**TÀI LIỆU KIẾN TRÚC HƯỚNG THÀNH PHẦN**

**Phần mềm Trường mầm non Hoa Trạng Nguyên -Thái Nguyên**

**Lịch sử sửa đồi**

|  |  |  |  |
| --- | --- | --- | --- |
| Ngày | Phiên bản | Mô tả | Người viết |
| 11/11/2021 | Phác thảo | Phác thảo ban đầu | Nguyễn Đức Tuân |
|  |  |  |  |
|  |  |  |  |

## Kiến trúc dựa trên hướng thành phần

### 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 bạn sẵn (commercial off-the-shelf - COTS) có thể tái sử dụng cho hệ thống thiện tại và chúng có thể tạo ra các thành phần tái sử dụng trong tương lai. Điều này làm thăng 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 sử dụng lại bằng cách kéo và thả. Nhiều thành phần là vô hình, đặc biệt là sử dụng 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 cáp cho khách hàng của nó; bản thân nói cũng có thể yêu cầu các dịch vụ từ 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. 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 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.

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

### Thành phần (cấu phần) là gì?

Có một sự thống nhất chung trong phát triển phần mềm hướng cấu phần rằng một thành phần 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.

*“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 Somerville, 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.

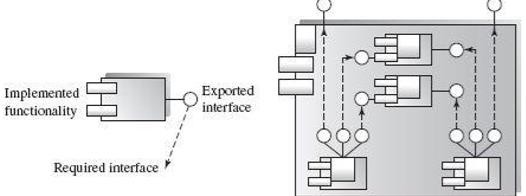
|  |  |
| --- | --- |
| **Các đặc điểm của cấu phần** | **Mô tả** |
| Chuẩn hóa (Standardized) | Tiêu chuẩn hóa thành phần có nghĩa là một thành phần được sử dụng trong quy trình CBSE phải phù hợp với mô hình thành phần tiêu chuẩn. Mô hình này có thể xác định các giao diện thành phần, siêu dữ liệu thành 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 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 pháo và thuộc tính của nó. |
| Composable  (có thể dùng một lần) | 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 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 1.1, 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.

Hình 1.1: Khái niệm chung về một thành phần và thành phần lồng nhau

Một thành phần được biểu diễn trong ký hiệu UML 2.0 được thể hiện trong hình sau:



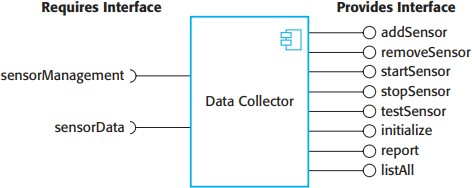
Hình 1.2: Ký hiệu thành phần trong UML

Các thành phần có hai giao diện liên quan, như trong Hình 1.1. 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 1.3 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.



Hình 1.3: Mô hình của thành phần thu thập dữ liệu

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.

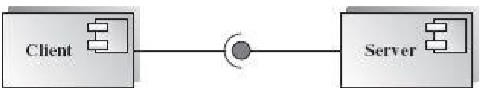
Giao diện ‘yêu cầu’ được sử dụng để kết nối thành phần với các cảm biến. Nó giả định rằng các cảm biến có giao diện dữ liệu, được truy cập thông qua sensorData và giao diện quản lý, được truy cập thông qua sensorManagement. Giao diện này được thiết kế để kết nối với các loại cảm biến khác nhau, do đó 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.

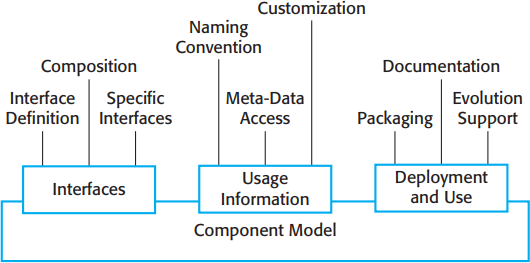
Sơ đồ trong Hình 1.4 cho thấy một cấu thành thành phần đơn giản trong đó thành phần khách yêu cầu (nhận) các dịch vụ từ một thành phần máy chủ thông qua cổng được cung cấp của máy chủ.



Hình 1.4: Component composition

### 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 1.5: 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. Được tóm tắt các phần tử mô hình này trong Hình 1.5. 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.

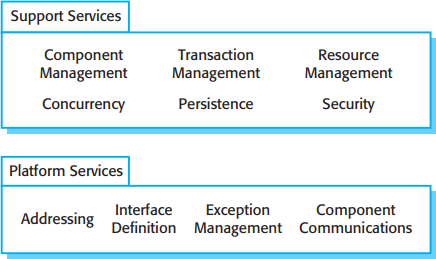
Siêu dữ liệu thành phần (component meta - data) là dữ liệu về chính thành phần đó, chẳng hạn như thông tin về các giao diện và thuộc tính của nó. Siêu dữ liệu rất quan trọng vì nó cho phép người dùng thành phần tìm hiểu những dịch vụ được cung cấp và yêu cầu. Việc triển khai mô hình thành phần thường 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 1.3) 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.

1. **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 1.6 cho thấy một số dịch vụ có thể được cung cấp bởi việc triển khai một mô hình thành phần.



Hình 1.6: Các dịch vụ phần mềm trung gian được xác định trong 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.

### Nguyên tắc thiết kế dựa trên thành phần

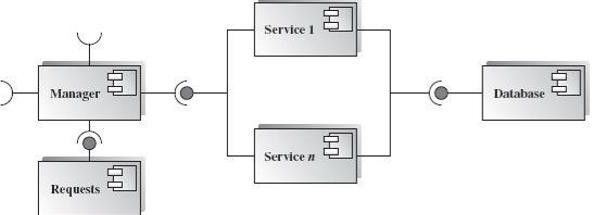


1.1.4.1 Kết nối của các thành phần

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 1.7 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 1.7: Kiến trúc dựa trên thành phần UML

Có ba loại kết nối:

* Kết nối Ball and cup (b2c hoặc c2b) : kết nói hao 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 mẫu 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.

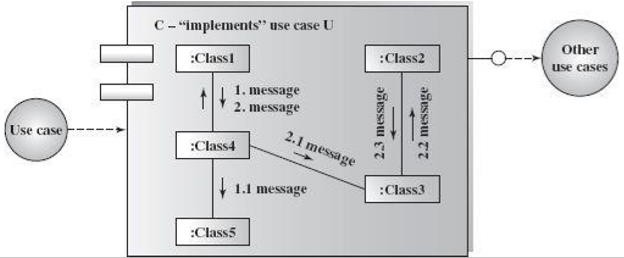
Chúng ta có thể thấy rằng thiết kế phần mềm dựa trên thành phần bắt đầu từ cấp độ trừu tượng hơn so với thiết kế OO

* + - 1. 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 1.8 minh họa việc triển khai thành phần của ca sử dụng U (xảy ra ở giữa hai ca sử dụng khác).



Hình 1.8: 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.

**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 | Nhược 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.  -Chi phí giảm .  -Dễ dàng phát triển . | -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. |

### 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.

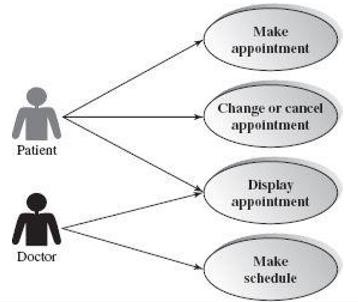
Thiết kế phần mềm dựa trên thành phần có nhiều lợi thế hơn thiết kế hướng đối tượng về các thuộc tính chất lượng của khả năng tái sử dụng, năng suất, độ tin cậy, khả năng mở rộng, tính di động và các thuộc tính khác. Các thách thức đối với việc áp dụng kiến trúc dựa trên thành phần bao gồm sự thích ứng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

### 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.

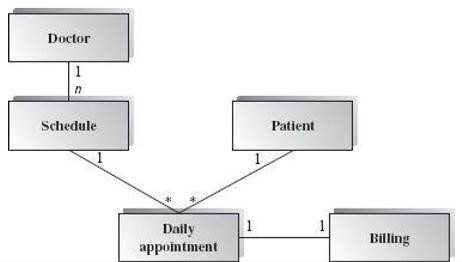
*Một phòng khám có hàng chục bác sĩ gia đình. Mỗi bác sĩ có một lịch trình hàng ngày với các khoảng thời gian 15 phút. Bệnh nhân có thể đặt lịch hẹn với bất kỳ bác sĩ nào. Bệnh nhân có thể thay đổi hoặc hủy bỏ cuộc hẹn. Bệnh nhân mới được chào đón. Tất cả các cuộc hẹn bao gồm tên bệnh nhân, tên bác sĩ, 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 1.9 cung cấp mô tả một phần của kịch bản. Người đọc có thể trau chuốt thêm.



Hình 1.9: Biểu đồ UC cho hệ thống đặt hẹn phòng khám

Sơ đồ khái niệm quy trình nghiệp vụ thể hiện trong Hình 1.10 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.



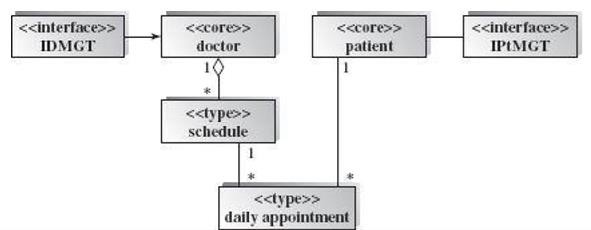
Hình 1.10: Biểu đồ khái niệm nghiệp vụ

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.

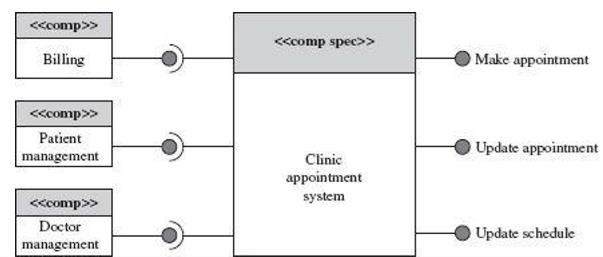
Giả sử không có sẵn sơ đồ lớp. Chúng tôi tinh chỉnh sơ đồ khái niệm nghiệp vụ với quy trình sau.

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 1.11: Xác định các thành phần

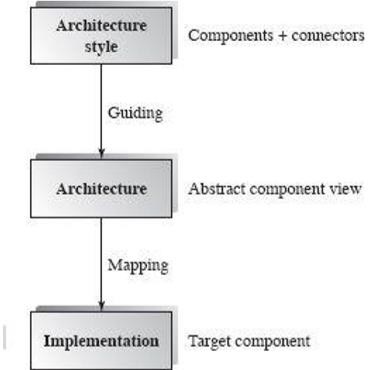


Hình 1.12: Lắt cắt đầu tiên của kiến trúc phần mềm dựa trên thành phần

Hình 1.12 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 1.13. 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 1.13: Vòng đời