

7

Thiết kế và thực hiện

Mục tiêu

Mục tiêu của chương này là giới thiệu thiết kế phần mềm hướng đối tượng sử dụng UML và nêu bật những vấn đề triển khai quan trọng. Khi bạn đọc xong chương này, bạn sẽ:

- hiểu các hoạt động quan trọng nhất trong quy trình thiết kế hướng đối tượng nói chung;
- hiểu một số mô hình khác nhau có thể được sử dụng để ghi lại một thiết kế hướng đối tượng;
- biết về ý tưởng của các mẫu thiết kế và chúng là một cách tái sử dụng kiến thức và kinh nghiệm thiết kế;
- đã được giới thiệu các vấn đề chính phải được xem xét khi triển khai phần mềm, bao gồm cả việc tái sử dụng phần mềm và phát triển nguồn mở.

Nội dung

- 7.1 Thiết kế hướng đối tượng sử dụng UML
- 7.2 Các mẫu thiết kế
- 7.3 Các vấn đề triển khai
- 7.4 Phát triển nguồn mở

Thiết kế và triển khai phần mềm là giai đoạn trong quy trình công nghệ phần mềm mà tại đó một hệ thống phần mềm thực thi được phát triển. Đối với một số hệ thống đơn giản, thiết kế và triển khai phần mềm là công nghệ phần mềm và tất cả các hoạt động khác đều được hợp nhất với quy trình này. Tuy nhiên, đối với các hệ thống lớn, thiết kế và triển khai phần mềm chỉ là một trong một tập hợp các quy trình (kỹ thuật yêu cầu, xác minh và xác nhận, v.v.) liên quan đến công nghệ phần mềm.

Các hoạt động thiết kế và triển khai phần mềm luôn luôn xen kẽ nhau. Thiết kế phần mềm là một hoạt động sáng tạo trong đó bạn xác định các thành phần phần mềm và mối quan hệ của chúng, dựa trên yêu cầu của khách hàng. Thực hiện là quá trình hiện thực hóa thiết kế như một chương trình. Đôi khi, có một giai đoạn thiết kế riêng biệt và thiết kế này được lập mô hình và ghi lại. Vào những thời điểm khác, một thiết kế nằm trong đầu người lập trình hoặc được phác thảo thô sơ trên bảng trắng hoặc tờ giấy. Thiết kế là làm thế nào để giải quyết một vấn đề, vì vậy luôn có một quá trình thiết kế. Tuy nhiên, không phải lúc nào cũng cần thiết hoặc thích hợp để mô tả chi tiết thiết kế bằng UML hoặc ngôn ngữ mô tả thiết kế khác.

Thiết kế và triển khai có mối liên hệ chặt chẽ với nhau và thông thường bạn nên tính đến các vấn đề triển khai khi phát triển một thiết kế. Ví dụ: sử dụng UML để ghi lại một thiết kế có thể là điều nên làm nếu bạn đang lập trình bằng ngôn ngữ hướng đối tượng như Java hoặc C#. Tôi nghĩ nó sẽ ít hữu ích hơn nếu bạn đang phát triển bằng một ngôn ngữ được gỡ động như Python và chẳng có ý nghĩa gì nếu bạn đang triển khai hệ thống của mình bằng cách định cấu hình một gói có sẵn. Như tôi đã thảo luận ở Chương 3, các phương pháp linh hoạt thường hoạt động từ các bản phác thảo thiết kế không chính thức và để lại nhiều quyết định thiết kế cho các lập trình viên.

Một trong những quyết định triển khai quan trọng nhất phải được đưa ra ở giai đoạn đầu của dự án phần mềm là liệu bạn có nên mua hoặc xây dựng phần mềm ứng dụng hay không. Trong nhiều lĩnh vực, hiện nay có thể mua các hệ thống có sẵn (COTS) có thể điều chỉnh và điều chỉnh theo yêu cầu của người dùng. Ví dụ: nếu bạn muốn triển khai hệ thống hồ sơ y tế, bạn có thể mua một gói đã được sử dụng trong bệnh viện. Sử dụng phương pháp này có thể rẻ hơn và nhanh hơn thay vì phát triển một hệ thống bằng ngôn ngữ lập trình thông thường.

Khi bạn phát triển một ứng dụng theo cách này, quy trình thiết kế sẽ quan tâm đến cách sử dụng các tính năng cấu hình của hệ thống đó để đáp ứng các yêu cầu của hệ thống. Bạn thường không phát triển các mô hình thiết kế của hệ thống, chẳng hạn như mô hình các đối tượng hệ thống và các tương tác của chúng. Tôi thảo luận về cách tiếp cận phát triển dựa trên COTS này ở Chương 16.

Tôi cho rằng hầu hết độc giả của cuốn sách này đều đã có kinh nghiệm thiết kế và thực hiện chương trình. Đây là thứ bạn có được khi học lập trình và nắm vững các thành phần của ngôn ngữ lập trình như Java hoặc Python. Bạn có thể đã học được cách thực hành lập trình tốt bằng các ngôn ngữ lập trình mà bạn đã học cũng như cách gỡ lỗi các chương trình mà bạn đã phát triển. Vì vậy, tôi không đề cập đến chủ đề lập trình ở đây. Thay vào đó, chương này có hai mục đích:

1. Để chỉ ra cách áp dụng mô hình hóa hệ thống và thiết kế kiến trúc (được đề cập trong Chương 5 và 6) trong việc phát triển thiết kế phần mềm hướng đối tượng.



Phương pháp thiết kế có cấu trúc

Các phương pháp thiết kế có cấu trúc đề xuất rằng thiết kế phần mềm nên được giải quyết một cách có phương pháp. Thiết kế một hệ thống bao gồm việc làm theo các bước của phương pháp và cải tiến thiết kế của một hệ thống ở mức độ ngày càng chi tiết. Vào những năm 1990, có một số phương pháp cạnh tranh dành cho thiết kế hướng đối tượng. Tuy nhiên, những người phát minh ra các phương pháp được sử dụng phổ biến nhất đã cùng nhau phát minh ra UML, nó thống nhất các ký hiệu được sử dụng trong các phương pháp khác nhau.

Thay vì tập trung vào các phương pháp, hầu hết các cuộc thảo luận hiện nay đều xoay quanh các quy trình trong đó thiết kế được coi là một phần của quy trình phát triển phần mềm tổng thể. Quy trình hợp nhất hợp lý (RUP) là một ví dụ điển hình về quy trình phát triển chung.

<http://www.SoftwareEngineering-9.com/Web/Structured-methods/>

- Giới thiệu các vấn đề triển khai quan trọng mà các sách lập trình thư ờng không đề cập đến. Chúng bao gồm tái sử dụng phần mềm, quản lý cấu hình và phát triển nguồn mở.

Vì có rất nhiều nền tảng phát triển khác nhau nên chương này không thiên về bất kỳ ngôn ngữ lập trình hoặc công nghệ triển khai cụ thể nào.

Vì vậy, tôi đã trình bày tất cả các ví dụ sử dụng UML thay vì bằng ngôn ngữ lập trình như Java hoặc Python.

7.1 Thiết kế hướng đối tượng sử dụng UML

Một hệ thống hướng đối tượng được tạo thành từ các đối tượng tương tác duy trì trạng thái cục bộ của riêng chúng và cung cấp các hoạt động trên trạng thái đó. Sự biểu diễn của trạng thái là riêng tư và không thể truy cập trực tiếp từ bên ngoài đối tượng. Quá trình thiết kế hướng đối tượng bao gồm việc thiết kế các lớp đối tượng và mối quan hệ giữa các lớp này. Các lớp này định nghĩa các đối tượng trong hệ thống và các tương tác của chúng. Khi thiết kế được thực hiện như một chương trình thực thi, các đối tượng sẽ được tạo động từ các định nghĩa lớp này.

Các hệ thống hướng đối tượng dễ thay đổi hơn các hệ thống được phát triển bằng cách sử dụng các phương pháp tiếp cận chức năng. Các đối tượng bao gồm cả dữ liệu và các thao tác để thao tác với dữ liệu đó. Do đó, chúng có thể được hiểu và sửa đổi như những thực thể độc lập. Việc thay đổi cách triển khai của một đối tượng hoặc thêm dịch vụ sẽ không ảnh hưởng đến các đối tượng hệ thống khác. Bởi vì các đối tượng được liên kết với các sự vật nên thư ờng có sự ánh xạ rõ ràng giữa các thực thể trong thế giới thực (chẳng hạn như các thành phần phần cứng) và các đối tượng điều khiển của chúng trong hệ thống. Điều này cải thiện tính dễ hiểu và do đó khả năng bảo trì của thiết kế.

Để phát triển một thiết kế hệ thống từ ý tưởng đến thiết kế chi tiết, hướng đối tượng, có một số điều bạn cần làm:

- Hiểu và xác định bối cảnh cũng như các tương tác bên ngoài với hệ thống.
- Thiết kế kiến trúc hệ thống.

3. Xác định các đối tượng chính trong hệ thống.

4. Xây dựng mô hình thiết kế.

5. Chỉ định giao diện.

Giống như tất cả các hoạt động sáng tạo, thiết kế không phải là một quá trình rõ ràng và tuần tự. Bạn phát triển một thiết kế bằng cách lấy ý tưởng, đề xuất giải pháp và tinh chỉnh các giải pháp này khi có thông tin. Bạn chắc chắn phải quay lại và thử lại khi có vấn đề phát sinh. Đôi khi bạn khám phá các lựa chọn một cách chi tiết để xem liệu chúng có hiệu quả hay không; vào những lúc khác, bạn bỏ qua các chi tiết cho đến cuối quá trình. Do đó, tôi đã cố tình không minh họa quá trình này dưới dạng một sơ đồ đơn giản vì điều đó có nghĩa là thiết kế có thể được coi là một chuỗi các hoạt động rõ ràng. Trên thực tế, tất cả các hoạt động trên đều đan xen và ảnh hưởng lẫn nhau.

Tôi minh họa các hoạt động của quy trình này bằng cách thiết kế một phần mềm cho trạm thời tiết ở vùng hoang dã mà tôi đã giới thiệu ở Chương 1. Các trạm thời tiết ở vùng hoang dã được triển khai ở các vùng sâu vùng xa. Mỗi trạm thời tiết ghi lại thông tin thời tiết địa phương và định kỳ chuyển thông tin này đến hệ thống thông tin thời tiết bằng cách sử dụng liên kết vệ tinh.

7.1.1 Bối cảnh và tư tưởng tác của hệ thống


Giai đoạn đầu tiên trong bất kỳ quy trình thiết kế phần mềm nào là phát triển sự hiểu biết về mối quan hệ giữa phần mềm đang được thiết kế và môi trường bên ngoài của nó. Điều này rất cần thiết để quyết định cách cung cấp chức năng hệ thống cần thiết và cách cấu trúc hệ thống để giao tiếp với môi trường của nó. Hiểu biết về bối cảnh cũng cho phép bạn thiết lập ranh giới của hệ thống.

Việc thiết lập ranh giới hệ thống giúp bạn quyết định những tính năng nào được triển khai trong hệ thống đang được thiết kế và những tính năng nào có trong các hệ thống liên quan khác. Trong tư tưởng hợp này, bạn cần quyết định cách phân bổ chức năng giữa hệ thống điều khiển cho tất cả các trạm thời tiết và phần mềm nhúng trong chính trạm thời tiết.

Các mô hình bối cảnh hệ thống và các mô hình tư tưởng tác trình bày các quan điểm bổ sung cho nhau về mối quan hệ giữa hệ thống và môi trường của nó:

1. Mô hình bối cảnh hệ thống là mô hình cấu trúc thể hiện các hệ thống khác trong môi trường của hệ thống đang được phát triển.
2. Mô hình tư tưởng tác là mô hình động thể hiện cách hệ thống tư tưởng tác với môi trường của nó khi nó được sử dụng.

Mô hình bối cảnh của một hệ thống có thể được biểu diễn bằng cách sử dụng các liên kết. Các liên kết chỉ đơn giản thể hiện rằng có một số mối quan hệ giữa các thực thể tham gia vào liên kết đó. Bản chất của các mối quan hệ hiện đã được xác định. Do đó, bạn có thể ghi lại môi trường của hệ thống bằng sơ đồ khối đơn giản, hiển thị các thực thể trong hệ thống và mối liên kết của chúng. Điều này được minh họa trong Hình 7.1, cho thấy rằng



Các trường hợp sử dụng trạm thời tiết

Báo cáo thời tiết—gửi dữ liệu thời tiết tới hệ thống thông tin thời tiết

Báo cáo trạng thái—gửi thông tin trạng thái tới hệ thống thông tin thời tiết

Khởi động lại—nếu trạm thời tiết bị tắt, hãy khởi động lại hệ thống

Tắt máy—tắt trạm thời tiết

Cấu hình lại—cấu hình lại phần mềm trạm thời tiết

Powersave—đưa trạm thời tiết vào chế độ tiết kiệm điện

Điều khiển từ xa—gửi lệnh điều khiển tới bất kỳ hệ thống con trạm thời tiết nào

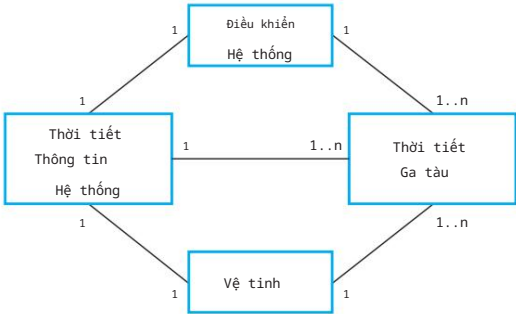
<http://www.SoftwareEngineering-9.com/Web/WS/Usecases.html>

Các hệ thống trong môi trường của mỗi trạm thời tiết là hệ thống thông tin thời tiết, hệ thống vệ tinh trên tàu và hệ thống điều khiển. Thông tin về số lượng trên liên kết cho thấy có một hệ thống điều khiển nhưng có nhiều trạm thời tiết, một vệ tinh và một hệ thống thông tin thời tiết chung.

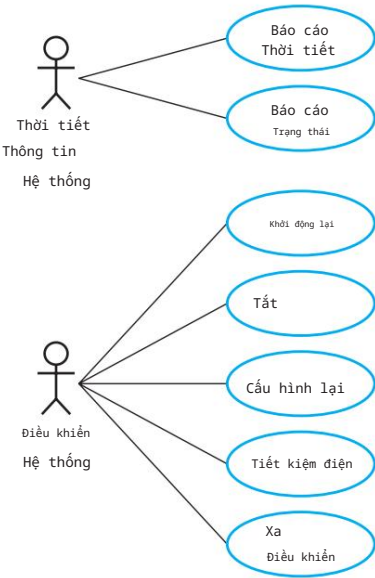
Khi lập mô hình tương tác của một hệ thống với môi trường của nó, bạn nên sử dụng một cách tiếp cận trừu tượng không bao gồm quá nhiều chi tiết. Một cách để làm điều này là sử dụng mô hình ca sử dụng. Như tôi đã thảo luận trong Chương 4 và 5, mỗi ca sử dụng thể hiện một tương tác với hệ thống. Mỗi tương tác có thể có được đặt tên theo hình elip và thực thể bên ngoài liên quan đến tương tác được biểu thị bằng hình que.

Mô hình ca sử dụng cho trạm thời tiết được thể hiện trong Hình 7.2. Điều này cho thấy trạm thời tiết tương tác với hệ thống thông tin thời tiết để báo cáo dữ liệu thời tiết và trạng thái phần cứng của trạm thời tiết. Các tương tác khác là với một hệ thống điều khiển có thể đưa ra các lệnh điều khiển trạm thời tiết cụ thể. Như tôi đã giải thích ở Chương 5, hình que được sử dụng trong UML để thể hiện các hệ thống khác cũng như người dùng con người.

Mỗi trường hợp sử dụng này phải được mô tả bằng ngôn ngữ tự nhiên có cấu trúc. Điều này giúp các nhà thiết kế xác định các đối tượng trong hệ thống và cung cấp cho họ sự hiểu biết về mục đích của hệ thống. Tôi sử dụng định dạng chuẩn cho mô tả này để xác định rõ ràng thông tin nào được trao đổi, cách bắt đầu tương tác và



Hình 7.1 Bối cảnh hệ thống trạm thời tiết



Hình 7.2 Các trường hợp sử dụng trạm thời tiết

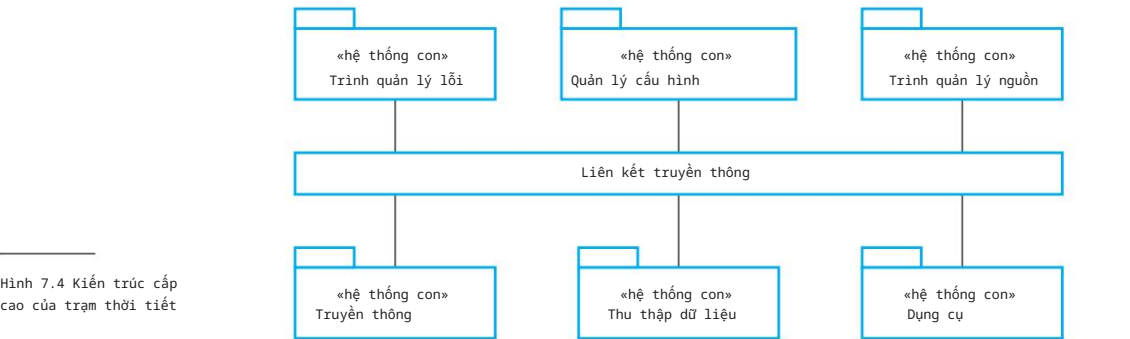
sớm. Điều này được thể hiện trong Hình 7.3, mô tả trường hợp sử dụng Báo cáo thời tiết từ Hình 7.2. Ví dụ về một số trường hợp sử dụng khác có trên Web.

7.1.2 Thiết kế kiến trúc

Khi các tương tác giữa hệ thống phần mềm và môi trường của hệ thống đã được xác định, bạn sử dụng thông tin này làm cơ sở để thiết kế kiến trúc hệ thống. Tất nhiên, bạn cần kết hợp điều này với kiến thức chung của bạn về các nguyên tắc thiết kế kiến trúc và với kiến thức miền chi tiết hơn.

Hình 7.3 Mô tả ca sử dụng-Báo cáo thời tiết

| | |
|--------------------|---|
| Hệ thống | Trạm thời tiết |
| Trường hợp sử dụng | Báo cáo thời tiết |
| Diễn viên | Hệ thống thông tin thời tiết, trạm thời tiết Trạm |
| Đạt | thời tiết gửi bản tóm tắt dữ liệu thời tiết đã được thu thập từ các thiết bị trong kỳ thu thập đến hệ thống thông tin thời tiết. Dữ liệu được gửi là nhiệt độ mặt đất và không khí tối đa, tối thiểu và trung bình; áp suất không khí tối đa, tối thiểu và trung bình; tốc độ gió tối đa, tối thiểu và trung bình; tổng lượng mưa; và hướng gió được lấy mẫu trong khoảng thời gian 5 phút. |
| Kích thích | Hệ thống thông tin thời tiết thiết lập liên kết vệ tinh với trạm thời tiết và yêu cầu truyền dữ liệu. |
| Phản ứng | Dữ liệu tóm tắt được gửi đến hệ thống thông tin thời tiết. |
| Bình luận | Các trạm thời tiết thường được yêu cầu báo cáo một lần mỗi giờ nhưng tần suất này có thể khác nhau giữa các trạm và có thể được sửa đổi trong tương lai. |



Hình 7.4 Kiến trúc cấp cao của trạm thời tiết

Bạn xác định các thành phần chính tạo nên hệ thống và các tương tác của chúng, sau đó có thể tổ chức các thành phần bằng cách sử dụng một mẫu kiến trúc chẳng hạn như mô hình lớp hoặc mô hình máy khách-máy chủ. Tuy nhiên, điều này không cần thiết ở giai đoạn này.

Thiết kế kiến trúc cấp cao cho phần mềm trạm thời tiết được thể hiện trong Hình 7.4. Trạm thời tiết bao gồm các hệ thống con độc lập giao tiếp bằng cách phát các thông báo trên cơ sở hạ tầng chung, được hiển thị dưới dạng liên kết Truyền thông trong Hình 7.4. Mỗi hệ thống con lắng nghe các tín hiệu trên cơ sở hạ tầng đó và chọn các tín hiệu dành cho chúng. Đây là một phong cách kiến trúc thư ờng được sử dụng khác ngoài những phong cách được mô tả trong Chương 6.

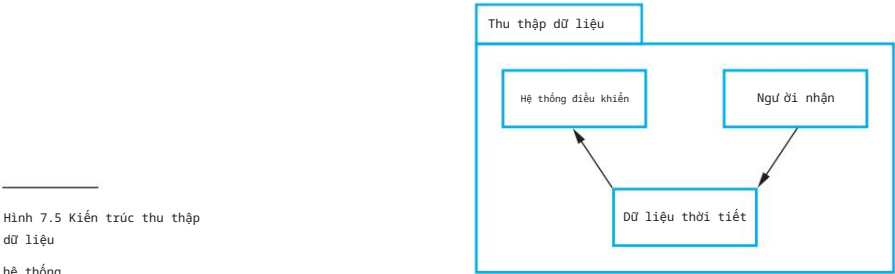
Ví dụ: khi hệ thống con truyền thông nhận được một lệnh điều khiển, chẳng hạn như tắt máy, lệnh này sẽ được mỗi hệ thống con khác tiếp nhận, sau đó các hệ thống con này sẽ tự tắt theo đúng cách. Lợi ích chính của kiến trúc này là dễ dàng hỗ trợ các cấu hình khác nhau của hệ thống con vì người gửi tín hiệu không cần gửi tín hiệu đến một hệ thống con cụ thể.

Hình 7.5 cho thấy kiến trúc của hệ thống con thu thập dữ liệu, được bao gồm trong Hình 7.4. Các đối tượng Máy phát và Máy thu liên quan đến việc quản lý thông tin liên lạc và đối tượng WeatherData đóng gói thông tin được thu thập từ các thiết bị và truyền đến thông tin thời tiết

hệ thống. Sự sắp xếp này tuân theo mô hình nhà sản xuất-người tiêu dùng, được thảo luận trong Chương 20.

7.1.3 Nhận dạng lớp đối tượng

Đến giai đoạn này của quá trình thiết kế, bạn nên có một số ý tưởng về các đối tượng thiết yếu trong hệ thống mà bạn đang thiết kế. Khi sự hiểu biết của bạn về thiết kế phát triển, bạn tinh chỉnh những ý tưởng này về các đối tượng hệ thống. Mô tả ca sử dụng giúp xác định các đối tượng và hoạt động trong hệ thống. Từ mô tả của thư ờng hợp sử dụng Báo cáo thời tiết, rõ ràng là sẽ cần có các đối tượng đại diện cho các công cụ thu thập dữ liệu thời tiết, cũng như một đối tượng đại diện cho bản tóm tắt dữ liệu thời tiết. Bạn cũng thư ờng cần một trình độ cao



Hình 7.5 Kiến trúc thu thập dữ liệu hệ thống

đối tượng hệ thống hoặc các đối tượng đóng gói các tương tác hệ thống được xác định trong các trường hợp sử dụng. Với những đối tượng này, bạn có thể bắt đầu xác định các lớp đối tượng trong hệ thống.

Đã có nhiều đề xuất khác nhau về cách xác định các lớp đối tượng trong hệ thống hướng đối tượng:

1. Sử dụng phân tích ngữ pháp của mô tả ngôn ngữ tự nhiên của hệ thống sẽ được xây dựng. Đối tượng và thuộc tính là danh từ; hoạt động hoặc dịch vụ đều là động từ (Abbott, 1983).
2. Sử dụng các thực thể hữu hình (đồ vật) trong miền ứng dụng như máy bay, các vai trò như người quản lý hoặc bác sĩ, các sự kiện như yêu cầu, tương tác như cuộc họp, địa điểm như văn phòng, đơn vị tổ chức như công ty, v.v. (Coad và Yourdon, 1990; Shlaer và Mellor, 1988; Wirfs-Brock và cộng sự, 1990).
3. Sử dụng phân tích dựa trên kịch bản trong đó các kịch bản sử dụng hệ thống khác nhau lần lượt được xác định và phân tích. Khi mỗi kịch bản được phân tích, nhóm chịu trách nhiệm phân tích phải xác định các đối tượng, thuộc tính và hoạt động cần thiết (Beck và Cunningham, 1989).

Trong thực tế, bạn phải sử dụng một số nguồn kiến thức để khám phá các lớp đối tượng. Các lớp đối tượng, thuộc tính và hoạt động được xác định ban đầu từ mô tả hệ thống không chính thức có thể là điểm khởi đầu cho thiết kế. Sau đó, thông tin sâu hơn từ kiến thức miền ứng dụng hoặc phân tích kịch bản có thể được sử dụng để tinh chỉnh và mở rộng các đối tượng ban đầu. Thông tin này có thể được thu thập từ các tài liệu yêu cầu, thảo luận với người dùng hoặc từ các phân tích về hệ thống hiện có.

Ở trạm thời tiết hoang dã, việc nhận dạng đối tượng dựa trên phần cứng hữu hình trong hệ thống. Tôi không có đủ không gian để bao gồm tất cả các đối tượng hệ thống ở đây, nhưng tôi đã chỉ ra năm lớp đối tượng trong Hình 7.6. Các đối tượng Nhiệt kế mặt đất, Máy đo gió và Máy đo áp suất là các đối tượng miền ứng dụng và các đối tượng WeatherStation và WeatherData đã được xác định từ mô tả hệ thống và mô tả kịch bản (ca sử dụng):

1. Lớp đối tượng WeatherStation cung cấp giao diện cơ bản của trạm thời tiết với môi trường của nó. Hoạt động của nó phản ánh các tương tác được thể hiện trong

Hình 7.6 Các đối tượng trạm thời tiết

Hình 7.6 Các đối tượng trạm thời tiết

2. Lớp đối tượng WeatherData có nhiệm vụ xử lý lệnh báo cáo thời tiết. Nó gửi dữ liệu tóm tắt từ các thiết bị trạm thời tiết đến hệ thống thông tin thời tiết.

3. Các lớp đối tượng Nhiệt kế mặt đất, Máy đo gió và Phong vũ biểu có liên quan trực tiếp đến các thiết bị trong hệ thống. Chúng phản ánh các thực thể phần cứng hữu hình trong hệ thống và các hoạt động liên quan đến việc kiểm soát phần cứng đó. Các đối tượng này hoạt động tự động để thu thập dữ liệu theo tần suất được chỉ định và lưu trữ dữ liệu được thu thập cục bộ. Dữ liệu này được gửi đến đối tượng WeatherData theo yêu cầu.

Bạn sử dụng kiến thức về miền ứng dụng để xác định các đối tượng, thuộc tính và dịch vụ khác. Chúng tôi biết rằng các trạm thời tiết thường được đặt ở những nơi xa xôi và bao gồm nhiều thiết bị khác nhau đôi khi gặp trục trặc. Lỗi thiết bị sẽ được báo cáo tự động. Điều này ngụ ý rằng bạn cần các thuộc tính và thao tác để kiểm tra hoạt động chính xác của các công cụ. Có nhiều trạm thời tiết ở xa nên mỗi trạm thời tiết cần có mã định danh riêng.

Ở giai đoạn này trong quá trình thiết kế, bạn nên tập trung vào chính các đối tượng mà không cần suy nghĩ về cách thực hiện những đối tượng này. Khi bạn đã xác định được các đối tượng, bạn sẽ tinh chỉnh thiết kế đối tượng. Bạn tìm kiếm những đặc điểm chung rồi thiết kế hệ thống phân cấp kế thừa cho hệ thống. Ví dụ: bạn có thể xác định một siêu lớp Công cụ, xác định các tính năng chung của tất cả các công cụ, chẳng hạn như mã định danh cũng như các hoạt động nhận và kiểm tra. Bạn cũng có thể thêm các thuộc tính và thao tác mới vào siêu lớp, chẳng hạn như thuộc tính duy trì tần suất thu thập dữ liệu.

7.1.4 Mô hình thiết kế

Các mô hình thiết kế hoặc hệ thống, như tôi đã thảo luận trong Chương 5, hiển thị các đối tượng hoặc các lớp đối tượng trong một hệ thống. Chúng cũng cho thấy sự liên kết và mối quan hệ giữa các thực thể này. Những mô hình này là cầu nối giữa các yêu cầu hệ thống và việc triển khai hệ thống. Chúng phải trừu tượng để những chi tiết không cần thiết không che giấu mối quan hệ giữa chúng và các yêu cầu hệ thống. Tuy nhiên, chúng cũng phải bao gồm đủ chi tiết để các lập trình viên đưa ra quyết định thực hiện.

Nói chung, bạn có thể giải quyết loại xung đột này bằng cách phát triển các mô hình ở các mức độ chi tiết khác nhau. Khi có mối liên kết chặt chẽ giữa các kỹ sư yêu cầu, người thiết kế và lập trình viên thì các mô hình trừu tượng có thể là tất cả những gì được yêu cầu. Các quyết định thiết kế cụ thể có thể được đưa ra khi hệ thống được triển khai, với các vấn đề được giải quyết thông qua các cuộc thảo luận không chính thức. Khi các liên kết giữa người xác định hệ thống, người thiết kế và người lập trình là gián tiếp (ví dụ: khi hệ thống được thiết kế ở một bộ phận của tổ chức nhưng được triển khai ở nơi khác), thì có thể sẽ cần các mô hình chi tiết hơn.

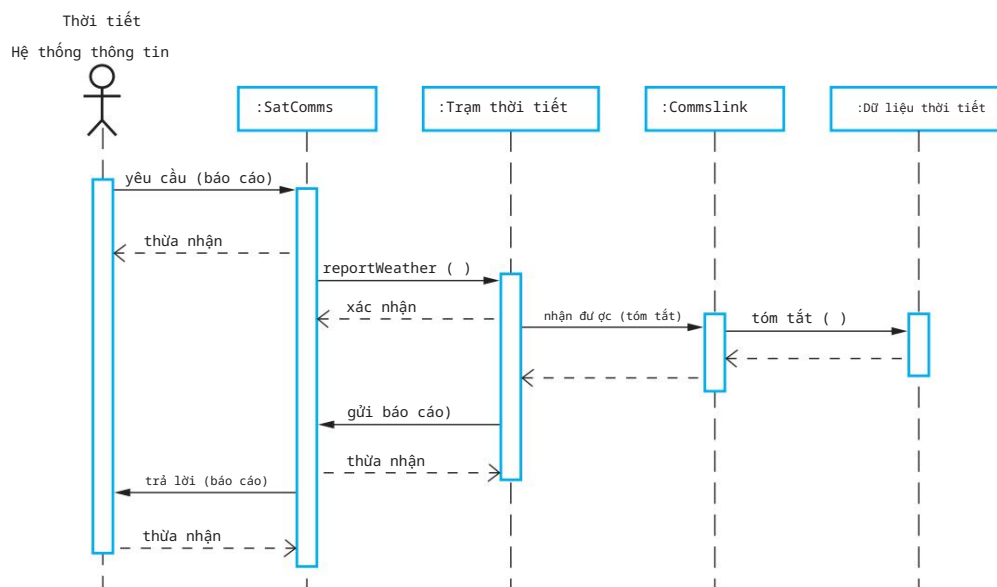
Do đó, một bước quan trọng trong quá trình thiết kế là quyết định các mô hình thiết kế mà bạn cần và mức độ chi tiết cần thiết trong các mô hình này. Điều này phụ thuộc vào loại hệ thống đang được phát triển. Bạn thiết kế một hệ thống xử lý dữ liệu tuần tự theo cách khác với hệ thống những thời gian thực, do đó bạn sẽ cần các mô hình thiết kế khác nhau. UML hỗ trợ 13 loại mô hình khác nhau, nhưng như tôi đã thảo luận ở Chương 5, bạn hiếm khi sử dụng tất cả các loại mô hình này. Giảm thiểu số lượng mô hình được sản xuất giúp giảm chi phí thiết kế và thời gian cần thiết để hoàn thành quá trình thiết kế.

Khi bạn sử dụng UML để phát triển một thiết kế, thông thường bạn sẽ phát triển hai loại mô hình thiết kế:

1. Các mô hình cấu trúc, mô tả cấu trúc tĩnh của hệ thống bằng cách sử dụng các lớp đối tượng và mối quan hệ của chúng. Các mối quan hệ quan trọng có thể được ghi lại ở giai đoạn này là các mối quan hệ khái quát hóa (kế thừa), các mối quan hệ sử dụng/được sử dụng bởi và các mối quan hệ thành phần.
2. Mô hình động, mô tả cấu trúc động của hệ thống và thể hiện sự tương tác giữa các đối tượng của hệ thống. Các tương tác có thể được ghi lại bao gồm chuỗi yêu cầu dịch vụ do các đối tượng thực hiện và các thay đổi trạng thái được kích hoạt bởi các tương tác đối tượng này.

Trong giai đoạn đầu của quá trình thiết kế, tôi nghĩ có ba mô hình đặc biệt hữu ích cho việc thêm chi tiết vào trường hợp sử dụng và mô hình kiến trúc:

1. Các mô hình hệ thống con, thể hiện các nhóm đối tượng logic thành các hệ thống con mạch lạc. Chúng được biểu diễn bằng dạng sơ đồ lớp với mỗi hệ thống con được hiển thị dưới dạng một gói có các đối tượng kèm theo. Các mô hình hệ thống con là các mô hình tĩnh (cấu trúc).



Hình 7.7 Sơ đồ trình tự mô tả việc thu thập dữ liệu

2. Các mô hình trình tự, thể hiện trình tự các tương tác của đối tượng. Chúng được biểu diễn bằng chuỗi UML hoặc sơ đồ cộng tác. Các mô hình tuần tự là các mô hình động.

3. Mô hình máy trạng thái, cho thấy các đối tượng riêng lẻ thay đổi trạng thái như thế nào để phản ứng với các sự kiện. Chúng được thể hiện trong UML bằng sơ đồ trạng thái. Mô hình máy trạng thái là mô hình động.

Mô hình hệ thống con là một mô hình tĩnh hữu ích vì nó cho thấy cách thiết kế được tổ chức thành các nhóm đối tượng có liên quan một cách logic. Tôi đã trình bày loại mô hình này trong Hình 7.4 để hiển thị các hệ thống con trong hệ thống bản đồ thời tiết. Cũng như các mô hình hệ thống con, bạn cũng có thể thiết kế các mô hình đối tượng chi tiết, hiển thị tất cả các đối tượng trong hệ thống và các liên kết của chúng (kế thừa, tổng quát hóa, tổng hợp, v.v.). Tuy nhiên, có một mối nguy hiểm khi làm quá nhiều mô hình. Bạn không nên đưa ra quyết định chi tiết về việc triển khai mà thực sự nên giao cho người lập trình hệ thống.

Các mô hình trình tự là các mô hình động mô tả trình tự các tương tác đối tượng diễn ra đối với từng chế độ tương tác. Khi ghi lại một thiết kế, bạn nên tạo mô hình trình tự cho từng tương tác quan trọng. Nếu bạn đã phát triển mô hình ca sử dụng thì cần có một mô hình trình tự cho từng trường hợp sử dụng mà bạn đã xác định.

Hình 7.7 là một ví dụ về mô hình trình tự, được hiển thị dưới dạng sơ đồ trình tự UML. Sơ đồ này hiển thị trình tự tương tác diễn ra khi hệ thống bên ngoài yêu cầu dữ liệu tóm tắt từ trạm thời tiết. Bạn đọc sơ đồ trình tự từ trên xuống dưới:

1. Đối tượng SatComms nhận được yêu cầu từ hệ thống thông tin thời tiết để thu thập báo cáo thời tiết từ trạm thời tiết. Nó xác nhận đã nhận được

yêu cầu này. Đầu mũi tên dính trên tin nhắn đã gửi cho biết hệ thống bên ngoài không chờ phản hồi mà có thể tiếp tục xử lý khác.

2. SatComms gửi tin nhắn đến WeatherStation, thông qua liên kết vệ tinh, để tạo bản tóm tắt dữ liệu thời tiết được thu thập. Một lần nữa, đầu mũi tên hình que chỉ ra rằng SatComms không tự treo để chờ phản hồi.
3. WeatherStation gửi tin nhắn đến đối tượng Commslink để tóm tắt dữ liệu thời tiết. Trong trường hợp này, kiểu đầu mũi tên bình phương chỉ ra rằng thể hiện của lớp đối tượng WeatherStation đang chờ phản hồi.
4. Commslink gọi phương thức tóm tắt trong đối tượng WeatherData và chờ

một câu trả lời.

5. Bản tóm tắt dữ liệu thời tiết được tính toán và gửi về WeatherStation thông qua Đối tượng Commslink.
6. WeatherStation sau đó gọi đối tượng SatComms để truyền dữ liệu tóm tắt đến hệ thống thông tin thời tiết, thông qua hệ thống liên lạc vệ tinh.

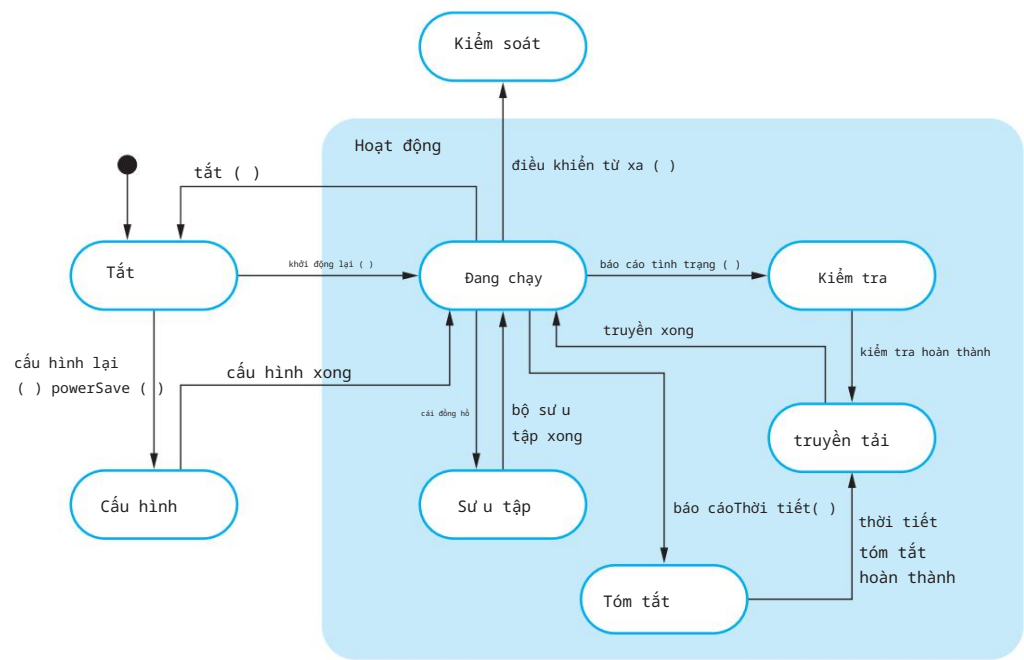
Các đối tượng SatComms và WeatherStation có thể được triển khai dưới dạng các quy trình đồng thời, quá trình thực thi của chúng có thể bị tạm dừng và tiếp tục lại. Phiên bản đối tượng SatComms lắng nghe các tin nhắn từ hệ thống bên ngoài, giải mã các tin nhắn này và bắt đầu các hoạt động của trạm thời tiết.

Sơ đồ tuần tự được sử dụng để mô hình hóa hành vi kết hợp của một nhóm đối tượng như bạn cũng có thể muốn tóm tắt hành vi của một đối tượng hoặc một hệ thống con để phản hồi các thông báo và sự kiện. Để thực hiện việc này, bạn có thể sử dụng mô hình máy trạng thái để hiển thị cách phiên bản đối tượng thay đổi trạng thái tùy thuộc vào thông báo mà nó nhận được. UML bao gồm các sơ đồ trạng thái, ban đầu được phát minh bởi Harel (1987) để mô tả các mô hình máy trạng thái.

Hình 7.8 là sơ đồ trạng thái của hệ thống trạm thời tiết cho thấy nó hoạt động như thế nào đáp ứng các yêu cầu về các dịch vụ khác nhau.

Bạn có thể đọc sơ đồ này như sau:

1. Nếu trạng thái hệ thống là Tắt máy thì nó có thể phản hồi với thông báo khởi động lại(), cấu hình lại() hoặc powerSave(). Mũi tên không được gắn nhãn có chấm màu đen cho biết trạng thái Tắt máy là trạng thái ban đầu. Thông báo khởi động lại() gây ra sự chuyển đổi sang hoạt động bình thường. Cả hai thông báo powerSave() và reconfigure() đều gây ra sự chuyển đổi sang trạng thái trong đó hệ thống tự cấu hình lại. Sơ đồ trạng thái cho thấy việc cấu hình lại chỉ được phép nếu hệ thống đã tắt.
2. Ở trạng thái Đang chạy, hệ thống mong đợi các tin nhắn tiếp theo. Nếu nhận được thông báo tắt máy(), đối tượng sẽ trở về trạng thái tắt máy.
3. Nếu nhận được thông báo reportWeather(), hệ thống sẽ chuyển sang trạng thái Tóm tắt. Khi bản tóm tắt hoàn tất, hệ thống sẽ chuyển sang trạng thái Đang truyền, tại đó thông tin được truyền đến hệ thống từ xa. Sau đó nó trở về trạng thái Đang chạy.



Hình 7.8 Sơ đồ trạng thái trạm thời tiết

4. Nếu nhận được thông báo reportStatus(), hệ thống sẽ chuyển sang trạng thái Kiểm tra, sau đó chuyển sang trạng thái Truyền, trừ khi quay lại trạng thái Đang chạy.

5. Nếu nhận được tín hiệu từ đồng hồ, hệ thống sẽ chuyển sang trạng thái Thu thập, nơi nó thu thập dữ liệu từ các thiết bị. Lần lượt từng nhạc cụ được xử lý để thu thập dữ liệu từ các cảm biến liên quan.

6. Nếu nhận được thông báo remoteControl(), hệ thống sẽ chuyển sang trạng thái được kiểm soát, trong đó hệ thống sẽ phản hồi một bộ thông báo khác từ phòng điều khiển từ xa. Những điều này không được hiển thị trên sơ đồ này.

Sơ đồ trạng thái là các mô hình cấp cao hữu ích của hệ thống hoặc hoạt động của đối tượng. Bạn thường không cần sơ đồ trạng thái cho tất cả các đối tượng trong hệ thống. Nhiều đối tượng trong một hệ thống tương đối đơn giản và mô hình trạng thái sẽ bổ sung thêm các chi tiết không cần thiết vào thiết kế.

7.1.5 Đặc tả giao diện

Một phần quan trọng của bất kỳ quá trình thiết kế nào là đặc tả các giao diện giữa các thành phần trong thiết kế. Bạn cần chỉ định các giao diện để các đối tượng và hệ thống con có thể được thiết kế song song. Khi một giao diện đã được chỉ định, các nhà phát triển của các đối tượng khác có thể cho rằng giao diện đó sẽ được triển khai.

Thiết kế giao diện liên quan đến việc xác định chi tiết giao diện cho một đối tượng hoặc một nhóm đối tượng. Điều này có nghĩa là xác định chữ ký và ngữ nghĩa của

Hình 7.9 Giao diện trạm thời tiết

| «giao diện» Báo cáo | «giao diện» Điều khiển từ xa |
|---|--|
| WeatherReport (WS-Ident): Báo cáo trạng thái báo cáo (WS-Ident): Báo cáo | startInstrument (công cụ): iStatus stopInstrument(công cụ): iStatus collData (công cụ): iStatus cung cấpData (công cụ): chuỗi |

các dịch vụ được cung cấp bởi đối tượng hoặc bởi một nhóm đối tượng. Các giao diện có thể được chỉ định trong UML bằng cách sử dụng ký hiệu giống như sơ đồ lớp. Tuy nhiên, không có phần thuộc tính và khuôn mẫu UML <<interface>> phải được đưa vào phần tên. Ngữ nghĩa của giao diện có thể được xác định bằng ngôn ngữ ràng buộc đối tượng (OCL). Tôi giải thích điều này trong Chương 17, nơi tôi đề cập đến kỹ thuật phần mềm dựa trên thành phần. Tôi cũng chỉ ra một cách khác để biểu diễn các giao diện trong UML.

Bạn không nên bao gồm các chi tiết về cách trình bày dữ liệu trong thiết kế giao diện vì các thuộc tính không được xác định trong đặc tả giao diện. Tuy nhiên, bạn nên bao gồm các thao tác truy cập và cập nhật dữ liệu. Vì cách biểu diễn dữ liệu bị ẩn nên có thể dễ dàng thay đổi nó mà không ảnh hưởng đến các đối tượng sử dụng dữ liệu đó. Điều này dẫn đến một thiết kế vốn dễ bảo trì hơn. Ví dụ: biểu diễn mảng của ngăn xếp có thể được thay đổi thành biểu diễn danh sách mà không ảnh hưởng đến các đối tượng khác sử dụng ngăn xếp. Ngược lại, việc trình bày các thuộc tính trong mô hình thiết kế tính trừu tượng có ý nghĩa vì đây là cách minh họa nhỏ gọn nhất các đặc điểm thiết yếu của đối tượng.

Không có mối quan hệ 1:1 đơn giản giữa các đối tượng và giao diện. Cùng một đối tượng có thể có nhiều giao diện, mỗi giao diện là một quan điểm về các phương thức mà nó cung cấp. Điều này được hỗ trợ trực tiếp trong Java, nơi các giao diện được khai báo tách biệt khỏi các đối tượng và giao diện 'thực hiện' của đối tượng. Tự nhiên, một nhóm đối tượng đều có thể được truy cập thông qua một giao diện duy nhất.

Hình 7.9 cho thấy hai giao diện có thể được xác định cho trạm thời tiết. Giao diện bên trái là giao diện báo cáo xác định tên hoạt động được sử dụng để tạo báo cáo trạng thái và thời tiết. Những ánh xạ này trực tiếp đến các hoạt động trong đối tượng WeatherStation. Giao diện điều khiển từ xa cung cấp bốn thao tác, ánh xạ vào một phương thức duy nhất trong đối tượng WeatherStation. Trong trừu tượng này, các thao tác riêng lẻ được mã hóa trong chuỗi lệnh liên quan đến phương thức remoteControl, được hiển thị trong Hình 7.6.

7.2 Mẫu thiết kế

Các mẫu thiết kế được bắt nguồn từ những ý tưởng được đưa ra bởi Christopher Alexander (Alexander và cộng sự, 1977), người cho rằng có một số mẫu chung nhất định về thiết kế tòa nhà vốn đã mang lại sự hài lòng và hiệu quả. Mẫu này là sự mô tả vấn đề và bản chất của giải pháp để giải pháp đó có thể được sử dụng lại trong

Tên mẫu: Ngươi quan sát Mô

tả: Tách cách hiển thị trạng thái của một đối tượng khỏi chính đối tượng đó và cho phép cung cấp các màn hình thay thế. Khi trạng thái đối tượng thay đổi, tất cả màn hình sẽ tự động được thông báo và cập nhật để phản ánh sự thay đổi.

Mô tả vấn đề: Trong nhiều trường hợp, bạn phải cung cấp nhiều màn hình thông tin trạng thái, chẳng hạn như màn hình đồ họa và màn hình dạng bảng. Không phải tất cả những điều này có thể được biết khi thông tin được chỉ định. Tất cả các bản trình bày thay thế phải hỗ trợ tương tác và khi trạng thái thay đổi, tất cả các màn hình phải được cập nhật.

Mẫu này có thể được sử dụng trong mọi trường hợp cần có nhiều hơn một định dạng hiển thị cho thông tin trạng thái và khi đối tượng duy trì thông tin trạng thái không cần thiết phải biết về các định dạng hiển thị cụ thể được sử dụng.

Mô tả giải pháp: Điều này liên quan đến hai đối tượng trừu tượng, Chủ thể và Ngươi quan sát, và hai đối tượng cụ thể, ConcreteSubject và ConcreteObject, kế thừa các thuộc tính của các đối tượng trừu tượng có liên quan. Các đối tượng trừu tượng bao gồm các thao tác chung có thể áp dụng trong mọi tình huống. Trạng thái được hiển thị được duy trì trong ConcreteSubject, kế thừa các hoạt động từ Chủ thể cho phép nó thêm và xóa Ngươi quan sát (mỗi ngươi quan sát tương ứng với một màn hình) và đưa ra thông báo khi trạng thái đã thay đổi.

ConcreteObserver duy trì một bản sao trạng thái của ConcreteSubject và triển khai giao diện Update() của Observer cho phép các bản sao này được lưu giữ theo từng bước. ConcreteObserver tự động hiển thị trạng thái và phản ánh các thay đổi bất cứ khi nào trạng thái được cập nhật.

Mô hình UML của mẫu được hiển thị trong Hình 7.12.

Hậu quả: Chủ thể chỉ biết Observer trừu tượng và không biết chi tiết về lớp cụ thể.

Do đó có sự kết hợp tối thiểu giữa các đối tượng này. Do thiếu kiến thức nên việc tối ưu hóa nhằm nâng cao hiệu suất hiển thị là không thực tế. Những thay đổi đối với chủ đề có thể tạo ra một tập hợp các cập nhật được liên kết cho ngươi quan sát, một số trong đó có thể không cần thiết.

Hình 7.10 Mẫu Observer

các cài đặt khác nhau. Mẫu này không phải là thông số kỹ thuật chi tiết. Đúng hơn, bạn có thể coi nó như một sự mô tả về trí tuệ và kinh nghiệm tích lũy được, một giải pháp đã được thử nghiệm thành công cho một vấn đề chung.

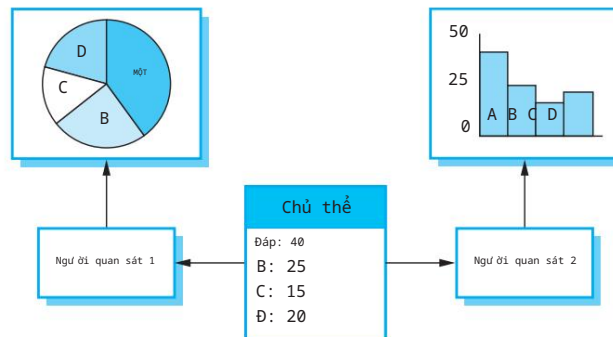
Một trích dẫn từ trang web của Hillside Group (<http://hillside.net>), dành riêng cho để duy trì thông tin về các mẫu, gói gọn vai trò của chúng trong việc tái sử dụng:

Mẫu và Ngôn ngữ mẫu là những cách để mô tả các phương pháp hay nhất, thiết kế tốt và nắm bắt trải nghiệm theo cách mà người khác có thể sử dụng lại trải nghiệm này.

Các mẫu đã có tác động rất lớn đến thiết kế phần mềm hướng đối tượng. Ngoài việc là giải pháp được thử nghiệm cho các vấn đề thông thường, chúng còn trở thành từ vựng để nói về một thiết kế. Do đó, bạn có thể giải thích thiết kế của mình bằng cách mô tả các mẫu mà bạn đã sử dụng. Điều này đặc biệt đúng đối với các mẫu thiết kế nổi tiếng nhất được mô tả ban đầu bởi 'Gang of Four' trong cuốn sách mẫu của họ, (Gamma et al., 1995). Những mô tả mẫu đặc biệt quan trọng khác là những mô tả được xuất bản trong loạt sách của các tác giả từ Siemens, một công ty công nghệ lớn ở Châu Âu (Buschmann và cộng sự, 1996; Buschmann và cộng sự, 2007a; Buschmann và cộng sự, 2007b; Kircher và Jain, 2004 ; Schmidt và cộng sự, 2000).

Các mẫu thiết kế thường gắn liền với thiết kế hướng đối tượng. Các mẫu được xuất bản thường dựa vào các đặc điểm của đối tượng như tính kế thừa và tính đa hình để cung cấp tính tổng quát. Tuy nhiên, nguyên tắc chung về việc gói gọn kinh nghiệm trong một

Hình 7.11 Nhiều màn hình



mẫu là mẫu có thể áp dụng như nhau cho bất kỳ loại thiết kế phần mềm nào. Vì vậy, bạn có thể có các mẫu cấu hình cho hệ thống COTS. Các mẫu là một cách sử dụng lại kiến thức và kinh nghiệm của các nhà thiết kế khác.

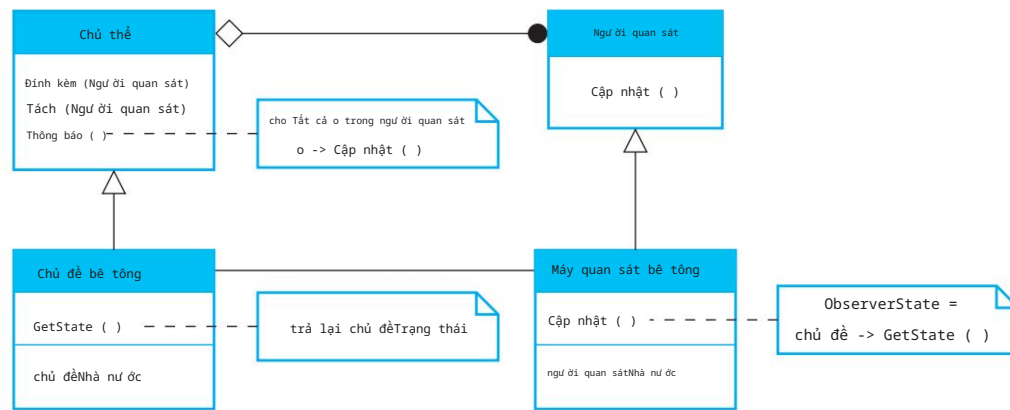
Bốn yếu tố thiết yếu của các mẫu thiết kế đã được 'Gang of Four' xác định trong cuốn sách mẫu của họ:

1. Tên tham chiếu có ý nghĩa cho mẫu.
2. Mô tả về khu vực có vấn đề giải thích khi nào mẫu có thể được áp dụng.
3. Mô tả giải pháp về các phần của giải pháp thiết kế, mối quan hệ và trách nhiệm của chúng. Đây không phải là một mô tả thiết kế cụ thể. Nó là một mẫu cho một giải pháp thiết kế có thể được khởi tạo theo nhiều cách khác nhau. Điều này thường được thể hiện bằng đồ họa và hiển thị mối quan hệ giữa các đối tượng và các lớp đối tượng trong giải pháp.
4. Tuyên bố về hệ quả-kết quả và sự đánh đổi-của việc áp dụng mô hình. Điều này có thể giúp các nhà thiết kế hiểu liệu một mẫu có thể được sử dụng trong một tình huống cụ thể hay không.

Gamma và các đồng tác giả của ông chia mô tả vấn đề thành động lực (mô tả lý do tại sao mô hình này hữu ích) và khả năng áp dụng (mô tả các tình huống trong đó mô hình có thể được sử dụng). Trong phần mô tả giải pháp, họ mô tả cấu trúc mẫu, người tham gia, sự hợp tác và cách triển khai.

Để minh họa mô tả mẫu, tôi sử dụng mẫu Observer, lấy từ cuốn sách của Gamma et al. (Gamma và cộng sự, 1995). Điều này được thể hiện trong Hình 7.10. Trong phần mô tả của mình, tôi sử dụng bốn yếu tố mô tả thiết yếu và cũng bao gồm một tuyên bố ngắn gọn về những gì mẫu có thể làm. Mẫu này có thể được sử dụng trong các tình huống yêu cầu các cách trình bày khác nhau về trạng thái của đối tượng. Nó tách đối tượng phải được hiển thị khỏi các hình thức trình bày khác nhau. Điều này được minh họa trong Hình 7.11, hiển thị hai bản trình bày đồ họa của cùng một tập dữ liệu.

Các biểu diễn đồ họa thường được sử dụng để minh họa các lớp đối tượng trong các mẫu và mối quan hệ của chúng. Chúng bổ sung cho mô tả mẫu và thêm



Hình 7.12 Mô hình UML của mẫu Observer

chi tiết đến mô tả giải pháp. Hình 7.12 là biểu diễn dưới dạng UML của mẫu Observer.

Để sử dụng các mẫu trong thiết kế của mình, bạn cần nhận ra rằng bất kỳ vấn đề thiết kế nào bạn đang gặp phải đều có thể có một mẫu liên quan có thể áp dụng được. Ví dụ về những vấn đề như vậy, được ghi lại trong cuốn sách mô hình ban đầu của 'Gang of Four', bao gồm:

1. Thông báo cho một số đối tượng rằng trạng thái của một số đối tượng khác đã thay đổi (Ngư ời quan sát mẫu).
2. Sắp xếp gọn gàng các giao diện cho một số đối tượng liên quan thường được phát triển dần dần (Mẫu mặt tiền).
3. Cung cấp một cách tiêu chuẩn để truy cập các phần tử trong bộ sưu tập, bất kể cách bộ sưu tập đó được triển khai (mẫu Iterator).
4. Cho phép khả năng mở rộng chức năng của lớp hiện có tại thời gian chạy (Mẫu trang trí).

Các mẫu hỗ trợ tái sử dụng khái niệm, cấp cao. Khi bạn cố gắng sử dụng lại các thành phần có thể thực thi được, bạn chắc chắn bị hạn chế bởi các quyết định thiết kế chi tiết do những ngư ời triển khai các thành phần này đưa ra. Những phạm vi này từ các thuật toán cụ thể đã được sử dụng để triển khai các thành phần cho đến các đối tượng và kiểu trong giao diện thành phần. Khi các quyết định thiết kế này mâu thuẫn với các yêu cầu cụ thể của bạn, việc sử dụng lại thành phần này là không thể hoặc gây ra sự thiếu hiệu quả trong hệ thống của bạn. Sử dụng các mẫu có nghĩa là bạn sử dụng lại các ý tưởng nhưng có thể điều chỉnh cách triển khai cho phù hợp với hệ thống mà bạn đang phát triển.

Khi bạn bắt đầu thiết kế một hệ thống, có thể khó biết trước liệu bạn có cần một mẫu cụ thể hay không. Do đó, việc sử dụng các mẫu trong quy trình thiết kế thường liên quan đến việc phát triển một thiết kế, gặp sự cố và sau đó nhận ra rằng một mẫu có thể được sử dụng. Điều này chắc chắn có thể thực hiện được nếu bạn tập trung vào 23 mục đích chung

các mẫu được ghi lại trong cuốn sách mẫu gốc. Tuy nhiên, nếu vấn đề của bạn là một vấn đề khác, bạn có thể gặp khó khăn trong việc tìm ra một mẫu thích hợp trong số hàng trăm mẫu khác nhau đã được đề xuất.

Mẫu là một ý tưởng tuyệt vời nhưng bạn cần có kinh nghiệm thiết kế phần mềm để sử dụng chúng một cách hiệu quả. Bạn phải nhận ra những tình huống có thể áp dụng một khuôn mẫu. Những lập trình viên thiếu kinh nghiệm, ngay cả khi họ đã đọc sách về mẫu, sẽ luôn khó quyết định liệu họ có thể sử dụng lại mẫu hay cần phát triển một giải pháp dành cho mục đích đặc biệt.

7.3 Các vấn đề thực hiện

Công nghệ phần mềm bao gồm tất cả các hoạt động liên quan đến phát triển phần mềm từ các yêu cầu ban đầu của hệ thống cho đến bảo trì và quản lý hệ thống đã triển khai. Tất nhiên, giai đoạn quan trọng của quá trình này là triển khai hệ thống, nơi bạn tạo phiên bản thực thi của phần mềm.

Việc triển khai có thể liên quan đến việc phát triển các chương trình bằng ngôn ngữ lập trình cấp cao hoặc cấp thấp hoặc điều chỉnh và điều chỉnh các hệ thống chung, sẵn có để đáp ứng các yêu cầu cụ thể của tổ chức.

Tôi cho rằng hầu hết độc giả của cuốn sách này sẽ hiểu các nguyên tắc lập trình và sẽ có một số kinh nghiệm lập trình. Vì chương này nhằm mục đích cung cấp một cách tiếp cận độc lập với ngôn ngữ nên tôi chưa tập trung vào các vấn đề về thực hành lập trình tốt vì chương này phải sử dụng các ví dụ dành riêng cho ngôn ngữ. Thay vào đó, tôi giới thiệu một số khía cạnh triển khai đặc biệt quan trọng đối với công nghệ phần mềm mà thư ờng không được đề cập trong các văn bản lập trình. Đó là:

1. Tái sử dụng Hầu hết các phần mềm hiện đại được xây dựng bằng cách sử dụng lại các thành phần hoặc hệ thống hiện có. Khi bạn đang phát triển phần mềm, bạn nên tận dụng tối đa mã hiện có.
2. Quản lý cấu hình Trong quá trình phát triển, nhiều phiên bản khác nhau của từng thành phần phần mềm được tạo ra. Nếu bạn không theo dõi các phiên bản này trong hệ thống quản lý cấu hình, bạn có thể đưa các phiên bản sai của các thành phần này vào hệ thống của mình.
3. Phát triển mục tiêu máy chủ Phần mềm sản xuất thư ờng không thực thi trên cùng một máy tính với môi trường phát triển phần mềm. Đúng hơn là bạn phát triển nó trên một máy tính (hệ thống máy chủ) và thực thi nó trên một máy tính riêng biệt (hệ thống đích). Hệ thống máy chủ và hệ thống đích đôi khi cùng loại nhưng thư ờng thì chúng hoàn toàn khác nhau.

7.3.1 Tái sử dụng

Từ những năm 1960 đến những năm 1990, hầu hết các phần mềm mới đều được phát triển từ đầu, bằng cách viết tất cả mã bằng ngôn ngữ lập trình cấp cao. Việc tái sử dụng đáng kể duy nhất hoặc

phần mềm là việc tái sử dụng các chức năng và đối tượng trong thư viện ngôn ngữ lập trình. Tuy nhiên, áp lực về chi phí và tiến độ có nghĩa là cách tiếp cận này ngày càng trở nên không khả thi, đặc biệt đối với các hệ thống thương mại và dựa trên Internet. Do đó, một cách tiếp cận phát triển dựa trên việc tái sử dụng phần mềm hiện có đã xuất hiện và hiện được sử dụng rộng rãi cho các hệ thống kinh doanh, phần mềm khoa học và ngày càng được sử dụng rộng rãi trong kỹ thuật hệ thống nhúng.

Có thể tái sử dụng phần mềm ở một số cấp độ khác nhau:

1. Mức độ trừu tượng Ở cấp độ này, bạn không trực tiếp sử dụng lại phần mềm mà sử dụng kiến thức về sự trừu tượng hóa thành công trong thiết kế phần mềm của bạn. Các mẫu thiết kế và các mẫu kiến trúc (được đề cập trong Chương 6) là những cách biểu diễn kiến thức trừu tượng để tái sử dụng.
2. Cấp độ đối tượng Ở cấp độ này, bạn trực tiếp sử dụng lại các đối tượng từ thư viện thay vì tự mình viết mã. Để triển khai kiểu tái sử dụng này, bạn phải tìm các thư viện thích hợp và khám phá xem các đối tượng và phương thức có cung cấp chức năng mà bạn cần hay không. Ví dụ: nếu bạn cần xử lý thư trong chương trình Java, bạn có thể sử dụng các đối tượng và phương thức từ thư viện JavaMail.
3. Thành phần cấp độ thành phần là tập hợp các đối tượng và lớp đối tượng hoạt động cùng nhau để cung cấp các chức năng và dịch vụ liên quan. Bạn thường phải điều chỉnh và mở rộng thành phần bằng cách thêm một số mã của riêng bạn. Một ví dụ về việc tái sử dụng ở cấp độ thành phần là nơi bạn xây dựng giao diện người dùng của mình bằng cách sử dụng framework. Đây là tập hợp các lớp đối tượng chung thực hiện xử lý sự kiện, quản lý hiển thị, v.v. Bạn thêm kết nối vào dữ liệu sẽ được hiển thị và viết mã để xác định chi tiết hiển thị cụ thể như bố cục màn hình và màu sắc.
4. Cấp độ hệ thống Ở cấp độ này, bạn sử dụng lại toàn bộ hệ thống ứng dụng. Điều này thường liên quan đến một số loại cấu hình của các hệ thống này. Điều này có thể được thực hiện bằng cách thêm và sửa đổi mã (nếu bạn đang sử dụng lại dòng sản phẩm phần mềm) hoặc bằng cách sử dụng giao diện cấu hình của chính hệ thống. Hầu hết các hệ thống thương mại hiện nay đều được xây dựng theo cách này, trong đó các hệ thống COTS (thương mại sẵn có) chung được điều chỉnh và tái sử dụng. Đôi khi cách tiếp cận này có thể liên quan đến việc sử dụng lại một số hệ thống khác nhau và tích hợp chúng để tạo ra một hệ thống mới.

Bằng cách sử dụng lại phần mềm hiện có, bạn có thể phát triển hệ thống mới nhanh hơn, ít rủi ro phát triển hơn và chi phí cũng thấp hơn. Vì phần mềm được sử dụng lại đã được thử nghiệm trong các ứng dụng khác nên nó đáng tin cậy hơn phần mềm mới. Tuy nhiên, có những chi phí liên quan đến việc tái sử dụng:

1. Chi phí thời gian tìm kiếm phần mềm để sử dụng lại và đánh giá xem nó có đáp ứng nhu cầu của bạn hay không. Bạn có thể phải kiểm thử phần mềm để đảm bảo rằng nó sẽ hoạt động trong môi trường của bạn, đặc biệt nếu môi trường này khác với môi trường phát triển của nó.
2. Chi phí mua phần mềm có thể tái sử dụng, nếu có. Đối với các sản phẩm lớn hệ thống kế, những chi phí này có thể rất cao.

3. Chi phí điều chỉnh và cấu hình các thành phần hoặc hệ thống phần mềm có thể tái sử dụng để phản ánh các yêu cầu của hệ thống mà bạn đang phát triển.
4. Chi phí tích hợp các thành phần phần mềm có thể tái sử dụng với nhau (nếu bạn sử dụng phần mềm từ các nguồn khác nhau) và với mã mới mà bạn có đã phát triển. Việc tích hợp phần mềm có thể tái sử dụng từ các nhà cung cấp khác nhau có thể khó khăn và tốn kém vì các nhà cung cấp có thể đưa ra các giả định trái ngược nhau về cách phần mềm tương ứng của họ sẽ được sử dụng lại.

Cách sử dụng lại kiến thức và phần mềm hiện có nên là điều đầu tiên bạn nên làm nghĩ đến khi bắt đầu một dự án phát triển phần mềm. Bạn nên xem xét khả năng sử dụng lại trước khi thiết kế phần mềm một cách chi tiết, vì bạn có thể muốn điều chỉnh thiết kế của bạn để sử dụng lại tài sản phần mềm hiện có. Như tôi đã thảo luận ở Chương 2, trong quy trình phát triển theo định hướng tái sử dụng, bạn tìm kiếm các phần tử có thể sử dụng lại sau đó sửa đổi yêu cầu và thiết kế để tận dụng tốt nhất những điều này.

Đối với một số lượng lớn các hệ thống ứng dụng, công nghệ phần mềm thực sự có nghĩa là tái sử dụng phần mềm. Do đó, tôi dành một số chương trong phần công nghệ phần mềm của cuốn sách cho chủ đề này (Chương 16, 17 và 19).

7.3.2 Quản lý cấu hình

Trong phát triển phần mềm, sự thay đổi luôn xảy ra, vì vậy quản lý thay đổi là hoàn toàn cần thiết. Khi một nhóm người đang phát triển phần mềm, bạn phải đảm bảo rằng các thành viên trong nhóm không can thiệp vào công việc của nhau. Tức là nếu hai mọi người đang làm việc trên một thành phần, những thay đổi của họ phải được phối hợp. Nếu không thì, một lập trình viên có thể thực hiện các thay đổi và ghi đè lên công việc của người khác. Bạn cũng phải đảm bảo rằng mọi người đều có thể truy cập các phiên bản cập nhật nhất của các thành phần phần mềm, nếu không các nhà phát triển có thể làm lại công việc đã được thực hiện. Khi cái gì gặp sự cố với phiên bản mới của hệ thống, bạn phải có khả năng quay lại phiên bản đang hoạt động của hệ thống hoặc thành phần đó.

Quản lý cấu hình là tên được đặt cho quy trình chung về quản lý một hệ thống phần mềm đang thay đổi. Mục đích của quản lý cấu hình là hỗ trợ quy trình tích hợp hệ thống để tất cả các nhà phát triển có thể truy cập mã và tài liệu dự án một cách có kiểm soát, tìm hiểu những thay đổi đã được thực hiện, biên dịch và liên kết các thành phần để tạo thành một hệ thống. Do đó, có ba hoạt động quản lý cấu hình cơ bản:

1. Quản lý phiên bản, nơi cung cấp hỗ trợ để theo dõi các phiên bản khác nhau phiên bản của các thành phần phần mềm. Hệ thống quản lý phiên bản bao gồm các phương tiện để điều phối sự phát triển của một số lập trình viên. Họ dùng một mã ghi đè của nhà phát triển đã được người khác gửi vào hệ thống.
2. Tích hợp hệ thống, nơi cung cấp hỗ trợ để giúp các nhà phát triển xác định những gì phiên bản của các thành phần được sử dụng để tạo ra từng phiên bản của hệ thống. Cái này Sau đó, mô tả được sử dụng để xây dựng hệ thống một cách tự động bằng cách biên dịch và liên kết các thành phần cần thiết.

3. Theo dõi sự cố, trong đó hỗ trợ được cung cấp để cho phép người dùng báo cáo lỗi và các sự cố khác, đồng thời cho phép tất cả các nhà phát triển biết ai đang xử lý các sự cố này và khi nào chúng được khắc phục.

Các công cụ quản lý cấu hình phần mềm hỗ trợ từng hoạt động trên.

Những công cụ này có thể được thiết kế để hoạt động cùng nhau trong một hệ thống quản lý thay đổi toàn diện, chẳng hạn như ClearCase (Bellagio và Milligan, 2005). Trong các hệ thống quản lý cấu hình tích hợp, các công cụ quản lý phiên bản, tích hợp hệ thống và theo dõi vấn đề được thiết kế cùng nhau. Chúng có chung kiểu giao diện người dùng và được tích hợp thông qua kho lưu trữ mã chung.

Ngoài ra, có thể sử dụng các công cụ riêng biệt, được cài đặt trong môi trường phát triển tích hợp. Quản lý phiên bản có thể được hỗ trợ bằng cách sử dụng hệ thống quản lý phiên bản như Subversion (Pilato và cộng sự, 2008), hệ thống này có thể hỗ trợ phát triển nhiều trang, nhiều nhóm. Hỗ trợ tích hợp hệ thống có thể được tích hợp vào ngôn ngữ hoặc dựa trên một bộ công cụ riêng biệt như hệ thống xây dựng GNU. Điều này bao gồm những gì có lẽ là công cụ tích hợp nổi tiếng nhất do Unix tạo ra. Hệ thống theo dõi lỗi hoặc theo dõi sự cố, chẳng hạn như Bugzilla, được sử dụng để báo cáo lỗi và các sự cố khác cũng như để theo dõi xem những lỗi này đã được sửa hay chưa.

Vì tầm quan trọng của nó trong công nghệ phần mềm chuyên nghiệp, tôi thảo luận về quản lý cấu hình và thay đổi chi tiết hơn trong Chương 25.

7.3.3 Phát triển mục tiêu máy chủ

Hầu hết việc phát triển phần mềm đều dựa trên mô hình máy chủ-đích. Phần mềm được phát triển trên một máy tính (máy chủ), nhưng chạy trên một máy riêng biệt (máy đích). Tổng quát hơn, chúng ta có thể nói về nền tảng phát triển và nền tảng thực thi. Nền tảng không chỉ là phần cứng. Nó bao gồm hệ điều hành được cài đặt cùng với phần mềm hỗ trợ khác như hệ thống quản lý cơ sở dữ liệu hoặc môi trường phát triển tư duy tác đối với các nền tảng phát triển.

Đôi khi, nền tảng phát triển và thực thi giống nhau, nên có thể phát triển phần mềm và thử nghiệm nó trên cùng một máy. Tuy nhiên, thông thường hơn, chúng khác nhau nên bạn cần chuyển phần mềm đã phát triển của mình sang nền tảng thực thi để thử nghiệm hoặc chạy trình mô phỏng trên máy phát triển của bạn.

Trình mô phỏng thường được sử dụng khi phát triển các hệ thống nhúng. Bạn mô phỏng các thiết bị phần cứng, chẳng hạn như cảm biến và các sự kiện trong môi trường mà hệ thống sẽ được triển khai. Trình mô phỏng tăng tốc quá trình phát triển cho các hệ thống nhúng vì mỗi nhà phát triển có thể có nền tảng thực thi riêng mà không cần tải phần mềm xuống phần cứng mục tiêu. Tuy nhiên, việc phát triển các trình mô phỏng rất tốn kém và do đó thường chỉ có sẵn cho các kiến trúc phần cứng phổ biến nhất.

Nếu hệ thống đích đã cài đặt phần mềm trung gian hoặc phần mềm khác mà bạn cần sử dụng thì bạn cần có khả năng kiểm tra hệ thống bằng phần mềm đó. Việc cài đặt phần mềm đó trên máy phát triển của bạn có thể là không thực tế, ngay cả khi nó giống với nền tảng mục tiêu do các hạn chế về giấy phép. Trong những trường hợp đó, bạn cần chuyển mã đã phát triển của mình sang nền tảng thực thi để kiểm tra hệ thống.



Sơ đồ triển khai UML

Sơ đồ triển khai UML cho thấy cách các thành phần phần mềm được triển khai vật lý trên bộ xử lý; nghĩa là sơ đồ triển khai hiển thị phần cứng và phần mềm trong hệ thống và phần mềm trung gian được sử dụng để kết nối các thành phần khác nhau trong hệ thống. Về cơ bản, bạn có thể coi sơ đồ triển khai như một cách xác định và ghi lại môi trường đích.

<http://www.SoftwareEngineering-9.com/Web/Deployment/>

Nền tảng phát triển phần mềm phải cung cấp nhiều công cụ để hỗ trợ các quy trình kỹ thuật phần mềm. Chúng có thể bao gồm:

1. Một trình biên dịch tích hợp và hệ thống chỉnh sửa theo cú pháp cho phép bạn tạo, chỉnh sửa và biên dịch mã.
2. Hệ thống gỡ lỗi ngôn ngữ.
3. Các công cụ chỉnh sửa đồ họa, chẳng hạn như công cụ chỉnh sửa mô hình UML.
4. Các công cụ kiểm tra, chẳng hạn như JUnit (Massol, 2003) có thể tự động chạy một loạt các bài kiểm tra trên phiên bản mới của chương trình.
5. Các công cụ hỗ trợ dự án giúp bạn sắp xếp mã cho các mục đích phát triển khác nhau dự án.

Ngoài các công cụ tiêu chuẩn này, hệ thống phát triển của bạn có thể bao gồm các công cụ chuyên biệt hơn như máy phân tích tĩnh (được thảo luận trong Chương 15). Thông thường, môi trường phát triển cho các nhóm cũng bao gồm một máy chủ dùng chung chạy hệ thống quản lý cấu hình và thay đổi, và có lẽ cả một hệ thống hỗ trợ quản lý yêu cầu.

Các công cụ phát triển phần mềm thường được nhóm lại để tạo ra môi trường phát triển tích hợp (IDE). IDE là một bộ công cụ phần mềm hỗ trợ các khía cạnh khác nhau của việc phát triển phần mềm, trong một số khuôn khổ chung và giao diện người dùng. Nói chung, IDE được tạo ra để hỗ trợ phát triển bằng một ngôn ngữ lập trình cụ thể như Java. IDE ngôn ngữ có thể được phát triển đặc biệt hoặc có thể là bản khởi tạo của IDE có mục đích chung, với các công cụ hỗ trợ ngôn ngữ cụ thể.

IDE đa năng là một khung lưu trữ các công cụ phần mềm cung cấp phư ơng tiện quản lý dữ liệu cho phần mềm đang được phát triển và các cơ chế tích hợp cho phép các công cụ hoạt động cùng nhau. IDE có mục đích chung được biết đến nhiều nhất là môi trường Eclipse (Carlson, 2005). Môi trường này dựa trên kiến trúc plug-in để nó có thể chuyên biệt cho các ngôn ngữ và miền ứng dụng khác nhau (Clayberg và Rubel, 2006). Do đó, bạn có thể cài đặt Eclipse và điều chỉnh nó cho phù hợp với nhu cầu cụ thể của mình bằng cách thêm các trình cắm thêm. Ví dụ: bạn có thể thêm một bộ plug-in để hỗ trợ phát triển hệ thống nối mạng bằng Java hoặc kỹ thuật hệ thống nhúng bằng C.

Là một phần của quá trình phát triển, bạn cần đưa ra quyết định về cách triển khai phần mềm đã phát triển trên nền tảng mục tiêu. Điều này thật đơn giản

đối với các hệ thống nhúng, trong đó mục tiêu thường là một máy tính. Tuy nhiên, đối với các hệ thống phân tán, bạn cần quyết định nền tảng cụ thể nơi các thành phần sẽ được triển khai. Các vấn đề bạn phải cân nhắc khi đưa ra quyết định này là:

1. Các yêu cầu về phần cứng và phần mềm của một thành phần Nếu một thành phần được thiết kế cho một kiến trúc phần cứng cụ thể hoặc dựa trên một số hệ thống phần mềm khác thì rõ ràng nó phải được triển khai trên nền tảng cung cấp hỗ trợ phần cứng và phần mềm cần thiết.
2. Yêu cầu về tính sẵn sàng của hệ thống Các hệ thống có tính sẵn sàng cao có thể yêu cầu các thành phần được triển khai trên nhiều nền tảng. Điều này có nghĩa là, trong trường hợp nền tảng bị lỗi, việc triển khai thành phần thay thế sẽ có sẵn.
3. Truyền thông thành phần Nếu có lưu lượng truyền thông giữa các thành phần ở mức cao, việc triển khai chúng trên cùng một nền tảng hoặc trên các nền tảng gần nhau về mặt vật lý thường là hợp lý. Điều này làm giảm độ trễ liên lạc, độ trễ giữa thời điểm tin nhắn được gửi bởi một thành phần và được thành phần khác nhận được.

Bạn có thể ghi lại các quyết định của mình về việc triển khai phần cứng và phần mềm bằng cách sử dụng sơ đồ triển khai UML, biểu thị cách các thành phần phần mềm được phân phối trên các nền tảng phần cứng.

Nếu bạn đang phát triển một hệ thống nhúng, bạn có thể phải tính đến các đặc điểm của mục tiêu, chẳng hạn như kích thước vật lý, khả năng cấp nguồn, nhu cầu phản hồi theo thời gian thực đối với các sự kiện cảm biến, đặc điểm vật lý của bộ truyền động và khả năng vận hành theo thời gian thực của nó. hệ thống. Tôi thảo luận về kỹ thuật hệ thống nhúng ở Chương 20.

7.4 Phát triển nguồn mở

Phát triển nguồn mở là một cách tiếp cận phát triển phần mềm trong đó mã nguồn của hệ thống phần mềm được xuất bản và các tình nguyện viên được mời tham gia vào quá trình phát triển (Raymond, 2001). Nguồn gốc của nó là từ Tổ chức Phần mềm Tự do (<http://www.fsf.org>), tổ chức ủng hộ rằng mã nguồn không nên là độc quyền mà phải luôn có sẵn để người dùng kiểm tra và sửa đổi theo ý muốn. Có giả định rằng mã sẽ được kiểm soát và phát triển bởi một nhóm nhỏ cốt lõi, thay vì người dùng mã.

Phần mềm nguồn mở đã mở rộng ý tưởng này bằng cách sử dụng Internet để tuyển dụng một lượng lớn các nhà phát triển tình nguyện. Nhiều người trong số họ cũng là người sử dụng mã.

Ít nhất về nguyên tắc, bất kỳ người đóng góp nào cho dự án nguồn mở đều có thể báo cáo và sửa lỗi cũng như đề xuất các tính năng và chức năng mới. Tuy nhiên, trên thực tế, các hệ thống nguồn mở thành công vẫn dựa vào nhóm các nhà phát triển nòng cốt, những người kiểm soát các thay đổi đối với phần mềm.

Tất nhiên, sản phẩm nguồn mở nổi tiếng nhất là hệ điều hành Linux.

được sử dụng rộng rãi như một hệ thống máy chủ và ngày càng được sử dụng như một môi trường máy tính để bàn.

Các sản phẩm nguồn mở quan trọng khác là Java, máy chủ web Apache và

hệ quản trị cơ sở dữ liệu MySQL. Những người chơi lớn trong ngành công nghiệp máy tính như

vì IBM và Sun hỗ trợ phong trào nguồn mở và xây dựng phần mềm của họ dựa trên nguồn mở

nguồn sản phẩm. Có hàng ngàn hệ thống nguồn mở khác ít được biết đến hơn

và các thành phần cũng có thể được sử dụng.

Việc mua phần mềm nguồn mở thường khá rẻ hoặc miễn phí. Bạn thường có thể tải xuống phần mềm nguồn mở miễn phí. Tuy nhiên, nếu bạn muốn có tài liệu và hỗ trợ thì bạn có thể phải trả tiền cho việc này, nhưng chi phí thường khá cao.

lợi ích chính khác của việc sử dụng các sản phẩm nguồn mở là nguồn mở hoàn thiện

hệ thống thường rất đáng tin cậy. Lý do cho điều này là họ có một lượng lớn người dùng sẵn

sàng tự khắc phục sự cố hơn là báo cáo những vấn đề này.

vấn đề cho nhà phát triển và chờ đợi bản phát hành mới của hệ thống. Các lỗi được phát hiện và sửa chữa nhanh hơn mức thường có thể thực hiện được bằng phần mềm độc quyền.

Đối với một công ty tham gia phát triển phần mềm, có hai nguồn mở những vấn đề cần xem xét:

1. Sản phẩm đang được phát triển có nên sử dụng các thành phần nguồn mở không?
2. Có nên sử dụng cách tiếp cận nguồn mở để phát triển phần mềm không?

Câu trả lời cho những câu hỏi này phụ thuộc vào loại phần mềm đang được phát triển. hoạt động cũng như nền tảng và kinh nghiệm của nhóm phát triển.

Nếu bạn đang phát triển một sản phẩm phần mềm để bán thì thời gian đưa ra thị trường và giảm chi phí là rất quan trọng. Nếu bạn đang phát triển trong một miền có chất lượng cao thống nguồn mở sẵn có, bạn có thể tiết kiệm thời gian và tiền bạc bằng cách sử dụng các hệ thống này. Tuy nhiên, nếu bạn đang phát triển phần mềm theo một tập hợp các yêu cầu cụ thể của tổ chức thì việc sử dụng các thành phần nguồn mở có thể không phải là một lựa chọn. Có thể bạn sẽ phải tích hợp phần mềm của bạn với các hệ thống hiện có không tương thích với các hệ thống sẵn có các hệ thống nguồn mở. Tuy nhiên, ngay cả khi đó, việc sửa đổi có thể nhanh hơn và rẻ hơn hệ thống nguồn mở thay vì phát triển lại chức năng mà bạn cần.

Ngày càng có nhiều công ty sản phẩm đang sử dụng cách tiếp cận nguồn mở để phát triển. Mô hình kinh doanh của họ không phụ thuộc vào việc bán sản phẩm phần mềm mà phụ thuộc vào hỗ trợ bán hàng cho sản phẩm đó. Họ tin rằng sự tham gia của cộng đồng nguồn mở sẽ cho phép phần mềm được phát triển với chi phí rẻ hơn, nhanh hơn và sẽ tạo ra một cộng đồng người sử dụng phần mềm. Tuy nhiên, một lần nữa, điều này thực sự chỉ áp dụng được cho các sản phẩm phần mềm nói chung hơn là các ứng dụng tổ chức cụ thể.

Nhiều công ty tin rằng việc áp dụng cách tiếp cận nguồn mở sẽ tiết lộ kiến thức kinh doanh bí mật cho đối thủ cạnh tranh của họ và do đó không muốn áp dụng điều này. mô hình phát triển. Tuy nhiên, nếu bạn đang làm việc trong một công ty nhỏ và bạn mở tìm nguồn phần mềm của bạn, điều này có thể trấn an khách hàng rằng họ sẽ có thể hỗ trợ phần mềm nếu công ty của bạn phá sản.

Việc xuất bản mã nguồn của một hệ thống không có nghĩa là mọi người ở phạm vi rộng hơn cộng đồng nhất thiết sẽ giúp đỡ sự phát triển của nó. Nguồn mở thành công nhất

sản phẩm là sản phẩm nền tảng hơn là hệ thống ứng dụng. Có một số lựa chọn hạn chế các nhà phát triển có thể quan tâm đến các hệ thống ứng dụng chuyên biệt. Như vậy, việc tạo ra một hệ thống phần mềm nguồn mở không đảm bảo sự tham gia của cộng đồng.

7.4.1 Cấp phép nguồn mở

Mặc dù nguyên tắc cơ bản của phát triển nguồn mở là mã nguồn phải được cung cấp miễn phí, nhưng điều này không có nghĩa là bất kỳ ai cũng có thể làm những gì họ muốn với mã đó. Về mặt pháp lý, nhà phát triển mã (công ty hoặc cá nhân) vẫn sở hữu mã. Họ có thể đặt ra các hạn chế về cách sử dụng nó bằng cách đưa các điều kiện ràng buộc về mặt pháp lý vào giấy phép phần mềm nguồn mở (St. Laurent, 2004). Một số nhà phát triển nguồn mở tin rằng nếu một thành phần nguồn mở được sử dụng để phát triển một hệ thống mới thì hệ thống đó cũng phải là nguồn mở. Những người khác sẵn sàng cho phép sử dụng mã của họ mà không có hạn chế này. Các hệ thống được phát triển có thể là độc quyền và được bán dưới dạng hệ thống nguồn đóng.

Hầu hết các giấy phép nguồn mở đều bắt nguồn từ một trong ba mô hình chung:

1. Giấy phép Công cộng GNU (GPL). Đây được gọi là giấy phép 'có đi có lại', nói một cách đơn giản, có nghĩa là nếu bạn sử dụng phần mềm nguồn mở được cấp phép theo giấy phép GPL, thì bạn phải làm cho phần mềm đó trở thành nguồn mở.
2. Giấy phép Công cộng Chung GNU Ít hơn (LGPL). Đây là một biến thể của giấy phép GPL nơi bạn có thể viết các thành phần liên kết tới mã nguồn mở mà không cần phải xuất bản nguồn của các thành phần này. Tuy nhiên, nếu bạn thay đổi thành phần được cấp phép thì bạn phải xuất bản thành phần này dưới dạng nguồn mở.
3. Giấy phép Phân phối Tiêu chuẩn Berkley (BSD). Đây là giấy phép không có đi có lại, có nghĩa là bạn không bắt buộc phải xuất bản lại bất kỳ thay đổi hoặc sửa đổi nào được thực hiện đối với mã nguồn mở. Bạn có thể đưa mã vào các hệ thống độc quyền được bán. Nếu bạn sử dụng các thành phần nguồn mở, bạn phải thừa nhận người tạo mã ban đầu.

Các vấn đề cấp phép là quan trọng vì nếu bạn sử dụng phần mềm nguồn mở như một phần của sản phẩm phần mềm thì bạn có thể bị ràng buộc bởi các điều khoản của giấy phép để biến sản phẩm của riêng bạn thành nguồn mở. Nếu bạn đang cố gắng bán phần mềm của mình, bạn có thể muốn giữ bí mật nó. Điều này có nghĩa là bạn có thể muốn tránh sử dụng phần mềm nguồn mở được cấp phép GPL trong quá trình phát triển nó.

Nếu bạn đang xây dựng phần mềm chạy trên nền tảng nguồn mở, chẳng hạn như Linux, thì giấy phép không phải là vấn đề. Tuy nhiên, ngay khi bạn bắt đầu đưa các thành phần nguồn mở vào phần mềm của mình, bạn cần thiết lập các quy trình và cơ sở dữ liệu để theo dõi những gì đã được sử dụng và các điều kiện cấp phép của chúng. Bayersdorfer (2007) gợi ý rằng các công ty quản lý dự án sử dụng nguồn mở nên:

1. Thiết lập hệ thống duy trì thông tin về các thành phần nguồn mở được tải xuống và sử dụng. Bạn phải giữ một bản sao giấy phép cho mỗi

thành phần hợp lệ tại thời điểm thành phần đó được sử dụng. Giấy phép có thể thay đổi nên bạn cần biết những điều kiện mà bạn đã đồng ý.

2. Nhận biết các loại giấy phép khác nhau và hiểu cách một thành phần được cấp phép trước khi nó được sử dụng. Bạn có thể quyết định sử dụng một thành phần trong hệ thống này như không sử dụng trong hệ thống khác vì bạn dự định sử dụng các hệ thống này theo những cách khác nhau.
3. Nhận thức được con đường tiến hóa của các thành phần. Bạn cần biết một chút về dự án nguồn mở nơi các thành phần được phát triển để hiểu chúng có thể thay đổi như thế nào trong tương lai.
4. Giáo dục mọi người về nguồn mở. Việc có sẵn các thủ tục để đảm bảo tuân thủ các điều kiện cấp phép là chưa đủ. Bạn cũng cần giáo dục các nhà phát triển về nguồn mở và cấp phép nguồn mở.
5. Có hệ thống kiểm toán tại chỗ. Các nhà phát triển, với thời hạn chặt chẽ, có thể muốn phá vỡ các điều khoản của giấy phép. Nếu có thể, bạn nên có sẵn phần mềm để phát hiện và ngăn chặn điều này.
6. Tham gia vào cộng đồng nguồn mở. Nếu bạn dựa vào các sản phẩm nguồn mở, bạn nên tham gia vào cộng đồng và giúp hỗ trợ sự phát triển của họ.

Mô hình kinh doanh phần mềm đang thay đổi. Việc xây dựng một doanh nghiệp bằng cách bán các hệ thống phần mềm chuyên dụng ngày càng trở nên khó khăn hơn. Nhiều công ty thích làm cho phần mềm của họ trở thành nguồn mở và sau đó bán hỗ trợ và tư vấn cho người dùng phần mềm. Xu hướng này có thể sẽ tăng tốc với việc sử dụng ngày càng nhiều phần mềm nguồn mở và ngày càng có nhiều phần mềm ở dạng này.

NHỮNG ĐIỂM CHÍNH

- Thiết kế và triển khai phần mềm là các hoạt động đan xen. Mức độ chi tiết trong thiết kế phụ thuộc vào loại hệ thống đang được phát triển và liệu bạn đang sử dụng cách tiếp cận theo kế hoạch hay linh hoạt.
- Quá trình thiết kế hướng đối tượng bao gồm các hoạt động thiết kế kiến trúc hệ thống, xác định các đối tượng trong hệ thống, mô tả thiết kế bằng cách sử dụng các mô hình đối tượng khác nhau và ghi lại các giao diện thành phần.
- Một loạt các mô hình khác nhau có thể được tạo ra trong quá trình thiết kế hướng đối tượng. Chúng bao gồm các mô hình tĩnh (mô hình lớp, mô hình tổng quát, mô hình liên kết) và mô hình động (mô hình trình tự, mô hình máy trạng thái).
- Các giao diện thành phần phải được xác định chính xác để các đối tượng khác có thể sử dụng chúng. một UML khuôn mẫu giao diện có thể được sử dụng để xác định giao diện.
- Khi phát triển phần mềm, bạn phải luôn xem xét khả năng sử dụng lại các phần mềm hiện có phần mềm, dư thừa các thành phần, dịch vụ hoặc hệ thống hoàn chỉnh.

202 Chương 7 ■ Thiết kế và thực hiện

- Quản lý cấu hình là quá trình quản lý các thay đổi đối với hệ thống phần mềm đang phát triển. Điều cần thiết là khi một nhóm người hợp tác để phát triển phần mềm.
- Hầu hết việc phát triển phần mềm là phát triển mục tiêu máy chủ. Bạn sử dụng IDE trên máy chủ để phát triển phần mềm, phần mềm này được chuyển đến máy đích để thực thi.
- Phát triển nguồn mở liên quan đến việc cung cấp mã nguồn của hệ thống một cách công khai. Điều này có nghĩa là nhiều người có thể đề xuất những thay đổi và cải tiến cho phần mềm.

ĐỌC THÊM

Các mẫu thiết kế: Các thành phần của Phần mềm hướng đối tượng có thể tái sử dụng. Đây là cẩm nang về mẫu phần mềm ban đầu nhằm giới thiệu các mẫu phần mềm cho cộng đồng rộng lớn. (E. Gamma, R. Helm, R. Johnson và J. Vlissides, Addison-Wesley, 1995.)

Áp dụng UML và Mẫu: Giới thiệu về Phân tích và Thiết kế hướng đối tượng và Phát triển lặp lại, ấn bản thứ 3. Larman viết rõ ràng về thiết kế hướng đối tượng cũng như thảo luận về việc sử dụng UML. Đây là phần giới thiệu hay về cách sử dụng các mẫu trong quá trình thiết kế. (C. Larman, Prentice Hall, 2004.)

Sản xuất phần mềm nguồn mở: Cách chạy một dự án phần mềm miễn phí thành công. Cuốn sách của ông là hướng dẫn toàn diện về nền tảng của phần mềm nguồn mở, các vấn đề cấp phép và tính thực tiễn của việc điều hành một dự án phát triển nguồn mở. (K. Fogel, O'Reilly Media Inc., 2008.)

Đọc thêm về tái sử dụng phần mềm được đề xuất trong Chương 16 và về quản lý cấu hình trong Chương 25.

BÀI TẬP

- 7.1. Sử dụng ký hiệu có cấu trúc được hiển thị trong Hình 7.3, chỉ định các trường hợp sử dụng trạm thời tiết cho Trạng thái báo cáo và Cấu hình lại. Bạn nên đưa ra những giả định hợp lý về chức năng được yêu cầu ở đây.
- 7.2. Giả sử rằng MHC-PMS đang được phát triển bằng cách sử dụng phương pháp hướng đối tượng. Vẽ sơ đồ ca sử dụng hiển thị ít nhất sáu trường hợp sử dụng có thể có cho hệ thống này.
- 7.3. Sử dụng ký hiệu đồ họa UML cho các lớp đối tượng, thiết kế các lớp đối tượng sau:
 - xác định các thuộc tính và hoạt động. Sử dụng kinh nghiệm của riêng bạn để quyết định các thuộc tính và thao tác cần được liên kết với các đối tượng này. ■ điện thoại ■ máy in
 - cho máy tính cá nhân ■ hệ thống âm thanh nổi cá nhân
 - tài khoản ngân hàng
 - danh mục thư viện