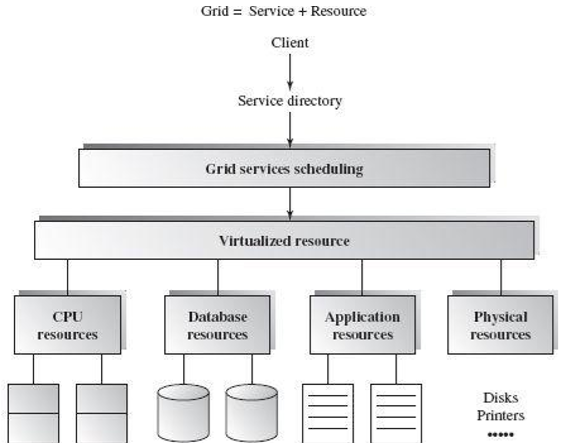


Hình 4.48. Mô hình xử lý nghiệp vụ dịch vụ Web

### **4.6.4 Triển khai SOA cho Điện toán Dịch vụ Lưới (Grid Service Computing)**

Tính toán dịch vụ lưới là một triển khai khác của SOA. Điện toán dịch vụ lưới sử dụng tất cả các tài nguyên khác nhau trải rộng trên các vị trí khác nhau thông qua mạng. Nó có thể cung cấp cả dịch vụ chương trình, chẳng hạn như dịch vụ web và dịch vụ phần cứng. Mỗi lưới có thể là lưới điện toán hỗ trợ tài nguyên tính toán hoặc lưới dữ liệu hỗ trợ tích hợp dữ liệu hoặc cả hai. Hình 4.49 cho thấy một kiến trúc dịch vụ lưới quản lý tất cả các tài nguyên như CPU, cơ sở dữ liệu, ứng dụng, lưu trữ và mạng, bằng cách quản lý tập thể và trực quan. Một bộ lập lịch dịch vụ lưới điều phối việc lập lịch cho tất cả việc chia sẻ tài nguyên.

Khi khách hàng gửi yêu cầu công việc, thư mục dịch vụ sẽ định vị dịch vụ và cho phép bộ lập lịch dịch vụ lưới chọn công việc phụ thuộc vào tính khả dụng của dịch vụ. Sau khi hoàn thành công việc, người dùng sẽ được hệ thống thông báo.



Hình 4.49. Kiến trúc dịch vụ lưới

# CHƯƠNG 5: QUY TRÌNH THIẾT KẾ KIẾN TRÚC

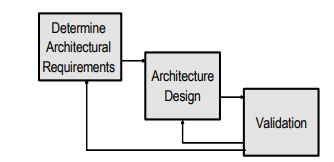
## **5.1. Tổng quan**

Vai trò của một kiến trúc sư không chỉ đơn giản là thực hiện một hoạt động thiết kế phần mềm. Kiến trúc sư thường phải:

* Làm việc với các bên liên quan khác nhau của ứng dụng: Kiến trúc sư đóng vai trò liên lạc quan trọng bằng cách đảm bảo rằng tất cả các nhu cầu của bên liên quan của ứng dụng được hiểu và kết hợp vào thiết kế. Ví dụ: ngoài các yêu cầu của người dùng doanh nghiệp đối với một ứng dụng, quản trị viên hệ thống sẽ yêu cầu ứng dụng đó có thể dễ dàng cài đặt, giám sát, quản lý và nâng cấp
* Dẫn dắt nhóm thiết kế kỹ thuật: xác định kiến trúc ứng dụng là một hoạt động thiết kế. Kiến trúc sư lãnh đạo một nhóm thiết kế, bao gồm các nhà thiết kế hệ thống (hoặc trên các dự án lớn, các kiến trúc sư khác) và các đầu mối kỹ thuật để tạo ra bản thiết kế kiến trúc
* Làm việc với quản lý dự án: Kiến trúc sư làm việc chặt chẽ với quản lý dự án, giúp lập kế hoạch dự án, ước tính và phân bổ nhiệm vụ và lập lịch thực hiện

Để hướng dẫn kiến trúc sư xác định kiến trúc ứng dụng, bạn nên tuân theo một quy trình kỹ thuật phần mềm đã xác định. Hình 5.1 cho thấy một quy trình kiến trúc lặp lại ba bước đơn giản có thể được sử dụng để hướng dẫn các hoạt động trong quá trình thiết kế. Tóm lại, ba bước là:

* Xác định các yêu cầu kiến trúc: Điều này liên quan đến việc tạo ra một tuyên bố hoặc mô hình của các yêu cầu sẽ thúc đẩy thiết kế kiến trúc.
* Thiết kế kiến trúc: Điều này liên quan đến việc xác định cấu trúc và trách nhiệm của các thành phần sẽ tạo nên kiến trúc.
* Xác thực: Điều này liên quan đến việc “kiểm tra” kiến trúc, thường bằng cách xem qua thiết kế, so với các yêu cầu hiện có và bất kỳ yêu cầu nào đã biết hoặc có thể có trong tương lai.



Hình 5.1. Sơ đồ ba bước của tiến trình thiết kế kiến trúc

Quá trình thiết kế kiến trúc này vốn là lặp đi lặp lại. Khi một thiết kế được đề xuất, việc xác nhận nó có thể cho thấy rằng thiết kees đó cần được sửa đổi hoặc các yêu cầu nhất định cần được xác định và hiểu rõ hơn. Cả hai điều này đều dẫn đến những cải tiến cho thiết kế, sau đó xác nhận…cho đến khi nhóm thiết kế hài lòng rằng các yêu cầu được đáp ứng.

Phần còn lại của chương này sẽ giải thích chi tiết hơn từng bước trong sơ đồ trên.

## **5.2 Quyết định các yêu cầu kiến trúc**

Trước khi có thể thiết kế một giải pháp kiến trúc, bạn cần phải có một ý tưởng khá tốt về các yêu cầu đối với kiến trúc ứng dụng. Các yêu cầu về kiến trúc, đôi khi còn gọi các yêu cầu thiết yếu về mặt kiến trúc hoặc các trường hợp sử dụng kiến trúc, về cơ bản là các yêu cầu về chức năng và chất lượng đối với một ứng dụng

### **5.2.1. Xác định các yêu cầu kiến trúc**

Hình 5.2. cho thấy các nguồn chính của các yêu cầu về kiến trúc chính là tài liệu đặc tả yêu cầu và các tài liệu khác mô tả yêu cầu khác nhau của các bên liên quan. Đầu ra của bước này là một tài liệu đưa ra các yêu cầu kiến trúc cho ứng dụng



Hình 5.2. Các đầu vào và đầu ra cho bước xác định yêu cầu kiến trúc

Xét ví dụ dưới đây: Một yêu cầu kiến trúc điển hình liên quan đến độ tin cậy của kết nối giữa các thành phần là:

“*Kết nối giữa các thành phần phải được đảm bảo thành công mà không bị mất thông điệp”*

Một số yêu cầu về kiến trúc thực sự là những ràng buộc, ví dụ:

*“Hệ thống phải sử dụng máy chủ web dựa trên IIS hiện có và sử dụng các trang máy chủ hoạt động (Active Server Page) để xử lý các yêu cầu web”*

Các ràng buộc đặt ra các hạn chế đối với kiến trúc và (hầu như luôn luôn) không thể thương lượng được. Chúng giới hạn phạm vi lựa chọn thiết kế mà một kiến trúc sư có thể thực hiện. Đôi khi điều này làm cho cuộc sống của một kiến trúc sư trở nên dễ dàng hơn nhưng đôi khi thì không. Bảng dưới đây liệt kê một số yêu cầu kiến trúc mẫu cùng với thuộc tính chất lượng mà chúng giải quyết

Bảng 5.1. Một vài ví dụ yêu cầu kiến trúc

|  |  |
| --- | --- |
| Thuộc tính chất lượng | Yêu cầu kiến trúc |
| Hiệu suất | Hiệu suất ứng dụng phải cung cấp thời gian phản hồi dưới bốn giây cho 90% yêu cầu |
| Bảo mật | Tất cả các thông tin liên lạc phải được xác thực và mã hoá bằng việc sử dụng |
| Quản lý tài nguyên | Thành phần máy chủ phải chạy trên một máy chủ văn phòng cấp thấp có bộ nhớ 512MB |
| Tính sử dụng được | Thành phần giao diện người dùng phải chạy trong trình duyệt Internet để hỗ trợ người dùng từ xa. |
| Tính sẵn dùng | Hệ thống phải chạy 24x7x365 với tính sẵn dùng tổng thể là 0,99 |
| Tính tin cậy | Không được phép mất tin nhắn và tất cả các tin nhắn gửi đi phải được biết trong 30 giây |
| Tính mở rộng | Ứng dụng phải có khả năng xử lý tải cao nhất là 500 người dùng đồng thời trong thời gian đăng ký |
| Khả năng sửa đổi | Kiến trúc phải hỗ trợ di chuyển theo từng giai đoạn từ phiên bản Ngôn ngữ thế hệ thứ 4 (4GL) hiện tại sang giải pháp công nghệ hệ thống .NET |

Bảng 5.2. Đưa ra một số ví dụ điển hình về các ràng buộc, cùng với nguồn của mỗi ràng buộc

Bảng 5.2. Một vài ví dụ ràng buộc

|  |  |
| --- | --- |
| Ràng buộc | Yêu cầu kiến trúc |
| Nghiệp vụ | Công nghệ phải này phải chạy dưới dạng một trình plug-in cho MS Biz Talk vì chúng tôi muốn bán phần mềm này Microsoft |
| Phát triển | Hệ thống phải được viết bằng Java để chúng tôi có thể sử dụng đội ngũ phát triển hiện có. |
| Lập lịch | Phiên bản đầu tiên của sản phẩm này phải được phân phối trong vòng sáu tháng |
| Nghiệp vụ | Chúng tôi muốn hợp tác chặt chẽ và phát triển hơn nữa tài trợ từ MegaHuge Tech Corp vì vậy chúng tôi cần sử dụng công nghệ trong ứng dụng của chúng tôi |

### **5.2.2. Ưu tiên các yêu cầu kiến trúc**

Có một điều hiếm khi xảy ra đó là tất cả các yêu cầu về kiến trúc cho một ứng dụng đều như nhau. Thông thường, danh sách các yêu cầu về kiến trúc chứa các mục có mức độ ưu tiên thấp hoặc “tính năng này có thì tốt, không có cũng không sao”. Do đó, điều quan trọng là phải xác định rõ ràng những điều này và xếp hạng các yêu cầu kiến trúc bằng cách sửu dụng mức độ ưu tiên. Ban đầu, thông thường chỉ cần phân bổ từng yêu cầu cho một trong ba danh mục, đó là:

1. Cao: ứng dụng phải hỗ trợ yêu cầu này. Những yêu cầu này định hướng thiết kế kiến trúc
2. Trung bình: yêu cầu này sẽ cần được hỗ trợ ở một số giai đoạn nhưng không nhất thiết phải có trong bản phát hành đầu tiên/tiếp theo;
3. Thấp: đây là một phần của danh sách mong muốn yêu cầu. Các giải pháp có thể đáp ứng các yêu cầu này là mong muốn, nhưng chúng không phải là định hướng của thiết kế.

Việc ưu tiên trở nên phức tạp khi đối mặt với các yêu cầu xung đột. Các ví dụ phổ biến là:

* Khả năng tái sử dụng của các thành phần trong giải pháp so với thời gian đưa ra thị trường nhanh chóng. Làm cho các thành phần được khái quát hoá và có thể tái sử dụng luôn mất thời gian và công sức hơn
* Chi phí tối tiểu cho các sản phẩm COTS so với giảm nỗ lực/chi phí phát triển. Các sản phẩm COTS có nghĩa là bạn phải phát triển ít mã hơn, nhưng chúng lại tốn tiền.

Không có giải pháp đơn giản nào cho những xung đột này. Công việc của kiến trúc sư là thảo luận những vấn đề này với các bên liên quan và đưa ra các tình huống giải pháp khả thi để giúp hiểu thấu đáo các vấn đề.

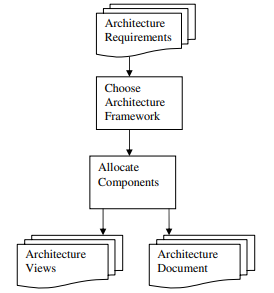
Các yêu cầu xung đột thậm chí có thể có cùng mức độ ưu tiên. Sau đó, trách nhiệm của giải pháp là cân nhắc những sự đánh đổi thích hợp và cố gắng tìm ra “ranh giới tốt” đáp ứng đầy đủ các hai yêu cầu mà không làm bất kỳ ai khó chịu hoặc có những hậu quả không mong muốn lớn đối với ứng dụng. Hãy nhớ rằng, kiến trúc sư giỏi biết cách nói “không”.

Trong một dự án có nhiều bên liên quan, thông thường là một ý tưởng hay để khiến từng nhóm bên liên quan ký tên vào mức độ ưu tiên này. Điều này đặc biệt đúng khi đối mặt với các yêu cầu trái ngược nhau. Khi điều này được đồng ý, thiết kế kiến trúc có thể bắt đầu.

## **5.3. Thiết kế kiến trúc**

Mặc dù tất cả các nhiệm vụ mà một kiến trúc sư thực hiện đều quan trọng, nhưng chất lượng của thiết kế kiến trúc mới thực sự quan trọng. Các tài liệu yêu cầu tuyệt vời và mạng lưới quan tâm của các bên liên quan chẳng có ý nghĩa gì nếu một thiết kế kém được tạo ra.

Không có gì ngạc nhiên khi thiết kế thường là nhiệm vụ khó khăn nhất mà một kiến trúc sư đảm nhận. Các kiến trúc sư giỏi dựa trên nhiều năm kinh nghiệm thiết kế và kỹ thuật phần mềm. Không có gì thay thế cho trải nghiệm này, vì vậy tất cả những gì chương này có thể làm là cố gắng giúp người đọc có được một số kiến thức cần thiết càng nhanh càng tốt.



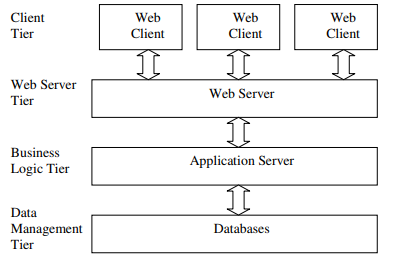
Hình 5.3 chỉ ra các đầu vào của bước thiết kế là yêu cầu kiến trúc. Bản thân giai đoạn thiết kế này có hai bước, có tính chất lặp đi lặp lại. Đầu tiên liên quan đến việc lựa chọn một chiến lược tổng thể cho kiến trúc, dựa trên các mẫu kiến trúc đã được chứng minh. Thứ hai liên quan đến việc chỉ định các thành phần riêng lẻ tạo nên ứng dụng, chỉ ra cách chúng phù hợp với khuôn khổ tổng thể và phân bổ trách nhiệm cho chúng. Các đầu ra là một tập hợp các khung nhìn kiến trúc nắm bắt thiết kế kiến trúc và một tài liệu thiết kế giải thích thiết kế, các lý do chính dẫn đến quyết định thiết kế chính và xác định các rủi ro liên quan đến việc thực hiện thiết kế.

### **5.3.1. Lựa chọn khung thiết kế**

Hầu hết các ứng dụng CNTT mà đã làm việc trong nhiều năm qua đều dựa trên một số lượng nhỏ các kiến trúc đã được chứng minh và hiểu rõ. Có một lý do chính đáng cho điều này – chúng hoạt động. Tận dụng các giải pháp đã biết sẽ giảm thiểu rủi ro ứng dụng bị lỗi do kiến trúc không phù hợp.

Vì vậy, bước thiết kế ban đầu liên quan đến việc lựa chọn một khung kiến trúc có vẻ như thoả mãn yêu cầu chính. Đối với các ứng dụng nhỏ, một mẫu kiến trúc đơn lẻ như máy khách – máy chủ (client – server), n – tầng có thể đủ. Đối với những ứng dụng phức tạp hơn, thiết kế sẽ kết hợp một hoặc nhiều mẫu đã biết, với kiến trúc sư chỉ định cách các mẫu này tích hợp để tạo thành kiến trúc tổng thể.

Không có công thức kỳ diệu nào để thiết kế khung kiến trúc. Tuy nhiên, điều kiện tiên quyết là phải hiểu cách mỗi mô hình kiến trúc chính giải quyết các thuộc tính chất lượng nhất định. Các phần phụ sau đây trình bày ngắn gọn một số mẫu chính được sử dụng và mô tả cách chúng giải quyết các yêu cầu chất lượng chung.



Hình 5.4. Ví dụ N – tier client server

***5.3.1.1. N – Tier Client Server***

Hình 5.4 cho thấy sơ đồ ứng dụng Web được minh hoạ. Các thuộc tính chính của mẫu này:

* Tách các mối quan tâm: tầng trình bày, tầng nghiệp vụ, tầng logic xử lý dữ liệu được phân chia rõ ràng theo các cấp khác nhau
* Đồng bộ giao tiếp: việc giao tiếp giữa các tầng là trả lời yêu cầu đồng bộ theo thứ tự. Các yêu cầu xuất phát theo một hướng từ khách hàng, thông qua Web và các cấp logic nghiệp vụ đến EIS. Mỗi tầng chờ phản hồi từ tầng kia trước khi tiếp tục.
* Triển khai linh hoạt: Không có hạn chế về cách một ứng dụng nhiều tầng được triển khai. Tất cả các cấp có thể chạy trên cùng một máy tính hoặc có thể mỗi tầng sẽ được triển khai trên các máy tính khác nhau. Trong các ứng dụng web, cấp tầng ứng dụng khác thường là trình duyệt chạy trên máy tính của người dùng, giao tiếp từ xa thông qua Internet với các thành phần tầng web.

Bảng 5. Các thuộc tính chất lượng cho mẫu N-tier client – server

|  |  |
| --- | --- |
| Thuộc tính chất lượng | Các vấn đề |
| Tính sẵn dùng | Máy chủ ở mỗi tầng có thể được sao chép, vì vậy nếu một không thành công, những người khác vẫn duy trì hoạt động. Nhìn chung, ứng dụng sẽ cung cấp chất lượng dịch vụ thấp hơn cho đến khi máy chủ bị lỗi được khôi phục |
| Xử lý thất bại | Nếu một máy khách đang giao tiếp với máy chủ bị lỗi, hầu hết các máy chủ web và ứng dụng sẽ triển khai chuyển đổi dự phòng chuyển vị. Điều này có nghĩa là một yêu cầu của khách hàng, mà máy chủ bị lỗi, yêu cầu này sẽ được chuyển hướng tới máy chủ bản sao để có thể đáp ứng yêu cầu. |
| Tính thay đổi | Việc tách biệt các mối quan tâm, giúp tăng cường khả năng sửa đổi vì tầng trình diễn, nghiệp vụ và quản lý dữ liệu đều được đóng gói rõ ràng. Mỗi tầng sẽ có thể có logic bên trong của nó được sửa đổi trong nhiều trường hợp mà không có những thay đổi xuất hiện ở tầng khác. |
| Hiệu suất | Kiến trúc này đã được chứng minh hiệu suất cao. Các vấn đề chính cần xem xét là số lượng luồng đồng thời được hỗ trợ trong mỗi máy chủ, tốc độ kết nối giữa các tầng và lượng dữ liệu được truyền. Cũng giống như các hệ thống phân tán, điều hợp lý là giảm thiểu các cuộc gọi cần thiết giữa các tầng để đáp ứng từng yêu cầu |
| Khả năng mở rộng | Vì các máy chủ trong mỗi tầng có thể được sao chép và nhiều phiên bản máy chủ hoạt động trên cùng một máy chủ hoặc các máy chủ khác nhau, kiến trúc sẽ mở rộng và tăng lên tốt. Trong thực tế, cấp quản lý dữ liệu thường trở thành điểm nghẽn về năng lực của hệ thống |

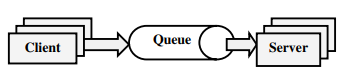
Bảng cho thấy các thuộc tính chất lượng chung có thể được giải quyết như thế nào với mẫu này. Cách giải quyết chính xác từng thuộc tính chất lượng phụ thuộc vào công nghệ máy chủ ứng dụng và web thực tế được sử dụng để triển khai ứng dụng. .NET, mỗi triển khai J2EE và các máy chủ ứng dụng độc quyền khác đều có các tính năng cụ thể khác nhau. Những điều này cần được hiểu trong quá trình thiết kế kiến trúc để không gặp phải những bất ngờ khó chịu sau này trong dự án, khi việc sửa chữa sẽ tốn kém hơn nhiều để thực hiện.

Mẫu N- tier Client Server thường được sử dụng và hỗ trợ trực tiếp từ các công nghệ máy chủ ứng dụng cho mẫu này làm cho việc triển khai các ứng dụng bằng mẫu này tương đối dễ dàng. Nói chung là thích hợp khi một ứng dụng phải hỗ trợ một số lượng lớn khách hàng tiềm năng và các yêu cầu đồng thời và mỗi yêu cầu mất một khoảng thời gian tương đối ngắn (vài miligiay) để xử lý.

***5.3.1.2. Thông điệp***

Trong hình 5.5, các thành phần cơ bản của thông điệp được hiển thị. Các thuộc tính chính của mẫu này là:

* Truyền thông không đồng bộ: khách hàng gửi yêu cầu đến hàng đợi, nơi thông điệp được lưu trữ cho đến khi một ứng dụng xoá nó. Sau đó, khách hàng đã viết thông điệp vào hàng đợi, nó tiếp tục mà không cần đợi tin nhắn bị xoá
* QoS có thể được cấu hình: hàng đợi có thể được cấu hình để phân phối tốc độ cao, không đáng tin cậy hoặc chậm hơn, đáng tin cậy. Các hoạt động hàng đợi có thể được phối hợp với các giao dịch cơ sở dữ liệu
* Kết nối lỏng lẻo: Không có ràng buộc trực tiếp giữa máy khách và máy chủ. Máy khách không biết máy chủ nào nhận được thông báo. Máy chủ không biết thông điệp đến từ máy khách nào?



Hình 5.5. Sơ đồ mẫu thông điệp

Bảng 5 cho thấy các thuộc tính chất lượng chung được giải quyết như thế nào bằng cách gửi thông điệp. Một lần nữa, hãy nhớ rằng, hỗ trợ chính xác các thuộc tính chất lượng này phụ thuộc vào thông điệp

Bảng 5. Các thuộc tính chất lượng cho Mẫu thông điệp

|  |  |
| --- | --- |
| Thuộc tính chất lượng | Các vấn đề |
| Tính sẵn dùng | Các hàng đợi vật lý có cùng tên logic có thể được sắp xếp lại trên các phiên bản máy chủ thông điệp khác nhau. Khi không thành công, khách hàng có thể gửi thông điệp đến hàng đợi bản sao |
| Xử lý lỗi | Nếu một máy khách đang giao tiếp với một hàng đợi bị lỗi, nó có thể tìm thấy một hàng đợi bản sao và đăng thông điệp ở đó. |
| Khả năng sửa đổi | Thông điệp vốn dĩ được kết hợp với nhau một cách lỏng lẻo, và tính năng chuyênn nghiệp này mang lại khả năng sửa đổi cao vì máy khách và máy chủ không bị ràng buộc trực tiếp qua một giao diện. Những thay đổi đối với định dạng thông điệp có thể gây ra những thay đổi đối với việc triển khai máy chủ. Các định dạng thông điệp tự mô tả, có thể khám phá có thể giúp giảm sự phụ thuộc này vào các định dạng thông điệp |
| Hiệu suất | Công nghệ hàng đợi thông điệp có thể gửi hàng nghìn tin nhắn mỗi giây. Thông điệp không đáng tin cậy nhanh hơn đáng tin cậy, với sự khác biệt phụ thuộc vào chất lượng của công nghệ thông điệp được sử dụng |
| Khả năng mở rộng | Hàng đợi có thể được lưu trữ trên các điểm cuối giao tiếp, hoặc được sao chép trên các cụm máy chủ thông điệp được lưu trữ trên một hoặc nhiều máy chủ. Điều này làm cho thông điệp trở thành một giải pháp có khả năng mở rộng cao |

Các ứng dụng có thể chia việc xử lý một yêu cầu thành một số bước rời rạc, được kết nối bằng hàng đợi, là một phần mở rộng cơ bản của mẫu thông điệp đơn giản. Điều này giống với mẫu “Ống và bộ lọc – Pipe and Filter”

### **5.3.2. Phân bổ các thành phần**

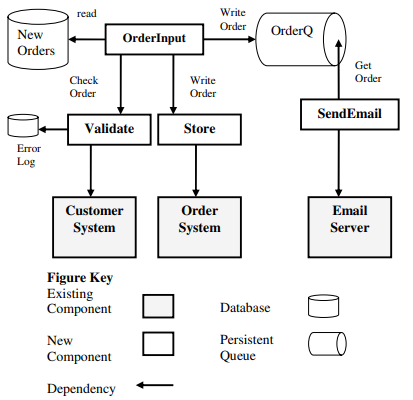
Khi một khung kiến trúc tổng thể đã được chọn, dựa trên một hoặc nhiều mẫu kiến trúc, nhiệm vụ tiếp theo là xác định các thành phần chính sẽ bao gồm thiết kế. Khuôn khổ xác định các mẫu giao tiếp tổng thể cho các thành phần. Điều này phải được tăng cường bởi những điều sau:

* Xác định các thành phần ứng dụng chính và cách chúng plug into vào khung làm việc
* Xác định giao diện hoặc các dịch vụ mà mỗi thành phần hỗ trợ
* Xác định các trách nhiệm của thành phần, nêu rõ nó có thể là gì dựa vào để làm khi nó nhận được một yêu cầu
* Xác định sự phụ thuộc giữa các thành phần
* Xác định các phân vùng trong kiến trúc là ứng viên để phân phối qua các máy chủ trong mạng

Các thành phần trong kiến trúc là những yếu tố trừu tượng chính sẽ tồn tại trong ứng dụng. Do đó, có lẽ không có gì ngạc nhiên khi dấu hiệu thành phần có nhiều điểm chung với các kỹ thuật thiết kế hướng đối tượng được sử dụng rộng rãi. Trên thực tế, các sơ đồ lớp và gói thường được sử dụng để mô tả các thành phần trong một kiến trúc.

Dưới đây là một vài hướng dẫn cho thiết kế thành phần:

* Giảm thiểu sự phụ thuộc giữa các thành phần. Cố gắng tạo ra một giải pháp cam kết lỏng lẻo trong đó các thay đổi đối với một thành phần không bị xáo trộn trong kiến trúc, lan truyền trên nhiều thành phần. Hãy nhớ rằng, mỗi khi bạn thay đổi điều gì, bạn phải kiểm tra lại nó.
* Thiết kế các thành phần đóng gói một tập hợp trách nhiệm có tính “gắn kết” cao. Độ kết dính là thước đo mức độ phù hợp của các bộ phần với nhau. Các thành phần có tính gắn kết cao có xu hướng có một nhóm nhỏ các trách nhiệm được xác định rõ để thực hiện một chức năng logic duy nhất. Ví dụ, một thành phần EnrollReports đóng gói tất cả các chức năng cần thiết đẻ tạo báo cáo về việc đăng ký học của sinh viên trong các khoá học. Nếu cần thay đổi định dạng hoặc loại báo cáo, thì khả năng các thay đổi sẽ được thực hiện trong thành phần này. Do đó, sự gắn kết mạnh mẽ giới hạn nhiều loại thay đổi đối với một thành phần duy nhất, giảm thiểu các nỗ lực kiểm thử và bảo trì
* Cô lập sự phụ thuộc vào phần mềm trung gian và bất kỳ công nghệ cơ sở hạ tầng COTS nào. Càng ít thành phần phụ thuộc vào các lệnh gọi API của phần mềm trung gian và các thành phần COTS cụ thể, thì việc thay đổi hoặc nâng cấp phần mềm trung gian hoặc các dịch vụ cơ sở hạ tầng khác càng dễ dàng. Tất nhiên, điều này cần nhiều nỗ lực hơn để xây dựng và đưa ra một hình phạt hiệu suất.
* Sử dụng phân rã để cấu trúc các thành phần theo thứ bậc. Thành phần cấp ngoài cùng xác định giao diện có sẵn công khai cho thành phần hỗn hợp. Bên trong, các lệnh gọi tới giao diện này được uỷ quyền cho các thành phần được xác định cục bộ, mà các giao diện của chúng không hiển thị bên ngoài.
* Giảm thiểu các cuộc gọi giữa các thành phần, vì chúng có thể gây tốn kém nếu các thành phần được phân phối. Cố gắng tổng hợp các chuỗi lệnh gọi giữa các thành phần thành một lệnh gọi duy nhất có thể thực hiện các xử lý cần thiết trong một yêu cầu duy nhất. Điều này tạo ra các phương thức hoặc dịch vụ thô hơn trong các giao diện thực hiện nhiều công việc hơn cho mỗi yêu cầu



Hình 5.6. Ví dụ kiến trúc về xử lý đơn đặt hàng

Hãy khám phá một ví dụ đơn giản minh hoạ cho vấn đề này. Hình 5.6. là một ví dụ về khung nhìn kiến trúc của một ứng dụng xử lý đơn đặt hàng, được xác định bằng cách sử dụng một ký hiệu không chính thức đơn giản. Đơn đặt hàng mới được nhận (từ nơi không liên quan – không nằm trong hệ thống con này) và được tải vào cơ sở dữ liệu. Mỗi đơn đặt hàng phải được xác thực dựa trên hệ thống chi tiết khách hàng hiện tại để kiểm tra thông tin khách hàng và các tuỳ chọn thanh toán hợp lệ có tồn tại hay không. Sau khi được xác thực, dữ liệu đơn đặt hàng chỉ được lưu trữ trong cơ sở dữ liệu xử lý đơn đặt hàng và một email được tạo cho khách hàng để thông báo rằng đơn đặt hàng của họ đang được xử lý.

Khung kiến trúc chung dựa trên thông điệp đơn giản. Chi tiết đơn đặt hàng của khách hàng được đọc từ cơ sở dữ liệu, được xác thực và nếu hợp lệ, chúng được lưu trữ trong ứng dụng đặt hàng và được ghi vào hàng đợi. Thông tin về mỗi đơn đặt hàng hợp lệ được xoá khỏi hàng đợi, được định dạng dưới dạng email và được gửi đến khách hàng bằng máy chủ thư. Do đó, bằng cách sử dụng hàng đợi tin nhắn, kiến trúc này sẽ tách quá trình xử lý đơn hàng khỏi định dạng và gửi email.

Bốn thành phần được giới thiệu để giải quyết vấn đề này. Những điều này được mô tả bên dưới, cùng với liên kết phản hồi của chúng.

* OrderInput: Điều này chịu trách nhiệm truy cập cơ sở dữ liệu đơn đặt hàng mới, đóng gói logic xử lý đơn hàng và ghi vào hàng đợi.
* Validate (xác thực): điều nay bao gồm trách nhiệm tương tác với hệ thống khách hàng để thực hiện xác thực và ghi vào nhật ký lỗi nếu đơn đặt hàng không hợp lệ.
* Store (cửa hàng): cái này có trách nhiệm tương tác với hệ thống đặt hàng để lưu trữ dữ liệu đơn hàng.
* SendEmail: thao tác này xoá thư khỏi hàng đợi, định dang thư email và gửi qua máy chủ email. Nó gói gọn tất cả kiến thức về định dạng email và quyền truy cập máy chủ email

Vì vậy, mỗi thành phần có các phụ thuộc rõ ràng và một tập hợp các trách nhiệm nhỏ, tạo ra một kiến trúc gắn kết và liên kết lỏng lẻo. Chúng tôi sẽ quay lại ví dụ và phân tích sâu hơn các thuộc tính của nó trong phần tiếp theo, trong đó thảo luận vêf việc xác thực thiết kế kiến trúc.

## **5.4. Thẩm định (validation)**

Trong quy trình kiến trúc, mục đích của giai đoạn thẩm định là tăng cường sự tin tưởng của nhóm thiết kế rằng kiến trúc phù hợp với mục tiêu của dự án. Việc xác nhận phải đạt được trong các ràng buộc dự án, vì thiết kế chi tiết và việc thực hiện thường không thể bắt đầu hoàn toàn cho đến khi kiến trúc được thống nhất. Bí quyết là phải chặt chẽ và hiệu quả nhất có thể.

Việc xác nhận một thiết kế kiến trúc đặt ra một số thách thức khó khăn. Cho dù đó là kiến trúc cho một ứng dụng mới hay một sự phát triển của hệ thống hiện có, thì thiết kế được đề xuất cũng chỉ là một thiết kế. Nó không thể được thực thi hoặc kiểm tra để thấy rằng nó đáp ứng các yêu cầu của nó. Nó cũng có thể sẽ bao gồm các thành phần mới phải được xây dựng và các thành phần có sẵn hộp đen như phần mềm trung gian và các thư viện chuyên biệt và các ứng dụng hiện có. Tất cả các bộ phận này phải được tích hợp và được làm cùng nhau.

Vì vậy, những gì có thể được thực hiện một cách hợp lý? Có hai kỹ thuật chính đã được chứng minh là hữu ích. Việc đầu tiên về cơ bản liên quan đến việc kiểm tra thủ công kiến trúc bằng cách sử dụng kịch bản kiểm thử. Thứ hai liên quan đến việc xây dựng một nguyên mẫu tạo ra một nguyên mẫu đơn giản của ứng dụng mong muốn, để khả năng thoả mãn các yêu cầu có thể được đánh giá chi tiết hơn thông quả thử nghiệm nguyên mẫu. Mục đích của cả hai là xác định các sai sót và điểm yếu tiềm ẩn trong thiết kế để chúng có thể được cải thiện trước khi bắt đầu triển khai. Những cách tiếp cận này nên được sử dụng để xác định rõ ràng các khu vực rủi ro tiềm ẩn để theo dõi và giám sát trong các hoạt động xây dựng tiếp theo.

# CHƯƠNG 6. TÀI LIỆU KIẾN TRÚC PHẦN MỀM

## **6.1. Giới thiệu tổng quan về tài liệu kiến trúc**

Ngay cả kiến ​​trúc tốt nhất, hoàn toàn phù hợp nhất cho công việc, về cơ bản sẽ trở nên vô dụng nếu những người cần sử dụng nó không biết nó là gì; không thể hiểu nó đủ tốt để sử dụng xây dựng hoặc sửa đổi nó; hoặc (tệ nhất là) hiểu sai nó và áp dụng nó không chính xác. Và tất cả nỗ lực, phân tích, làm việc chăm chỉ và thiết kế sâu sắc của đội kiến ​​trúc sẽ bị lãng phí.

Việc tạo một kiến trúc chưa đủ. Nó phải được truyền đạt theo cách để các bên liên quan sử dụng nó đúng cách để thực hiện công việc của họ. Nếu bạn gặp khó khăn trong việc tạo ra một kiến trúc mạnh mẽ, một kiến trúc mà bạn mong đợi sẽ đứng vững trước thử thách của thời gian, thì bạn phải gặp khó khăn trong việc mô tả nó đủ chi tiết, không mơ hồ và tổ chức nó để người khác có thể nhanh chóng tìm thấy và cập nhật thông tin cần thiết.

Tài liệu nói cho kiến trúc sư. Nó nói lên kiến trúc sư ngày nay, khi kiến trúc sư nên làm những việc khác ngoài việc trả lời hàng trăm câu hỏi về kiến trúc. Và nó nói cho kiến trúc sư vào ngày mai, khi anh ta hoặc cô ta đã rời khỏi dự án và bây giờ một người khác phụ trách quá trình phát triển và bảo trì của nó.

Sự thật đáng buồn là tài liệu kiến trúc ngày nay, nếu nó được thực hiện, thường được coi là một suy nghĩ sau, một cái gì đó mọi người làm vì họ phải làm. Có thể một hợp đồng yêu cầu nó. Có thể một khách hàng yêu cầu nó. Có thể quy trình tiêu chuẩn của một công ty yêu cầu nó. Trên thực tế, đây đều có thể là những lý do chính đáng. Nhưng không ai trong số họ đủ thuyết phục để tạo ra tài liệu chất lượng cao.

Tại sao kiến trúc sư nên dành thời gian và năng lượng quý báu để người quản lý có thể kiểm tra một công trình có thể giao được?

Các kiến trúc sư giỏi nhất tạo ra tài liệu tốt không phải vì nó "bắt buộc" mà vì họ thấy rằng điều cần thiết là tạo ra một sản phẩm chất lượng cao, có thể dự đoán được và ít phải làm lại nhất có thể. Họ coi các bên liên quan trực tiếp của họ là những người liên quan mật thiết nhất đến công việc này: nhà phát triển, người triển khai, người thử nghiệm và nhà phân tích.

Nhưng các kiến trúc sư cũng coi tài liệu là cung cấp giá trị cho chính họ. Tài liệu đóng vai trò là nơi lưu trữ kết quả của các quyết định thiết kế quan trọng khi chúng được xác nhận. Một sơ đồ tài liệu được suy nghĩ kỹ lưỡng có thể giúp quá trình thiết kế diễn ra suôn sẻ và có hệ thống hơn nhiều. Tài liệu giúp (các) kiến trúc sư lập luận về thiết kế kiến trúc và truyền đạt nó trong khi tiến trình xây dựng kiến trúc, cho dù trong giai đoạn thiết kế sáu tháng hay chạy nước rút Agile sáu ngày

## **6.2. Việc sử dụng và đối tượng cho tài liệu kiến trúc**

Một tài liệu Kiến trúc phải phục vụ các mục đích khác nhau. Nó phải đủ minh bạch và dễ tiếp cận để nhân viên mới nhanh chóng hiểu được. Nó phải đủ cụ thể để làm bản thiết kế xây dựng. Nó phải có đủ thông tin để làm cơ sở cho việc phân tích. Tài liệu kiến trúc là cả quy tắc và mô tả. Đối với một số khán giả, nó quy định điều gì phải đúng, đặt ra những hạn chế đối với các quyết định chưa được đưa ra. Đối với các đối tượng khác, nó mô tả những gì là sự thật, kể lại các quyết định đã được đưa ra về thiết kế của hệ thống. Chẳng hạn, tài liệu kiến trúc tốt nhất để phân tích hiệu suất có thể khác với tài liệu kiến trúc tốt nhất mà chúng tôi muốn giao cho người triển khai. Và cả hai điều này sẽ khác với những gì chúng tôi đưa ra trong gói " welcome aboard " của người thuê mới hoặc một cuộc họp báo mà chúng tôi tổng hợp cho một giám đốc điều hành. Khi lập kế hoạch và xem xét tài liệu, bạn cần đảm bảo hỗ trợ tất cả các nhu cầu liên quan.

Chúng ta có thể thấy rằng nhiều loại người khác nhau sẽ quan tâm đến một tài liệu kiến trúc. Họ hy vọng và mong đợi rằng tài liệu kiến trúc sẽ giúp họ thực hiện các công việc tương ứng. Hiểu cách sử dụng của chúng trong tài liệu kiến trúc là điều cần thiết, vì những cách sử dụng đó xác định thông tin quan trọng cần nắm bắt.

Về cơ bản, tài liệu kiến trúc có ba cách sử dụng:

1. Tài liệu kiến trúc phục vụ như một phương tiện giáo dục. Việc sử dụng giáo dục bao gồm việc giới thiệu mọi người đến hệ thống. Mọi người có thể là thành viên mới của nhóm, các nhà phân tích bên ngoài, hoặc thậm chí là một kiến trúc sư mới. Trong nhiều trường hợp, người "mới" là khách hàng mà bạn đang trình bày giải pháp của mình lần đầu tiên, một bài thuyết trình mà bạn hy vọng sẽ nhận được tài trợ hoặc tiếp tục được chấp thuận.
2. Tài liệu kiến trúc đóng vai trò như một phương tiện chính để giao tiếp giữa các bên liên quan. Việc sử dụng chính xác kiến trúc như một phương tiện giao tiếp phụ thuộc vào việc các bên liên quan đang thực hiện giao tiếp
3. Tài liệu kiến trúc làm cơ sở cho việc phân tích và xây dựng hệ thống. Kiến trúc cho người triển khai biết những gì cần triển khai. Mỗi mô-đun có các giao diện phải được cung cấp và sử dụng các giao diện từ các mô-đun khác. Điều này không chỉ cung cấp hướng dẫn về các giao diện được cung cấp và sử dụng, mà còn xác định những gì nhóm khác mà nhóm phát triển cho mô-đun phải giao tiếp

## **6.3. Ký hiệu cho Tài liệu Kiến trúc**

Các ký hiệu cho các dạng xem tài liệu khác nhau đáng kể về mức độ hình thức của chúng. Nói một cách đại khái, có ba loại ký hiệu chính:

* Ký hiệu không chính thức. Các khung nhìn được mô tả (thường bằng đồ thị) bằng cách sử dụng các công cụ chỉnh sửa và sơ đồ có mục đích chung cũng như các quy ước trực quan được chọn cho hệ thống trong tầm tay. Ngữ nghĩa của mô tả được đặc trưng bằng ngôn ngữ tự nhiên, và chúng không thể được phân tích chính thức. Theo kinh nghiệm của chúng tôi, công cụ phổ biến nhất cho các ký hiệu không chính thức là Power Point.
* Ký hiệu bán chính thức. Các khung nhìn được thể hiện dưới dạng ký hiệu chuẩn hóa quy định các yếu tố đồ họa và quy tắc xây dựng, nhưng nó không cung cấp cách xử lý ngữ nghĩa hoàn chỉnh về ý nghĩa của các yếu tố đó. Phân tích thô sơ có thể được áp dụng để xác định xem một mô tả có thỏa mãn các thuộc tính cú pháp hay không. UML là một ký hiệu bán chính thức theo nghĩa này
* Ký hiệu chính thức. Các khung nhìn được mô tả trong một ký hiệu có ngữ nghĩa chính xác (thường dựa trên toán học). Có thể phân tích chính thức cả cú pháp và ngữ nghĩa. Có rất nhiều ký hiệu chính thức cho kiến trúc phần mềm có sẵn. Thường được gọi là ngôn ngữ mô tả kiến trúc (ADL), chúng thường cung cấp cả từ vựng đồ họa và ngữ nghĩa cơ bản để biểu diễn kiến trúc. Trong một số trường hợp, các ký hiệu này được chuyên biệt hóa cho các khung nhìn kiến trúc cụ thể. Ở những người khác, chúng cho phép nhiều chế độ xem, hoặc thậm chí cung cấp khả năng xác định chính thức các chế độ xem mới. Tính hữu ích của ADL nằm ở khả năng hỗ trợ tự động hóa thông qua các công cụ liên quan: tự động hóa để cung cấp phân tích hữu ích về kiến trúc hoặc hỗ trợ tạo mã. Trong thực tế, việc sử dụng các ký hiệu như vậy là rất hiếm.

## **6.4. Khung nhìn**

Có lẽ khái niệm quan trọng nhất liên quan đến tài liệu kiến trúc phần mềm là chế độ xem. Kiến trúc phần mềm là một thực thể phức tạp không thể được mô tả theo kiểu một chiều đơn giản. Khung nhìn là một biểu diễn của một tập hợp các phần tử hệ thống và các quan hệ giữa chúng không phải là tất cả các phần tử hệ thống, mà là các phần tử của một kiểu cụ thể. Ví dụ, một khung nhìn phân lớp của một hệ thống sẽ hiển thị các phần tử của kiểu "lớp", nghĩa là, nó sẽ hiển thị sự phân hủy của hệ thống thành các lớp và mối quan hệ giữa các lớp đó. Tuy nhiên, một chế độ xem phân lớp thuần túy sẽ không hiển thị các dịch vụ của hệ thống, máy khách và máy chủ, hoặc mô hình dữ liệu hoặc bất kỳ loại phần tử nào khác.

Do đó, các khung nhìn cho phép chúng ta phân chia thực thể đa chiều là một kiến trúc phần mềm thành một số (chúng tôi hy vọng) các đại diện thú vị và có thể quản lý được của hệ thống. Khái niệm về khung nhìn cung cấp cho chúng ta nguyên tắc cơ bản nhất của tài liệu kiến trúc:

*Tài liệu hóa một kiến trúc là vấn đề ghi lại các khung nhìn có liên quan và sau đó thêm tài liệu áp dụng cho nhiều khung nhìn*

Câu châm ngôn này mang lại cho chúng tôi cách tiếp cận tài liệu tên của nó: Views và Beyond.

Các quan điểm có liên quan là gì? Điều này hoàn toàn phụ thuộc vào mục tiêu của bạn. Như chúng ta đã thấy trước đây, tài liệu kiến trúc có thể phục vụ nhiều mục đích: tuyên bố sứ mệnh cho người triển khai, cơ sở để phân tích, đặc điểm kỹ thuật để tạo mã tự động, điểm bắt đầu để hiểu hệ thống và khôi phục tài sản hoặc bản thiết kế để lập kế hoạch dự án.

Các chế độ xem khác nhau cũng thể hiện các thuộc tính chất lượng khác nhau ở các mức độ khác nhau. Do đó, các thuộc tính chất lượng mà bạn và các bên liên quan khác quan tâm nhất trong sự phát triển của hệ thống sẽ ảnh hưởng đến việc lựa chọn các chế độ xem để lập tài liệu. Ví dụ: chế độ xem phân lớp sẽ cho phép bạn lý luận về tính di động của hệ thống, chế độ xem triển khai sẽ cho phép bạn lý luận về hiệu suất hệ thống và độ tin cậy, v.v.

Các quan điểm khác nhau hỗ trợ các mục tiêu và mục đích sử dụng khác nhau. Đây là lý do tại sao chúng tôi không ủng hộ một quan điểm cụ thể hoặc một tập hợp các quan điểm. Các dạng xem nên ghi lại phụ thuộc vào cách sử dụng mà bạn mong đợi để tạo ra tài liệu. Các quan điểm khác nhau sẽ làm nổi bật các yếu tố và quan hệ hệ thống khác nhau. Có bao nhiêu quan điểm khác nhau để đại diện là kết quả của một quyết định chi phí / lợi ích. Mỗi chế độ xem đều có chi phí và lợi ích và bạn nên đảm bảo rằng lợi ích của việc duy trì một chế độ xem cụ thể lớn hơn chi phí của nó

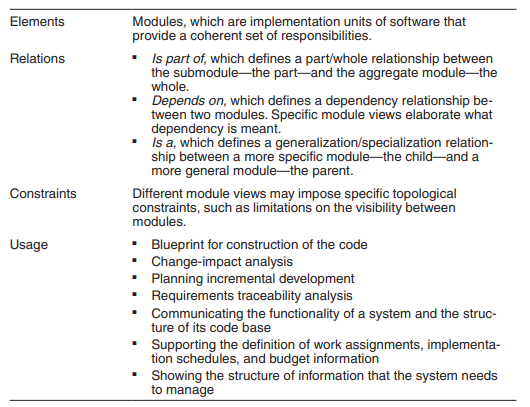
Các khung nhìn Mô đun

Một mô-đun là một đơn vị hợp thành cung cấp một tập hợp các trách nhiệm nhất quán. Một mô-đun có thể có dạng một lớp, một tập hợp các lớp, một lớp, một khía cạnh hoặc bất kỳ phân tách nào của đơn vị triển khai. Chế độ xem mô-đun ví dụ là phân tách, sử dụng và các lớp. Mỗi mô-đun có một tập hợp các thuộc tính được gán cho nó. Các thuộc tính này nhằm thể hiện thông tin quan trọng liên quan đến mô-đun, cũng như các ràng buộc đối với mô-đun. Thuộc tính mẫu là trách nhiệm, thông tin hiển thị và lịch sử sửa đổi. Các quan hệ mà các mô-đun có với nhau bao gồm là một phần, phụ thuộc vào, và là một.

Cách thức mà phần mềm của hệ thống được phân tách thành các đơn vị có thể quản lý được vẫn là một trong những dạng cấu trúc hệ thống quan trọng. Ở mức tối thiểu, điều này xác định cách mã nguồn của hệ thống được phân tách thành các đơn vị, các loại giả định mà mỗi đơn vị có thể thực hiện về các dịch vụ do các đơn vị khác cung cấp và cách các đơn vị đó được tổng hợp thành các tập hợp lớn hơn. Nó cũng bao gồm các cấu trúc dữ liệu toàn cầu tác động và bị tác động bởi nhiều đơn vị. Cấu trúc mô-đun thường xác định các thay đổi đối với một phần của hệ thống có thể ảnh hưởng đến các phần khác như thế nào và do đó khả năng của hệ thống hỗ trợ khả năng sửa đổi, tính di động và tái sử dụng.

Không chắc rằng tài liệu của bất kỳ kiến trúc phần mềm nào có thể hoàn chỉnh mà không có ít nhất một chế độ xem mô-đun.

Bảng 6.1 tóm tắt các yếu tố, quan hệ, ràng buộc và mục đích của các khung nhìn mô-đun nói chung. Sau đó, chúng tôi cung cấp thông tin này cụ thể cho từng dạng xem mô-đun thường được sử dụng.



Các thuộc tính của mô-đun giúp hướng dẫn thực hiện hoặc được đầu vào để phân tích phải được ghi lại như một phần của tài liệu hỗ trợ cho chế độ xem mô-đun. Danh sách các thuộc tính có thể khác nhau nhưng có thể bao gồm những điều sau:

* Tên. Tất nhiên, tên của mô-đun là phương tiện chính để đề cập đến nó. Tên của mô-đun thường gợi ý điều gì đó về vai trò của nó trong hệ thống. Ngoài ra, tên của mô-đun có thể phản ánh vị trí của nó trong một hệ thống phân cấp phân tách; Ví dụ, tên A.B.C đề cập đến mô-đun C là mô-đun con của mô-đun B, bản thân nó là mô-đun con của A.
* Các trách nhiệm. Thuộc tính trách nhiệm cho một mô-đun là một cách để xác định vai trò của nó trong hệ thống tổng thể và thiết lập danh tính cho nó ngoài tên gọi. Trong khi tên của một mô-đun có thể gợi ý vai trò của nó, thì một tuyên bố về trách nhiệm thiết lập nó một cách chắc chắn hơn nhiều. Các trách nhiệm phải được mô tả đầy đủ chi tiết để làm rõ cho người đọc những gì mỗi mô-đun làm
* Khả năng hiển thị của (các) giao diện. Khi một mô-đun có các mô-đun con, một số giao diện của các mô-đun con là công khai và một số có thể là riêng tư; nghĩa là, các giao diện chỉ được sử dụng bởi các mô-đun con trong mô-đun mẹ bao quanh. Các giao diện riêng tư này không hiển thị bên ngoài ngữ cảnh đó
* Thông tin triển khai. Mô-đun là đơn vị thực hiện. Do đó, rất hữu ích khi ghi lại thông tin liên quan đến việc thực hiện chúng trên quan điểm quản lý sự phát triển và xây dựng hệ thống chứa chúng. Điều này có thể bao gồm những điều sau:
  + Ánh xạ tới các đơn vị mã nguồn. Điều này xác định các tệp cấu thành việc triển khai một mô-đun. Ví dụ: một Tài khoản mô-đun, nếu được triển khai bằng Java, có thể có một số tệp cấu thành việc triển khai nó: IAccount.java (một giao diện), Accountimpl.java (triển khai chức năng Tài khoản), AccountBean.java (một lớp để giữ trạng thái của một tài khoản trong bộ nhớ), AccountOrmMapping.xml (một tệp xác định ánh xạ giữa AccountBean và ánh xạ quan hệ đối tượng trong bảng cơ sở dữ liệu) và thậm chí có thể là một bài kiểm tra đơn vị AccountTest.java
  + Thông tin kiểm tra. Kế hoạch kiểm thử của mô-đun, các trường hợp kiểm thử, và dữ liệu kiểm tra là những tài liệu quan trọng. Thông tin này có thể chỉ đơn giản là một con trỏ đến vị trí của những hiện vật này.
  + Thông tin quản lý. Người quản lý có thể cần thông tin về lịch trình và ngân sách dự đoán của mô-đun. Thông tin này có thể chỉ đơn giản là một con trỏ đến vị trí của những hiện vật này.
  + Ràng buộc thực hiện. Trong nhiều trường hợp, kiến trúc sư sẽ có một chiến lược triển khai cho một mô-đun hoặc có thể biết các ràng buộc mà việc triển khai phải tuân theo.
  + Lịch sử sửa đổi. Biết lịch sử của một mô-đun bao gồm các tác giả và các thay đổi cụ thể có thể hữu ích khi bạn hoàn thiện các hoạt động bảo trì.

Vì các mô-đun phân vùng hệ thống, nên có thể xác định cách các yêu cầu chức năng của một hệ thống được hỗ trợ bởi các trách nhiệm của mô-đun. Chế độ xem mô-đun hiển thị sự phụ thuộc giữa các mô-đun hoặc các lớp (là các nhóm mô-đun có kiểu sử dụng được phép cụ thể) cung cấp cơ sở tốt để phân tích tác động thay đổi. Các mô-đun thường được sửa đổi do báo cáo sự cố hoặc yêu cầu thay đổi. Phân tích tác động yêu cầu một mức độ hoàn chỉnh nhất định của thiết kế và tính toàn vẹn của mô tả mô-đun. Đặc biệt, thông tin phụ thuộc phải có sẵn và chính xác để có thể tạo ra kết quả hữu ích.

Chế độ xem mô-đun có thể được sử dụng để giải thích chức năng của hệ thống cho những người không quen thuộc với nó. Các mức độ chi tiết khác nhau của việc phân rã mô-đun cung cấp bản trình bày từ trên xuống về các trách nhiệm của hệ thống và do đó có thể hướng dẫn quá trình học tập. Đối với một hệ thống đã có triển khai, các dạng xem mô-đun, nếu được cập nhật, sẽ rất hữu ích, vì chúng giải thích cấu trúc của cơ sở mã cho một nhà phát triển mới trong nhóm. Do đó, các chế độ xem mô-đun cập nhật có thể đơn giản hóa và thường xuyên việc bảo trì hệ thống.

## **6.5. Mẫu tài liệu kiến trúc phần mềm**

Tài liệu kiến trúc phần mềm bao gồm:

**1.** Giới thiệu

Phần giới thiệu này cung cấp cái nhìn tổng quan về toàn bộ Tài liệu Kiến trúc Phần mềm Nó bao gồm mục đích, phạm vi, định nghĩa, từ viết tắt, chữ viết tắt, tài liệu tham khảo và tổng quan về hệ thống.

* 1. Mục đích
  2. Phạm vi
  3. Các định nghĩa, từ viết tắt
  4. Tài liệu tham khảo
  5. Tổng quan

2. Mục tiêu và các ràng buộc của kiến túc

3. Đại diện kiến trúc

3.1. Các khung nhìn kiến trúc

3.2. Mẫu kiến trúc

3.3. Tiến trình kiến trúc

4. Phân rã khung nhìn kiến trúc

4.1. Use – case view

4.2. Khung nhìn thiết kế/khái niệm/logic

4.3. Khung nhìn xử lý

4.4. Khung nhìn thành phần

4.5. Khung nhìn triển khai

5. Kích thước và hiệu suất

6. Chất lượng

7. Tài liệu tham khảo

**6.6. Case study**

# CHƯƠNG 7: KIẾN TRÚC KHÔNG ĐỒNG NHẤT – HETEROGENEOUS

Mục tiêu của Chương này

* Giới thiệu phương pháp luận chung để đưa ra quyết định về kiến trúc
* Tóm tắt lợi ích và hạn chế của từng phong cách kiến trúc
* Trình bày cách áp dụng các nguyên tắc chung trong các ví dụ nghiên cứu tình huống

## **7.1 Tổng quan**

Trong thực tế, nhiều phong cách kiến trúc thường phải được sử dụng trong cùng một dự án. Hãy tưởng tượng bạn là kiến trúc sư của một lâu đài thời trung cổ. Không có khả năng rằng cùng một phong cách kiến trúc có thể được sử dụng cho tất cả các bộ phận của lâu đài (ví dụ: hào, tháp, cầu kéo, tường rèm, sảnh lâu đài, các tòa nhà dân dụng). Tương tự, đối với một dự án phần mềm quy mô lớn, các phong cách kiến trúc không đồng nhất được sử dụng để kết hợp lợi ích của nhiều phong cách và đảm bảo chất lượng và tính phù hợp.

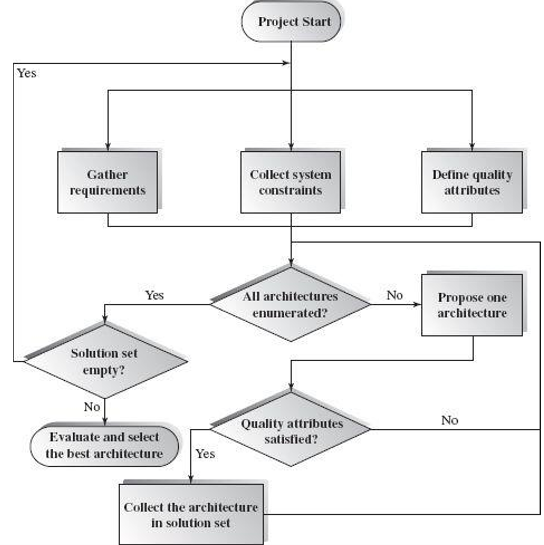
Chương này cung cấp một nghiên cứu điển hình về kiến trúc không đồng nhất. Nó kiểm tra việc phân tích và thiết kế một dự án quy mô tương đối lớn (một nhà sản xuất và cung cấp máy tính trực tuyến) bằng cách sử dụng phương pháp “chia và chinh phục” đã được kiểm nghiệm thời gian. Bài tập tình huống sẽ giúp bạn ôn lại kiến thức đã học trong các chương trước và thực hành các nguyên tắc chung trong một thiết kế thực tế hơn.

## **7.2 Phương pháp luận về quyết định kiến ​​trúc**

Một chỉ huy quân đội thành công phải tính đến tất cả các yếu tố trước khi đưa binh lính ra ngoài. Tương tự, trách nhiệm của một kiến ​​trúc sư phần mềm giỏi không phải là tìm ra một giải pháp khả thi mà là xác định kiến ​​trúc nào phù hợp nhất với nhu cầu nghiệp vụ. Một số kiểu kiến ​​trúc đã được đề cập, bao gồm kiến ​​trúc luồng dữ liệu (tuần tự hàng loạt, bộ lọc đường ống, điều khiển quy trình), kiến ​​trúc trung tâm dữ liệu (kho lưu trữ, bảng đen), phân cấp, giao tiếp không đồng bộ ngầm định, hướng tương tác, phân tán và dựa trên thành phần kiến trúc.

Với số lượng lớn các phong cách kiến ​​trúc thay thế có sẵn, làm thế nào để bạn chọn đúng phong cách sẽ đạt được các mục tiêu của dự án một cách tối ưu (ví dụ: với chi phí tối thiểu)? Phần này hướng dẫn bạn giới thiệu ngắn gọn về phương pháp luận chung để xác định kiến ​​trúc phần mềm theo các ràng buộc hệ thống nhất định.

Quá trình lựa chọn kiến ​​trúc của một hệ thống phần mềm liên quan chặt chẽ đến việc phân tích các yêu cầu. Các yêu cầu của hệ thống, mức độ ưu tiên của từng yêu cầu và các ràng buộc của hệ thống (ngân sách dự án, ngày phát hành, v.v.) đều xác định kiến ​​trúc sẽ được sử dụng, ngay cả khi nó có thể không phải là kiến ​​trúc đẹp nhất, nhanh nhất hoặc nhất tiết kiệm. Kiến trúc được chọn phải “tối ưu” và không nhất thiết phải tập trung vào một khía cạnh cụ thể của các ràng buộc hệ thống. Lấy thiết kế của một hệ điều hành làm ví dụ. Kiến trúc phân lớp và các biến thể của nó được các nhà thiết kế của nhiều hệ điều hành, chẳng hạn như Mac OS, OS2, Unix và các biến thể của nó, cũng như các phiên bản khác nhau của hệ điều hành Windows. Một điểm mạnh của kiến ​​trúc phân lớp là tính mạnh mẽ và điều này rất quan trọng đối với một hệ điều hành; vì lỗi không truyền từ lớp này sang lớp khác, nếu lớp trên gặp sự cố thì nó sẽ không ảnh hưởng đến các dịch vụ của lớp dưới cùng.



Hình 7.1. Biểu đồ luồng của tiến trình lựa chọn kiến trúc

Hình 7.1 trình bày quy trình lựa chọn kiến ​​trúc đơn giản hóa kết hợp với phân tích yêu cầu. Khi một dự án bắt đầu, các nhà phân tích hệ thống và kiến ​​trúc sư phần mềm tương tác với các bên liên quan của dự án, thu thập các yêu cầu của dự án, xác định các ràng buộc hệ thống và xác định các thuộc tính chất lượng của hệ thống. Sau đó, các kiến ​​trúc sư phần mềm phân rã các chức năng và đề xuất các phong cách kiến ​​trúc khả thi. Đối với mỗi thiết kế kiến ​​trúc được đề xuất, các chức năng cần thiết được ánh xạ tới các thành phần của kiến ​​trúc. Khi tất cả các chức năng cần thiết được thỏa mãn, các thuộc tính chất lượng được đánh giá dựa trên thiết kế kiến ​​trúc. Các kết quả, cùng với thiết kế kiến ​​trúc, được ghi lại và thu thập trong một tập hợp các giải pháp ứng viên. Quá trình được lặp lại cho tất cả các kiểu kiến ​​trúc có thể có và kết quả được đánh giá. Nếu không có thiết kế kiến ​​trúc được đề xuất nào hoạt động, các nhà phân tích hệ thống phải trao đổi với các bên liên quan về những thay đổi yêu cầu của dự án như thời hạn kéo dài, tăng ngân sách hoặc ít yêu cầu chức năng hơn. Khi tập hợp các yêu cầu và ràng buộc hệ thống được tinh chỉnh, toàn bộ quá trình bắt đầu lại từ đầu, cho đến khi xác định được kiến ​​trúc phù hợp nhất (hoặc kết thúc dự án).

## **7.3 Thuộc tính chất lượng**

Mặc dù mô hình tổng thể của quá trình lựa chọn kiến ​​trúc là rõ ràng, nhưng cách nó được thực hiện có thể khác nhau rất nhiều tùy thuộc vào các kiến trúc sư. Một bước đặc biệt khó khăn trong quy trình là xác định và đánh giá các thuộc tính chất lượng của dự án. Sự thành công của bước này phụ thuộc vào chuyên môn và kinh nghiệm của các kiến ​​trúc sư phần mềm và nhà phân tích hệ thống.

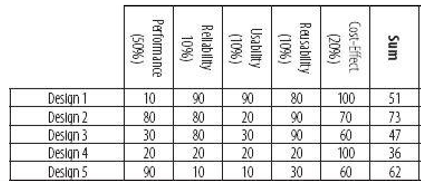
Thuộc tính chất lượng là một phần thiết yếu của các yêu cầu phi chức năng của hệ thống. Các nhà phân tích hệ thống có trách nhiệm tập hợp một danh sách đầy đủ các thuộc tính chất lượng trước khi quá trình thiết kế chi tiết có thể bắt đầu. Đối với hầu hết các hệ thống máy tính, các thuộc tính chất lượng thường bao gồm hiệu quả (hiệu quả thời gian, tiết kiệm tài nguyên), chức năng (tính hoàn chỉnh, bảo mật, khả năng tương tác), khả năng bảo trì (khả năng mở rộng, khả năng sửa đổi, khả năng kiểm tra), tính di động (độc lập phần cứng, độc lập phần mềm, khả năng cài đặt, khả năng tái sử dụng), độ tin cậy (khả năng chịu lỗi, tính khả dụng) và khả năng sử dụng (tính khả dụng của mã, giao diện người dùng, khả năng học hỏi).

Mỗi hệ thống, với miền ứng dụng cụ thể, có thể có các yêu cầu đặc biệt đối với các thuộc tính chất lượng. Các nhà phân tích hệ thống cần tương tác với các bên liên quan để xác định tầm quan trọng tương đối của các thuộc tính chất lượng. Lưu ý rằng, ngay cả đối với cùng một loại hệ thống phần mềm, các bên liên quan khác nhau có thể thích các thuộc tính chất lượng khác nhau. Ví dụ, hệ điều hành trên máy tính lớn của IBM, được triển khai rộng rãi trong các ngân hàng, có tiêu chuẩn rất cao về khả năng bảo trì, độ tin cậy và bảo mật. Tuy nhiên, hiệu quả và khả năng sử dụng có thể ít được quan tâm hơn. Mặt khác, một số bên liên quan có thể thích hiệu quả hơn bảo mật.

Trong nhiều trường hợp, các thuộc tính chất lượng có thể được định lượng. Ví dụ, để xác định thuộc tính hiệu suất của một máy chủ phát sóng phương tiện trực tuyến, nhiều yếu tố có thể được xác định và đánh giá định lượng, chẳng hạn như băng thông, thông lượng, độ trễ, tỷ lệ mất và tốc độ rung. Đối với các thuộc tính chất lượng khác, chẳng hạn như tính dễ hiểu và độ tin cậy, đánh giá định tính cũng có thể được chuyển đổi thành các giá trị định lượng. Các nhận xét như "xuất sắc" và "kém" có thể được ánh xạ thành các điểm số trong thang điểm 100.

Khi tất cả các thuộc tính chất lượng được thể hiện một cách định lượng, các kiến ​​trúc sư phần mềm có thể liệt kê các thiết kế kiến ​​trúc khả thi và đánh giá từng thuộc tính chất lượng cho mỗi thiết kế. Vì mỗi thuộc tính chất lượng được chỉ định một hệ số trọng số, được xác định trong quá trình phân tích yêu cầu trước đó, nên tổng điểm có thể được tính cho mỗi thiết kế. Khi quá trình đánh giá hoàn tất, thiết kế có điểm cao nhất có thể được chọn.

Hình 7.2 là một ví dụ đơn giản về đánh giá định lượng các thiết kế kiến ​​trúc. Giả sử rằng các nhà phân tích hệ thống trong một dự án xác định năm thuộc tính chất lượng trong quá trình phân tích yêu cầu: hiệu suất, độ tin cậy, khả năng sử dụng, khả năng tái sử dụng và hiệu quả chi phí. Mỗi thuộc tính chất lượng được ấn định một hệ số trọng số phần trăm; ví dụ, hiệu suất chiếm 50% mối quan tâm của các bên liên quan trong số tất cả các thuộc tính chất lượng. Đối với mỗi thuộc tính chất lượng, đánh giá được thể hiện bằng giá trị từ 0 đến 100.



Hình 7.2 Mẫu đánh giá định lượng các thuộc tính chất lượng

Điểm có trọng số có thể được tính cho mỗi thiết kế. Ví dụ, tổng điểm của Thiết kế 1 được tính như sau:

10 \* 50% + 90 \* 10% + 90 \* 10% + 80 \* 10% + 100 \* 20% = 51

Bằng cách so sánh tổng điểm của tất cả các thiết kế, chúng tôi thấy rằng lựa chọn là Thiết kế 2.

Khi đã xác định được khung đánh giá định lượng, kiến trúc sư phần mềm có thể tiến hành thiết kế kiến trúc bằng hai bước sau:

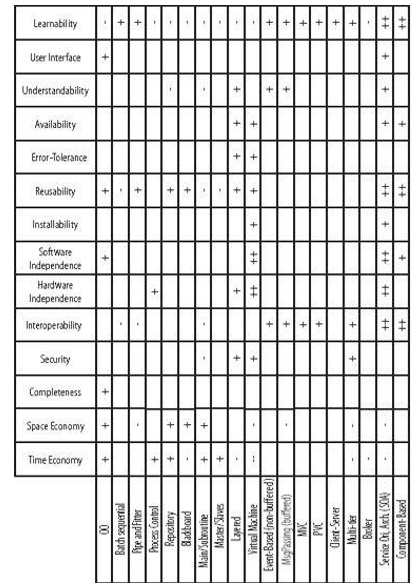
1. Chọn một phong cách kiến trúc thích hợp.

2. Trang bị các chi tiết của thiết kế kiến trúc; ví dụ, khi kiểu bộ lọc ống được chọn, kiến trúc sư phần mềm vẫn phải xác định bộ lọc sẽ là gì và cách kết nối chúng.\

## **7.4 Lựa chọn kiểu kiến ​​trúc**

Việc lựa chọn phong cách kiến ​​trúc thường phụ thuộc vào chuyên môn của các kiến ​​trúc sư phần mềm. Tuy nhiên, một công cụ hữu ích là sáng kiến ​​của Grady Booch để tạo ra Sổ tay Kiến trúc Phần mềm (Booch, 2004). Nó kiểm kê các hệ thống phần mềm và thảo luận về các ứng dụng của các phong cách kiến ​​trúc khác nhau. Đối với các nhà phát triển làm việc trên các dự án thông thường (ví dụ: ứng dụng web, ứng dụng cơ sở dữ liệu quan hệ, trò chơi trực tuyến, kiểm soát lưu lượng, v.v.), sổ tay là một cách nhanh chóng và thuận tiện để tìm các dự án liên quan trong cùng một miền ứng dụng và tìm hiểu về các kiểu kiến ​​trúc đã được sử dụng. [Ngoài ra, sự tiến bộ trong kiến ​​trúc dòng sản phẩm cũng đầy hứa hẹn.

Tuy nhiên, vẫn tồn tại hướng chung về cách lựa chọn phong cách kiến ​​trúc dựa trên các yêu cầu và ràng buộc của dự án. Hình 7.3 trình bày sự so sánh ngắn gọn của các phong cách kiến ​​trúc được giới thiệu trong cuốn sách này và liệt kê các thuộc tính chất lượng. Mỗi ô cho biết kiểu kiến ​​trúc có đáp ứng thuộc tính chất lượng tương ứng hay không như sau: “+” biểu thị tốt, “++” biểu thị rất tốt, “-” biểu thị xấu và “-” biểu thị rất xấu. Một ô trống biểu thị không có phán đoán rõ ràng cho cặp kiểu / thuộc tính trong trường hợp chung. Bằng cách kiểm tra kiểu kiến ​​trúc dựa trên từng thuộc tính chất lượng được yêu cầu trong Hình 7.3 và miền ứng dụng của mỗi kiểu kiến ​​trúc, một kiến ​​trúc sư phần mềm có thể có được ý tưởng sơ bộ về khả năng áp dụng của kiểu kiến ​​trúc trong một dự án.



Hình 7.3. Bảng thông tin về mẫu kiến trúc

## **7.5 Đánh giá thiết kế kiến ​​trúc**

Tiếp theo, chúng tôi giải quyết bước thứ hai trong việc lựa chọn kiến ​​trúc: cách đánh giá một thiết kế kiến ​​trúc. Có nhiều cách tiếp cận đánh giá hệ thống, chẳng hạn như ATAM (Phương pháp phân tích đánh đổi kiến ​​trúc), SAAM ((Software Architecture Analysis Method - Phương pháp phân tích kiến ​​trúc phần mềm) và ARID ((Active Reviews for Intermediate Designs - Đánh giá tích cực cho các thiết kế trung gian). Phần này thảo luận ngắn gọn về việc áp dụng phương pháp SAAM (Kazman, Abowd, Bass và Clements, 1996).

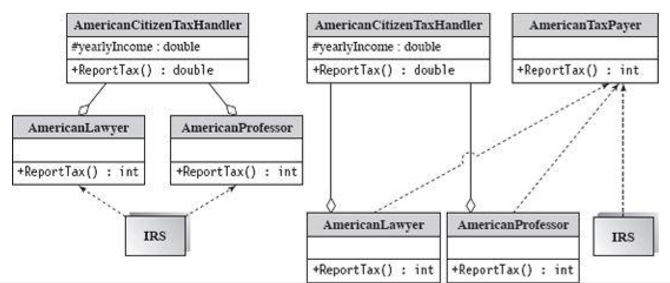
Ý tưởng chung của SAAM là đánh giá các thiết kế kiến ​​trúc ứng viên bằng cách sử dụng một tập hợp các kịch bản. Một kịch bản thiết kế thể hiện cách sử dụng quan trọng của một hệ thống và phản ánh khung nhìn của các bên liên quan. Lưu ý rằng một kịch bản tương tự, nhưng không giống với trường hợp sử dụng. Một ca sử dụng ghi lại cách sử dụng cần thiết của hệ thống; tuy nhiên, một kịch bản có thể chứa các tình huống không có trong phạm vi hiện tại của dự án. Ví dụ, để đánh giá khả năng sửa đổi của hệ thống đăng ký trực tuyến của một trường đại học, một tình huống có thể là: Các nhà phát triển nên làm gì khi cách trình bày dữ liệu nội bộ của gói ứng dụng tốt nghiệp bị thay đổi? Đây là một kịch bản được sử dụng trong SAAM, nhưng không phải là một trường hợp sử dụng của hệ thống.

Quá trình phân tích SAAM thường bao gồm ba giai đoạn:

1. Xác định một tập hợp các kịch bản thiết kế bao gồm các yêu cầu chức năng và phi chức năng. Các thuộc tính chất lượng cần được phản ánh
2. Thực hiện đánh giá tất cả các thiết kế kiến ​​trúc ứng viên, sử dụng tập hợp các kịch bản.
3. Thực hiện phân tích mối quan hệ tương tác giữa các kịch bản.

Nghiên cứu điển hình sau đây chứng minh hai giai đoạn đầu tiên trong SAAM. Nghiên cứu điển hình dựa trên ví dụ về người nộp thuế trong Chương 4. Giả sử rằng bạn sẽ thiết kế một hệ thống xử lý thuế trực tuyến cho Sở Thuế vụ (IRS). Người điều hành nhập thông tin của từng người đóng thuế vào hệ thống, thông tin này được ghi lại trong một ví dụ của một lớp như AmericanLawyer và AmericanProfessor, dựa trên nghề nghiệp của người đóng thuế. Một phiên bản của lớp IRS được sử dụng để xử lý thông tin thuế cho từng người đóng thuế. Hai thiết kế kiến ​​trúc ứng viên (cả hai kiểu OO) được trình bày trong Hình 7.4. Các bên liên quan quan tâm đến các thuộc tính chất lượng sau:

* Khả năng mở rộng: Theo thời gian, nhiều loại nghề nghiệp hơn có thể được thêm vào hệ thống, chẳng hạn như AmericanFarmer, AmericanBusinessOwner, v.v.
* Hiệu suất: Vì hàng triệu trường hợp có thể được xử lý mỗi ngày trong thời gian cao điểm, nên hiệu quả về thời gian là rất quan trọng. Tuy nhiên, hiệu quả về không gian ít được quan tâm hơn vì các bên liên quan có đủ kinh phí để mua bộ lưu trữ máy tính (rẻ hơn mỗi ngày).
* Khả năng thay đổi: Định dạng biểu mẫu thuế và phương pháp tính thuế suất thay đổi hàng năm. Hệ thống phải thích ứng với những thay đổi đó.



Để đánh giá hai thiết kế ứng viên trong Hình 7.4, ba tình huống sau được thiết kế:

* Tình huống 1: Thêm một nghề nghiệp nữa, được gọi là AmericanFarmer, vào hệ thống. Kịch bản này kiểm tra khả năng mở rộng của hệ thống.
* Tình huống 2: Thực hiện kiểm tra toàn diện ảo trên hệ thống. Hệ thống sẽ hoạt động như thế nào nếu nó phải xử lý một triệu người nộp thuế mỗi ngày? Đối tượng IRS bao gồm danh sách một triệu người nộp thuế và nó phải xử lý từng đối tượng bằng cách gọi hoạt động ReportTax(). Kịch bản này kiểm tra hiệu suất của hệ thống.
* Tình huống 3: Ví dụ: thay đổi thuật toán tính thuế suất trong ReportTax() để thay đổi quy tắc khấu trừ theo từng khoản. Kịch bản này kiểm tra khả năng sửa đổi của hệ thống.

Bây giờ hãy xem xét cách đánh giá hai thiết kế trong Hình 7.4.

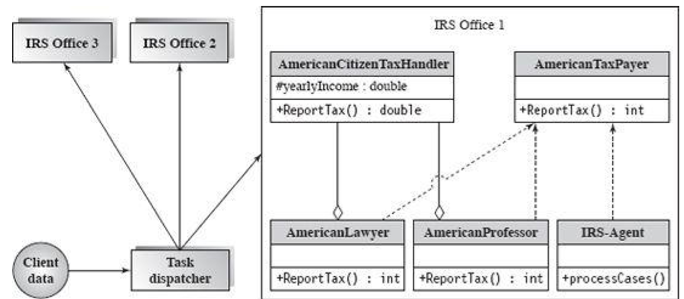
*Tình huống 1:* Trong Thiết kế 1 (bên trái của Hình 7.4), nếu một lớp American-Farmer mới được thêm vào, mã của lớp IRS phải được sửa đổi. Khi duyệt qua danh sách người nộp thuế, đối tượng IRS có một trường hợp chuyển đổi lớn để kiểm tra kiểu của từng cá thể người nộp thuế và gọi thao tác tương ứng. Một trường hợp phải được thêm vào câu lệnh trường hợp chuyển đổi đó. Trong Thiết kế 2 (bên phải Hình 7.4), lập trình viên có thể chỉ cần chèn tệp lớp mới của AmericanFarmer và tất cả các lớp khác vẫn còn nguyên vẹn. Rõ ràng, Thiết kế 2 đánh bại Thiết kế 1 trong Kịch bản 1.

*Tình huống 2:* Có thể dễ dàng nhận thấy hai thiết kế kiến ​​trúc tương tự nhau về công năng. Thiết kế 1 sẽ có chi phí cao hơn một chút trong việc quyết định thao tác Repo`rtTax () nào sẽ gọi dựa trên loại cá thể của người nộp thuế. Số lượng loại nghề nghiệp trong hệ thống càng lớn, câu lệnh trường hợp chuyển đổi càng dài và tổng chi phí càng lớn. Thiết kế 2 không có vấn đề này do tính đa hình, có chi phí phụ là một giá trị không đổi. Tuy nhiên, sự khác biệt về tổng chi phí của chúng là không đáng kể so với sự phức tạp của việc tính thuế suất. Do đó, cả hai thiết kế đều hoạt động như nhau trên Kịch bản 2.

Một câu hỏi khác cần xem xét là liệu cả hai thiết kế có vượt qua được bài kiểm tra của Kịch bản 2. Một triệu trường hợp phải được xử lý mỗi ngày, tương đương 11,57 trường hợp mỗi giây. Mục tiêu này gần như có thể đạt được nếu không có I/O nặng nào tham gia. Tuy nhiên, nếu các chức năng bổ sung được yêu cầu (ví dụ: để tạo báo cáo thuế tự động điền cho từng trường hợp), điều này có thể dễ dàng vượt qua khả năng của cả hai thiết kế kiến ​​trúc. Do đó, cả hai thiết kế có thể có vấn đề về hiệu suất khi các yêu cầu chức năng bổ sung được thêm vào.

Để cải thiện hiệu quả thời gian, có thể xem xét việc sao chép các thiết kế hiện tại (như trong Hình 7.5). Người điều phối tác vụ truy cập dữ liệu người dùng tuần tự và phân phối dữ liệu đến một số văn phòng IRS. Mỗi văn phòng IRS về cơ bản là một triển khai của Thiết kế 2 trong Hình 7.4. Lưu ý rằng kiến ​​trúc có thể dễ dàng mở rộng để giới thiệu nhiều văn phòng IRS hơn nhằm đạt được hiệu suất cao hơn nữa. Người điều phối nhiệm vụ cũng có thể được nhân rộng vì không có mối quan hệ phụ thuộc lẫn nhau giữa các trường hợp của người nộp thuế. Hình 7.5 về cơ bản là một kiểu kiến ​​trúc tuần tự hàng loạt.

*Tình huống 3:* Cả hai thiết kế trong Hình 7.4 sẽ hoạt động như nhau trong Kịch bản 3. Nếu thuật toán tính thuế thay đổi, lớp AmericanCitizenTaxHandler phải được sửa đổi trong cả hai thiết kế.



Hình 7.5 Người điều phối công việc song song



Hình 7.6. Các kết quả phân tích SAAM

Hình 7.6 tóm tắt kết quả phân tích SAAM trên hai thiết kế kiến trúc trong hình 7.4 và hình 7.5.

## **7.6 Nghiên cứu điển hình: Nhà cung cấp máy tính trực tuyến**

Phần này trình bày một nghiên cứu điển hình về cách xây dựng kế hoạch thiết kế cho một nhà cung cấp máy tính trực tuyến (OCV). Hệ thống kết quả được gọi là OCVS, trong đó chữ “S” cuối cùng là viết tắt của “Hệ thống”.

Mục tiêu kinh doanh của OCV là giảm thiểu giá thành sản phẩm và tăng tỷ lệ hài lòng của khách hàng thông qua công nghệ mới nhất. OCV bán máy tính trực tiếp cho khách hàng qua Internet. Khách hàng trực tiếp chỉ định những gì họ cần trực tuyến. Khi nhận được đơn đặt hàng, các bộ phận được lắp ráp để đáp ứng yêu cầu của khách hàng. Bán hàng trực tiếp cho phép loại bỏ những người trung gian và đại lý, do đó làm giảm giá thành sản phẩm. Sản xuất đúng lúc (JIT) được sử dụng để giảm thiểu chi phí tồn kho và tối đa hóa hiệu quả sản xuất. Các bộ phận chỉ được đặt hàng khi có yêu cầu. Toàn bộ quá trình xử lý đơn hàng và sản xuất sản phẩm được giám sát bởi một hệ thống máy tính (OCVS), và các đơn đặt hàng chính xác của các bộ phận được gửi đến nhà cung cấp mỗi ngày để các yêu cầu về bộ phận có thể được đáp ứng chính xác. Thành công của OCV dựa vào hệ thống máy tính, hệ thống kết nối thông tin giữa tất cả các phòng ban và tối đa hóa hiệu quả của toàn bộ tổ chức. Các yêu cầu chung của OCVS được nêu như sau:

1. Yêu cầu 1 Một cổng thông tin điện tử được cung cấp cho khách hàng để đặt hàng. Một đơn đặt hàng có thể được tùy chỉnh cao trong đó khách hàng có thể đặt cấu hình sản phẩm theo từng bước. Khách hàng có thể thanh toán trực tiếp qua giỏ hàng hoặc lưu lại giỏ hàng để sử dụng sau. Thẻ tín dụng là phương thức thanh toán duy nhất được chấp nhận. Cổng thông tin điện tử có thể phục vụ đồng thời 10.000 khách hàng.
2. Yêu cầu 2 Một hệ thống con được cung cấp cho các nhà thiết kế sản phẩm. Khi một sản phẩm được phát hành, nhà thiết kế có thể nhập cấu hình cơ bản của sản phẩm vào OCVS. Một thiết kế cũng có thể chỉ định quá trình cấu hình của sản phẩm. Khi thông số kỹ thuật của một sản phẩm mới được hoàn thành, các trang web tương ứng sẽ được tạo tự động trong OCVS. Khoảng không quảng cáo và hệ thống con cho dây chuyền lắp ráp cũng có thể kết hợp thông tin ngay lập tức.
3. Yêu cầu 3 Một hệ thống con là để hỗ trợ các dây chuyền lắp ráp. Mỗi công nhân trong một dây chuyền lắp ráp được trang bị một hệ thống máy tính mô tả chi tiết công việc mà công nhân đó phải thực hiện.
4. Yêu cầu 4 Một hệ thống con quản lý hàng tồn kho tự động được phát triển để theo dõi hàng tồn kho. Công cụ lập lịch tự động báo cáo số lượng bộ phận ước tính được yêu cầu mỗi ngày. Các báo cáo, nếu được người quản lý doanh nghiệp chấp thuận, sẽ được chuyển thành đơn đặt hàng bộ phận cho nhà cung cấp.
5. Yêu cầu 5 OCVS nên kết nối thông tin giữa tất cả các bộ phận để tối đa hóa hiệu quả của tổ chức. Các bộ phận liên quan bao gồm xử lý đơn hàng, sản xuất và vận chuyển, thiết kế sản phẩm và chiến lược kinh doanh.
6. Yêu cầu 6 Các thuộc tính chất lượng cần thiết của OCVS bao gồm:

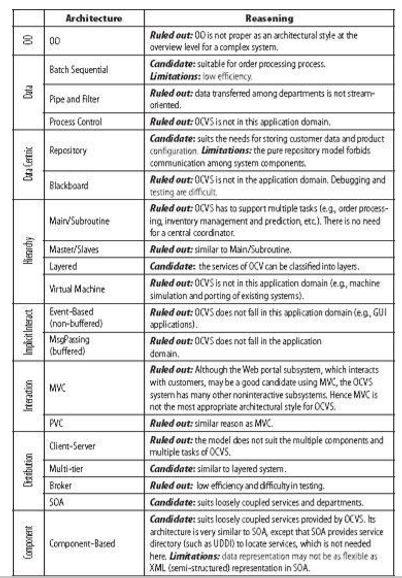
* khả năng mở rộng, để các sản phẩm mới (và quy trình cấu hình của chúng) có thể dễ dàng được thêm vào hệ thống;
* khả năng sửa đổi, để cho phép sửa đổi đại diện nội bộ của sản phẩm và quy trình cấu hình theo thời gian, vì các quy tắc kinh doanh (ví dụ: dự đoán sử dụng bộ phận), trang web sản phẩm và quy trình cấu hình sẽ thường xuyên thay đổi;
* tính khả dụng và độ tin cậy, đặc biệt đối với các mô-đun tương tác với khách hàng và dây chuyền lắp ráp sản xuất; và
* hiệu quả về thời gian, cho tất cả các mô-đun đáp ứng yêu cầu sản xuất một triệu máy tính cá nhân mỗi năm. Yêu cầu thời gian cao điểm của cổng thông tin khách hàng là hỗ trợ đồng thời 10.000 khách hàng tại bất kỳ thời điểm nào.

Nhiệm vụ tiếp theo là xây dựng một bản thiết kế kiến ​​trúc khả thi của OCVS. Thực hiện theo chiến lược từ trên xuống: cung cấp thiết kế kiến ​​trúc tổng thể trước và sau đó phân rã hệ thống thành các thành phần, nơi thiết kế của từng thành phần được trang bị.

### **7.6.1 Thiết kế kiến ​​trúc tổng thể của OCVS**

Nhớ lại rằng quá trình thiết kế kiến ​​trúc bao gồm hai bước: (1) lựa chọn phong cách kiến ​​trúc, và (2) thiết kế kiến ​​trúc. Bây giờ chúng ta phải đưa ra quyết định về (các) phong cách kiến ​​trúc của cấu trúc tổng thể của OCVS. Có rất nhiều phong cách kiến ​​trúc để bạn lựa chọn và một số trong số chúng có thể được sàng lọc một cách dễ dàng. Hình 7.7 minh họa quy trình sàng lọc. OCVS không rơi vào miền ứng dụng của nhiều kiểu kiến ​​trúc được liệt kê, chẳng hạn như điều khiển quy trình (ưu tiên các hệ thống nhúng), đường ống và bộ lọc (yêu cầu luồng dữ liệu giữa các thành phần), chương trình chính/con (không hỗ trợ kiến ​​trúc phân tán), MVC (chỉ phù hợp với thành phần cổng web của OCVS), v.v.

Vì các kiểu kho lưu trữ và tuần tự theo lô thuần túy không hoạt động riêng lẻ cho OCVS, chúng được kết hợp như một kiểu kiến ​​trúc ứng cử viên (không đồng nhất). Do đó, sau khi sàng lọc ban đầu trong Hình 11.7, các kiểu kiến ​​trúc ứng cử viên là: (1) tuần tự theo lô + kho lưu trữ, (2) phân lớp, (3) đa tầng, (4) kiến ​​trúc hướng dịch vụ (SOA) và (5) kiến ​​trúc dựa trên thành phần (CBA)



Hình 7.7. Sàng lọc các kiểu kiến trúc cho OCVS

Hai cặp phong cách kiến trúc rất giống nhau: lớp so với đa tầng và SOA so với CBA. Kiến trúc đa tầng có thể được hiểu đơn giản là phong cách kiến trúc phân lớp trong hệ thống phân tán. Đầu tiên, bạn phải phá vỡ ràng buộc giữa SOA và CBA.

***7.6.1.1 SOA so với CBA***

Trong cả SOA và CBA, các thành phần (“dịch vụ” trong SOA) có các giao diện được xác định rõ. CBA là một kỹ thuật cũ hơn SOA, được hỗ trợ chủ yếu bởi các kỹ thuật phần mềm trung gian như CORBA. Ngoài ra, CBA có các dịch vụ thư mục yếu hơn các dịch vụ được cung cấp trong SOA (chẳng hạn như UDDI, các dịch vụ Mô tả chung, Khám phá và Tích hợp).

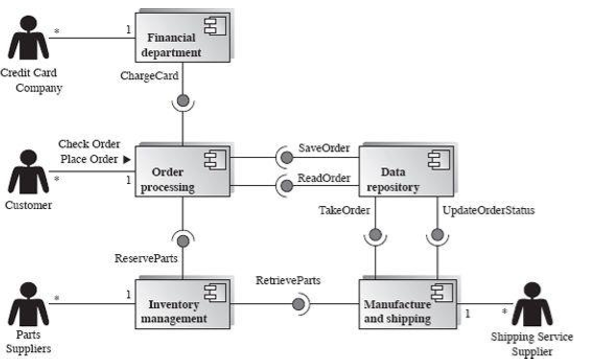
Tuy nhiên, trong trường hợp OCVS, CBA có những lợi thế riêng. Thứ nhất, vì tất cả các thành phần đều nổi tiếng và thường không được công bố ra thế giới bên ngoài, nên OCVS không cần đến nhà môi giới dịch vụ bổ sung và các dịch vụ định vị như UDDI trong SOA. Thứ hai, hiệu quả có thể là một mối quan tâm trong việc thực hiện SOA. Mặc dù tính linh hoạt cao hơn được cung cấp bởi việc sử dụng XML trong việc truyền tham số, SOA về cơ bản là không trạng thái và thiếu khả năng truyền các tham chiếu đối tượng từ xa. Tất cả các tham chiếu đối tượng từ xa phải được tuần tự hóa, điều này tạo ra chi phí mạng lớn hơn nhiều cho việc triển khai SOA (Brown, Johnston và Kelly, 2002). Ngoài ra, chi phí mạng của việc truyền cùng một bản tin bằng SOA lớn hơn đáng kể so với CBA. Do đó, khi xem xét các yêu cầu của OCVS, phong cách kiến ​​trúc CBA được ưu tiên hơn.

***7.6.1.2 Quyết định cuối cùng về việc lựa chọn kiến ​​trúc***

Quyết định được thực hiện để chọn kiến ​​trúc dựa trên thành phần (CBA) thay vì các lựa chọn thay thế khác. Tuần tự theo lô đến kho dữ liệu bị loại trừ do đối số sau: Tuần tự theo lô phù hợp để xử lý đơn hàng; tuy nhiên, nó chỉ hỗ trợ quy trình làm việc của một tác vụ cụ thể. OCVS phải thực hiện đồng thời một số nhiệm vụ chính.

Với chế độ xem nhiều tầng, bạn có thể sử dụng ba tầng sau để lập mô hình hệ thống: (1) front-end, (2) business model và (3) database. Lưu ý rằng nhiều đơn vị chức năng, chẳng hạn như Cổng thông tin điện tử và hệ thống sản xuất, phải được chia thành ba cấp và các thành phần của chúng được trộn lẫn trong cùng một cấp. Do đó, về mặt logic, hệ thống không giống như một sự lựa chọn rõ ràng cho mẫu này. Sự so sánh trước dẫn đến sự lựa chọn của CBA.

Hình 7.8 trình bày thiết kế kiến ​​trúc của cấu trúc tổng thể của OCVS. Hệ thống thường bao gồm năm thành phần: Xử lý đơn đặt hàng, Kho dữ liệu, Quản lý hàng tồn kho, Sản xuất và Vận chuyển, và Phòng Tài chính. Lưu ý rằng mỗi thành phần có thể được triển khai trên một hoặc nhiều máy tính và hệ thống OCVS tổng thể là một hệ thống phân tán. Có bốn tác nhân tương tác với OCVS: Khách hàng, Nhà cung cấp linh kiện, Công ty thẻ tín dụng và Nhà cung cấp dịch vụ vận chuyển. Các thành phần tương tác với nhau thông qua cuộc gọi thủ tục từ xa. Ví dụ, khi bộ phận Xử lý đơn hàng kết thúc giao dịch đặt hàng của một khách hàng, giao diện SaveOrder của Kho dữ liệu được sử dụng để lưu đơn hàng. Trong thời gian chờ đợi, giao diện ReserveParts của Quản lý khoảng không quảng cáo được sử dụng để đặt trước các bộ phận cho đơn đặt hàng đó. Sau đó, thành phần Sản xuất và Vận chuyển sẽ TakeOrder từ Data Repository, và RetrieveParts từ Inventory Management. Bộ phận Sản xuất và Vận chuyển sẽ định kỳ UpdateOrderStatus lên Data Repository để khách hàng có thể theo dõi trạng thái đơn hàng. Khi đơn hàng kết thúc, Nhà cung cấp dịch vụ vận chuyển sẽ liên hệ để giao sản phẩm cho khách hàng.



Hình 7.8. Kiến trúc tổng thể của OCVS

Nhiệm vụ tiếp theo là đánh giá cách thiết kế kiến ​​trúc trong Hình 7.8 thỏa mãn các yêu cầu của OCVS.

1. Yêu cầu 1 (cổng thông tin điện tử của khách hàng) được ánh xạ tới thành phần Xử lý đơn hàng.
2. Yêu cầu 2 (chèn cấu hình sản phẩm) chưa được thiết kế kiến ​​trúc đáp ứng. Các nhà phân tích hệ thống và kiến ​​trúc sư phần mềm có thể đề xuất loại bỏ yêu cầu này. Lập luận là để đạt được Yêu cầu 2, một ngôn ngữ mô tả sản phẩm cụ thể phải được xác định và trình thông dịch ngôn ngữ tương ứng phải được triển khai. Vì OCVS không phát hành quá 20 sản phẩm mỗi năm nên không cần xây dựng hệ thống diễn giải và ngôn ngữ mô tả sản phẩm. Một cách thuận tiện hơn là kỹ sư hệ thống chèn thủ công các mô-đun cấu hình sản phẩm vào hệ thống xử lý đơn hàng.
3. Yêu cầu 3 (hệ thống sản xuất có sự hỗ trợ của máy tính) được ánh xạ tới thành phần Sản xuất và Vận chuyển.
4. Yêu cầu 4 (quản lý khoảng không quảng cáo thông minh) được ánh xạ tới thành phần Quản lý khoảng không quảng cáo.
5. Yêu cầu 5 được phản ánh trong thiết kế của cấu trúc tổng thể trong đó thành phần Kho dữ liệu chịu trách nhiệm duy trì dữ liệu trực tiếp
6. Yêu cầu 6 là về thuộc tính chất lượng. Yêu cầu về khả năng mở rộng có thể được giải quyết bằng cách bổ sung thủ công các mô-đun cấu hình sản phẩm vào thành phần Xử lý đơn hàng (không tốt bằng giải pháp ngôn ngữ mô tả sản phẩm, nhưng ít tốn kém hơn nhiều). Hiệu quả và tính khả dụng có thể đạt được thông qua việc nhân rộng các thành phần, ví dụ, các thành phần Xử lý Đơn hàng và Kho dữ liệu. Thuộc tính chất lượng duy nhất không được xử lý tốt là khả năng sửa đổi. Khi phần trình bày dữ liệu nội bộ của hồ sơ khách hàng hoặc chi tiết đơn hàng bị thay đổi, Kho dữ liệu và các thành phần tương quan khác phải được sửa đổi. Kiến trúc SOA có thể hoạt động tốt hơn một chút do XML được cấu trúc hóa. Tuy nhiên, khả năng sửa đổi được đánh đổi bằng hiệu quả thời gian và sự thuận tiện khi thực hiện.

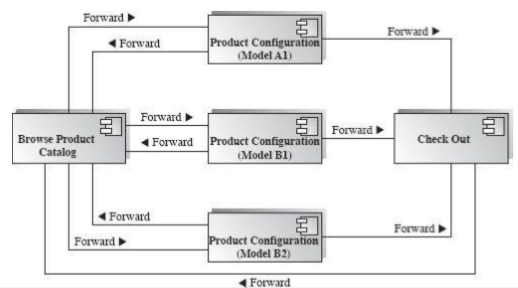
Tiếp theo, chúng tôi sẽ đề cập đến thiết kế kiến trúc của Xử lý đơn hàng, Quản lý hàng tồn kho và các thành phần Sản xuất và Vận chuyển. Phần thảo luận về Bộ phận Tài chính và Kho dữ liệu được bỏ qua ở đây vì thiết kế rất đơn giản: Bộ phận Tài chính liên hệ với các công ty thẻ tín dụng để tính phí thẻ và Kho dữ liệu là một lớp bao bọc đơn giản của cơ sở dữ liệu quan hệ.

### **7.6.2 Thiết kế kiến ​​trúc của thành phần xử lý đơn hàng**

Thành phần Xử lý Đơn hàng tương tự như Hệ thống Xử lý Đơn hàng (OPS) và ví dụ về JSP / Servlet MVC II được thảo luận trong Chương 4. Tóm lại, thành phần Xử lý Đơn hàng đương nhiên nằm trong miền ứng dụng của kiểu kiến ​​trúc tương tác dựa trên web. Có hai lựa chọn về kiến ​​trúc tương tác: MVC và PAC. Mô hình MVC được chọn do tính đơn giản của nó.

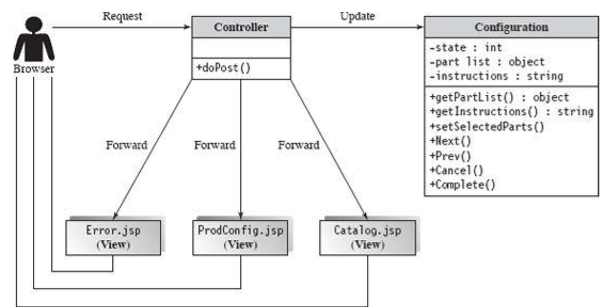
Theo yêu cầu của các bên liên quan của OCV, việc thêm thông tin sản phẩm mới và quy trình cấu hình sản phẩm vào thành phần Xử lý đơn hàng sẽ tương đối dễ dàng. Để đạt được điều này, cấu trúc tổng thể của thành phần được tổ chức bằng cách sử dụng kiến ​​trúc dựa trên thành phần, trong đó mỗi thành phần đại diện cho tập hợp các trang JSP tương quan, các lớp Java Servlet và Java Beans tương ứng cho một “giai đoạn” nhất định trong phiên mua sắm. Các thành phần này không gọi bất kỳ chức năng nào của nhau và kiểu liên kết duy nhất giữa chúng là một thành phần có thể chuyển tiếp (tức là chuyển hướng) đến một thành phần khác.

Hình 7.9 trình bày cấu trúc tổng thể của thành phần Xử lý Đơn hàng. Lưu ý rằng có nhiều thành phần cho cấu hình sản phẩm (ví dụ: A1, B1 và ​​B2). Nếu một sản phẩm mới được phát hành, thành phần cấu hình tương ứng có thể được chèn vào thành phần Xử lý Đơn hàng cùng với tất cả các thành phần cấu hình sản phẩm khác và thành phần Kiểm tra vẫn còn nguyên vẹn. Chỉ thành phần Duyệt danh mục sản phẩm mới cần được biên dịch lại nếu một sản phẩm mới được thêm vào.



Hình 7.9. Các thành phần chính của thành phần Order Processing

Thiết kế kiến trúc của các thành phần Duyệt qua Danh mục Sản phẩm (Browse Product Catalog), Kiểm tra (Check Out) và Cấu hình Sản phẩm (Product Configuration) sẽ tương tự. Tất cả chúng đều sử dụng kiến trúc Model-View-Controller (MVC), cho phép phân tách trình bày và logic nghiệp vụ, do đó linh hoạt trong việc mở rộng và sửa đổi. Ý tưởng cơ bản của MVC là sử dụng một mô hình để duy trì dữ liệu của hệ thống, một khung nhìn để trình bày dữ liệu và một bộ điều khiển để phục vụ như một người trung gian giữa khung nhìn và mô hình. MVC có thể được thực hiện trên nhiều nền tảng bao gồm cả .Net và Java. Ví dụ, trên nền tảng Java, các trang JSP thường được sử dụng làm View, các lớp Java Servlet được sử dụng làm Controller, và các đối tượng Java Bean và cơ sở dữ liệu back-end được sử dụng làm Model. Thiết kế kiến trúc của thành phần Cấu hình Sản phẩm (Product Configuration) được trình bày trong Hình 7.10



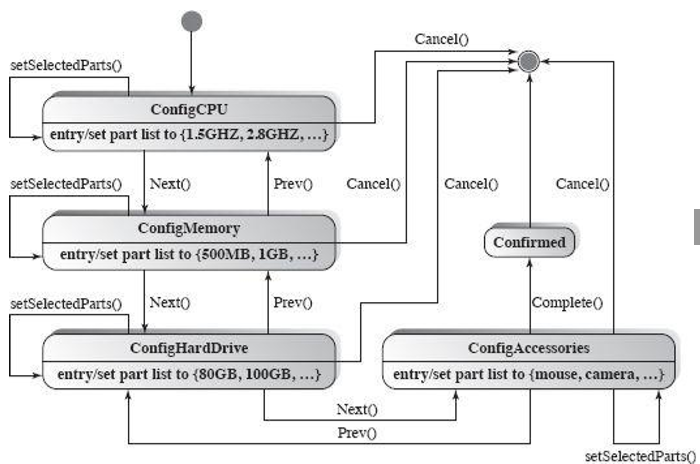
Hình 7.10 Thiết kế kiến trúc của Product Configuration

Trong Hình 7.10, “View” của thành phần Cấu hình Sản phẩm (Product Configuration) bao gồm ba trang JSP, một trang để hiển thị thông báo lỗi, một trang để hiển thị danh mục sản phẩm (thực sự là một phần của thành phần Duyệt danh mục sản phẩm - Browse Product) và trang cuối cùng (ProdConfig.jsp ) để tương tác với khách hàng trong từng bước của quy trình cấu hình. Mỗi trang JSP có nội dung HTML tĩnh, được thiết kế bởi các nhà thiết kế đồ họa, có thể thay đổi thường xuyên. Nội dung động được thiết lập bởi Controller Servlet. Ví dụ: số bước của quy trình cấu hình, hướng dẫn cho bước cấu hình, cũng như danh sách các bộ phận của các nhà cung cấp khác nhau để khách hàng lựa chọn, tham số hóa ProdConfig.jsp. Tất cả những nội dung này cuối cùng được truy xuất từ mô hình, một Java Bean được gọi là Cấu hình (Configuration).

Mô hình hiển thị trong Hình 7.10 hoạt động như sau. Sau khi khách hàng chọn một mẫu sản phẩm trong thành phần Duyệt qua Danh mục sản phẩm (Browse Product), controller của Duyệt danh mục sản phẩm (Browse Product) sẽ chuyển hướng trình duyệt đến ProdConfig.jsp. Bộ điều khiển Servlet được tạo và đến lượt nó, tạo ra bean Cấu hình mô hình. Sau đó, trang ProdConfig.jsp giúp khách hàng định cấu hình máy tính từng bước — chọn kiểu CPU, loại và kích thước bộ nhớ, cạc đồ họa, v.v. Khách hàng có thể chọn chuyển sang bước tiếp theo hoặc quay lại bước trước bươc. Khi một bước kết thúc, danh sách các phần được chọn trong bước đó sẽ được chuyển tiếp đến

Bộ điều khiển (Controller) và sau đó đến Cấu hình. Khi toàn bộ cấu hình được hoàn thành và được người dùng xác nhận, yêu cầu AddToCart được gửi đến Controller và cơ sở dữ liệu trung tâm sẽ được cập nhật.

Lựa chọn triển khai các mô-đun của Cấu hình Sản phẩm là hiển nhiên, đó là mô hình hướng đối tượng. Một sơ đồ trạng thái máy có thể nắm bắt được thiết kế của cấu hình một cách đẹp mắt, và một ví dụ được thể hiện trong Hình 7.11. Mỗi trạng thái của sơ đồ trạng thái máy đại diện cho một bước trong cấu hình. Khi máy trạng thái vào một trạng thái, biến danh sách bộ phận được đặt tương ứng cho bước đó. Người dùng có thể lùi và tiến giữa các trạng thái. Từ mỗi trạng thái, một lệnh gọi của hoạt động Hủy sẽ khiến máy trạng thái kết thúc ở trạng thái cuối cùng. Lưu ý rằng thiết kế trong Hình 7.11 không cung cấp tất cả các chi tiết. Ví dụ, để cho phép khách hàng lùi lại, cần thiết lập một ngăn xếp để ghi lại lịch sử của các bộ phận đã chọn. Khi máy trạng thái chuyển sang trạng thái tiếp theo, danh sách bộ phận mới nhất do khách hàng gửi sẽ được đẩy vào ngăn xếp; khi máy trạng thái lùi lại, một hoạt động bật phải được tiến hành. Một vấn đề khác với thiết kế hiện tại là từ mỗi trạng thái, quá trình chuyển đổi Tiếp theo sẽ bị vô hiệu hóa nếu người dùng chưa gửi các phần được chọn cho bước đó. Bạn có thể tinh chỉnh thiết kế trong Hình 7.11.



Hình 7.11. Biểu đồ máy trạng thái của lớp Cấu hình (Configuration)

### **7.6.3 Thiết kế kiến ​​trúc của quản lý hàng tồn kho**

Như thể hiện trong Hình 7.8, thành phần Quản lý Hàng tồn kho có các trách nhiệm sau:

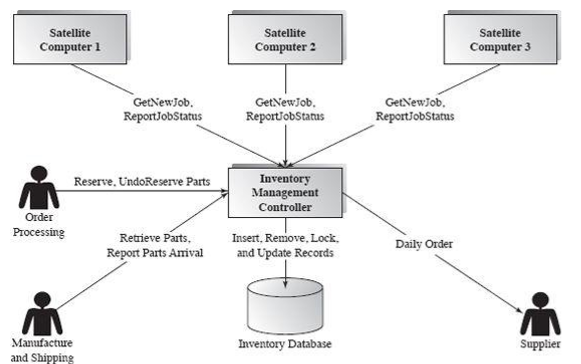
* Nó duy trì các bản ghi của tất cả các bộ phận (ví dụ: CPU, chip bộ nhớ, ổ cứng, màn hình, v.v.). Kho của OCVS được quản lý kỹ thuật số. Mỗi mặt hàng có đầy đủ hồ sơ trong hệ thống cơ sở dữ liệu của Quản lý hàng tồn kho (Inventory Management) và mỗi mặt hàng (không phải loại của nó) có một mã vạch riêng.
* Nó cho phép khách hàng đặt trước các bộ phận cho một đơn đặt hàng. Khi đặt trước thành công, mã vạch của các mặt hàng đã đặt trước được liên kết với đơn đặt hàng. Khách hàng nhận được phản hồi ngay lập tức khi các bộ phận họ đặt hàng không có sẵn. Khi khách hàng lùi lại cấu hình sản phẩm, khóa đặt trước trên các bộ phận tương ứng sẽ được giải phóng bằng cách gọi thao tác UndoReserve.
* Nó cho phép bộ phận Sản xuất (Manufacture) và Vận chuyển (Shipping) kiểm tra các bộ phận bằng cách gọi RetrieveParts. Mỗi công nhân trong kho đều có bàn làm việc được trang bị máy tính và máy quét mã vạch. Khi lệnh gọi RetrieveParts từ xa xảy ra, Quản lý khoảng không quảng cáo tạo ra một lệnh công việc, trong đó chỉ định các mục cần kiểm tra và điểm đến của chúng, đồng thời gán mã vạch và hộp gói thư nội bộ cho lệnh công việc. Một công nhân nhận được đơn đặt hàng công việc qua máy tính, chọn các mặt hàng yêu cầu, lắp ráp sản phẩm và sau đó gửi đi. Bất cứ khi nào một mặt hàng được đưa vào một gói hàng, mã vạch của nó trước tiên phải được quét. Khi hộp gói hàng đến Xưởng sản xuất & Vận chuyển, mã vạch của gói hàng sẽ được quét lại để xác nhận việc hoàn thành đơn đặt hàng.
* Nó thường xuyên kiểm tra hàng tồn kho của tất cả các bộ phận. Dựa trên lịch sử hàng tồn kho, nó đưa ra đơn đặt hàng hàng ngày cho từng loại phụ tùng cho các nhà cung cấp. Mỗi nhà cung cấp đã thiết lập một máy chủ giao dịch phù hợp với giao diện chuỗi cung ứng của OCVS.

Từ những yêu cầu trên, có thể dễ dàng suy ra rằng việc quản lý hàng tồn kho cần bao gồm:

* một cơ sở dữ liệu lưu trữ thông tin về tất cả các bộ phận được lưu trữ trong kho;
* một hệ thống điều khiển có thể chấp nhận yêu cầu từ xa từ các bộ phận Xử lý Đơn hàng (Order Processing) và Sản xuất (Manufacture) và Vận chuyển (Shipping); và
* máy tính vệ tinh (với máy quét mã vạch) được sử dụng bởi công nhân.

Bước tiếp theo là quyết định cách tổ chức các mô-đun này. Miền ứng dụng của Quản lý khoảng không quảng cáo sớm dẫn đến phong cách kiến ​​trúc không đồng nhất: sự kết hợp giữa kho dữ liệu và kiểu máy chủ khách. Sơ đồ khối trong Hình 7.12 hiển thị cấu trúc chung của Quản lý Hàng tồn kho (Inventory Management). Cả Bộ điều khiển Inventory Managementv và Satellite Computers (Máy tính Vệ tinh) đều tương tác với cơ sở dữ liệu quan hệ để chèn, xóa và cập nhật các bản ghi bộ phận máy tính. Mối quan hệ giữa Máy tính Vệ tinh và Bộ điều khiển Quản lý Hàng tồn kho ((Inventory Management).) là máy khách/máy chủ. Bộ điều khiển (Inventory Management) lưu giữ một danh sách các lệnh công việc.

Khi một Satellite Computer (Máy tính Vệ tinh) đã hoàn thành một công việc, nó sẽ truy xuất các chi tiết của một thứ tự công việc khác và hiển thị công việc đó cho nhân viên.



Hình 7.12 Thiết kế kiến trúc của thành phần Quản lý kho

Rõ ràng, Bộ điều khiển Quản lý kho có thể được phát triển bằng cách sử dụng mô hình OO và Máy tính vệ tinh có thể được phát triển bằng kiến trúc MVC (vì nó tương tác với công nhân). Chúng ta sẽ xem xét chi tiết thiết kế của hai mô-đun này dưới dạng bài tập.

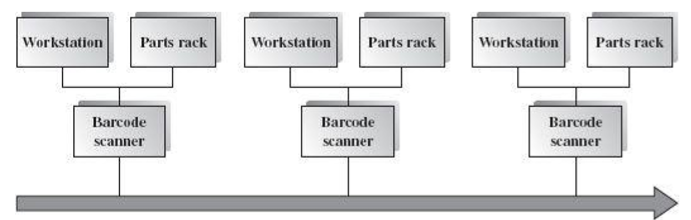
### **7.6.4 Thiết kế kiến trúc của thành phần sản xuất và vận chuyển**

Nhiệm vụ cuối cùng của chương này là thiết kế kiến trúc cho thành phần Sản xuất và Vận chuyển. Hệ thống con này phải quản lý luồng thông tin của bộ phận sản xuất, bộ phận chịu trách nhiệm sản xuất hơn một triệu chiếc PC mỗi năm

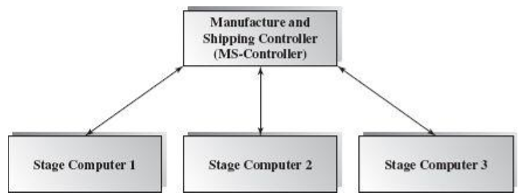
Cơ sở Thiết bị chính của bộ phận sản xuất là dây chuyền lắp ráp.

Ý tưởng chính của dây chuyền lắp ráp là tận dụng nguyên tắc kinh tế “phân công lao động”. Nếu các nhà quản lý chia nhỏ việc sử dụng lao động thành các nhiệm vụ được xác định rõ ràng, chuyên biệt và lặp đi lặp lại, họ có thể tăng hiệu quả của đầu ra lên rất nhiều. Cấu trúc của dây chuyền lắp ráp OCV được trình bày trong Hình 7.13. Thành phần chính của nó là băng tải, có chức năng chuyển vật liệu về phía trước. Nhiều công đoạn được gắn vào băng tải (số lượng công đoạn có thể thay đổi và có thể được điều chỉnh linh hoạt). Phần cứng của mỗi giai đoạn bao gồm:

* Một máy trạm cung cấp không gian để công nhân lắp ráp các bộ phận. Máy trạm được trang bị một máy tính (được gọi là “máy tính giai đoạn”) hiển thị chi tiết công việc cho người vận hành và cho phép người vận hành gửi thông điệp đến bộ điều khiển trung tâm (ví dụ: báo cáo lỗi kiểm tra, sự cố của các bộ phận, v.v.).
* Giá đỡ các bộ phận để giữ các bộ phận cần thiết cho sân khấu. Bộ phận kiểm kê điền vào giá các bộ phận. Với sự hỗ trợ của OCVS, nó đảm bảo rằng khi khung máy tính đến một giai đoạn, các bộ phận cần thiết ở giai đoạn này đã nằm trong giá linh kiện.
* Máy quét mã vạch được sử dụng: (1) để kiểm tra các bộ phận khi chúng được giao bởi bộ phận kiểm kê; (2) để kiểm tra khung máy tính khi nó đến; và (3) để kiểm tra khung máy tính khi công việc hoàn thành ở giai đoạn này.



Hình 7.13 Sơ đồ trực quan của dây chuyền lắp ráp OCVS



Hình 7.14. Kiến trúc của thành phần Sản xuất và Vận chuyển

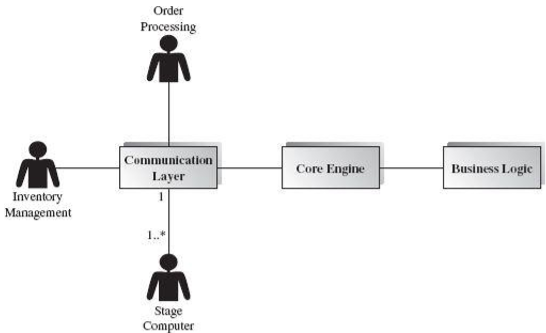
**Cấu trúc tổng thể của thành phần sản xuất và vận chuyển:** Mục tiêu của dự án là điều phối luồng thông tin để đẩy nhanh quá trình sản xuất. Phạm vi của thành phần Sản xuất và Vận chuyển bao gồm bộ điều khiển trung tâm tương tác với các thành phần khác của OCVS và các máy tính giai đoạn (với máy quét mã vạch); điều khiển ròng rọc băng tải ra khỏi phạm vi.

Với các yêu cầu và bản chất của dây chuyền lắp ráp, một lựa chọn tự nhiên cho cấu trúc tổng thể của thành phần Sản xuất và Vận chuyển là kiến ​​trúc máy khách-máy chủ, được thể hiện trong Hình 7.14. Phần máy chủ trong thiết kế được gọi là MS-Controller.

**Thiết kế của MS-Controller** Nhớ lại rằng trách nhiệm của MS-Controller là: (1) tương tác với các thành phần Xử lý Đơn hàng và Quản lý Hàng tồn kho để chuỗi cung ứng suôn sẻ; và (2) quản lý tất cả các máy tính công đoạn để điều phối quá trình sản xuất, chẳng hạn như hiển thị chi tiết công việc cho người vận hành khi khung máy tính đến công đoạn hoặc chấp nhận các báo cáo ngoại lệ từ người vận hành.

Vì MS-Controller chịu nhiều trách nhiệm nên một kiến ​​trúc phân lớp có thể được sử dụng để đơn giản hóa thiết kế của nó. Ý tưởng cơ bản là chia MS-Controller thành ba lớp: (1) một lớp giao tiếp chấp nhận các thông điệp được gửi bởi các máy tính giai đoạn và giao tiếp với các thành phần từ xa khác như Xử lý Đơn hàng và Quản lý Hàng tồn kho; (2) lớp động cơ cốt lõi xử lý các yêu cầu từ máy tính giai đoạn và kiểm soát toàn bộ quy trình sản xuất; và (3) lớp logic nghiệp vụ xác định và giải thích các quy tắc nghiệp vụ.

Hình 7.15 cho thấy thiết kế kiến ​​trúc của MS-Controller. Lớp giao tiếp phải giao tiếp với các máy tính / thành phần từ xa.

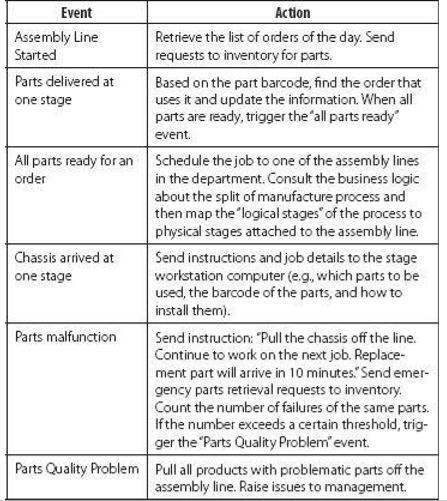


Hình 7.15. Thiết kế kiến trúc của MS-Controller

Các yêu cầu từ thế giới bên ngoài được chuyển tiếp đến bộ máy cốt lõi để xử lý. Mô-đun nghiệp vụ nằm phía sau tham số hóa quá trình xử lý của công cụ cốt lõi.

Bước tiếp theo là đi sâu vào thiết kế của ba mô-đun của MS-Controller. Các chi tiết thiết kế của Lớp Giao tiếp (Communication) và Logic nghiệp vụ (Business Logic) được giao cho bạn dưới dạng nhiệm vụ. Sau đây, trọng tâm là thiết kế của Core Engine.

Core Engine phải thực hiện nhiều nhiệm vụ khác nhau. Ví dụ, khi các dây chuyền lắp ráp được bắt đầu, nó có trách nhiệm truy xuất các đơn đặt hàng công việc hàng ngày từ Kho dữ liệu. Nếu có bất kỳ sự cố nào xảy ra (ví dụ: card âm thanh bị trục trặc), Core Engine phải gửi hướng dẫn tới giai đoạn để thực hiện các hành động ngay lập tức (ví dụ: kéo sản phẩm ra khỏi dòng và tạm thời dừng tất cả các sản phẩm có cùng số model). Có một số lựa chọn thay thế khả thi cho việc thiết kế Core Engine. Chúng bao gồm bảng đen (Blackboard) và mẫu kiến ​​trúc dựa trên sự kiện. Tuy nhiên, giải pháp tốt nhất là sử dụng kiến ​​trúc thông điệp được đệm (buffered message architecture). Có thể dễ dàng thiết lập dịch vụ đăng ký thông điệp bằng cách sử dụng các dịch vụ nhắn tin như Java Message Service. Sau đó, các sự kiện có thể được thêm vào dịch vụ đăng ký thông điệp và các trình xử lý được đăng ký. Hình 7.16 trình bày một số sự kiện mẫu và các hành động tương ứng sẽ được thực hiện.



Hình 7.16. Các sự kiện mẫu và trình xử lý của Core Engine

Một đặc điểm nổi bật của hệ thống thông điệp đệm là tính linh hoạt của nó. Các sự kiện và trình xử lý mới có thể được xác định một cách thuận tiện mà không làm gián đoạn các mô-đun hiện có. Mặc dù quyết định về kiến ​​trúc của Core Engine là sử dụng kiểu thông báo đệm, nhưng thiết kế chi tiết của nó có thể được thực hiện một cách thuận tiện bằng cách sử dụng phương pháp OO.

## **7.7 Tóm tắt**

Lựa chọn thiết kế kiến ​​trúc tốt nhất cho một hệ thống phần mềm thực sự là một nghệ thuật. Chương này giới thiệu một số phương pháp luận để đưa ra các quyết định về kiến ​​trúc và đánh giá các thiết kế kiến ​​trúc thay thế. Các nguyên tắc chung đã được thực hành trong một nghiên cứu điển hình về xây dựng hệ thống nhà cung cấp máy tính trực tuyến. Các kiểu kiến ​​trúc khác nhau như dựa trên thành phần, hướng dịch vụ, tuần tự theo lô, Model-View-Controller, máy khách-máy chủ, kho dữ liệu và hệ thống thông báo đệm đã được xem xét trong thiết kế của nó. Bạn được khuyến khích hoàn thành các mô-đun chưa hoàn thành của hệ thống nhà cung cấp máy tính trực tuyến. Hãy nhớ rằng: chuyên môn đến từ kinh nghiệm!

# CHƯƠNG 8: KIẾN TRÚC VÀ MẪU THIẾT KẾ - DESIGN PATTERN

Mục tiêu của chương này:

* Giới thiệu về khái niệm, vai trò của mẫu thiết kế
* Các mẫu thiết kế

## **8.1. Khái niệm mẫu thiết kế**

Trong phát triển phần mềm hiện đại, kiến trúc tổng thể của dự án đóng một vai trò quan trọng, đặc biệt với bộ khung (framework) và mẫu thiết kế (design pattern). Sử dụng mẫu thiết kế là một giải pháp đã được áp dụng rộng rãi để giải quyết những vấn đề thường hay xảy ra giống nhau trong thực tế. Khái niệm mẫu thiết kế lần đầu tiên được đề xuất trong ngành xây dựng bởi kiến trúc sư Christopher Alexander vào những năm 1970. Ông cho rằng mẫu thiết kế là một giải pháp hiệu quả để giải quyết nhiều vấn đề thường hay lặp đi lặp lại trong thực tế kiến trúc.

Tại một Hội nghị về ngôn ngữ lập trình Hướng đối tượng năm 1987, Kent Beack và War Cunningham đã trình bày áp dụng những ý tưởng của Alexander cho thiết kế và phát triển phần mềm đặc biệt dành cho lập trình hướng đối tượng. Bài báo này đã thu hút nhiều quan tâm nghiên cứu xây dựng mẫu thiết kế cho phát triển phần mềm. Đến năm 1994, cuốn sách “Design Pattern: Elements of Reusable Object – Oriented Software” của Gang of Four (GoF) – Erich Gamma, Richard Helm, Ralph Johnson and John Vlissdes – với đề xuất 23 mẫu thiết kế đã lan rộng và ảnh hưởng mạnh mẽ ứng dụng của các mẫu thiết kế. Để hiểu mẫu thiết kế là gì, chúng ta hãy xem xét một số ý kiến sau đây:

* Mẫu thiết kế là một cách hiệu quả đề truyền tải những thiết kees có chất lượng. Nó có thể là tự vựng của một phần tử thiết kế để giải quyết vấn đề đang gặp phải hay là một ngôn ngữ chung để trao đổi kinh nghiệm
* Mẫu thiết kế không phải là một phát minh mà đúng ra là một phần tài liệu bàn về cách giải quyết một vấn đề được xem là hữu hiệu nhất qua kinh nghiệm xây dựng nhiều hệ thống phần mềm.
* Không phải cái gì tốt nhất trong thực tế đều được xem là mẫu mà phải thoả mãn Quy luật “ít nhất ba”. Luật này cho rằng mẫu đó phải được chứng tỏ là lặp đi lặp lại và là tốt nhất trong ít nhất ba hệ thống đã xây dựng. Ý nghĩa của luật này là nhằm đảm bảo rằng đã có một số chuyên gia phần mềm ứng dụng giải pháp đề xuất để giải quyết một số vấn đề trong thiết kế phần mềm rồi.

Mặc dù mẫu thiết kế thường được xem xét trong bối cảnh phát triển phần mềm hướng đối tượng nhưng chúng có thể được áp dụng trong nhiều lĩnh vực khác với các cách tiếp cận lập trình khác nhau. Chỉ một số thay đổi nhỏ, một mẫu thiết kế có thể được điều chỉnh để tạo ra mẫu thiết kế mới.

Các mẫu thiết kế không cung cấp các giải pháp cho mọi vấn đề trong thiết kế và phát triển phần mềm thực tế mà chỉ đưa ra giải pháp tái sử dụng để giải quyết các vấn đề phát triển phần mềm thường gặp trong một bối cảnh cụ thể nào đó. Điều này có ý nghĩa rằng nó chỉ là các mô hình cung cấp các giải pháp tốt nhất cho một vấn đề trong một ngữ cảnh cụ thể nào đó nhưng không thể hiệu quả trong ngữ cảnh khác. Vì vậy, các giải pháp của các mẫu thiết kế được đề xuất có thể áp dụng trong ngữ cảnh này nhưng không áp dụng được trong một số ngữ cảnh khác.

Mẫu thiết kế và Framework

Mẫu thiết kế không tồn tại dưới dạng thành phần phần mềm mà phải được cài đặt tại thời điểm sử dụng, không nên nhầm lẫn giữa framework với mẫu thiết kế. Framework là một dạng phần mềmm gồm các thành phần phối hợp hoạt động với nhau để cung cấp một kiến trúc cho một miền ứng dụng nên nó có thể cài đặt nhiều mẫu thiết kế. Bảng 8.1 sau đây nêu lên một số đặc điểm thể hiện sự khác biệt này [10]

Bảng 7.1 Mẫu thiết kế và Framework

|  |  |
| --- | --- |
| Mẫu thiết kế | Framework |
| Mẫu thiết kế là giải pháp lặp đi lặp lại đối với vấn đề nảy sinh trong vòng đời của ứng dụng phần mềm | Framwork là nhóm thành phần cộng hợp với nhau để cung cấp một kiến trúc có thể sử dụng lại những ứng dụng |
| Mục đích là:   * Giúp cải tiến chất lượng phần mềm (sử dụng lại, bảo trì, mở rộng,…) * Giảm thời gian phát triển | Mục đích là:   * Giúp cải tiến chất lượng phần mềm (sử dụng lại, bảo trì, mở rộng,…) * Giảm thời gian phát triển |
| Mẫu là một cấu trúc logic | Framework là một dạng phần mềm |
| Mô tả mẫu thường độc lập với ngôn ngữ lập trình và chi tiết cài đặt | Framework đặc trưng cho cài đặt |
| Mẫu là tổng quát hơn và có thể sử dụng trong nhiều ứng dụng | Framework cung cấp chức năng cho miền cụ thể |
| Mẫu thiết kế không tồn tại dưới dạng thành phần phần mềm. Nó cần được cài đặt mỗi khi sử dụng | Framework không phải là một ứng dụng đầy đủ, các ứng dụng có thể xây dựng bằng cách kế thừa từ các thành phần có sẵn |
| Mẫu thiết kế là cách thiết kế tốt và được sửu dụng để thiết kế các Framework | Mẫu thiết kế có thể sử dụng trong thiết kế và cài đặt các Framework nghĩa là Framework có thể chứa nhiều mẫu thiết kế |

## **8.2. Định dạng mẫu thiết kế**

Vì các mẫu thiết kế được sử dụng bởi nhiều người nên chúng cần được thể hiện theo một định dạng được chấp nhận rộng rãi. Nghĩa là, kiểu định dạng phải dễ hiểu và tổng quát để có thể cài đặt với nhiều ngôn ngữ lập trình khác nhau. Hơn nữa, vấn đề quan trọng là cần phải thể hiện được cách sử dụng các mẫu này trong thực tế ngữ cảnh tương ứng. Sau đây là một số đặc trưng của các mẫu thiết kế:

- Tên mẫu: Tên viết ngắn gọn của mẫu thường là một hoặc hai từ. Ví dụ, tên mẫu phải chọn làm sao thể hiện được mục đích của mô hình và được sửu dụng như một tên duy nhất đặc trưng cho mẫu đó.

- Phân loại: Các mẫu được phân chia thành nhiều loại khác nhau. Nhóm mẫu tạo dựng (creational) là các mẫu tập trung vào cách tạo các đối tượng, nhóm mẫu cấu trúc (structural) tập trung vào việc sắp xếp các đối tượng với nhau thành một cấu trúc tổng thể lớn hơn; nhóm mẫu hành vi (behaviroal) liên quan đến hành vi, cộng tác giữa các đối tượng để thực hiện một mục tiêu nào đó.

- Mục đích: một mô tả chung ngắn gọn (một hoặc hai câu) về mô hình. Ví dụ mẫu *Observer* mô tả phụ thuộc 1-nhiều giuawx các đối tượng để khi một đối tượng thay đổi trạng thái thì tất cả phụ thuộc của nó có thể nhận biết và cập nhật tự động

- Định danh: Các mẫu đều có định danh của mẫu đó.

- Động lực: Xác định vấn đề mà mẫu thiết kế này có thể được sử dụng để giải quyết

- Khả năng áp dụng: Lĩnh vực mà mẫu thiết kế này có thể được sử dụng để giải quyết

- Cấu trúc: những biểu đồ lớp và thành phần với UML nhằm thể hiện cách hoạt động mẫu thiết kế này

- Các bên tham gia: Xác định các đối tượng liên quan hoặc những công việc mà đối tượng đó được yêu cầu thực hiện

- Cộng tác: Xác định sự cộng tác của các bên tham gia

- Kết quả: Lợi ích và hạn chế của mẫu thiết kế này (thêm những gợi ý để giải quyết các vấn đề về hạn chế).

- Cài đặt: những gợi ý, chỉ dẫn về cách sử dụng mẫu này bao gồm cả những thủ thuật, các cách ứng dụng hiệu quả và những điều nên tránh

- Mã nguồn mẫu: có ví dụ triển khai đầy đủ cho mẫu này được viết dưới dạng ngôn ngữ C++, Java, C#...

- Các áp dụng: Ví dụ về các lĩnh vực mà mẫu này được áp dụng trong thực tế

- Các mẫu liên quan: Các mẫu thiết kế tương tự như mẫu này hoặc cũng mang lại hiệu quả tương tự khi sử dụng như mẫu hiện thời.

**8.3. Phân loại mẫu thiết kế**

Từ khi các mẫu thiết kế được đưa ra bởi nhóm GoF, đã có nhiều mẫu được đề xuất thêm. Tài liệu này trình bày một số mẫu thông dụng được đưa ra bởi nhóm GoF. Hệ thống các mẫu này có thể nói là khá đầy đủ cho những người mới làm quen với mẫu thiết kế để giải quyết các vấn đề của thiết kế và phát triển phần mềm hiện nay. Hệ thống 23 mẫu thiết kế của GoF được chia thành 3 nhóm: Nhóm tạo dựng, nhóm cấu trúc và nhóm hành vi.

**Nhóm mẫu tạo dựng**

* Mục đích của mẫu nhóm này nhằm giải quyết công việc thường xuyên là tạo ra các đối tượng
* Các mẫu sẽ tạo ra một cơ chế đơn giản, thống nhất khi tạo các thể hiện của đối tượng
* Cho phép đóng gói các chi tiết về các lớp nào được khởi tạo và cách các thể hiện này được tạo ra.
* Nó khuyến khích sử dụng interface nhằm giảm độ liên kết
* Nhóm này gồm các mẫu thiết kế sau đây: *Abstract Factory, Factory Method, Builder, Protype, và Singleton*

**Nhóm mẫu cấu trúc**

* Nhóm này chủ yếu giải quyết vấn đề một đối tượng uỷ thác trách nhiệm cho những đối tượng khác. Điều này dẫn đến kiến trúc phân tầng của các thành phần với độ kết nối thấp.
* Tạo điều kiện giao tiếp giữa các đối tượng khi một đối tượng không thể truy nhập đối tượng khác theo cách thông thường hay khi một đối tượng không thể sử dụng được do giao diện không tương thích.
* Cung cấp các cách để cấu trúc một đối tượng với quan hệ hợp thành được tạo ra đầy đủ. Nhóm này liên quan tới các quan hệ cấu trúc giữa các thể hiện, bằng cách sử dụng các quan hệ kế thừa, liên kết và phụ thuộc. Để xem thông tin về mẫu này phải dựa vào biểu đồ lớp của mẫu
* Nhóm này gồm các mẫu sau: *Adapter, Bridge, Composite, Decorator, Façade, Proxy and Flyweight.*

**Nhóm mẫu hành vi**

* Nhóm mẫu thiết kế này liên quan đến các quan hệ gán trách nhiệm để cung cấp các chức năng giữa các đối tượng trong hệ thống
* Mô tả cơ chế giao tiếp giữa các đối tượng và xác định cơ chế chọn các thuật toán khác nhau bởi các đối tượng khác nhau tại thời gian chạy
* Đối với các mẫu thuộc nhóm này ta có thể dựa vào biểu đồ giao tiếp và biểu đồ tuần tự để giải thích cách chuyển giao của các chức năng
* Nhóm này gồm có một số mẫu sau đây: *Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy and Visitor.*

Ngoài ra, hai mẫu interface và abstract được xem là thuộc nhóm mẫu cơ bản và đã được sử dụng rộng rãi trong thiết kế các ngôn ngữ lập trình như Java. Nhiều ví dụ với mã nguồn đầy đủ về các mẫu thiết kế bạn đọc có thể tham khảo tại trang: https://www.tutorialspoint.com/design\_pattern/index.htm

## **8.4. Sử dụng mẫu thiết kế**

### **8.4.1. Khi nào sử dụng mẫu thiết kế**

Mẫu thiết kế là mô tả tổng quát của một giải pháp đối với một vấn đề được “lặp đi lặp lại” trong các dự án phần mềm. Như vậy, một mẫu thiết kế có thể được xem như là một “khuôn mẫu” có sẵn được áp dụng để giải quyết một vấn đề trong các tình huống khác nhau.

Mẫu thiết kế được hiểu theo nghĩa tái sử dụng ý tưởng hơn là tái sử dụng mã lệnh. Mẫu thiết kế cho phép các nhà thiết kế ngồi lại cùng nhau để giải quyết một vấn đề nào đó mà không phải mất nhiều thời gian tranh cãi. Nhiều dự án phần mềm thất bại là do những người phát triển không có được sử hiểu biết chung về các vấn đề trong kiến trúc phần mềm. Do đó, mẫu thiết kế có thể sử dụng như một nguồn cung cấp những thuật ngữ và khái niệm để nhanh chóng hình dung ra “bức tranh” của giải pháp trong quá trình thiết kế. Hơn nữa nếu mẫu thiết kế được sử dụng một cách hiệu quả thì dễ dàng đưa ra được kiến trúc hệ thống ban đầu và việc bảo trì phần mềm sau này cũng được tiến hành thuận lợi hơn.

Mẫu thiết kế hỗ trợ tái sử dụng kiến trúc và mô hình thiết kế phần mềm quy mô lớn. Cần phân biệt mẫu thiết kế với framework. Framework hỗ trợ tái sử dụng mô hình thiết kế và mã nguồn ở mức chi tiết hơn. Trong khi đó, mẫu thiết kế được vận dụng ở mức tổng quát hơn, giúp các nhà phát triển hình dung và nhận biết các cấu trúc tĩnh và động cũng như quan hệ tương tác giữa các giải pháp trong quá trình thiết kế ứng dụng đối với một lĩnh vực riêng biệt

Mẫu thiết kế là đa tương thích nghĩa là nó không phụ thuộc vào ngôn ngữ lập trình, công nghệ hoặc các nền tảng lớn như J2EE của Sun hay của Microsoft.NET. Tiềm năng ứng dụng của mẫu thiết kế là rất lớn. Các kiến trúc dựa trên mẫu thiết kế đã được sử dụng khác nhiều trong các phần mềm mã nguồn mở, trong nền tảng J2EE hoặc .NET …Trong các dạng ứng dụng này, có thể dễ dàng nhận ra một số tên lớp chứa các mẫu thiết kế như Factory, Proxy, Adapter…

### **8.4.2. Sử dụng mẫu thiết kế như thế nào**

Các mẫu thiết kế cần phải thay đổi để phù hợp với tình huống sử dụng cụ thể. Sau đây là một số lý do vì sao cần sự điều chỉnh như vậy:

* Sự khác biệt về ngôn ngữ biên dịch: các ngôn ngữ lập trình kể cả các ngôn ngữ hướng đối tượng đều có nuhwngx khác biệt nhau như cú pháp, biên dịch…Do đó, các mẫu thiết kế cần phải hiệu chỉnh sao cho phù hợp với ngôn ngữ đang sử dụng cho dự án
* Sự khác biệt về quan điểm: thông thường, có thể có nhiều giải pháp đối với mỗi vấn đề đặt ra trong khi thiết kế. Do đó, một mẫu thiết kế được đưa ra bởi một chuyên gia hoặc một nhóm các chuyên gia không có nghĩa là nó đã là hoàn hảo.
* Sự khác biệt về kiểu: Thiết kế và kiểu mã hoá có thể nảy sinh từ nhiều nguồn như kinh nghiệm, nguyên tắc viết mã của doanh nghiệp, các yêu cầu của một thư viện hoặc framework. Nếu một mô hình không phù hợp với kiểu mà dự án sử dụng thì chúng ta phải điều chỉnh nó.
* Vấn đề khác nhau: Mỗi một mẫu thiết kế phải được thể hiện hết sức tổng quát nên một số mô hình lớp trình bày trong mẫu có thể sẽ không được sử dụng. Ví dụ, trong mẫu thiết kế *Observer*, các lớp *ConcreteObserver* và *ConcreteSubject* có thể không được sử dụng vì không phù hợp với tình huống thiết kế của chúng ta.
* Thành phần, kế thừa và đa kế thừa: các ngôn ngữ hướng đối tượng và nhà phát triển thường có sự khác biệt trong cách sử dụng kế thừa và thành phần. Ví dụ, trong C++ có đa kế thừa nhưng Java thì lại không có
* Đơn giản hoá: Một mẫu thiết kế thường là tổng quát hoá cho nhiều tính năng và do đố có thể sẽ phức tạp hơn thực tế thiết kế của yêu cầu dự án đề ra. Vì vậy, chúng ta có thể làm đơn giản mẫu đó nhằm mang lại hiệu quả cao hơn.