

BÀI GIẢNG

PHÂN TÍCH & THIẾT KẾ HỆ THỐNG THÔNG TIN

*Tài liệu dành cho sinh viên chính quy
ngành Công nghệ Thông tin*

TRẦN ĐÌNH QUÉ

LỜI NÓI ĐẦU

Phân tích và thiết kế hướng đối tượng là môn học tiếp theo sau của môn học Nhập môn Công nghệ phần mềm. Trong khi môn công nghệ phần mềm trình bày nhiều khía cạnh trong phát triển phần mềm, môn học này tập trung vào các pha Phân tích và Thiết kế phần mềm dựa trên cách tiếp cận hướng đối tượng. Tài liệu “thô” này được biên soạn chủ yếu dựa vào cuốn sách “Object-Oriented Analysis and Design: Understanding System Development with UML 2.0” của Mike O’Docherty nhằm giúp cho những sinh viên gặp khó khăn khi đọc tài liệu nguyên bản tiếng Anh. Nội dung tài liệu bao gồm:

Chương 1: Mở đầu

Giới thiệu các kiểu hệ thống thông tin và mô hình hệ thống dựa vào cách tiếp cận hướng đối tượng. Các khái niệm đối tượng và lớp, đóng gói, quan hệ giữa các lớp và vấn đề sử dụng lại mã nguồn.

Chương 2 : Phương pháp luận phát triển phần mềm

Chương này trình bày các pha phát triển phần mềm cùng các phân tích, nhận định các phương pháp luận truyền thống. Nội dung chủ yếu trình bày các phương pháp luận phát triển phần mềm hướng đối tượng được sử dụng phổ biến trong công nghiệp phần mềm hiện nay.

Chương 3. UML và công cụ phát triển phần mềm

Nội dung bao gồm phần giới thiệu về UML, các biểu đồ UML và sử dụng các biểu đồ UML trong phân tích và thiết kế hướng đối tượng cùng công cụ phát triển.

Chương 4. Xác định yêu cầu

Nội dung tập trung trình bày pha xác định yêu cầu dựa trên quan điểm nghiệp vụ và quan điểm người phát triển. Một case study về Quản lý đăng ký học theo tín chỉ sẽ được xem xét.

Chương 5. Phân tích yêu cầu

Nội dung bao gồm tổng quan quá trình phân tích và phân tích tĩnh cũng như phân tích động. Phần phân tích tĩnh đề cập đến việc xác định các lớp và quan hệ giữa các lớp, các thuộc tính. Phần phân tích động bàn về việc thực thi các lớp, các lớp biên, điều khiển và thực thể, các biểu đồ giao tiếp, phương thức trong lớp và cách xây dựng dựa trên gán trách nhiệm cho lớp và biểu đồ trạng thái.

LỜI NÓI ĐẦU

Chương 6. Thiết kế kiến trúc hệ thống

Trình bày các bước trong thiết kế hệ thống, chọn topo hệ thống mạng cho thiết kế, thiết kế đồng thời vấn đề và xác định độ ưu tiên thiết kế.

Chương 7. Lựa chọn công nghệ

Trình bày một số vấn đề liên quan đến công nghệ trong phát triển phần mềm. Nội dung bao gồm: Các công nghệ tầng client; Các công nghệ tầng trung gian; Các công nghệ tầng trung gian đến tầng dữ liệu; Các kiểu cấu hình; Các gói theo UML.

Chương 8. Thiết kế các hệ thống con

Chương này trình bày ánh xạ mô hình lớp phân tích thành mô hình lớp thiết kế, xử lý lưu trữ với cơ sở dữ liệu quan hệ, một số vấn đề liên quan đến giao diện người sử dụng, thiết kế các dịch vụ nghiệp vụ, sử dụng pattern, framework và thư viện.

Chương 9. Các mẫu thiết kế sử dụng lại

Nội dung trình bày các mẫu thiết kế sử dụng mẫu; một số vấn đề liên quan đến tìm kiếm, kết hợp và hiệu chỉnh mẫu thiết kế.

Chương 10. Đặc tả các giao diện của lớp

Nội dung trình bày vai trò của đặc tả, các kiểu đặc tả, đặc tả hướng đối tượng, thiết kế theo hợp đồng, đặc tả trong java.

Tác giả vô cùng cảm ơn sinh viên Khoa Công nghệ Thông tin thuộc các Khóa D05, D06, D07 đã giúp soạn thảo tài liệu này. Nhiều thuật ngữ, câu văn, hình vẽ... chưa được Việt hóa hay chưa được sáng sửa, rất mong nhận được nhiều ý kiến đóng góp của các Thầy – Cô cùng các bạn sinh viên để tài liệu ngày được hoàn thiện hơn. Tận đáy lòng mình, tác giả mong sinh viên hãy đọc và lĩnh hội nguyên bản tiếng Anh với sự giúp đỡ của giảng viên trên lớp và hãy xem tài liệu này như là bản “nháp”.

Tác giả

TÀI LIỆU THAM KHẢO

- [1] Mike O'Docherty, *Object-Oriented Analysis and Design: Understanding System Development with UML 2.0*, John Wiley & Sons, 2005.
- [2] Huỳnh Văn Đức, Đoàn Thiện Ngân, *Giáo trình nhập môn UML*, NXB Lao động Xã hội, 2003.
- [3] Đặng Văn Đức, *Phân tích và thiết kế hướng đối tượng*, NXB Giáo Dục, 2002
- [4] S. Schach, *Object-oriented and classical software engineering*, Sixth Edition, McGrawHill, 2006.
- [5] J. A. Hoffer, J. George, *Modern systems analysis and design*, Prentice Hall, 2002.
- [6] Trần Đình Quέ và Nguyễn Mạnh Sơn, Phân tích và Thiết kế hướng đối tượng, Bài giảng cho Sinh viên Đại học Từ xa, Học viện CNBCVT, 2005

MỤC LỤC

CHƯƠNG 1 MỞ ĐẦU.....	11
 1.1 GIỚI THIỆU.....	11
 1.2 CÁC KIỂU HỆ THỐNG THÔNG TIN.....	11
 1.3 MÔ HÌNH HỆ THỐNG DỰA TRÊN CÁCH TIẾP CẬN HƯỚNG ĐÓI TƯỢNG	15
1.3.1 Mô hình hệ thống.....	15
1.3.2 UML.....	15
 1.4 ĐỐI TƯỢNG VÀ LỐP.....	18
1.4.1 Đối tượng.....	18
1.4.2 Lốp.....	19
 1.5 ĐÓNG GÓI.....	19
 1.6 QUAN HỆ GIỮA CÁC LỐP.....	22
1.6.1 Các quan hệ phu thuộc: Association, aggregation và Composition.....	22
1.6.2 Ké thừa.....	26
 1.7 CÁCH SỬ DỤNG LẠI MÃ NGUỒN.....	30
 1.8 KẾT LUẬN.....	31
 BÀI TẬP.....	32
 TÀI LIỆU THAM KHẢO THÊM.....	32
CHƯƠNG 2 PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM.....	33
 2.1 MỞ ĐẦU.....	33
 2.2 CÁC PHA PHÁT TRIỂN PHẦN MỀM CỔ DIỄN.....	35
2.2.1 Xác định yêu cầu.....	35
2.2.2 Phân tích.....	35
2.2.3 Thiết kế.....	35
2.2.4 Đặc tả.....	35
 2.2.5 Cài đặt.....	36
2.2.6 Kiểm thử.....	36
2.2.7 Triển khai.....	36
2.2.8 Bảo trì.....	36
2.2.9 Một số câu hỏi.....	36
 2.3 CÁC PHƯƠNG PHÁP LUẬN TRUYỀN THỐNG.....	36
2.3.1 Phương pháp luận theo mô hình thác nước.....	36
2.3.2 Phương pháp luận xoắn ốc.....	37
2.3.3 Phương pháp luận lặp.....	38
2.3.4 Phương pháp luận tăng dần.....	39
2.3.5 Kết hợp các phương pháp luận.....	39
 2.4 PHƯƠNG PHÁP LUẬN HƯỚNG ĐÓI TƯỢNG.....	41
2.4.1 Tiến trình hợp nhất (UP: Unified Process).....	41
2.4.2 Rational Unified Process (RUP).....	41

MỤC LỤC

<u>2.5 KẾT LUẬN.....</u>	<u>52</u>
<u>BÀI TẬP.....</u>	<u>52</u>
<u>TÀI LIỆU THAM KHẢO THÊM.....</u>	<u>52</u>
CHƯƠNG 3 . UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM.....	54
3.1 GIỚI THIỆU VỀ UML.....	54
3.1.1 Lịch sử ra đời của UML.....	54
3.1.2 UML – Ngôn ngữ mô hình hoá hướng đối tượng.....	55
3.1.3 Các khái niệm cơ bản trong UML.....	56
3.2 CÁC BIỂU ĐỒ UML.....	58
3.2.1 Biểu đồ use case.....	60
3.2.2 Biểu đồ lớp.....	62
3.2.3 Biểu đồ trạng thái.....	68
3.2.4 Biểu đồ tương tác đang tuần tự.....	71
3.2.5 Biểu đồ tương tác đang công tác.....	73
3.2.6 Biểu đồ hoạt động.....	74
3.2.7 Biểu đồ thành phần.....	77
3.2.8 Biểu đồ triển khai hệ thống.....	78
3.3 GIỚI THIỆU CÔNG CỤ RATIONAL ROSE	78
3.4 KẾT LUẬN.....	81
<u>BÀI TẬP.....</u>	<u>82</u>
CHƯƠNG 4 XÁC ĐỊNH YÊU CẦU.....	83
4.1 GIỚI THIỆU.....	83
4.2 SỰ RA ĐỜI CỦA MỘT HỆ THỐNG.....	83
4.3 USE CASE.....	83
4.4 QUAN ĐIỂM NGHIỆP VỤ.....	84
4.4.1 Xác định các tác nhân nghiệp vụ (business actors).....	84
4.4.2 Xây dựng Bảng Thuật ngữ của dự án (Project Glossary).....	85
4.4.3 Xác định các trường hợp sử dụng nghiệp vụ (Business Use Cases).....	86
4.4.4 Minh họa các use case bằng biểu đồ giao tiếp (Communication diagram)....	87
4.4.5 Minh họa các use case bằng biểu đồ hoạt động (Activity diagram).....	88
4.5 QUAN ĐIỂM NHÀ PHÁT TRIỂN.....	89
4.5.1 Xác định các actor của hệ thống.....	90
4.5.2 Danh sách các use case.....	90
4.5.3 Biểu đồ use case.....	91
4.5.4 Quan hệ giữa các actor.....	91
4.5.5 Quan hệ giữa các use case.....	92
4.5.6 Điều kiện trước (preconditions), điều kiện sau (postconditions) và kế thừa (inheritance).....	94
4.5.7 Khảo sát Use case.....	95
4.5.8 Các yêu cầu phụ.....	96
4.5.9 Chi tiết của use case.....	96
4.5.10 Bản phác họa giao diện người dùng.....	97
4.5.10 Xếp mức độ ưu tiên các use case hệ thống.....	99
4.6 TỔNG KẾT.....	100
<u>BÀI TẬP.....</u>	<u>100</u>
CHƯƠNG 5 PHÂN TÍCH YÊU CẦU.....	101

MỤC LỤC

5.1 GIỚI THIỆU.....	101
5.2 PHÂN TÍCH TĨNH.....	103
5.2.1. Xác định các lớp.....	103
5.2.2 Quan hệ giữa các lớp.....	104
5.2.3 Thể hiện biểu đồ lớp và biểu đồ đối tượng.....	104
5.2.4 Biểu diễn các quan hệ.....	105
5.2.5 Thuộc tính	108
5.2.6 Lớp liên kết (association classes).....	110
5.2.7 Đối tượng thực và đối tượng không thực (tangible versus intangible objects)	111
5.3 PHÂN TÍCH ĐỘNG.....	112
5.3.1 Lý do phân tích động.....	112
5.3.2 Hiện thực hóa Use Case.....	113
5.3.3 Biểu diễn hiện thực hóa UseCase.....	113
5.3.4 Các lớp biên, lớp thực thể và lớp điều khiển.....	114
5.3.5 Các thành phần của biểu đồ giao tiếp.....	115
5.3.6 Thêm các phương thức vào lớp.....	115
5.3.7 Trách nhiệm (Responsibility).....	116
5.4 CASE STUDY: HỆ QUẢN LÝ ĐĂNG KÝ HỌC THEO TÍN CHỈ.....	118
5.4.2. Phân tích tĩnh (Static Analysis).....	149
B. Biểu đồ tuần tự.....	162
5.5 KẾT LUẬN.....	173
BÀI TẬP.....	173
CHƯƠNG 6 THIẾT KẾ KIẾN TRÚC HỆ THỐNG.....	174
6.1 GIỚI THIỆU.....	174
6.2. ƯU TIÊN THIẾT KẾ.....	175
6.3. CÁC BƯỚC TRONG THIẾT KẾ HỆ THỐNG.....	175
6.4 LỰA CHỌN HÌNH TRẠNG HỆ THỐNG.....	176
6.4.1 Lịch sử của các cấu trúc mạng.....	176
6.4.2 Kiến trúc đơn tầng.....	177
6.4.3 Kiến trúc hai tầng.....	178
6.4.4 Kiến trúc ba tầng.....	182
6.4.5 Máy tính cá nhân.....	184
6.4.6 Mang máy tính.....	184
6.4.5. Internet và World Wide Web.....	185
6.4.6. Intranets.....	185
6.4.7. Các mạng extranet và các mạng riêng ảo.....	186
6.4.8. Kiến trúc Client – Server và kiến trúc phân tán.....	187
6.4.9. Biểu diễn cấu hình mạng trong UML.....	189
6.5 THIẾT KẾ CHO TƯƠNG TRANH.....	189
6.6 AN TOÀN THIẾT KẾ	192
6.6.1 Các khía cạnh bảo mật.....	192
6.6.2 Mã hóa và giải mã	194
6.6.3 Những quy luật chung về an toàn.....	195
6.7 PHÂN RÃ PHẦN MỀM.....	196
6.7.1 Hệ thống và hệ thống con	196
6.7.2 Các tầng.....	197

MỤC LỤC

<u>Java Layers: Applet plus RMI</u>	200
<u>Luồng truyền tin trong các tầng</u>	201
<u>6.8 KẾT LUẬN</u>	204
<u>BÀI TẬP</u>	204
CHƯƠNG 7 LỰA CHỌN CÔNG NGHỆ.....	205
7.1 GIỚI THIỆU	205
7.2 CÔNG NGHỆ TẦNG CLIENT.....	205
7.3 GIAO THỨC GIỮA TẦNG CLIENT VÀ TẦNG GIỮA.....	206
7.4 CÔNG NGHỆ TẦNG GIỮA.....	207
7.5 CÔNG NGHỆ TẦNG GIỮA ĐẾN TẦNG DỮ LIỆU.....	208
7.6 CÁC CÔNG NGHỆ KHÁC.....	209
7.6.1 Tổng quan về J2EE.....	209
Mở đầu.....	209
7.6.2 Giao thức tầng Client và tầng giữa.....	210
7.7 CÁC GÓI UML.....	213
7.8 KẾT LUẬN.....	214
<u>BÀI TẬP</u>	214
CHƯƠNG 8 THIẾT KẾ CÁC HỆ THỐNG CON	215
8.1 GIỚI THIỆU.....	215
8.2 ÁNH XẠ TỪ MÔ HÌNH LỚP PHÂN TÍCH SANG MÔ HÌNH LỚP THIẾT KẾ.....	216
8.2.1 Ánh xạ các phương thức.....	216
8.2.2 Các kiểu biến.....	217
8.2.3 Phạm vi của các trường.....	217
8.2.4 Các toán tử truy nhập.....	218
8.2.5 Ánh xạ các lớp, thuộc tính và hợp thành.....	218
8.2.6 Ánh xạ các kiểu kết hợp khác.....	219
8.2.7 Định danh phổ quát.....	225
8.3 XỬ LÝ LUU TRỮ VỚI CƠ SỞ DỮ LIỆU QUAN HỆ.....	227
8.3.1 Hệ quản trị cơ sở dữ liệu.....	227
8.3.2 Mô hình quan hệ.....	227
8.3.3 Ánh xạ các lớp thực thể.....	230
8.3.4 Ánh xạ hợp thành.....	231
8.3.5 Ánh xạ trạng thái đối tượng.....	232
8.4 HOÀN THIỆN GIAO DIỆN NGƯỜI DÙNG.....	235
8.5.1 Sử dụng proxy và copy.....	242
8.5.2 Phân loại các dịch vụ nghiệp vụ.....	244
8.5.3 Xác nhận các phiên.....	246
8.5.4 Hiệu thực hóa dịch vụ nghiệp vụ.....	247
8.6 SỬ DỤNG FRAMEWORK, KHUÔN MẪU, VÀ THU VIỆN.....	248
8.7 GIAO DỊCH.....	249
8.7.1 Định nghĩa giao dịch.....	249
8.7.2 Đồng thời.....	250
8.7.3 Sử dụng giao dịch với đối tượng.....	251
8.7.4 Giao dịch ở tầng trên.....	251
8.8 XỬ LÝ ĐA NHIỆM.....	251

MỤC LỤC

<u>8.8.1 Quản lý đa nhiệm</u>	252
<u>8.8.2 Quản lý đa tiến trình (Controlling Multiple Threads).....</u>	253
<u>8.8.3 An toàn luồng (Thread Safety).....</u>	254
<u>8.9 KẾT LUẬN.....</u>	256
<u>BÀI TẬP.....</u>	257
CHƯƠNG 9 MẪU THIẾT KẾ SỬ DỤNG LẠI.....	258
<u>9.1 GIỚI THIỆU.....</u>	258
<u>9.2 MẪU THIẾT KẾ LÀ GÌ?.....</u>	259
<u>9.3 HỆ THỐNG CÁC MẪU THIẾT KẾ</u>	260
<u>9.4 NỘI DUNG CÁC MẪU THIẾT KẾ.....</u>	261
9.4.1 Nhóm Creational	261
9.4.2 Nhóm Structural.....	276
9.4.3 Nhóm Behavior.....	296
<u>9.5 CASE STUDY: VÍ ĐỀ ĐỐI TƯỢNG TRUY CẬP DỮ LIỆU (J2EE Patterns).....</u>	321
<u>9.6 KẾT LUẬN</u>	330
<u>BÀI TẬP.....</u>	331
CHƯƠNG 10 ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP.....	332
<u>10.1 GIỚI THIỆU.....</u>	332
<u>10.2 ĐẶC TẢ LÀ GÌ?.....</u>	333
<u>10.3 ĐẶC TẢ HÌNH THỨC.....</u>	334
<u>10.4 ĐẶC TẢ KHÔNG HÌNH THỨC.....</u>	336
<u>10.5 KIỂM TRA ĐỘNG</u>	337
<u>10.6 ĐẶC TẢ HƯỚNG ĐỐI TƯỢNG.....</u>	340
10.6.1 Đặc tả hình thức với OCL.....	341
10.6.2 Đặc tả phi hình thức trong Eiffel.....	342
<u>10.7 THIẾT KẾ BẰNG HỢP ĐỒNG.....</u>	343
10.7.1 Hợp đồng và kế thừa.....	347
10.7.2 Giảm lỗi – kiểm tra mã.....	349
10.7.3 Việc tuân theo các hợp đồng.....	350
<u>10.8 ĐẶT TẢ PHI HÌNH THỨC TRONG JAVA.....</u>	351
10.8.1 Viết tài liệu một hợp đồng với chủ thích.....	351
10.8.2 Kiểm tra các điều kiện một cách linh động.....	352
10.8.3 Đánh dấu các vi phạm hợp đồng sử dụng ngoại lệ.....	352
<u>10.9 KẾT LUẬN.....</u>	353

CHƯƠNG 1 MỞ ĐẦU

1.1 GIỚI THIỆU

Hiện nay, phát triển phần mềm theo cách tiếp cận hướng đối tượng đã trở thành phổ biến trong công nghiệp phần mềm. Mục đích của chương này nhằm trình bày các dạng hệ thống thông tin, những khái niệm cơ bản trong hướng đối tượng như đối tượng, lớp, đóng gói, kế thừa... Sau đó, sẽ đi sâu phân tích các quan hệ giữa các lớp, ưu điểm của đóng gói, các cách sử dụng lại mã nguồn đặc biệt là mẫu thiết kế (design pattern).

1.2 CÁC KIỂU HỆ THỐNG THÔNG TIN

Trong thực tế có rất nhiều kiểu hệ thống thông tin và để dễ phân loại người ta chia các hệ thống thông tin thành 4 mức:

- Các hệ thống mức vận hành (Operational-level Systems)
- Các hệ thống mức tri thức (Knowledge-level Systems)
- Các hệ thống mức quản lý (Management-level Systems)
- Các hệ thống mức chiến lược (Strategic-level Systems)

Mức	Sử dụng	Mục đích	Các kiểu hệ thống	Ví dụ
Các hệ thống vận hành mức vận hành (operational)	Nhà quản lý vận hành hay người phụ trách hệ thống (operational managers).	Trả lời các câu hỏi thường ngày. Lưu vết trình tự các hoạt động và giao dịch hàng ngày.	Hệ thống xử lý giao dịch (TPS: Transaction Processing Systems)	Điều khiển máy móc hay dây chuyền, lưu trữ giao dịch, lập lịch, xử lý đơn đặt hàng.
Các hệ thống mức tri thức (knowledge and data workers).	Nhân viên tri thức và dữ liệu (knowledge and data workers).	Giúp tổ chức, phát hiện và tích hợp tri thức mới từ tri thức đang tồn tại vào nghiệp vụ của họ; điều khiển luồng công việc	Hệ thống hoạt động dựa trên tri thức. Hệ thống tự động hóa văn phòng	Hệ xử lý ngôn ngữ, lập lịch
Các hệ	Nhà quản lý	Phục vụ việc giám sát, điều khiển luồng công việc	Hệ thông tin	Quản lý bán

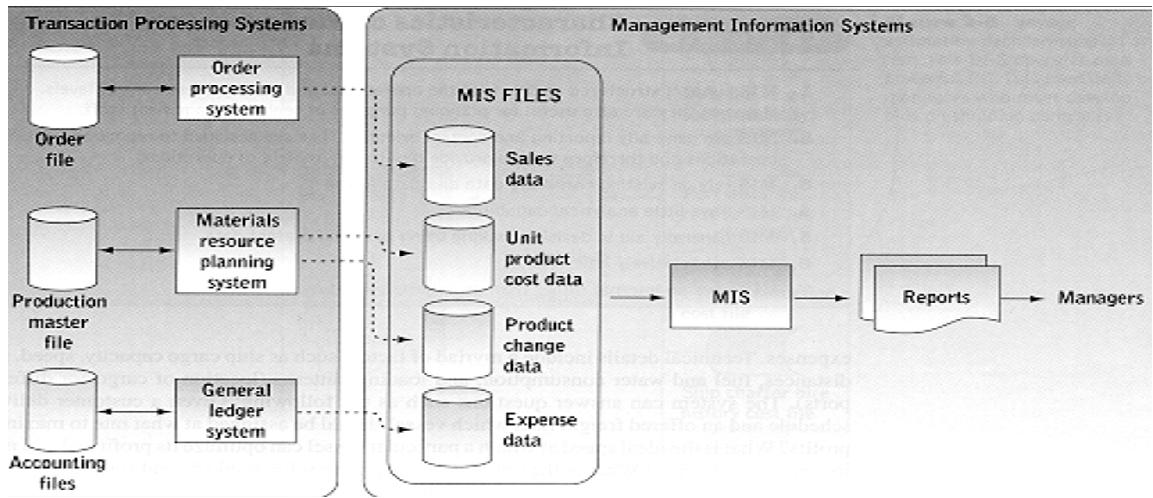
CHƯƠNG 1. MỞ ĐẦU

thống mức quản lý	trung gian điều khiển, ra quyết định và các hoạt động quản trị. Hỗ trợ ra các quyết định quan trọng (ít có cấu trúc) với các yêu cầu về thông tin không rõ ràng.	quản lý (MIS: Management Information Systems), Hệ hỗ trợ quyết định (DSS: Decision Support Systems)	hàng, hàng tồn kho, ngân sách hàng năm, lập kế hoạch sản xuất, phân tích chi phí, phân tích giá cả/lợi nhuận.	
Các hệ thống mức chiến lược	Nhà quản lý chính (senior managers).	Giúp giải quyết và vạch ra các vấn đề chiến lược và các xu hướng lâu dài: so sánh khả năng của tổ chức với những thay đổi và cơ hội xảy ra trong khoảng thời gian dài của môi trường bên ngoài.	Hệ hỗ trợ thực thi (Executive Support Systems – ESS)	Dự đoán ngân sách, xu hướng bán hàng trong 5 năm. Kế hoạch hoạt động trong 5 năm. Kế hoạch về lợi nhuận, nhân lực.

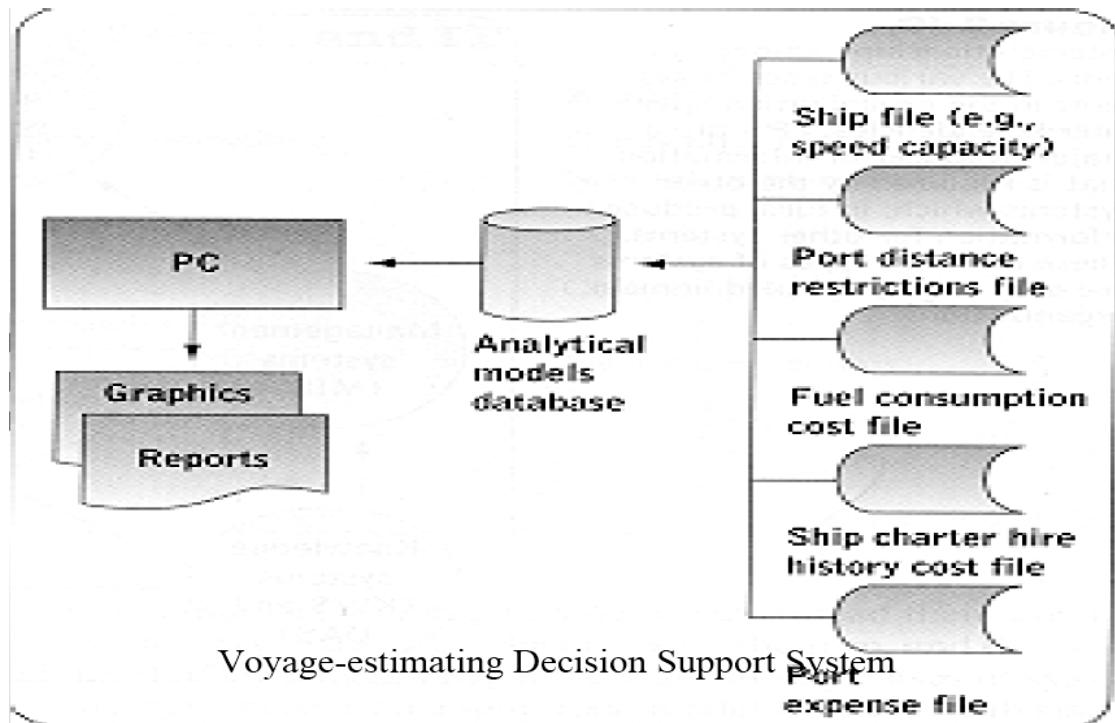
Như vậy, có 6 kiểu hệ thống thông tin tương ứng với 4 mức:

- Hệ thống xử lý giao dịch (Transaction Processing Systems – TPS).
- Hệ thống hoạt động dựa trên tri thức (Knowledge Work Systems – KWS).
- Hệ thống tự động hóa văn phòng (Office Automation Systems – OAS).
- Hệ thống thông tin quản lý (Management Information Systems – MIS).
- Hệ trợ giúp quyết định (Decision Support Systems – DSS).
- Hệ trợ giúp thực thi (Executive Support Systems – ESS).

Tương ứng với các kiểu hệ thống thông tin, các quá trình xử lý được cho trong các hình sau đây:

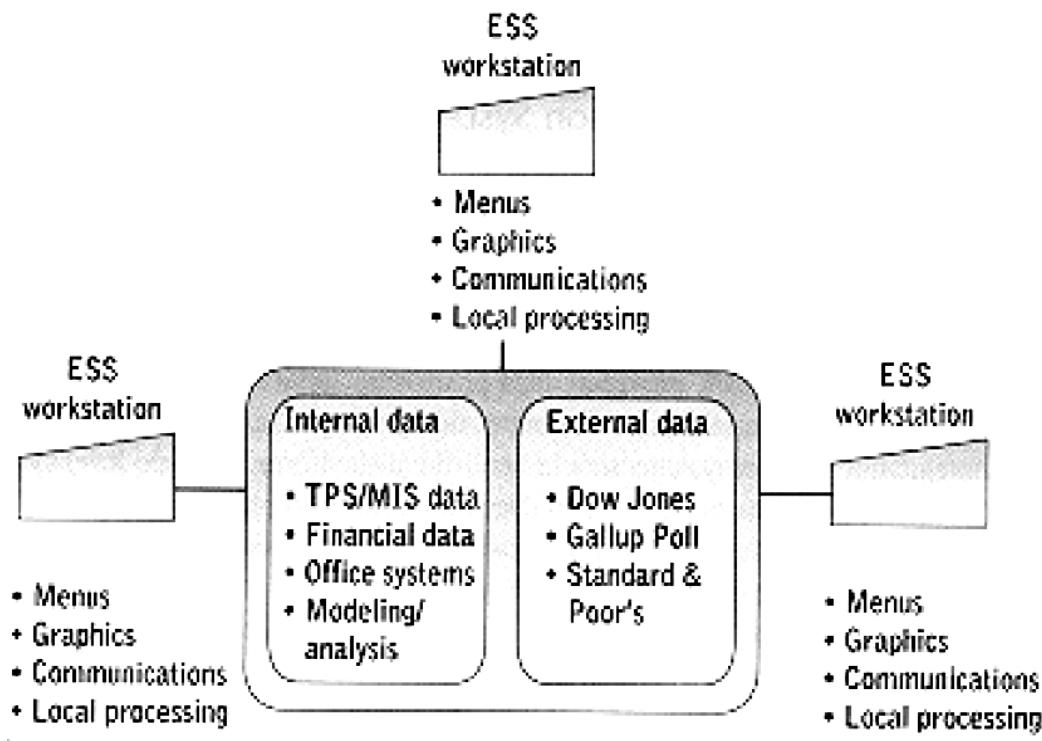


Hình 1.1: Quá trình xử lý trong TPS và MIS



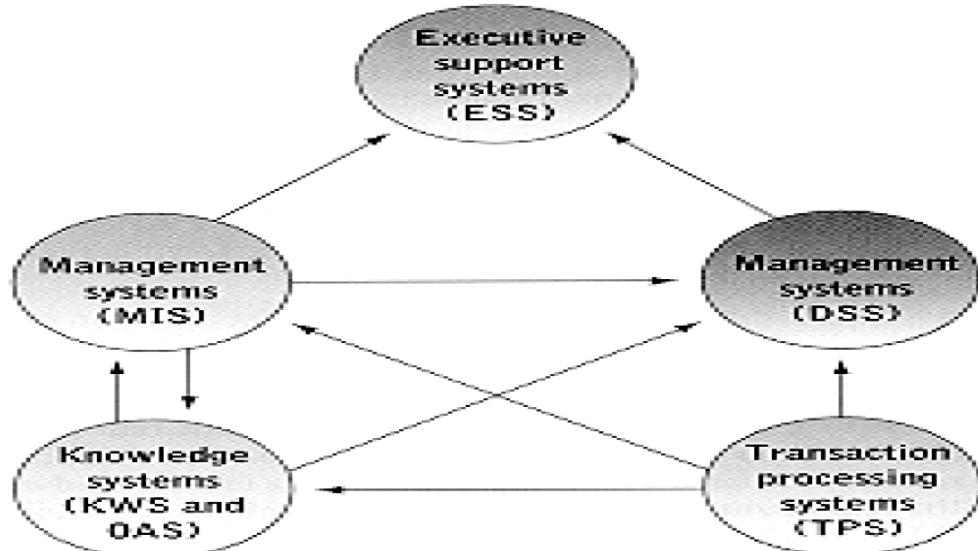
Hình 1.2: Quá trình xử lý trong DPS

CHƯƠNG 1. MỞ ĐẦU



Hình 1.3: Quá trình xử lý trong ESS

Quan hệ giữa các kiểu hệ thống thông tin được cho trong Hình 1.4



Hình 1.4: Quan hệ giữa các kiểu hệ thống thông tin

1.3 MÔ HÌNH HỆ THỐNG DỰA TRÊN CÁCH TIẾP CẬN HƯỚNG ĐỐI TƯỢNG

1.3.1 Mô hình hệ thống

Mô hình hệ thống dựa vào cách tiếp cận hướng đối tượng là phương pháp phân tích các yêu cầu của hệ thống nhằm mục đích xác định các hệ thống con thỏa mãn các tính chất sau:

- Mỗi hệ thống con có trách nhiệm rõ ràng trong việc thực hiện một phần của tác vụ.
- Mỗi hệ thống con nên là độc lập và tự chủ.
- Mỗi hệ thống con nên biết các hệ thống con khác làm gì và làm thế nào để gửi thông điệp để yêu cầu chúng hợp tác và hoàn thành công việc của mình.
- Hệ thống con nên ẩn giấu dữ liệu mà nó sử dụng so với thế giới bên ngoài.
- Hệ thống con nên được thiết kế để có thể dùng lại.

Mô hình hệ thống dựa vào cách tiếp cận hướng đối tượng được sử dụng rộng rãi trong thực tế vì:

- Làm cho những thay đổi trong phát triển các chức năng suốt vòng đời của hệ thống trở nên dễ dàng hơn.
- Làm cho việc dùng lại code của các hệ thống con trong khi thiết kế các hệ thống hơn trở nên dễ dàng hơn.
- Làm cho việc tích hợp các hệ thống con thành hệ thống lớn trở nên dễ dàng hơn.
- Làm cho việc thiết kế các hệ thống phân tán trở nên dễ dàng hơn.

Mô hình hệ thống dựa vào cách tiếp cận hướng đối tượng được sử dụng trong những tình huống sau đây:

- Cần thay đổi một hệ thống đang tồn tại bằng cách thêm chức năng mới.
- Hệ thống lớn có thể được thiết kế bằng cách thu thập các đối tượng sẵn có để sử dụng lại.

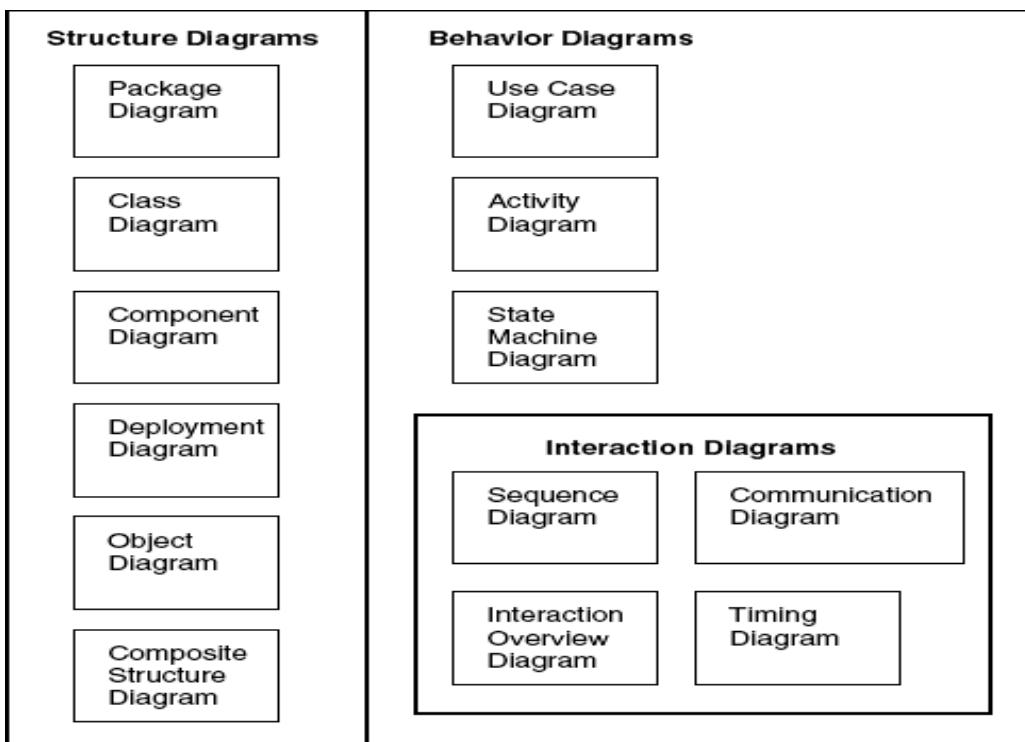
1.3.2 UML

Trong môn Công nghệ phần mềm, chúng ta đã làm quen với UML. Phần này trình bày tổng quan để có cái nhìn đầy đủ hơn 13 kiểu biểu đồ trong UML. Đặc tả UML không nói các biểu đồ này nên sử dụng ở đâu trong từng phương pháp luận nên chúng ta được phép tự do sử dụng chúng ở bất cứ lúc nào thấy thích hợp.

- Biểu đồ use case

CHƯƠNG 1. MỞ ĐẦU

- Biểu đồ lớp (class diagrams)
- Biểu đồ đối tượng (Object diagrams)
- Biểu đồ hoạt động (Activity diagrams)
- Biểu đồ máy trạng thái (State machine diagrams)
- Biểu đồ truyền thông (Communication diagrams)
- Biểu đồ tuần tự (Sequence diagrams)
- Biểu đồ gói (Package diagrams)
- Biểu đồ triển khai (Deployment diagrams)
- Biểu đồ thành phần (Component diagrams)
 - Biểu đồ tương tác chung (Interaction overview diagrams)
 - Biểu đồ thời gian (Timing diagrams)
 - Biểu đồ cấu trúc hợp (Composite structure diagram)



Hình 1.5: Phân loại các kiểu biểu đồ UML

Các biểu đồ nhóm cấu trúc (Structure diagram) thường được dùng kết hợp với nhóm biểu đồ hành vi (behavior diagram) để mô tả một khía cạnh nào đó của hệ thống. Mỗi lớp có một biểu đồ máy trạng thái (state machine diagram) tương ứng để xác định hành vi được điều khiển bởi sự kiện của các thể hiện của lớp đó. Nếu kết hợp biểu đồ đối tượng (object

diagram) với kịch bản (scenario), ta có biểu đồ tương tác (interaction diagram) để biểu diễn thứ tự các thông điệp theo thời gian hoặc theo sự kiện.

- **Biểu đồ gói (Package diagram)**: dùng để tổ chức các sản phẩm của quá trình phát triển, cung cấp cho chúng ta khả năng biểu diễn các thành phần UML theo nhóm.
- **Biểu đồ thành phần (Component diagram)**: dùng để mô hình các module, các thành phần của hệ thống. Một thành phần là một phần của hệ thống được đóng gói, có thể dùng lại và có khả năng thay thế. Ví dụ: loggers, XML parsers, online shopping carts... Đây là các thành phần chung được cộng đồng sử dụng, nhưng bạn có thể tự tạo các thành phần riêng cho mình.
- **Biểu đồ triển khai (Deployment diagram)**: mô hình phần mềm gắn với phần cứng trong thế giới thực. Nó chỉ ra cách tổ chức vật lý và cách giao tiếp giữa phần mềm với phần cứng.
- **Biểu đồ đối tượng (Object diagram)**: dùng khi muốn miêu tả cách mà các đối tượng trong hệ thống sẽ làm việc với nhau trong một trường hợp cụ thể.
- **Biểu đồ cấu trúc hợp (Composite structure diagram)**: chỉ ra cách các đối tượng tạo nên một bức tranh lớn. Chúng mô hình cách mà các đối tượng làm việc với nhau trong một lớp, hoặc cách các đối tượng đạt được mục đích.
- **Biểu đồ use case (Use case diagram)**: mô hình các chức năng mà hệ thống cung cấp.
- **Biểu đồ hoạt động (Activity diagram)**: mô hình các tiến trình nghiệp vụ. Một tiến trình nghiệp vụ là sự kết hợp của các tác vụ nhằm đạt được một mục đích nghiệp vụ nào đó. Ví dụ, phân phối đơn hàng của khách hàng...
- **Biểu đồ máy trạng thái (State machine diagram)**: mô hình các trạng thái của một đối tượng.
- **Biểu đồ tuần tự (Sequence diagram)**: cho phép mô hình một cách chính xác cách các thành phần của hệ thống tương tác với nhau..
- **Biểu đồ giao tiếp (Communication diagram)**: tập trung vào liên kết tương tác. Nó chỉ ra các liên kết nào cần thiết để truyền thông điệp tương tác giữa các bên tham gia
- **Biểu đồ thời gian (Timing diagram)**: mỗi sự kiện có thông tin về thời gian tương ứng với nó. Biểu đồ này miêu tả chính xác khi nào sự kiện xảy ra, mất bao

CHƯƠNG 1. MỞ ĐẦU

lâu để bên tham gia nhận được sự kiện đó và mất bao lâu để bên nhận sự kiện chuyển trạng thái.

- **Biểu đồ tổng quan tương tác (Interaction overview diagram):** cung cấp cái nhìn mức cao về cách mà một số tương tác làm việc với nhau để cài đặt hệ thống.

1.4 ĐỐI TƯỢNG VÀ LỚP

1.4.1 Đối tượng

Đối tượng là một sự vật, một thực thể, một danh từ hoặc bất cứ cái gì mà bạn có thể nhặt lên hoặc đá đi, hay những gì mà bạn có thể tưởng tượng ra với một số đặc tính của nó. Trong khi một số đối tượng thì đang tồn tại, một số thì không có hiện thời. Ví dụ, xe hơi, người, ngôi nhà, cái bàn, chậu cây hoặc áo mưa, hóa đơn bán hàng...

Các đối tượng đều có một số thuộc tính. Ví dụ, xe hơi có các thuộc tính là nhà sản xuất, số hiệu model, màu sắc, giá... Con chó có các thuộc tính là giống chó, tuổi thọ, màu lông, đồ chơi yêu thích. Các đối tượng cũng có hành vi của nó như xe hơi có thể di chuyển từ nơi này đến nơi khác, chó có thể sủa...

Mô tả đối tượng

Một khi đã quyết định làm việc với đối tượng, ta cần phải có cách để biểu diễn chúng theo biểu đồ để có thể miêu tả và suy nghĩ về chúng. Biểu đồ đối tượng trong UML có dạng như hình vẽ:

anObject
attribute1
attribute2
operation1()
operation2()
operation3()

Mỗi đối tượng được mô tả bởi 3 thành phần:

- Tên của đối tượng (có gạch chân).
- Các thuộc tính (tri thức về đối tượng).
- Các thao tác (hành vi của đối tượng).

Ví dụ Đối tượng *máy pha cà phê tự động* **aCoffeeMachine** bao gồm các thao tác:

- Hiển thị đồ uống
- Lựa chọn đồ uống.
- Nhận tiền.
- Pha chế đồ uống.

Máy pha cà phê cần biết những gì để có thể thực hiện các thao tác trên:

- Các đồ uống đang có sẵn.
- Giá đồ uống.
- Công thức pha chế đồ uống.

Như vậy ta có thể biểu diễn đối tượng máy pha cà phê như sau:

aCoffeeMachine

1.4.2 Lớp

Một lớp đóng gói các đặc điểm chung của một nhóm các đối tượng. Các nhà phát triển hướng đối tượng sử dụng các lớp để mô tả các thành phần mà các dạng đối tượng cụ thể sẽ có. Không có lớp, ta sẽ phải thêm

drinkPrices
availableDrinks
drinkRecipes
displayDrinks()
selectDrink()
dispenseDrink()
acceptMoney()

các thành phần này vào từng đối tượng riêng biệt. Biểu diễn lớp trong UML tương tự như biểu diễn đối tượng nhưng tên lớp không gạch chân.

1.5 ĐÓNG GÓI

Một trong các cách cơ bản để đảm bảo thiết kế lớp tốt là tạo ra việc *đóng gói dữ liệu*. Ví dụ, tạo lớp Employee chứa các thông tin mà ứng dụng cần để miêu tả một nhân viên:

```
public class Employee{
    public int employeeID;
    public String firstName;
    public String lastName;
}
```

Các thuộc tính đều ghi là public nghĩa là chúng có thể được truy cập từ mọi lớp bằng các câu lệnh sau:

```
Employee emp = new Employee();
emp.employeeID = 123456;
emp.firstName = "John";
emp.lastName = "Smith";
```

Mặc dù Java cho phép bạn đọc và sửa đổi các trường theo cách này, nhưng thông thường thì bạn không nên làm như vậy. Thay vào đó, ta cần thay đổi phạm vi truy cập tới các trường để giới hạn khả năng truy cập của chúng bằng cách sử dụng các phương thức set, get cho mỗi trường để có thể truy cập tới thuộc tính.

```
public class Employee {
    protected int employeeID;
    protected String firstName;
```

CHƯƠNG 1. MỞ ĐẦU

```
protected String lastName;
public int getEmployeeID() {
    return employeeID;
}
public void setEmployeeID(int id) {
    employeeID = id;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String name) {
    firstName = name;
}
public String getLastname() {
    return lastName;
}
public void setLastName(String name) {
    lastName = name;
}
```

```
}
```

Lợi ích của việc đóng gói

- Đóng gói an toàn
- Đóng gói làm đơn giản hóa việc chuyển đổi một lớp đang tồn tại thành một lớp được dùng để tạo các đối tượng phân tán (từ xa). Đối tượng phân tán thường nằm ở máy chủ, các phương thức của nó có thể được gọi bởi các ứng dụng nằm trên các máy khác (nói cách khác là có thể được gọi thông qua mạng). Do đó, các trường thuộc đối tượng đó không thể được gọi trực tiếp như với đối tượng cục bộ. Tuy nhiên, nếu định nghĩa các phương thức set và get, ta có thể làm được điều này.
- Ngoài ra, việc sử dụng các phương thức set và get sẽ cô lập chúng ta khỏi những thay đổi trong cài đặt một tính chất. Ví dụ, có thể đổi employeeID từ kiểu int sang kiểu String mà không ảnh hưởng tới các lớp khác, miễn là chúng ta thực hiện việc chuyển đổi thích hợp trong các phương thức set và get.

Đóng gói ẩn giấu việc vài đặt chi tiết

```
public class Employee {
    protected String employeeID;
    protected String firstName;
    protected String lastName;
```

```

public int getEmployeeID() {
    return Integer.parseInt(employeeID);
}
public void setEmployeeID(int id) {
    employeeID = Integer.toString(id);
}
public String getFirstName() {
    return firstName;
}
public void setFirstName(String name) {
    firstName = name;
}
public String getLastName() {
    return lastName;
}
public void setLastName(String name) {
    lastName = name;
}
}

```

Mặc dù, việc cài đặt thuộc tính employeeID bị thay đổi, nhưng các lớp khác khi đọc và sửa đổi thuộc tính này sẽ không nhìn thấy bất kỳ thay đổi nào trong hành vi của nó, vì việc thay đổi trong cài đặt được che giấu bởi các phương thức set và get. Đóng gói các thuộc tính của lớp cho phép định nghĩa các giá trị dẫn xuất có khả năng truy cập. Ví dụ, bạn có thể định nghĩa phương thức getFullName() trong lớp Employee để trả về họ tên đầy đủ dưới dạng một chuỗi đơn:

```

public String getFullName() {
    Return firstName + " " + lastName;
}

```

Tất nhiên, có thể lấy giá trị dẫn xuất mà không cần tạo phương thức get, nhưng thường sẽ tạo ra các đoạn mã trùng nhau khi dẫn xuất giá trị. Ví dụ, để dẫn xuất tên đầy đủ ở một số vị trí trong ứng dụng, bạn phải copy đoạn mã (firstName + “ “ + lastName) vào các vị trí đó và nếu thay đổi cài đặt, bạn phải thay đổi từng vị trí như khi bạn muốn có thêm middleName. Nhưng khi sử dụng phương thức getFullName() thì bạn chỉ cần thay đổi ở một vị trí trong mã nguồn mà thôi.

CHƯƠNG 1. MỞ ĐẦU

1.6 QUAN HỆ GIỮA CÁC LỚP

1.6.1 Các quan hệ phụ thuộc: Association, aggregation và Composition

Không có một đối tượng nào có thể tồn tại riêng lẻ. Tất cả các đối tượng đều được liên kết với các đối tượng khác, trực tiếp hoặc gián tiếp, mạnh hoặc yếu. Các liên kết cho phép ta tìm ra nhiều thông tin và hành vi phụ. Ví dụ, nếu ta đang xử lý một đối tượng Customer biểu diễn tên Lan và ta muốn gửi cho Lan một bức thư thì ta cần phải biết Lan sống ở số 42 Nguyễn Trãi, Hà nội. Ta muốn có thông tin về địa chỉ được lưu trữ trong đối tượng Address, vì vậy cần có kết nối giữa Customer và Address để biết thư được gửi đến đâu.

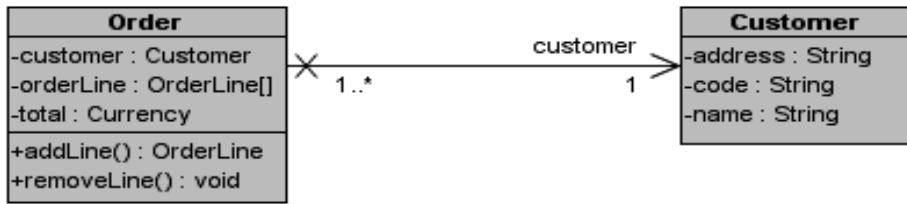
Khi mô hình hóa đối tượng, cần liên kết chúng theo hai cách chính sau đây: association và aggregation. Rất khó để tìm ra sự khác nhau giữa 2 cách này, sau đây là một số gợi ý:

- **Association** là một dạng liên kết yếu; các đối tượng là một phần của một nhóm, hoặc một họ các đối tượng nhưng chúng không hoàn toàn phụ thuộc vào đối tượng khác. Ví dụ, xét các đối tượng xe hơi, người lái xe và hai khách đi xe. Khi người lái xe và 2 khách đi xe ở trong xe, họ được liên kết với nhau vì họ cùng đi về một hướng, họ cùng chiếm một khoảng không gian...nhưng liên kết sẽ mất khi xe trả một vị khách nào đó đến nơi yêu cầu, vị khách đó sẽ không còn liên kết với các đối tượng khác nữa.
- **Aggregation**: nghĩa là đặt các đối tượng có liên kết với nhau cùng nhau để tạo ra đối tượng lớn hơn. Ví dụ, máy tính được tạo bởi các bộ phận màn hình, ố cứng, bàn phím...Aggregation thường có dạng phân cấp **bộ phận-toàn thể** (part-whole). Aggregation ám chỉ sự phụ thuộc giữa bộ phận và toàn thể, ví dụ, màn hình vẫn là màn hình nếu lấy nó ra khỏi máy tính, nhưng máy tính sẽ mất tác dụng nếu thiếu màn hình.

Nhu đã nói ở trên, thật khó để phân biệt giữa association và aggregation. Tuy nhiên, có thể đặt câu hỏi “Điều gì sẽ xảy ra nếu bỏ đi một trong các đối tượng?” để xác định quan hệ đó là association hay aggregation. Nhưng không phải là lúc nào nó cũng giải quyết được vấn đề mà cần phải suy nghĩ thường xuyên và cần có kinh nghiệm. Ta thường phải chọn lựa giữa association và aggregation, vì lựa chọn đó có thể ảnh hưởng đến cách ta thiết kế phần mềm.

Ví dụ “xử lý hóa đơn”

Liên kết (Association): Quan hệ Association thường được miêu tả giống như quan hệ tham chiếu (reference), trong đó một đối tượng nắm giữ một tham chiếu đến đối tượng khác.



Hình 1.6: Quan hệ liên kết

Cài đặt:

```

package relation;
public class Customer {
    private String _address;
    private String _code;
    private String _name;
}

package relation;
public class Order {
    private relation.Customer _customer;
    private OrderLine[] _orderLine;
    private Currency _total;
    public OrderLine addLine() {
        throw new UnsupportedOperationException();
    }
    public void removeLine() {
        throw new UnsupportedOperationException();
    }
}
  
```

Đây là quan hệ dễ cài đặt nhất, trong UML nó được biểu diễn là một đường thẳng nối giữa 2 lớp. Chiều mũi tên nói lên rằng ta gọi đối tượng Customer từ đối tượng Order nhưng không gọi Order từ Customer.

Tập hợp (Aggregation)

Quan hệ giữa lớp OrderList và lớp Order là Aggregation. Nghĩa là danh sách OrderList bao gồm nhiều Order nhưng các Order có đời sống riêng của nó và không cần phải là một bộ phận của danh sách OrderList cụ thể.

CHƯƠNG 1. MỞ ĐẦU



Hình 1.7: Quan hệ tập hợp

Cài đặt:

```

package relation;
public class Order {
    private relation.Customer _customer;
    private OrderLine[] _orderLine;
    private Currency _total;

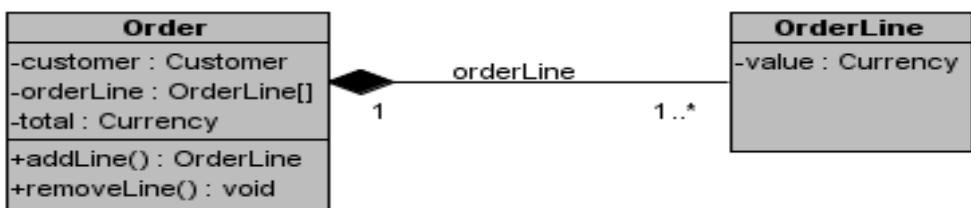
    public OrderLine addLine() {
        throw new UnsupportedOperationException();
    }
    public void removeLine() {
        throw new UnsupportedOperationException();
    }
}
package relation;
import java.util.Vector;
import aggregation.Order;

public class OrderList {
    Vector<Order> _order = new Vector<Order>();
    public void add() {
        throw new UnsupportedOperationException();
    }
    public int getCount() {
        throw new UnsupportedOperationException();
    }
    public OrderIterator getIterator() {
        throw new UnsupportedOperationException();
    }
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
  
```

Aggregation biểu diễn trong UML là đường thẳng có hình quả trám rỗng ở 1 đầu. Điều này xác định không có quan hệ sở hữu trong quan hệ này và các thẻ hiện của lớp được kết hợp sẽ được quản lý bên ngoài lớp đang kết hợp. Hình chữ nhật nhỏ có từ Customer chỉ ra một giới hạn, trong hàm khởi tạo của lớp OrderList có một tham số là Customer để giới hạn số lượng Order tương ứng với Customer đó trong Danh sách Order.

Hợp thành, cấu thành (Composition)

Quan hệ giữa Order và OrderLine là Composition. Các OrderLine của một Order đều thuộc về Order và không có ý nghĩa bên ngoài Order đó. Order có trách nhiệm hoàn toàn trong việc tạo, quản lý, và xóa bất kỳ OrderLine nào trong Order đó. Biểu diễn trong UML là đường thẳng một đầu có hình quả trám màu đen.



Hình 1.8: Quan hệ cấu thành

Cài đặt:

```

package relation;
public class OrderLine {
    private Currency _value;
    aggregation.Order _orderLine;
}

package relation;
public class Order {
    private Customer _customer;
    private OrderLine[] _orderLine;
    private Currency _total;
    aggregation.OrderList _unnamed_OrderList_;

    public OrderLine addLine() {
        throw new UnsupportedOperationException();
    }
    public void removeLine() {
        throw new UnsupportedOperationException();
    }
}
  
```

CHƯƠNG 1. MỞ ĐẦU

1.6.2 Kế thừa

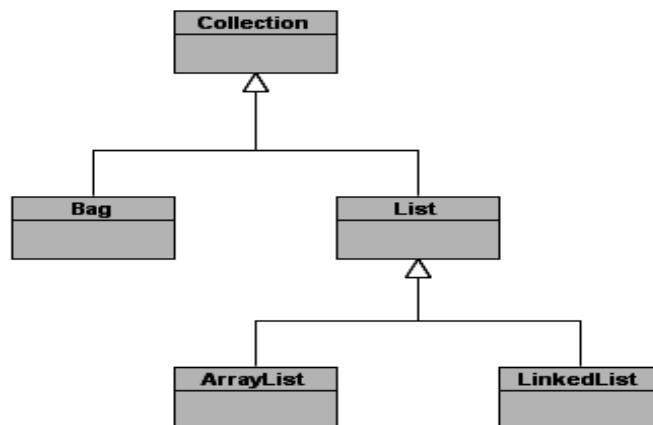
Kế thừa cho phép một lớp lấy lại một số đặc điểm, thuộc tính từ lớp cha và sau đó thêm vào một số đặc trưng riêng của nó. Kế thừa cho phép ta nhóm một số lớp thành một lớp tổng quát hơn để có thể đưa ra các đối tượng rộng hơn về thế giới mà ta đang sống.

Thiết kế kiến trúc phân cấp lớp

Ví dụ, ta muốn mô hình các bộ sưu tập để lưu giữ các đối tượng cho sử dụng về sau này. Sau khi phân tích, ta thấy có 4 kiểu bộ sưu tập như sau:

- **List**: lưu giữ các đối tượng theo thứ tự mà nó được đưa vào.
- **Bag**: lưu đối tượng nhưng không theo thứ tự.
- **LinkedList**: lưu các đối tượng theo thứ tự bằng cách cài đặt một chuỗi các đối tượng trong đó, mỗi đối tượng chỉ tới một đối tượng khác trong chuỗi. Danh sách liên kết cho phép cập nhật dễ dàng, nhưng truy cập chậm vì ta phải duyệt toàn bộ danh sách.
- **ArrayList**: lưu các đối tượng theo thứ tự sử dụng như là một mảng, nghĩa là một chuỗi các ô nhớ liên tiếp. Mảng cho phép truy cập nhanh nhưng cập nhật chậm vì ta có thể phải dịch các phần tử hoặc tạo một mảng mới mỗi khi cập nhật.

Làm thế nào thiết kế kiến trúc phân cấp kiểu kế thừa? Điểm mấu chốt là xem xét sự tương đồng giữa các khái niệm với nhau. Rõ ràng, chúng đều là các bộ sưu tập, vì vậy lớp Collection sẽ đưa lên đầu. Trong 4 bộ sưu tập trên, chỉ có Bag là không lưu các đối tượng theo thứ tự, còn lại đều lưu đối tượng theo thứ tự. Do đó, ta đặt Bag trực tiếp ngay bên dưới Collection trong một nhánh riêng. Ta thấy, List không có ràng buộc gì về cài đặt bên trong, trong khi LinkedList và ArrayList thì có. Vì vậy, List sẽ là lớp cha của LinkedList và ArrayList. Ta có bản thiết kế sau:

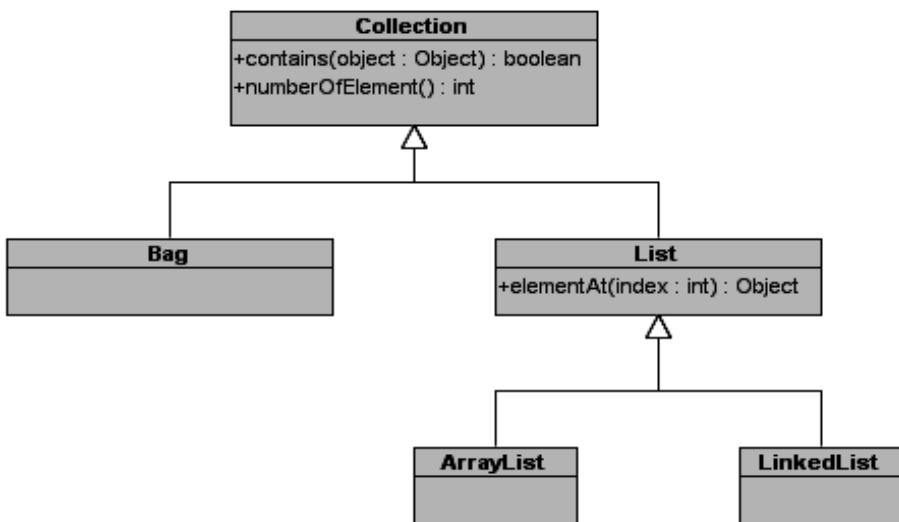


Hình 1.9: Thiết kế kiến trúc phân cấp

Trong thực tế, ta thường làm ngược lại. Trước hết, ta miêu tả các lớp ở mức lá (Bag, ArrayList và LinkedList) và sau đó, ta tìm kiếm các khái niệm tổng quát hơn. Trong khi phát triển mô hình kế thừa, ta tìm các *message* có thể chia sẻ, đưa chúng vào mô hình kế thừa càng ở các lớp trên càng tốt. Ta sẽ tìm các message trước khi tìm các thành phần lớp khác vì các message biểu diễn giao tiếp giữa các đối tượng với thế giới bên ngoài. Xét các message trong mô hình phân cấp Collection:

- `contains(:Object)`: boolean Tìm các đối tượng trong bộ sưu tập và trả về true nếu bộ sưu tập chứa tham số, ngược lại trả về false.
- `elementAt(:int)`: Object trả về đối tượng ở vị trí được xác định bởi tham số truyền vào.
- `numberOfElements()`: int trả về số nguyên là số đối tượng trong bộ sưu tập.

Đặt các message này vào lớp nào? Contains() có thể dùng đối với mọi bộ sưu tập, vì vậy đặt nó trong Collection. ElementAt(: int) lấy đối tượng ở vị trí xác định nên phải đặt trong List, để tránh sự lặp lại nếu để trong ArrayList và LinkedList. NumberOfElement() có thể dùng với mọi bộ sưu tập nên để nó trong Collection. Ta có bản thiết kế sau đây:



Hình 1.10: Đưa các message vào các lớp

Cài đặt phân cấp lớp

Vì thuật toán tìm kiếm khác nhau đối với bộ sưu tập theo thứ tự và không theo thứ tự, nên contains() không thể cài đặt trong Collection. Ở Bag và List ta sẽ cài đặt hàm contains() khác nhau.

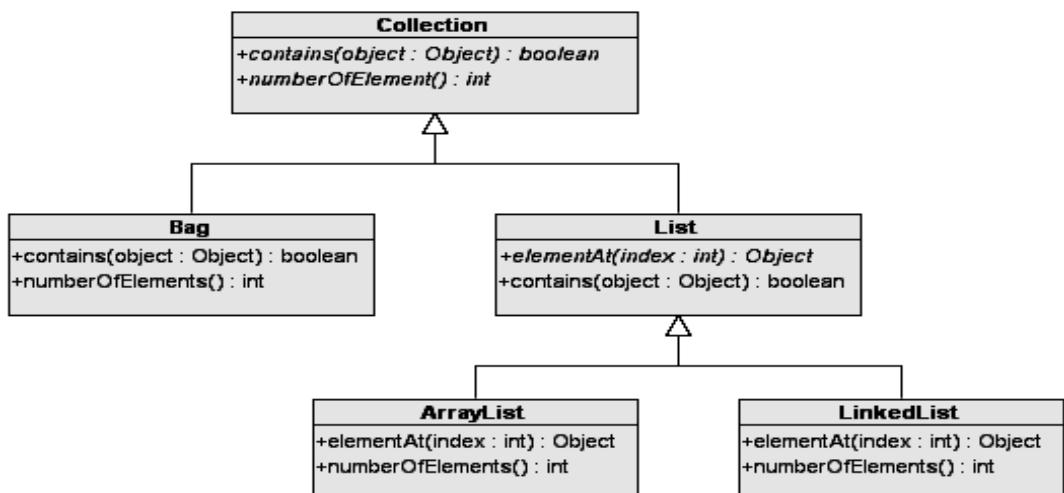
CHƯƠNG 1. MỞ ĐẦU

```
//Cài đặt hàm contains trong lớp List  
boolean contains(Object obj) {  
    for (int i = 0; i < numberOfElements(); ++i) {  
        if (elementAt(i) == obj) {  
            return true;  
        }  
    }  
    return false;  
}
```

Hàm elementAt được cài đặt khác nhau trong hai lớp ArrayList và LinkedList. Vì vậy, ta phải có hai phương thức elementAt riêng: một cho lớp ArrayList – truy cập các phần tử một cách trực tiếp, một cho lớp LinkedList – duyệt toàn bộ danh sách. Cài đặt phương thức numberOfElements() phụ thuộc vào việc ta lưu giá trị đó trong một trường hay tính nó khi cần.

- *Lưu số phần tử trong một trường*: trường này sẽ tăng khi thêm phần tử và giảm khi xóa phần tử. Cách này cho phép ghi số phần tử một cách nhanh chóng tùy thuộc vào bộ nhớ nhưng chậm trong việc thêm và xóa đối tượng.
- *Tính toán số phần tử khi cần*: đối với LinkedList thì chậm vì phải duyệt toàn bộ số phần tử. Đối với ArrayList và Bag, các đối tượng bên trong có thể lưu trữ số phần tử, do đó sẽ nhanh hơn. Cách này sẽ không tốn bộ nhớ và không bị chậm trong việc thêm và xóa đối tượng.

Ta có thiết kế lớp như sau:



Hình 1.11: Đưa các thông điệp vào các lớp

Các phương thức in nghiêng là các phương thức ảo, còn lại là các phương thức thực.

Phương thức ảo chỉ có tên mà không có các dòng code cài đặt, còn phương thức thực thì ngược lại.

Các lớp ảo

Lớp ảo là lớp có ít nhất một phương thức ảo – nó có thể nằm trong lớp đó hoặc được kế thừa từ lớp cha. Khi thiết kế phân cấp lớp, ta nên hình dung trong đầu rằng lớp cha cao nhất là ảo.

Định nghĩa lại các phương thức

Hướng đối tượng cho phép ta định nghĩa lại các phần tử thành kế thừa. Ở dạng đơn giản nhất, định nghĩa lại cho phép lớp con thay đổi việc cài đặt phương thức được kế thừa. Tên phương thức vẫn như cũ nhưng các dòng code trong thân sẽ được thay thế. Hoặc tạo chuyển message từ private sang public; đổi tên hoặc kiểu của một thuộc tính.

Ở đây, ta chỉ tập trung vào định nghĩa lại nội dung của phương thức, vì đó là lý do quan trọng nhất cho việc định nghĩa lại. Có 3 lý do chính giải thích tại sao ta phải định nghĩa lại:

- Phương thức được kế thừa là ảo và ta muốn biến nó thành thực, bằng cách thêm vào một số dòng code. Ví dụ, contains là ảo trong Collection nhưng cần là thực trong Bag và List
- Phương thức cần phải làm thêm một số công việc nằm ở lớp con
- Ta có thể cung cấp một cài đặt tốt hơn cho lớp con. Ví dụ, nếu thêm một chỉ số vào lớp LinkedList, ta có thể định nghĩa lại contains để làm việc nhanh hơn thuật toán tuần tự được dùng bởi List.

Khi ta thêm công việc, hãy chắc chắn rằng định nghĩa lớp cha vẫn làm mọi thứ bình thường – để tăng việc chia sẻ mã nguồn và đơn giản hóa việc bảo trì (ví dụ, nếu sửa đổi định nghĩa của lớp cha, lớp con sẽ tự động có hành vi mới). Mỗi ngôn ngữ hướng đối tượng cho phép phương thức được định nghĩa lại có thể gọi phương thức trong lớp cha.

```
//Ví dụ trong Java:  
void initialize() {  
    //invoke the inherited initialize method  
    super.initialize();  
    ...  
}
```

Đa kế thừa

Mỗi lớp con có nhiều lớp cha. Java có một dạng đa kế thừa đối với interface và lớp abstract (không cài đặt phương thức).

CHƯƠNG 1. MỞ ĐẦU

1.7 CÁCH SỬ DỤNG LẠI MÃ NGUỒN

Có một số cách để sử dụng lại mã nguồn:

- **Dùng lại các chức năng trong một hệ thống:** Dạng đơn giản nhất là dùng lại code (được sử dụng trong phát triển các hệ thống theo cách truyền thống) liên quan đến việc viết các hàm tiện ích được gọi từ nhiều nơi. Ví dụ, bạn thấy một số phần trong hệ thống cần được tìm kiếm thông qua một danh sách tên khách hàng, do đó bạn viết một hàm tìm kiếm chung. Việc viết các hàm có khả năng dùng lại khác với viết các hàm chia tiến trình phức tạp thành các bước đơn giản.
- **Dùng lại các phương thức trong một đối tượng:** Các phương thức được đóng gói trong một đối tượng có thể được gọi từ các phương thức khác. Ví dụ, phương thức drawFilledRectangle trong lớp GUIComponent có thể được sử dụng bởi bất kỳ phương thức nào của GUIComponent để tô màu một vùng màn hình với màu nền xác định. Bạn nên nghĩ đến việc sử dụng lại các phương thức trong một đối tượng bất cứ khi nào cần. Các phương thức nonpublic trong một đối tượng thường được dùng để chia nhỏ các tiến trình phức tạp theo kiểu truyền thống.
- **Dùng lại các lớp trong một hệ thống:** Nhiều lớp mà ta định nghĩa có thể được dùng trong các phần khác nhau của hệ thống. Ví dụ, nếu bạn định nghĩa lớp Customer để sử dụng trong hệ thống marketing, bạn muốn đối tượng Customer tương tự xuất hiện trong nhiều phần khác nhau của mã nguồn hệ thống. Kiểu dùng lại này là cơ sở của cách tiếp cận hướng đối tượng.
- **Dùng lại các hàm trong nhiều hệ thống:** các chức năng chung có thể được dùng lại (trong phát triển hệ thống kiểu truyền thống cũng như hướng đối tượng) trong các hệ thống khác mà bạn và các đồng nghiệp tạo ra. Ví dụ, bạn có thể viết một chức năng lấy ra năm tham gia vào công ty từ số thứ tự bảng lương của nhân viên. Với một chức năng được dùng lại bởi đồng nghiệp, bạn phải giải thích cho họ hiểu nó, có thể để vào nơi chứa tài nguyên dùng lại, một cơ sở dữ liệu các chức năng mà các lập trình viên có thể xem xét khi viết mã nguồn mới.
- **Dùng lại các lớp trong các hệ thống:** Ta có thể tạo ra và dùng lại toàn bộ một lớp (bao gồm các thuộc tính và các phương thức) hơn là chỉ một hàm đơn độc. Ví dụ đối tượng Employee đóng gói các thuộc tính và một tập các thao tác của nhân viên được sử dụng trong toàn bộ công ty. Những người ưa chuộng hướng đối tượng là những người muốn tạo ra nơi chứa tài nguyên dùng lại để chứa các lớp hơn là các hàm.
- **Dùng lại các lớp trong tất cả các hệ thống:** một bộ phận trong phần mềm cũng tương tự như một bộ phận trong phần cứng. Các thành phần trong

phần mềm được thiết kế để có thể dùng lại trong nhiều ngữ cảnh, được đóng gói chặt chẽ (bên yêu cầu không biết công việc bên trong là gì), đi cùng với phong cách chuẩn của interface, và được cung cấp sẵn từ bên thứ 3, thường là phải trả chi phí. Mỗi ngôn ngữ lập trình hướng đối tượng có một khuôn mẫu các thành phần trong phần mềm, ví dụ Java có JavaBeans.

- **Các thư viện hàm:** các hàm có liên quan, có chất lượng tốt có thể được nhóm lại thành một thư viện, vì vậy chúng luôn sẵn có. Ví dụ thư viện stdio, bắt nguồn từ các hệ thống Unix, cung cấp khả năng I/O cho các lập trình viên C. Các thư viện hàm được dùng trong cả việc phát triển phần mềm truyền thống cũng như phát triển phần mềm hướng đối tượng. Các thư viện được thiết kế tốt đôi khi được chuẩn hóa bởi các tổ chức như ISO hay ANSI. Các thư viện hàm có thể xuất phát từ bên trong công ty, sử dụng miễn phí hoặc bán để kiếm lời.
- **Các thư viện lớp:** là một sự phát triển của các thư viện chức năng, thường là các lớp hoàn chỉnh chứ không đơn thuần là các hàm. Viết một thư viện lớp đòi hỏi có nhiều kinh nghiệm. Ví dụ thư viện J2EE, cung cấp mã nguồn cho tất cả các kiểu dùng lại được liệt kê ở trên.
- **Mẫu thiết kế:** một mẫu thiết kế là một sự miêu tả cách tạo ra các phần của hệ thống HDT một cách hiệu quả. Các bản mẫu cũng được áp dụng trong các lĩnh vực khác như kiến trúc hệ thống. Mỗi mẫu là một miêu tả ngắn, chi tiết, cho biết khi nào dùng nó và code minh họa. Thiết kế các mẫu đòi hỏi có nhiều kinh nghiệm, nhưng không bằng việc tạo ra thư viện lớp.
- **Framework:** là một cấu trúc đã có sẵn để bạn đưa code vào. Trong trường hợp HDT, một framework bao gồm một số lớp được viết trước, cùng với tài liệu miêu tả hướng dẫn lập trình viên các quy tắc phải tuân theo. Ví dụ EJB framework – bao gồm thư viện J2EE và tài liệu đặc tả, dài hàng trăm trang – chỉ ra cách để lập trình viên viết được các thành phần có khả năng dùng lại, và cách để các bên thứ 3 cài đặt máy chủ ứng dụng Java. Phần lớn các framework được thiết kế bởi các chuyên gia cao cấp.

1.8 KẾT LUẬN

Chương này đã trình bày khái quát một số vấn đề trong phân tích và thiết kế hướng đối tượng. Nội dung tập trung vào những khái niệm đơn giản như đối tượng, lớp... đến những vấn đề phức tạp như xác định quan hệ giữa các lớp. Những khái niệm này đã được trình bày theo chuẩn UML cùng với những ví dụ minh họa. Tuy nhiên, phân tích và thiết kế theo hướng đối tượng bao gồm nhiều vấn đề và cần sự nghiên cứu lâu dài cũng như trải nghiệm thực tế mới có thể trở thành chuyên gia trong lĩnh vực thú vị này. Các chương tiếp theo sẽ giúp sinh viên hiểu rõ hơn những vấn đề này.

CHƯƠNG 1. MỞ ĐẦU

BÀI TẬP

1. Sinh viên tìm hiểu và khảo sát các hệ quản lý học tập theo tín chỉ sau đó đề xuất các yêu cầu của hệ thống này. Liệt kê các actor cùng các use case có thể có cho hệ thống.
2. Hãy chọn ra các lớp từ hệ quản lý học tập theo tín chỉ trong câu 1 và chỉ ra các quan hệ giữa chúng
3. Hãy thêm các thuộc tính và các phương thức vào các lớp chọn được
4. Hãy giải thích lý do chọn lựa các mức độ truy nhập các thuộc tính trên
5. Hãy giải thích lý do gán phương thức vào các lớp
6. Tương tự như các Câu 1 – 5 cho hệ quản lý Thư viện điện tử
7. Sinh viên có thể tham khảo các tài liệu thêm để viết ví dụ một mẫu thiết kế

TÀI LIỆU THAM KHẢO THÊM

[1] Pro Java Programming, Second Edition - BRETT SPELL, 2007.

[2] Data Access Object Pattern

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>

[3] Gregor Engels, Object-Oriented Modeling: A Roadmap, <http://wwwcs.uni-paderborn.de/cs/ag-engels/Papers/2000/EG00objectorientedModelling.pdf>

CHƯƠNG 2 PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

2.1 MỞ ĐẦU

Sản xuất phần mềm dù kích cỡ lớn hay nhỏ, dù phát triển đầy đủ trọn vẹn cả một phần mềm hay chỉ là một phần của phần mềm thì sản phẩm đó cũng thường được tạo ra bởi nhiều người và như vậy cần phải có một *phương pháp luận* (methodology) để sản xuất phần mềm đó. Ngay cả khi một phần nhỏ của phần mềm được phát triển bởi một người vẫn có thể được cải tiến bằng cách luôn luôn có một phương pháp luận trong đầu.

Phương pháp luận được hiểu một cách nôm na là cách thức làm việc một cách có hệ thống. Bản chất của nó là một *tiến trình* (process) có thể lặp đi lặp lại từ trạng thái đầu tiên trong việc phát triển phần mềm (một ý tưởng được hình thành hoặc một cơ hội kinh doanh mới) và xuyên suốt đến phần bảo trì một hệ phần mềm đã được cài đặt. Cũng giống như tiến trình, một phương pháp luận cần phải chỉ rõ ra cái mà chúng ta mong được sản xuất ra khi ta thực hiện theo tiến trình đó (những cái mà sản phẩm cuối cùng nên có). Một phương pháp luận cũng có thể bao gồm cả những lời khuyên hay kỹ thuật để quản lý tài nguyên, kế hoạch, lập lịch và các nhiệm vụ quản lý khác.

Mặc dù hầu hết các phương pháp luận được đề xuất dành cho các nhóm phát triển sản xuất các hệ phần mềm cỡ lớn, nhưng sự hiểu biết căn bản về một phương pháp luận tốt là rất cần thiết cho các nhà phát triển đơn độc đang nhắm đến giải quyết những vấn đề nhỏ. Điều này đúng vì:

- Một phương pháp luận có thể giúp áp đặt những quy định trong viết code.
- Ngay cả các bước cơ bản của một phương pháp luận cũng có thể giúp làm cho chúng ta hiểu vấn đề hơn, từ đó cải tiến được chất lượng của việc giải quyết vấn đề.
- Viết code chỉ là một trong nhiều hoạt động của việc phát triển phần mềm nên khi tiến hành một số các hoạt động khác sẽ giúp chúng ta hiểu được các lỗi về khái niệm và thực tế trước khi chúng ta tiến hành viết mã nguồn.
- Ở mọi giai đoạn, phương pháp luận đều đặc tả cái mà chúng ta nên làm tiếp theo, vì vậy chúng ta không cần phải hỏi và nghĩ “Được, thế thì làm cái gì bây giờ?”.
- Phương pháp luận giúp chúng ta viết code dễ mở rộng hơn (dễ thay đổi hơn), dễ sử dụng lại hơn (có thể áp dụng cho những bài toán khác) và dễ dàng hơn khi soát lỗi (bởi vì nó có nhiều tài liệu hơn).

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

Những dự án phát triển lớn cũng có lợi từ:

- **Tài liệu:** Tất cả các phương pháp luận đều chú trọng khâu viết tài liệu trong mọi giai đoạn phát triển, để khi một hệ thống đã hoàn thành không trở thành một khối không thể hiểu được.
- **Giảm được độ trễ:** Vì các luồng công việc, hoạt động, vai trò và những sự phụ thuộc bên trong được hiểu một cách tốt hơn nên có ít cơ hội cho con người nhัน rỗi.
- **Cải thiện may rủi:** Những sự may rủi trong việc phân phối thời gian và ngân sách được cải thiện
- **Cải tiến giao tiếp:** Giao tiếp tốt hơn giữa những người sử dụng, những người bán sản phẩm, những người quản lý và những nhà phát triển. Một phương pháp luận tốt dựa trên lý luận và trực giác chung cho nên sẽ dễ dàng cho những người tham gia nắm bắt được những cái cơ bản. Vì thế, sự phát triển sẽ tuân theo trật tự hơn, ít có những hiểu lầm hơn.
- **Tính lặp lại:** Vì những hoạt động được trình bày rõ ràng nên những dự án giống nhau có thể được phân theo những giai đoạn tương tự nhau và chi phí như nhau. Nếu chúng ta sản xuất ra những hệ thống tương tự nhau lặp đi lặp lại cho những khách hàng khác nhau (ví dụ như những gian hàng thương mại điện tử), chúng ta có thể tổ chức phương pháp luận tốt hơn theo thứ tự để chỉ tập trung vào những khía cạnh duy nhất của quá trình phát triển muộn nhất. Kết quả là chúng ta có thể tự động hóa một số phần trong quy trình phát triển và thậm chí là bán tự động.
- **Tính toán chi phí chính xác hơn:** Vì có thể dựa trên kinh nghiệm đã có để đưa ra ước lượng chính xác

Một phương pháp luận tốt thường đề cập đến những khía cạnh sau đây:

- Lập kế hoạch (Planning) : Quyết định xem cần thực hiện những gì.
- Lập lịch (Scheduling): Lập lịch để thực hiện các công việc
- Tài nguyên (Resourcing): Ước lượng và thu thập nhân lực, phần mềm, phần cứng và các tài nguyên cần thiết khác.
- Luồng công việc: các tiến trình con
- Hoạt động (Activites): Nhiệm vụ của mỗi cá nhân bên trong luồng công việc, như là kiểm thử một thành phần, vẽ một biểu đồ lớp hay một usecase...
- Vai trò (Roles): Vai trò của các nhân trong phương pháp luận (developer, tester, sales person)

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

- Tài liệu (Artifacts): Các sản phẩm có liên quan đến phát triển phần mềm như một phần của phần mềm, tài liệu thiết kế, kế hoạch huấn luyện và hướng dẫn sử dụng.
- Đào tạo (Education): Quyết định cách huấn luyện cho khách hàng, người dùng cuối, người bán hàng....

2.2 CÁC PHA PHÁT TRIỂN PHẦN MỀM CỔ ĐIỂN

2.2.1 Xác định yêu cầu

Đây là pha đầu tiên của phát triển phần mềm. Nó bao gồm 2 khía cạnh:

Mô hình hóa nghiệp vụ: liên quan đến việc hiểu về nghiệp vụ chung của phần mềm cần làm trong văn cảnh cụ thể của nó.

Mô hình hóa yêu cầu hệ thống: xác định được yêu cầu thực sự của khách hàng, xem khách hàng cần gì để biết được ta sẽ phải làm gì và không cần làm gì và xác định càng chi tiết càng rõ ràng càng tốt.

2.2.2 Phân tích

Phân tích có nghĩa là ta phải xem xét xem ta đang phải đối mặt với vấn đề gì? Trước khi đi vào thiết kế ta cần phân tích vấn đề một cách rõ ràng. Từ đó mới biết được ta đã thực sự hiểu về sản phẩm mà khách hàng yêu cầu hay chưa? Điều này liên quan đến khách hàng, người dùng cuối....

2.2.3 Thiết kế

Trong pha thiết kế chúng ta hành động để giải quyết vấn đề hay nói cách khác chúng ta đưa ra những quyết định dựa trên kinh nghiệm, ước lượng hay trực giác. Ta thiết kế và xem liệu nó sẽ được triển khai như thế nào. Thiết kế một hệ thống thường phân ra thành hệ thống con logic (tiến trình) và hệ thống con vật lý (máy tính, mạng), quyết định cách thức máy móc làm việc với nhau từ đó lựa chọn công nghệ...

2.2.4 Đặc tả

Pha đặc tả thường hay bị bỏ qua. Thuật ngữ đặc tả được dùng theo nhiều cách khác nhau và bởi nhiều nhà phát triển khác nhau. Ví dụ như đầu ra của pha yêu cầu là một đặc tả về cái mà hệ thống cần phải làm, thuật ngữ đặc tả được dùng để mô tả các hành vi mong đợi của một các thành phần trong chương trình. Đặc tả có thể được dùng trong các trường hợp sau:

- Là cơ sở để kiểm tra thiết kế phần mềm trong khi thực thi hệ thống
- Văn bản các thành phần phần mềm có thể cài đặt bởi bên thứ ba
- Để mô tả cách thức code hay là sử dụng lại code trong các ứng dụng khác.

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

2.2.5 Cài đặt

Pha này sẽ viết các đoạn code nhỏ cho các hệ thống con. Công việc chính là viết phần thân các phương thức trong lớp thiết kế.

2.2.6 Kiểm thử

Sau khi phần mềm đã hoàn thành thì cần phải kiểm tra lại xem nó đã đáp ứng các yêu cầu của khách hàng chưa. Cách tốt nhất mà người ta hay làm là thực hiện các test nhỏ trong suốt quá trình phát triển phần mềm để đảm bảo chất lượng.

2.2.7 Triển khai

Pha này quan tâm đến phần cứng và phần mềm tại nơi người dùng cuối.

2.2.8 Bảo trì

Sau khi hệ thống được triển khai, có thể sẽ có nhiều lỗi hệ thống hay cũng có thể khách hàng yêu cầu sửa đổi, nâng cấp....vì vậy cần có quá trình bảo trì hệ thống.

2.2.9 Một số câu hỏi

Các câu hỏi sau đây sẽ giúp bạn xác định được mục đích của các pha:

Xác định yêu cầu: “Phạm vi công việc của ta là gì?”, “Những công việc nào chúng ta cần phải hoàn thành?”

Phân tích: “Cần phải đổi mới với những thực thể nào?” “Làm thế nào để đảm bảo những điều đó là đúng?”

Thiết kế: “Giải quyết vấn đề như thế nào?” “Phần cứng , phần mềm nào cần để hoàn thành hệ thống?”, “ Thực thi các giải pháp như thế nào?”

Đặc tả: “Các quy tắc nào chi phối giao diện giữa các thành phần hệ thống?”

Cài đặt: “Làm thế nào code các thành phần cho thỏa mãn đặc tả?”

Kiểm thử: “Hệ thống có thỏa mãn yêu cầu của khách hàng không?”

Triển khai: “Người quản trị hệ thống phải làm gì?”, “Huấn luyện người sử dụng đầu cuối thế nào?”

Bảo trì: “Chúng ta có thể tìm và sửa lỗi được không?”, “Chúng ta có thể cải tiến hệ thống được không?”

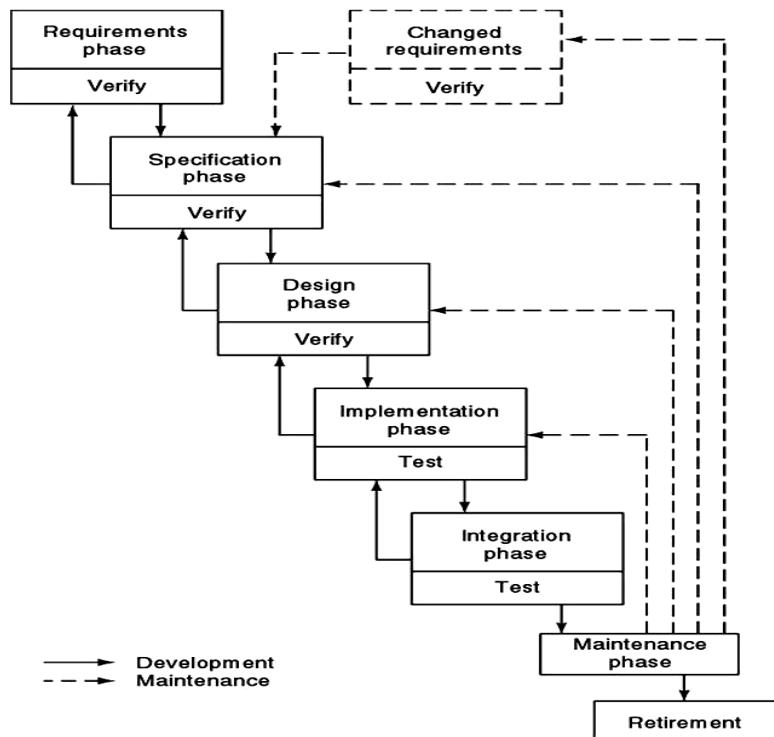
2.3 CÁC PHƯƠNG PHÁP LUẬN TRUYỀN THÔNG

2.3.1 Phương pháp luận theo mô hình thác nước

Mô hình thác nước lần đầu tiên được đề nghị bởi Royce (1970). Trước hết yêu cầu được xác định và kiểm tra bởi khách hàng và các thành viên trong nhóm SQA. Sau đó đặc tả được xây dựng và kiểm tra bởi SQA rồi đưa cho khách hàng xem xét. Khi khách hàng

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

chấp nhận tài liệu đặc tả thì nhóm phát triển lập kế hoạch quản lý dự án phần mềm và lập lịch trình chi tiết để phát triển phần mềm. Nhóm SQA kiểm tra kế hoạch đó.



Hình 1: Mô hình thác nước

Đặc trưng của mô hình thác nước là hướng tài liệu nghĩa là không có pha nào được gọi là hoàn thành nếu tài liệu pha đó chưa hoàn thành.

Ưu điểm

- Tài liệu đầy đủ
- Dễ bảo trì vì có tài liệu

Nhược điểm

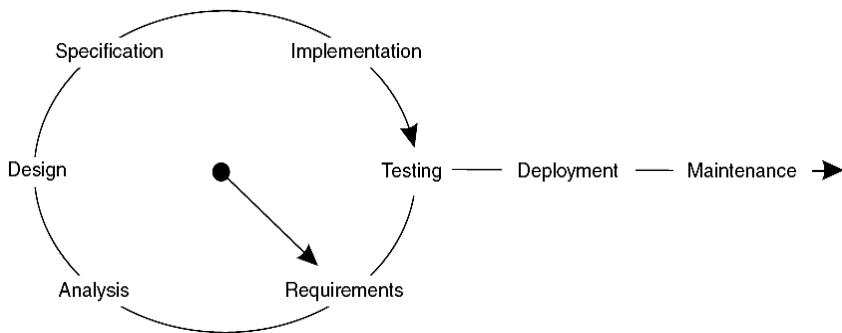
- Đôi khi tài liệu đặc tả quá mang tính chất kỹ thuật nên khách hàng khó mà hiểu được hết và do đó dễ hiểu sai yêu cầu khách hàng.
- Làm tuần tự từng bước một mà không thể nói những khó khăn trong suốt quá trình phát triển dự án vì thế một vài phase sẽ bị chậm hơn thời gian dự kiến.
- Làm tuần tự và đến tận cuối cùng mới bàn giao sản phẩm hoàn thiện thì rất dễ làm ra sản phẩm không phải là cái khách hàng cần.

2.3.2 Phương pháp luận xoắn ốc

Bắt đầu phương pháp luận xoắn ốc này là những yêu cầu (những yêu cầu này có thể đầy đủ hoặc khá mơ hồ ở giai đoạn này). Sau đó ta có thể thực hiện một số phân tích để tăng

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

độ hiểu biết về vấn đề được đặt ra ở pha xác định yêu cầu. Sau đó ta sẽ phác thảo ra một hệ thống mà ta cảm thấy phù hợp nhất với yêu cầu ban đầu. Mặc dù những bước trước là chưa hoàn chỉnh nhưng chúng ta vẫn viết code. Khi đã nỗ lực hoàn thành code ban đầu, chúng ta đưa sản phẩm đó cho người tài trợ (có thể là người dùng cuối, người quản lý và khách hàng trả tiền cho hệ thống).



Hình 2.2: Mô hình xoắn ốc

Bằng cách thông qua mỗi chu trình, chúng ta sẽ tăng thêm sự hiểu biết của ta về một miền vấn đề nào đó và về các giải pháp được đề xuất. Qua nhiều lần xoắn ốc, chúng ta có thể mổ xé các yêu cầu và phân tích thiết kế một cách chính xác hơn để đáp ứng với yêu cầu nhiều hơn.

Sau khi hệ thống của chúng ta hoàn thành, có lẽ sau 3 hoặc 4 xoắn ốc, chúng ta có thể thực hiện kiểm tra hệ thống một cách nghiêm ngặt. So với phương pháp thác nước, chúng ta làm việc rất giống như việc tạo ra một bức tượng điêu khắc: chúng ta cùng nhau đưa ra một khung cơ bản, sau đó gắn đát sét, lớp của lớp cho đến khi ta cảm thấy đạt yêu cầu mong muốn.

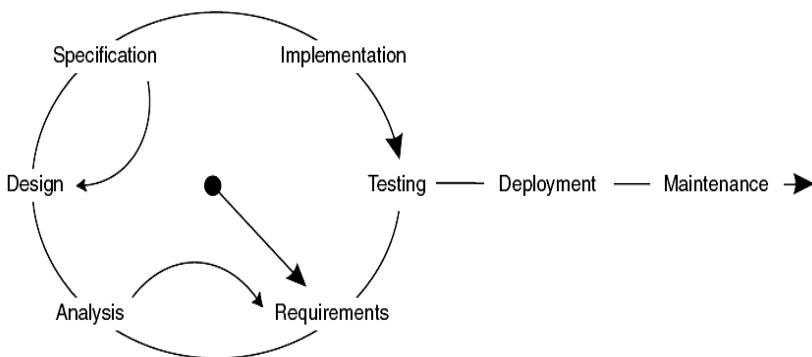
Phương pháp xoắn ốc không hoàn hảo do ở chỗ chúng ta thực hiện mô hình thác nước ba hay bốn lần và điều đó có nghĩa là những vấn đề dù là nhỏ nhất chúng ta cũng không bỏ qua. Có một số điểm không thể uốn cong bởi vì ta đang phải làm theo một trật tự qua các giai đoạn cổ điển, nếu ta tìm thấy những sai lầm, ta không thể sửa chữa cho đến các giai đoạn tiếp theo. Vì vậy, phương pháp xoắn ốc thường kết hợp với một phương pháp luận khác.

2.3.3 Phương pháp luận lặp

Điều mà ta cần là một phương pháp luận cho phép ta có thể lặp qua giai đoạn, di chuyển ngược trở lại, chuyên tiếp. Điều này dẫn tới phương pháp lặp được miêu tả trong hình 2.3. (Ở đây, lặp đi lặp lại được hiển thị trong vòng xoắn ốc, nhưng ta cũng có thể áp dụng cho một thác nước). Nay giờ chúng ta có nhiều cách hơn để thực hiện phần mềm của chúng ta. Nhưng chúng ta sẽ băn khoăn, liệu rằng có bị rối loạn hay không? Những pha cơ bản nhắc nhở chúng ta nên làm gì trong mỗi giai đoạn:

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

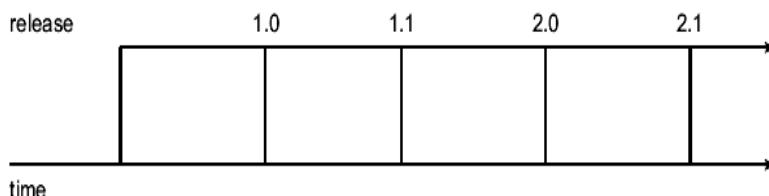
- Những bản tài liệu (biểu đồ, mô tả, code....) mà chúng ta cung cấp khi làm trong những phase cổ điển, sẽ được hoàn thiện khi chúng ta đi triển khai.
- Các công cụ hỗ trợ sản xuất phần mềm sẽ hỗ trợ ta lựa chọn những phương pháp luận phù hợp và đảm bảo rằng các tài liệu làm ra sẽ được lưu trữ tại một nơi nào đó



Hình 3: Phương pháp phát triển lặp

2.3.4 Phương pháp luận tăng dần

Hãy quay trở lại với điêu khắc, nếu ta được yêu cầu sản xuất một già đình bằng các tác phẩm điêu khắc khác (gồm cha, mẹ, con, con mèo, con chó). Các nhà tài trợ muốn nhìn thấy sự tiến bộ của ta ở mỗi một phần việc, vì vậy ta có thể cắt từng đoạn công việc, sau đó hoàn tất chúng và có thể tập trung vào các công việc tiếp theo. Trong phát triển phần mềm, gọi đó là phương pháp gia tăng.



Hình 4: Phương pháp luận tăng dần

Với phương pháp tăng dần, ta có thể cố gắng để cung cấp phiên bản 1.0 của hệ thống với một số chức năng cơ bản quan trọng. Sau một thời gian, ta có thể nâng cấp lên thành phiên bản 1.1 với những tính năng bổ sung (như là một phần thay thế phiên bản 1.0). Tiếp theo, ta có thể cung cấp phiên bản 2.0 với một số thay đổi và cứ như thế trong suốt vòng đời của hệ thống.

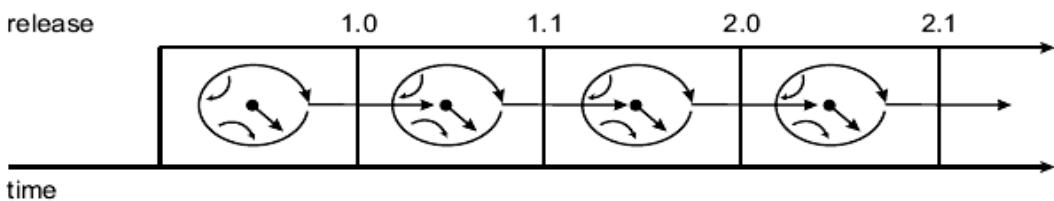
2.3.5 Kết hợp các phương pháp luận

Như vậy, phương pháp thác nước là không đủ trong hầu hết các trường hợp- mặc dù nó có các giai đoạn theo thứ tự đúng - và các phương pháp thay thế (xoắn ốc, lặp và tăng

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

dần) đều có đặc tính mong muốn nhưng không phương pháp luận nào trong số đó là tốt và đầy đủ cả. Vì vậy chúng ta phải kết hợp cả bốn phương pháp luận đó, nhưng vấn đề là làm thế nào.

Chúng ta biết rằng trong phương pháp tăng dần cần phải lập lịch cho kết quả của mỗi lần tăng. Trong mỗi lần tăng, phương pháp luận xoắn ốc gợi ý rằng chúng ta phải có ít nhất 2 lần tăng dần. Mặt khác, từ các phương pháp gia tăng mà chúng ta phải lên kế hoạch cho một sản phẩm tiếp theo của từng bước. Trong vòng mỗi gia tăng, các phương pháp xoắn ốc cho thấy rằng chúng ta cần phải có ít nhất hai nỗ lực để sản xuất mỗi gia tăng. Trong vòng xoắn ốc, các phương pháp thác nước xác định giai đoạn và thứ tự mà chúng xảy ra. Trong vòng mỗi thác nước con, các phương pháp lặp cho phép chúng ta lặp lại các giai đoạn, hoặc kết hợp các giai đoạn, như chúng ta thấy phù hợp (ví dụ, một vài chu kỳ của các yêu cầu và phân tích); phương pháp lặp cũng cho phép chúng ta sửa chữa một vấn đề ngay khi phát hiện ra nó (ví dụ, có thể khám phá ra trong quá trình thiết kế hệ thống con rằng việc thiết kế hệ thống chung làm cho một số phần chức năng không thể kết hợp, nên phải sửa chữa thiết kế hệ thống trước khi tiến hành thiết kế chi tiết). Sự kết hợp của các phương pháp được thể hiện trong Hình 2.5.



Hình 2.5: Kết hợp các mô hình

Không có nghiên cứu nào có thể cho biết cách chúng ta lên kế hoạch và tiến hành một dự án cụ thể. Điều đó phụ thuộc vào kích cỡ của dự án, số lượng các nhà phát triển, kinh nghiệm của các nhà phát triển, kinh nghiệm của các nhà quản lý trong quy hoạch và lập kế hoạch phát triển hình thức này... Như vậy, theo phát triển hướng đối tượng, không nên tuyệt đối hóa việc kết hợp này. Nó chỉ được xem là khá hợp lý cho các nhà quản lý, nhà tư vấn và nhà phát triển có kinh nghiệm khi quyết định tăng, xoắn ốc, lặp đi lặp lại và sản phẩm phù hợp cho từng pha hợp. Sau đó, quy hoạch phải được điều chỉnh theo hiểu biết ngày càng tăng với các yêu cầu thay đổi.

Mặc dù hầu hết các nhà lý thuyết đồng ý về bản chất của phương pháp thác nước, nhưng cũng có một số bất đồng về các thuật ngữ được sử dụng. Trong tài liệu này, chúng ta đã nhìn thấy các định nghĩa cụ thể của các phương pháp xoắn ốc, lặp và gia tăng nhưng bạn nên biết rằng những người khác sử dụng các thuật ngữ khác nhau hoặc thậm chí là từ đồng nghĩa.

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

2.4 PHƯƠNG PHÁP LUẬN HƯỚNG ĐÓI TƯỢNG

2.4.1 Tiến trình hợp nhất (UP: Unified Process)

Tiến trình hợp nhất (Unified Software Development Process hay Unified Process) là quy trình phát triển phần mềm thông nhất, viết tắt là UP. UP là một phương pháp phát triển phần mềm lặp và gia tăng. Nó khắc phục các nhược điểm của các mô hình truyền thống như thác nước hoặc xoắn ốc. UP có thể thay đổi tùy theo các công ty, tổ chức và các dự án riêng biệt. Khái niệm UP là khái niệm chung dùng để mô tả quy trình chung trong quá trình phát triển phần mềm.

Nguyên lý lặp và gia tăng trong mô hình UP cho phép chia nhỏ các pha trong vòng đời dự án thành các giai đoạn nhỏ, mỗi giai đoạn có thể được lặp đi lặp lại và gia tăng theo thời gian. Thường thì chúng ta sẽ chia nhỏ các pha chuẩn bị, xây dựng và bàn giao thành các giai đoạn nhỏ, trong các dự án lớn thì pha khởi động cũng có thể được chia nhỏ thành nhiều giai đoạn. Các giai đoạn này sẽ được phát triển lặp lại theo thời gian, có thể là theo các phiên bản (version) của sản phẩm trong dự án. Chúng ta có thể thêm và cải tiến các chức năng so với phiên bản trước của sản phẩm trong cùng một giai đoạn. Việc cho phép thêm và cải tiến chức năng này gọi là sự phát triển gia tăng của các giai đoạn theo thời gian. Một số mô hình UP phổ biến và có nhiều ưu điểm:

- Agile Unified Process (AUP)
- Basic Unified Process (BUP)
- Enterprise Unified Process (EUP)
- Essential Unified Process (EssUP)
- Open Unified Process (OUP)
- Rational Unified Process (RUP)
- Rational Unified Process – System Engineering (RUP-SE)

2.4.2 Rational Unified Process (RUP)

Hình 2.6 mô tả vòng đời của RUP. Ta có thể thấy RUP có 9 disciplines và 4 phase. Mỗi một discipline ước lượng thời gian trong từng pha mà nó chiếm là bao nhiêu. Như “Business Modeling” chiếm phần lớn thời gian ở trong pha Inception nhưng trong các pha còn lại thì ít hơn hay “Implementation” chiếm phần lớn thời gian ở trong pha Construction....Có bốn đặc trưng của vòng đời RUP

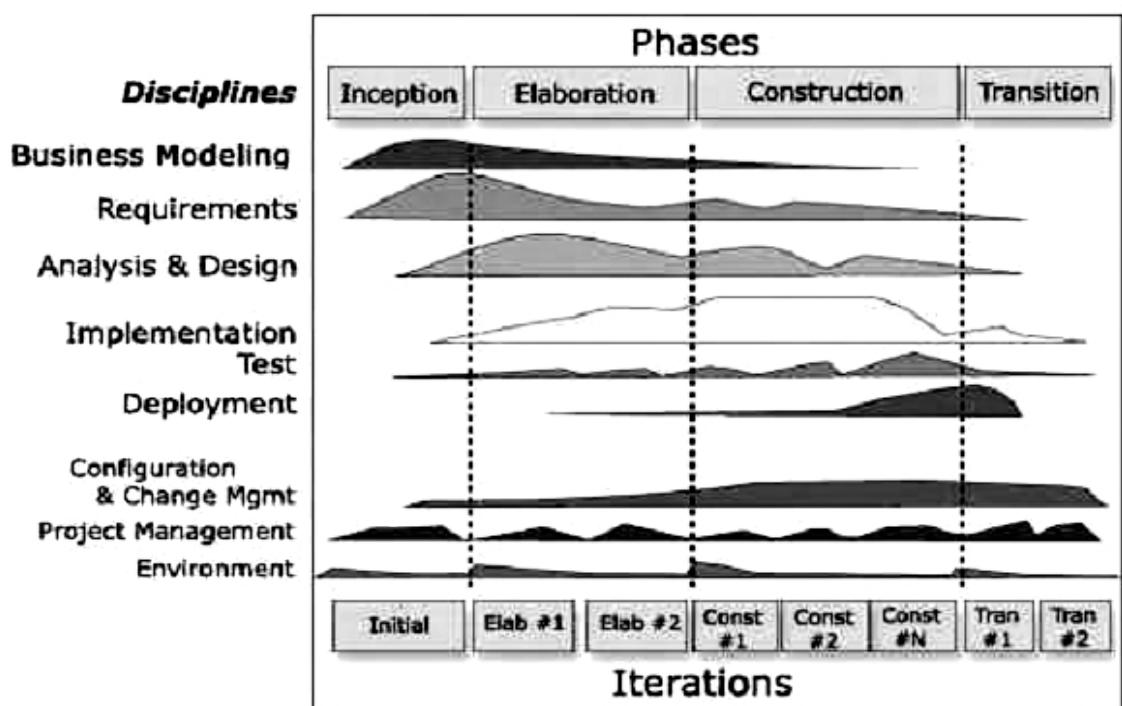
- Tuân tự theo các pha
- Lặp trong từng pha
- Cung cấp các bản phát hành gia tăng theo thời gian

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

- Tuân thủ các thực tế tốt nhất.

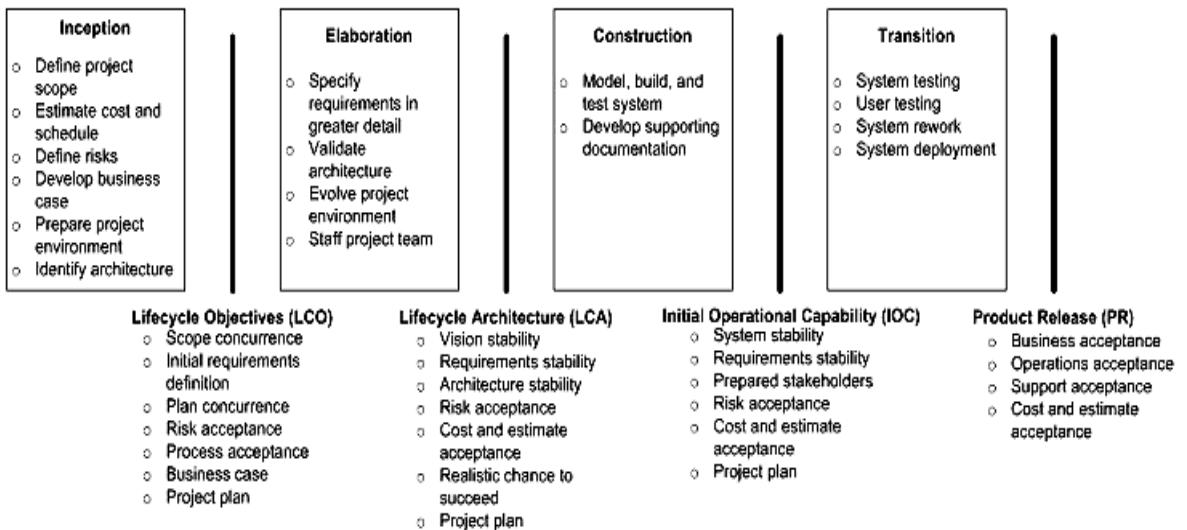
a. Tuần tự theo các pha

RUP là cấu trúc gồm 2 thành phần: Pha và Disciplines. Mô hình RUP chia vòng đời dự án thành bốn pha: Pha khởi động (Inception phase), Pha chuẩn bị (Elaboration phase), Pha xây dựng (Construction phase), Pha chuyển giao (Transition phase). Trong đó 10% thời gian cho Inception phase, 25% cho Elaboration phase, 55% cho Construction phase và 10% in Transition phase.



Hình 6: RUP

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM



Pha Inception

Đây là pha đầu tiên và ngắn nhất trong vòng đời dự án, thực tế pha này nên được triển khai càng ngắn gọn càng tốt. Nếu thực hiện pha này kéo dài thì có nghĩa là đang có sự dư thừa trong việc đặc tả hệ thống và đi ngược lại với tinh thần của mô hình RUP. Trong pha này chúng ta cần đạt được các mục đích sau:

- Xác định phạm vi, điều kiện và các giới hạn của dự án
- Xác định yêu cầu ban đầu của hệ thống (dù không có nhiều chi tiết)
- Xác định kế hoạch phát triển phần mềm
- Xác định các rủi ro cho dự án đang được quản lý một cách thích hợp
- Xác định các nghiệp vụ chính và các chức năng chính của dự án
- Xác định các thiết kế riêng phù hợp với quá trình phát triển.
- Chuẩn bị tài liệu kế hoạch

Mốc bàn giao yêu cầu và mục đích của dự án đánh dấu sự kết thúc của pha khởi động.

Pha Elaboration

Trong pha này, nhóm dự án sẽ phải hiểu được các chức năng chính của yêu cầu hệ thống. Tuy nhiên mục đích chính của nó là phải chỉ ra được các nhân tố gây ra rủi ro, thiết lập và kiểm tra lại kiến trúc hệ thống.

Kiến trúc hệ thống sẽ được kiểm tra chủ yếu thông qua việc phân tách hệ thống, có nghĩa là chia kiến trúc hệ thống thành các module nhỏ hơn, thông qua việc kiểm tra đánh giá từng module mà đưa ra được đánh giá tổng thể về kiến trúc hệ thống. Các module con này bao gồm nhân của hệ thống, các thành phần chính và các thành phần

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

phụ. Các thành phần này được phân nhỏ và được cài đặt theo từng giai đoạn. Ở cuối pha chuẩn bị này, chúng ta phải đưa ra được một kiến trúc hệ thống ổn định, có thể triển khai tất cả các yêu cầu chức năng chính và tối ưu theo các khía cạnh thực thi, khả năng mở rộng và giá thành phù hợp. Kết quả cuối cùng của pha chuẩn bị là phải đưa ra được kế hoạch (bao gồm ước lượng về giá thành và thời gian) cho pha xây dựng. Kế hoạch đưa ra phải đảm bảo đúng đắn và chính xác dựa trên kinh nghiệm. Kết thúc pha này dự án phải đảm bảo rằng:

- Có sự hình dung về dự án một cách chân thực.
- Yêu cầu đối với dự án.
- Kiến trúc ổn định để thỏa mãn các yêu cầu
- Rủi ro tiếp tục được quản lý
- Ước lượng chi phí hiện tại là hợp lý và chấp nhận được, ước lượng chi phí và lịch trình cho tương lai.
- Xác định cơ hội thực tế thành công của nhóm dự án.
- Lập kế hoạch chi tiết lắp ở trong Construction phase như là một mức cao hơn của quản lý dự án.

Mốc bàn giao kế hoạch và kiến trúc hệ thống đánh dấu kết thúc cho pha chuẩn bị.

Pha Construction

Đây là pha dài nhất trong vòng đời một dự án. Tại pha này, tất cả các chức năng của hệ thống sẽ được cài đặt. Việc cài đặt sẽ được chia thành nhiều giai đoạn nhỏ, mỗi giai đoạn cài đặt một vài chức năng. Kết quả của mỗi giai đoạn sẽ là việc phát hành các module chức năng có thể thực thi được. Nếu cần thiết, bản phát hành đầu của hệ thống được triển khai, hoặc là nội bộ hay bên ngoài, để có được người sử dụng phản hồi. Kết thúc pha này dự án phải đảm bảo rằng :

- Phần mềm và tài liệu hỗ trợ được chấp nhận để triển khai.
- Những bên liên quan sẵn sàng cho hệ thống được triển khai
- Rủi ro tiếp tục được quản lý hiệu quả
- Ước lượng chi phí hiện tại là hợp lý và chấp nhận được, ước lượng chi phí và lịch trình cho tương lai.
- Lập kế hoạch chi tiết lắp cho Transition phase, như là một mức cao hơn của kế hoạch dự án đưa ra.

Mốc phát hành sản phẩm sẽ đánh dấu kết thúc của pha xây dựng.

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

Pha Transition

Đây là pha cuối cùng trong vòng đời của một dự án. Sản phẩm làm ra sẽ được triển khai tại khách hàng đầu cuối. Các phản hồi nhận được trong quá trình chuyển giao sẽ được ghi nhận và đưa vào yêu cầu chức năng mới hoặc cải tiến chức năng trong phiên bản tiếp theo của sản phẩm. Pha chuyển giao cũng bao gồm sự chuyển đổi hệ thống và huấn luyện hệ thống mới cho người dùng. Kết thúc pha này hệ thống phải đảm bảo rằng:

- Hệ thống, bao gồm cả hỗ trợ tài liệu và huấn luyện sẵn sàng được triển khai
- Ước lượng chi phí hiện tại là hợp lý và chấp nhận được đã được thực hiện cho chi phí tương lai
- Hệ thống có thể hoạt động một khi nó được sản xuất
- Hệ thống có thể được hỗ trợ một cách thích hợp khi nó được sản xuất.

Mốc bàn giao sản phẩm sẽ đánh dấu sự kết thúc của pha chuyển giao.

Lặp trong từng pha

Trong từng pha của RUP được chia ra thành nhiều vòng lặp ngắn. Lặp đi lặp lại duy nhất một phần của toàn bộ hệ thống đang được phát triển chứ không như mô hình thác nước cố gắng làm tất cả cùng lúc. RUP có 9 nguyên lý :

1. Mô hình nghiệp vụ (Business Modeling)
2. Thu thập yêu cầu (Requirements)
3. Phân tích và thiết kế (Analysis and Design)
4. Cài đặt (Implementation)
5. Kiểm thử (Test)
6. Triển khai (Deployment)
7. Quản lý cấu hình và thay đổi (Configuration and Change Management)
8. Quản lý dự án (Project Management)
9. Môi trường (Environment)

Mô hình nghiệp vụ

Mục đích là để hiểu về công việc nghiệp vụ của tổ chức, đánh giá tình trạng hiện thời của tổ chức, bao gồm cả khả năng của nhóm phát triển trong việc hỗ trợ một hệ thống mới.

- Tìm hiểu các quy trình nghiệp vụ hiện nay, vai trò và trách nhiệm.
- Xác định và đánh giá chiến lược nghiệp vụ tiềm năng cho các quá trình tái cấu trúc

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

- Phát triển một mô hình miền nhằm phản ánh tập con của các nghiệp vụ.

Thu thập yêu cầu

Mục đích là thu thập yêu cầu từ khách hàng để xác định phạm vi công việc cần làm những gì và không phải làm những gì. Thông tin này được sử dụng bởi các nhà phân tích, thiết kế, lập trình viên, xây dựng hệ thống, người kiểm tra hệ thống, và bởi người quản lý dự án để lập kế hoạch và quản lý dự án. Hoạt động của thu thập yêu cầu bao gồm:

- Làm việc chặt chẽ với các bên tham gia dự án để hiểu rõ nhu cầu của họ. Xác định phạm vi của hệ thống.
- Khám phá cách sử dụng, quy định nghiệp vụ, giao diện người dùng, và các yêu cầu (phi chức năng) thông qua mô hình thích hợp.
- Xác định và ưu tiên các yêu cầu mới hoặc thay đổi khi trong suốt một dự án.

Phân tích và thiết kế

Mục đích là phân tích các yêu cầu của hệ thống và thiết kế các giải pháp có thể được thực thi, đưa các yêu cầu ràng buộc của khách hàng vào hệ thống. Hoạt động này bao gồm:

- Làm mịn một kiến trúc đề cử cho một hệ thống
- Hiểu (phân tích) các yêu cầu của hệ thống
- Thiết kế các thành phần, dịch vụ và/hoặc các module
- Thiết kế mạng, giao diện người dùng và cơ sở dữ liệu.

Thực thi

Mục đích của thực thi là đưa thiết kế vào thực thi code và để thực hiện kiểm thử ở mức cơ bản, như là unit test. Các hoạt động chính bao gồm:

- Hiểu và tiền hóa mô hình thiết kế.
- Viết chương trình nguồn
- Thực thi các thành phần, các dịch vụ hay các module
- Kiểm thử đơn vị mã nguồn
- Tích hợp code vào hệ thống con hay hệ thống xây dựng có thể triển khai được

Kiểm thử

Mục đích của kiểm thử là các hoạt động để đảm bảo chất lượng. Nó bao gồm việc tìm ra lỗi, xác minh các công việc của hệ thống như thiết kế, kiểm tra các yêu cầu đã có. Các hoạt động chính bao gồm :

- Xác định và lập kế hoạch kiểm thử

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

- Phát triển các test case
- Tổ chức các test suite
- Chạy kiểm thử
- Báo cáo lỗi

Triển khai

Mục đích là lập kế hoạch phân phối hệ thống và thực thi kế hoạch để làm cho hệ thống khả dụng với người sử dụng cuối. Hoạt động bao gồm:

- Lập kế hoạch về chiến lược triển khai
- Phát triển hỗ trợ và vật chất....
- Tạo các gói triển khai
- Phát triển phần mềm để cài đặt trên các site
- Huấn luyện cho người sử dụng cuối
- Quản lý kiểm thử chấp nhận

Quản lý cấu hình và thay đổi

Mục đích là để quản lý sự truy cập tới sản phẩm của dự án . Nó bao gồm không chỉ là phiên bản qua thời gian nhưng nó cũng điều khiển và quản lý sự thay đổi đó. Các hoạt động bao gồm:

- Quản lý các yêu cầu thay đổi
- Lập kế hoạch điều khiển cấu hình
- Cài đặt môi trường
- Giám sát và báo cáo trạng thái cấu hình
- Thay đổi và phân phối cấu hình
- Quản lý baselines và sản phẩm.

Quản lý dự án

Mục đích để quản lý trực tiếp các hoạt động của dự án. Nó bao gồm cả quản lý rủi ro, con người (phân công nhiệm vụ...) và gắn kết con người và hệ thống bên ngoài phạm vi dự án để chắc chắn rằng việc phát hành sản phẩm đúng thời gian và trong kinh phí cho phép. Các công việc quan trọng bao gồm:

- Khởi tạo một dự án mới
- Quản lý nhân sự của dự án

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

- Quan hệ giữa nhóm bên ngoài và tài nguyên
- Quản lý rủi ro
- Ước lượng, lập lịch và kế hoạch
- Quản lý các vòng lặp
- Đóng các phase hay dự án.

Môi trường

Mục đích là để hỗ trợ phần còn lại của công việc như hướng dẫn (tiêu chuẩn và nguyên tắc) các công cụ (phần cứng phần mềm..) có sẵn cho các nhóm khi cần thiết. Các hoạt động quan trọng bao gồm:

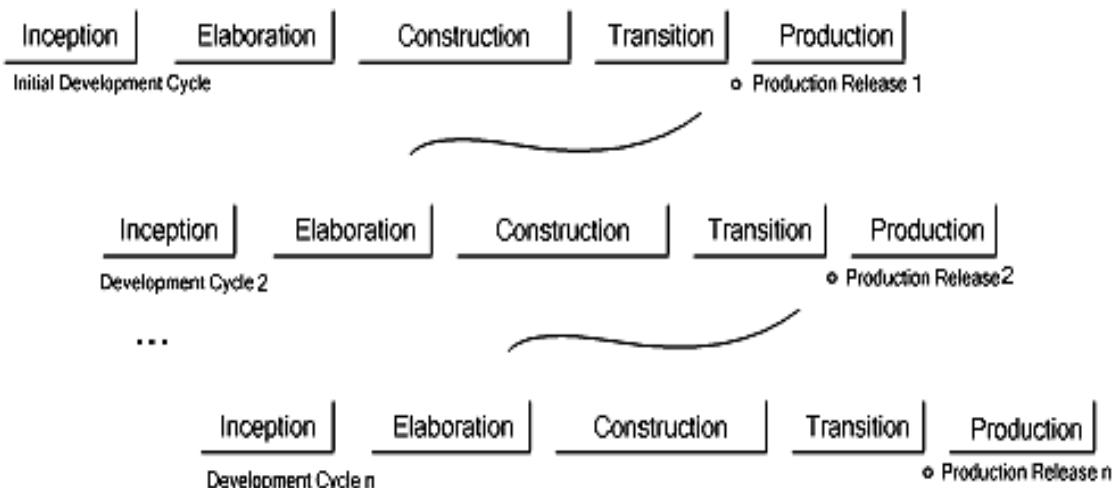
- Các tài liệu quá trình thiết kế riêng cho một nhóm dự án cá nhân
- Xác định và đánh giá các công cụ
- Cài đặt và thiết lập công cụ cho các nhóm dự án
- Hỗ trợ các công cụ trong suốt quá trình dự án.

Phân phối phiên bản

Phiên bản đầu tiên của một hệ thống thì thường không phải là phiên bản cuối cùng. Nếu như vậy thì một hệ thống sẽ chỉ dừng lại ở version 1.0. Thay vào đó hệ thống tiến triển theo thời gian. Khi một chức năng mới được thêm vào, yêu cầu của khách hàng được thêm, các chuẩn được chấp nhận và hỗ trợ. RUP là mô hình thông qua 4 phase để tạo ra một version duy nhất của một hệ thống gọi là sản phẩm phát hành.

Trong hoặc sau pha Transition một dự án mới có thể được bắt đầu để cập đến các yêu cầu nổi bật. Dự án mới sẽ bắt đầu trở lại từ Inception phase, tuy nhiên một số nhóm sẽ quyết định bắt đầu từ Elaboration phase hoặc thậm chí là Construction phase nếu thích hợp. Hình dưới đây mô tả có thể tiếp tục vô hạn, miễn là yêu cầu mới được xác định các bên liên quan đồng ý rằng đó là một phiên bản phần mềm mới.

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM



Tuân thủ các thực tế tốt nhất

Trong quá khứ thực hành bản gốc tốt nhất (phát triển lặp, quản lý yêu cầu, sử dụng thành phần kiến trúc, mô hình trực quan (UML), liên tục kiểm tra chất lượng và quản lý các thay đổi) phản ánh các vấn đề phát triển phần mềm. Những hoạt động tốt nhất cho RUP là:

- Thích nghi tiến trình (**Adapt process**)
- Cân bằng các ưu tiên yêu cầu của khách hàng (**Balance Competing Stakeholder Priorities**)
- Cộng tác giữa các nhóm (**Collaborate Across Teams**)
- Chứng tỏ giá trị lặp (**Demonstrate Value Iteratively**)
- Nâng cao mức độ trùu tượng
- Tập trung liên tục vào chất lượng (**Focus Continuously on Quality**)

Thừa nhận tiến trình (**Adapt process**)

Mỗi người, nhóm dự án, và tổ chức là khác nhau trong môi trường họ làm việc. Nên bạn phải chỉnh quá trình phần mềm của bạn để đáp ứng nhu cầu chính xác của bạn. Nếu bạn cố gắng áp dụng RUP một cách máy móc thì rất có khả năng đội dự án của bạn sẽ không có được các tài liệu cần thiết.

Cân bằng ưu tiên yêu cầu khách hàng (**Balance Competing Stakeholder Priorities**)

Bất kỳ một dự án nào cũng có một loạt các bên liên quan như: người dùng cuối, giới kinh doanh, nhà quản lý, khách hàng bên ngoài.... và họ sẽ có nhu cầu và sự ưu tiên khác nhau. Một trong những khó khăn nhất của phát triển phần mềm là cân bằng được những ưu tiên đó. Đặc biệt là khi họ thường xuyên thay đổi trong khi dự án tiến triển.

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

Cộng tác giữa các nhóm (Collaborate Across Teams)

Phần mềm được phát triển bởi các tài năng, những người thúc đẩy giao tiếp và cộng tác hiệu quả. Để đạt được điều này thành công cần khuyến khích nhân viên không chỉ làm tốt công việc của họ mà còn tích cực học hỏi những kỹ năng mới từ đồng nghiệp hay các nguồn khác như sách báo, internet....

Chứng tỏ giá trị lặp (Demonstrate Value Iteratively)

Ý tưởng cơ bản là muốn giảm bớt chu kỳ phản hồi bằng cách cung cấp các phần mềm sớm và thường xuyên hơn, ta sẽ làm điều này bằng cách tổ chức các dự án được lặp đi lặp lại.

Nâng cao mức độ trùu tượng

Phát triển phần mềm là khó khăn và ngày càng khó khăn. Hiệu quả đội phát triển là nâng cao trình độ trùu tượng của việc áp dụng các công cụ mô hình, sử dụng lại các sản phẩm đã có và bằng cách tập trung vào kiến trúc để suy xét thông qua những vấn đề lớn trong dự án.

Tập trung liên tục vào chất lượng (Focus Continuously on Quality)

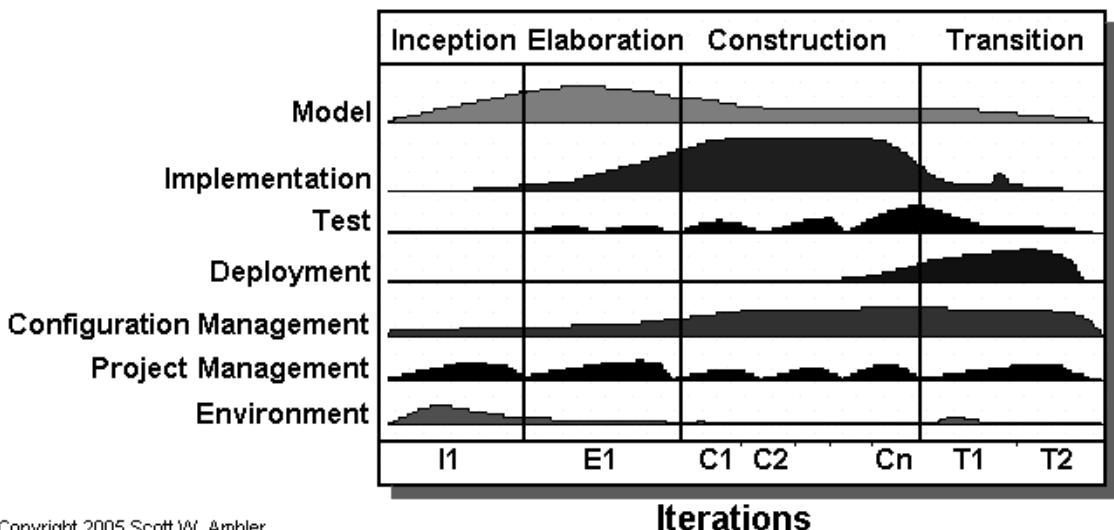
Toàn nhóm chịu trách nhiệm về chất lượng chứ không chỉ là các đội kiểm tra. Đây là một trong những lý do chính tại sao kiểm thử và xác nhận cần trong suốt RUP – mỗi kỷ luật bao gồm đánh giá hoặc là chính thức hoặc là không chính thức, các sản phẩm công việc tạo ra và kiểm thử là hoạt động rất quan trọng trong Implementation và Test. Chất lượng là rất quan trọng để các nhà phát triển Test Driven Development (TDD) tiếp cận đến Implementation: viết một unit test trước khi viết mã sản phẩm đầy đủ để hoàn thành kiểm thử, và sau đó refactor lại mã để nó có chất lượng cao nhất có thể.

2.4.3 Agile Unified Process (AUP)

AUP là một cách phát triển phần mềm dựa trên IBM Rational Unified Process (RUP). Các vòng đời của Agile UP cung cấp bản phát hành gia tăng theo thời gian

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

Phases



Copyright 2005 Scott W. Ambler

AUP

Các bước được thực hiện lặp đi lặp lại để xác định các hoạt động mà những người phát triển cần thực hiện để xây dựng, xác nhận và cung cấp phần mềm đáp ứng nhu cầu của khách hàng. Các bước bao gồm:

- Mô hình (Mô hình)
- Cài đặt/ thực thi ([Implementation](#))
- Kiểm thử ([Test](#))
- Triển khai ([Deployment](#))
- Quản lý cấu hình ([Configuration Management](#))
- Quản lý dự án ([Project Management](#))
- Môi trường ([Environment](#))

[Agile UP Phases:](#)

- [Inception](#)
- [Elaboration](#)
- [Construction](#)
- [Transition](#)

AUP

Model : Mục tiêu là để hiểu về nghiệp vụ của tổ chức, các vấn đề được đề cập đến trong dự án, và để xác định một giải pháp khả thi cho giải quyết các vấn đề.

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

Implementation : Các mục tiêu là biến đổi mô hình thành mã thực thi và thực hiện kiểm tra ở mức độ cơ bản (unit testing)

Test : Mục đích của kiểm tra là để thực hiện một đánh giá khách quan nhằm đảm bảo chất lượng. Điều này bao gồm việc tìm khuyết điểm, phê chuẩn cho hệ thống hoạt động như thiết kế, và xác minh rằng các yêu cầu có được đáp ứng đúng hay không.

Deployment: Mục tiêu là lên kế hoạch cho việc phân phối của hệ thống và thực thi kế hoạch cho người dùng cuối.

Configuration Management: Mục tiêu là để quản lý quyền truy cập vào các sản phẩm của dự án. Điều này bao gồm không chỉ theo dõi các phiên bản sản phẩm làm việc theo thời gian mà còn kiểm soát và quản lý các thay đổi sản phẩm.

Project Management : Mục tiêu là để chỉ đạo các hoạt động diễn ra trong dự án. Điều này bao gồm quản lý rủi ro, chỉ đạo nhân viên (giao nhiệm vụ, theo dõi tiến độ...), và phối hợp với nhân viên và các hệ thống bên ngoài phạm vi của dự án để đảm bảo rằng sản phẩm được giao đúng thời gian và trong kinh phí cho phép

Environment: Mục tiêu là hỗ trợ phần còn lại bằng cách bảo đảm rằng quá trình thích nghi, hướng dẫn (tiêu chuẩn và nguyên tắc), và các công cụ (phần cứng, phần mềm, vv) có sẵn cho đội khi cần thiết.

2.5 KẾT LUẬN

Chương này đã trình bày các phương pháp luận phát triển phần mềm. Sau khi trình bày tổng quan và đánh giá các phương pháp luận truyền thống, nội dung tập trung vào phương pháp luận hướng đối tượng được sử dụng phổ biến hiện nay.

BÀI TẬP

1. Trình bày các yếu tố liên quan đến phương pháp luận phát triển phần mềm và các vấn đề nào cần phải xem xét khi chọn phương pháp phù hợp cho một dự án phần mềm.
2. Sinh viên xem phần tài liệu tham khảo thêm và chọn một chủ đề để viết và trình bày tại lớp (có thể theo nhóm 3 sinh viên).

TÀI LIỆU THAM KHẢO THÊM

[1]The Agile Unified Process (AUP),

<http://www.ambyssoft.com/unifiedprocess/agileUP.html#Overview>

CHƯƠNG 2. PHƯƠNG PHÁP LUẬN PHÁT TRIỂN PHẦN MỀM

<http://www.agilemodeling.com/essays/agileModelingRUP.htm>

[2] Enterprise, Unified Process (AUP) <http://www.enterpriseunifiedprocess.com/>

[3] Rational Unified Process (RUP)

<http://www.ambyssoft.com/downloads/managersIntroToRUP.pdf>

[4] Mike O'Docherty, Object-Oriented Analysis and Design: Understanding System Development with UML 2.0, John Wiley, 2005

[5]http://www.ibm.com/developerworks/rational/library/05/0816_Louis/?S_TACT=105AGX99&S_CMP=CP

[6] <http://www.ibm.com/developerworks/rational/library/05/kunal/>

[7] <http://hosteddocs.ittoolbox.com/RP092305.pdf>

CHƯƠNG 3 . UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

Chương này nhằm giới thiệu ngôn ngữ mô hình hoá thống nhất UML và công cụ phát triển phần mềm hướng đối tượng. Nội dung cụ thể bao gồm:

- Giới thiệu UML
- Các biểu đồ trong UML
- Các bước phân tích thiết kế hướng đối tượng sử dụng UML
- Giới thiệu bộ công cụ Rational Rose

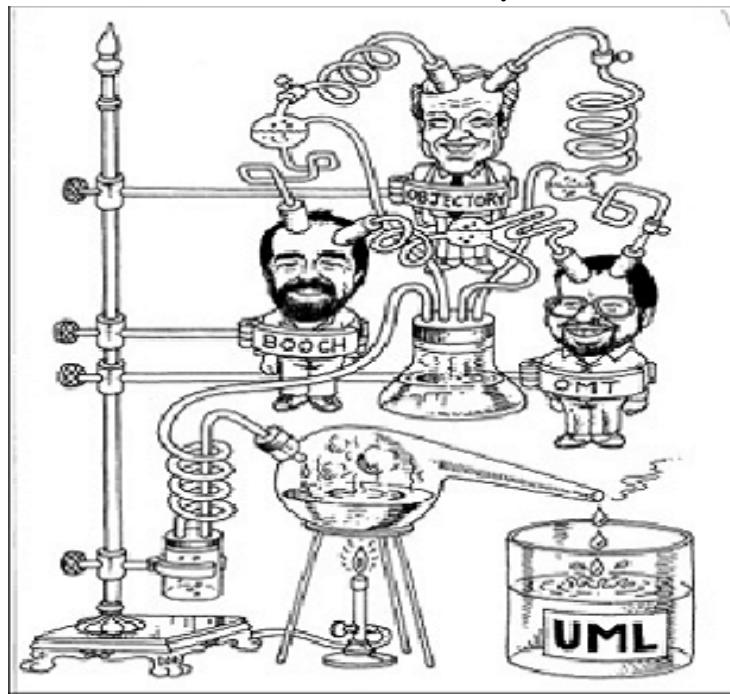
3.1 GIỚI THIỆU VỀ UML

3.1.1 Lịch sử ra đời của UML

Việc áp dụng rộng rãi phương pháp hướng đối tượng đã đặt ra yêu cầu cần phải xây dựng một phương pháp mô hình hóa để có thể sử dụng như một chuẩn chung cho những người phát triển phần mềm hướng đối tượng trên khắp thế giới. Trong khi các ngôn ngữ hướng đối tượng ra đời khá sớm, ví dụ như Simula-67 (năm 1967), Smalltalk (đầu những năm 1980), C++, CLOS (giữa những năm 1980)... thì những phương pháp luận cho phát triển hướng đối tượng lại ra đời khá muộn. Cuối những năm 80, đầu những năm 1990, một loạt các phương pháp luận và ngôn ngữ mô hình hóa hướng đối tượng mới ra đời, như Booch của Grady Booch, OMT của James Rumbaugh, OOSE của Ivar Jacobson, hay OOA and OOD của Coad và Yordon.

Mỗi phương pháp luận và ngôn ngữ trên đều có hệ thống ký hiệu riêng, phương pháp xử lý riêng và công cụ hỗ trợ riêng. Chính điều này đã thúc đẩy những người tiên phong trong lĩnh vực mô hình hóa hướng đối tượng ngồi lại cùng nhau để tích hợp những điểm mạnh của mỗi phương pháp và đưa ra một mô hình thống nhất chung. Nỗ lực thống nhất đầu tiên bắt đầu khi Rumbaugh gia nhập nhóm nghiên cứu của Booch tại tập đoàn Rational năm 1994 và sau đó Jacobson cũng gia nhập nhóm này vào năm 1995.

James Rumbaugh, Grady Booch và Ivar Jacobson đã cùng cố gắng xây dựng được một Ngôn Ngữ Mô Hình Hoá Thống Nhất và đặt tên là UML (Unifield Modeling Language) (Hình 3.1). UML đầu tiên được đưa ra năm 1997 và sau đó được chuẩn hóa để trở thành phiên bản 1.0. Chương này trình bày ngôn ngữ UML phiên bản 2.0.



Hình 3.1: Sự ra đời của UML

3.1.2 UML – Ngôn ngữ mô hình hoá hướng đối tượng

UML (Unified Modelling Language) là ngôn ngữ mô hình hoá tổng quát được xây dựng để đặc tả, phát triển và viết tài liệu cho các khía cạnh trong phát triển phần mềm hướng đối tượng. UML giúp người phát triển hiểu rõ và ra quyết định liên quan đến phần mềm cần xây dựng. UML bao gồm một tập các khái niệm, các ký hiệu, các biểu đồ và hướng dẫn. UML hỗ trợ xây dựng hệ thống hướng đối tượng dựa trên việc nắm bắt khía cạnh cấu trúc tĩnh và các hành vi động của hệ thống.

- Các cấu trúc tĩnh định nghĩa các kiểu đối tượng quan trọng của hệ thống, nhằm cài đặt và chỉ ra mối quan hệ giữa các đối tượng đó.
- Các hành vi động (dynamic behavior) định nghĩa các hoạt động của các đối tượng theo thời gian và tương tác giữa các đối tượng hướng tới đích.

Các mục đích của ngôn ngữ mô hình hoá thống nhất UML:

- Mô hình hoá các hệ thống sử dụng các khái niệm hướng đối tượng.
- Thiết lập sự liên hệ từ nhận thức của con người đến các sự kiện cần mô hình hoá.
- Giải quyết vấn đề về mức độ thừa kế trong các hệ thống phức tạp với nhiều ràng buộc khác nhau.
- Tạo một ngôn ngữ mô hình hoá có thể sử dụng được bởi người và máy.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

UML quy định một loạt các ký hiệu và quy tắc để mô hình hoá các pha trong quá trình phát triển phần mềm hướng đối tượng dưới dạng các biểu đồ.

3.1.3 Các khái niệm cơ bản trong UML

a) Khái niệm mô hình

Mô hình là một biểu diễn của sự vật hay một tập các sự vật trong một lĩnh vực áp dụng nào đó theo một cách khác. Mô hình nhằm nắm bắt các khía cạnh quan trọng của sự vật, bỏ qua các khía cạnh không quan trọng và biểu diễn theo một tập ký hiệu và quy tắc nào đó. Các mô hình thường được xây dựng sao cho có thể vẽ được thành các biểu đồ dựa trên tập ký hiệu và quy tắc đã cho.

Khi xây dựng các hệ thống, mô hình được sử dụng nhằm thoả mãn các mục đích sau:

- Nắm bắt chính xác yêu cầu và tri thức miền mà hệ thống cần phát triển
- Thể hiện tư duy về thiết kế hệ thống
- Trợ giúp ra quyết định thiết kế dựa trên việc phân tích yêu cầu
- Tổ chức, tìm kiếm, lọc, kiểm tra và sửa đổi thông tin về các hệ thống lớn.
- Làm chủ được các hệ thống phức tạp

Các thành phần trong một mô hình bao gồm:

- Ngữ nghĩa và biểu diễn: Ngữ nghĩa là nhằm đưa ra ý nghĩa, bản chất và các tính chất của tập các ký hiệu. Biểu diễn là phương pháp thể hiện mô hình theo cách sao cho có thể nhìn thấy được.
- Ngữ cảnh: mô tả tổ chức bên trong, cách sử dụng mô hình trong tiến trình phần mềm ...

b) Các hướng nhìn (View) trong UML

Các mô hình trong UML nhằm mục đích hỗ trợ phát triển các hệ thống phần mềm hướng đối tượng. Trong phương pháp luận hướng đối tượng không có sự phân biệt rạch ròi giữa các pha hay các bước. Tuy nhiên, thông thường UML vẫn được chia thành một số hướng nhìn và nhiều loại biểu đồ.

Một hướng nhìn trong UML là một tập con các biểu đồ UML được xây dựng để biểu diễn một khía cạnh nào đó của hệ thống.

Sự phân biệt giữa các hướng nhìn là rất linh hoạt. Có thể có những biểu đồ UML có mặt trong cả hai hướng nhìn. Các hướng nhìn cùng các biểu đồ tương ứng được mô tả trong bảng sau:

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

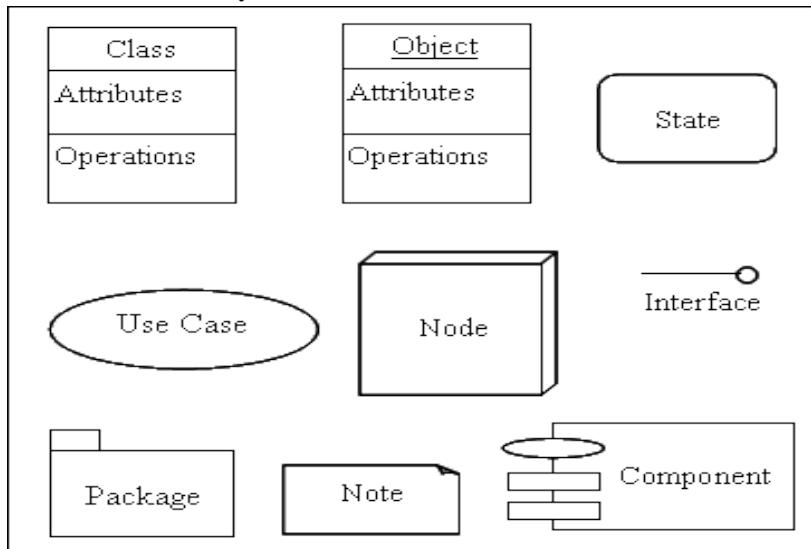
Khía cạnh chính	Hướng nhìn	Các biểu đồ	Các khái niệm chính
<i>Khía cạnh cấu trúc hệ thống</i>	Hướng nhìn tĩnh (static view)	Biểu đồ lớp	lớp, liên hệ, kế thừa, phụ thuộc, giao diện
	Hướng nhìn use case (Use case view)	Biểu đồ use case	Use case, tác nhân, liên hệ, extend, include ...
	Hướng nhìn cài đặt (implementation view)	Biểu đồ thành phần	Thành phần, giao diện, quan hệ phụ thuộc ...
	Hướng nhìn triển khai (deployment view)	Biểu đồ triển khai	Node, thành phần, quan hệ phụ thuộc, vị trí (location)
<i>Khía cạnh động</i>	Hướng nhìn máy trạng thái (state machine view)	Biểu đồ trạng thái	Trạng thái, sự kiện, chuyển tiếp, hành động
	Hướng nhìn hoạt động (activity view)	Biểu đồ động	Trạng thái, sự kiện, chuyển tiếp, kết hợp, đồng bộ ...
	Hướng nhìn tương tác (interaction view)	Biểu đồ tuần tự	Tương tác, đối tượng, thông điệp, kích hoạt ...
		Biểu đồ cộng tác	Cộng tác, vai trò cộng tác, thông điệp ...
<i>Khía cạnh quản lý mô hình</i>	Hướng nhìn quản lý mô hình	Biểu đồ lớp	Gói, hệ thống con, mô hình
<i>Khía cạnh khả năng mở rộng</i>	Tất cả	Tất cả	Các ràng buộc, stereotype, ...

Bảng 2.1: Các hướng nhìn trong UML

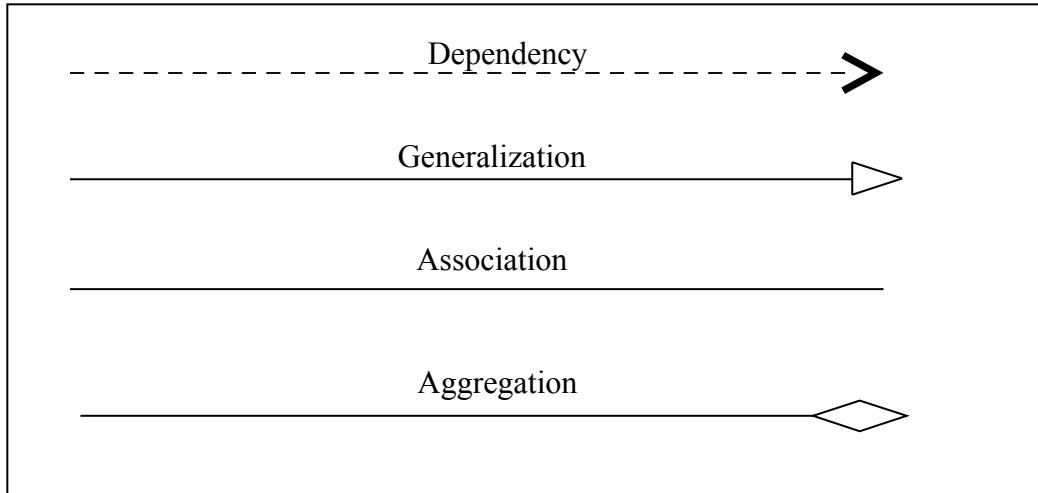
c) Các phần tử mô hình và các quan hệ

Một số ký hiệu để mô hình hướng đối tượng thường gặp trong UML được biểu diễn trong Hình 3.2. Đi kèm với các phần tử mô hình này là các *quan hệ*. Các quan hệ này có thể xuất hiện trong bất cứ mô hình nào của UML dưới các dạng khác nhau (như quan hệ giữa các use case, quan hệ trong biểu đồ lớp ...) (Hình 3.3).

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



Hình 3.2: Một số phần tử mô hình thường gặp trong UML



Hình 3.3: Một số dạng quan hệ trong UML

Ý nghĩa của các phần tử mô hình và các quan hệ sẽ được giải thích cụ thể hơn trong các chương sau.

3.2 CÁC BIỂU ĐỒ UML

Thành phần mô hình chính trong UML là các biểu đồ:

- **Biểu đồ use case** biểu diễn sơ đồ chức năng của hệ thống. Từ tập yêu cầu của hệ thống, biểu đồ use case sẽ phải chỉ ra hệ thống cần thực hiện điều gì để thỏa mãn các yêu cầu của người dùng hệ thống đó. Đi kèm với biểu đồ use case là các kịch bản.
- **Biểu đồ lớp** chỉ ra các lớp đối tượng trong hệ thống, các thuộc tính và phương thức của từng lớp và các mối quan hệ giữa những lớp đó.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

- **Biểu đồ trạng thái** tương ứng với mỗi lớp sẽ chỉ ra các trạng thái mà đối tượng của lớp đó có thể có và sự chuyển tiếp giữa những trạng thái đó.
- **Các biểu đồ tương tác** biểu diễn mối liên hệ giữa các đối tượng trong hệ thống và giữa các đối tượng với các tác nhân bên ngoài. Có hai loại biểu đồ tương tác:
 - **Biểu đồ tuần tự:** Biểu diễn mối quan hệ giữa các đối tượng và giữa các đối tượng và tác nhân theo thứ tự thời gian.
 - **Biểu đồ cộng tác:** Biểu diễn mối quan hệ giữa các đối tượng và giữa các đối tượng và tác nhân nhưng nhấn mạnh đến vai trò của các đối tượng trong tương tác.
- **Biểu đồ hoạt động** biểu diễn các hoạt động và sự đồng bộ, chuyển tiếp các hoạt động, thường được sử dụng để biểu diễn các phương thức phức tạp của các lớp.
- **Biểu đồ thành phần** định nghĩa các thành phần của hệ thống và mối liên hệ giữa các thành phần đó.
- **Biểu đồ triển khai hệ thống** mô tả hệ thống sẽ được triển khai như thế nào, thành phần nào được cài đặt ở đâu, các liên kết vật lý hoặc giao thức truyền thông nào được sử dụng.

Dựa trên tính chất của các biểu đồ, UML chia các biểu đồ thành hai lớp mô hình¹:

- **Biểu đồ mô hình cấu trúc (Structural Modeling Diagrams):** biểu diễn các cấu trúc tĩnh của hệ thống phần mềm được mô hình hóa. Các biểu đồ trong mô hình tĩnh tập trung biểu diễn khía cạnh tĩnh của hệ thống, liên quan đến cấu trúc cơ bản cũng như các phần tử chính trong miền quan tâm của bài toán. Các biểu đồ trong mô hình tĩnh bao gồm:
 - Biểu đồ gói
 - Biểu đồ đối tượng và lớp
 - Biểu đồ thành phần
 - Biểu đồ triển khai
- **Biểu đồ mô hình hành vi (Behavioral Modeling Diagrams):** Nắm bắt đến các hoạt động và hành vi của hệ thống, cũng như tương tác giữa các phần tử bên trong và bên ngoài hệ thống. Các dạng biểu đồ trong mô hình động bao gồm:
 - Biểu đồ use case
 - Biểu đồ tương tác dạng tuần tự
 - Biểu đồ tương tác dạng cộng tác
 - Biểu đồ trạng thái
 - Biểu đồ động

¹ Tham khảo http://www.sparxsystems.com.au/resources/uml2_tutorial/

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

Chúng ta sẽ lần lượt xem xét chi tiết các biểu đồ UML, mỗi biểu đồ sẽ được trình bày ý nghĩa của nó, tập kí hiệu UML cho biểu đồ đó và một ví dụ.

3.2.1 Biểu đồ use case

3.2.1.1 Ý nghĩa

Biểu đồ use case biểu diễn sơ đồ chức năng của hệ thống. Từ tập yêu cầu của hệ thống, biểu đồ use case sẽ phải chỉ ra hệ thống cần thực hiện điều gì để thoả mãn các yêu cầu của người dùng hệ thống đó. Đi kèm với biểu đồ use case là các kịch bản (scenario). Có thể nói, biểu đồ use case chỉ ra sự tương tác giữa các *tác nhân* và hệ thống thông qua các use case.

Mỗi *use case* mô tả một chức năng mà hệ thống cần phải có xét từ quan điểm người sử dụng. *Tác nhân* là con người hay hệ thống thực khác cung cấp thông tin hay tác động tới hệ thống.

Một *biểu đồ use case* là một tập hợp các tác nhân, các use case và các mối quan hệ giữa chúng. Các use case trong biểu đồ use case có thể được phân rã theo nhiều mức khác nhau.

3.2.1.2 Tập kí hiệu UML cho biểu đồ use case

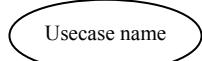
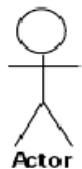
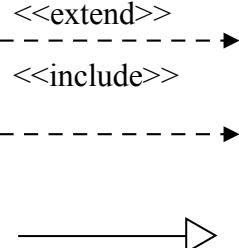
Một biểu đồ Use Case chứa các phần tử mô hình biểu thị hệ thống, tác nhân cũng như các trường hợp sử dụng và các mối quan hệ giữa các Use Case. Chúng ta sẽ lần lượt xem xét các phần tử mô hình này:

- *Hệ thống*: Với vai trò là thành phần của biểu đồ use case, hệ thống biểu diễn ranh giới giữa bên trong và bên ngoài của một chủ thể trong phần mềm chúng ta đang xây dựng. Chú ý rằng một hệ thống ở trong biểu đồ use case không phải bao giờ cũng nhất thiết là một hệ thống phần mềm; nó có thể là một chiếc máy, hoặc là một hệ thống thực (như một doanh nghiệp, một trường đại học, ...).
- *Tác nhân (actor)*: là người dùng của hệ thống, một tác nhân có thể là một người dùng thực hoặc các hệ thống máy tính khác có vai trò nào đó trong hoạt động của hệ thống. Như vậy, tác nhân thực hiện các use case. Một tác nhân có thể thực hiện nhiều use case và ngược lại một use case cũng có thể được thực hiện bởi nhiều tác nhân.
- *Các use case*: Đây là thành phần cơ bản của biểu đồ use case. Các use case được biểu diễn bởi các hình elip. Tên các use case thể hiện một chức năng xác định của hệ thống.
- *Mối quan hệ giữa các use case*: giữa các use case có thể có các mối quan hệ như sau:
 - *Include*: use case này sử dụng lại chức năng của use case kia.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

- *Extend*: use case này mở rộng từ use case kia bằng cách thêm vào một chức năng cụ thể.
- *Generalization*: use case này được kế thừa các chức năng từ use case kia.

Các phần tử mô hình use case cùng với ý nghĩa và cách biểu diễn của nó được tổng kết trong Bảng 2.2.

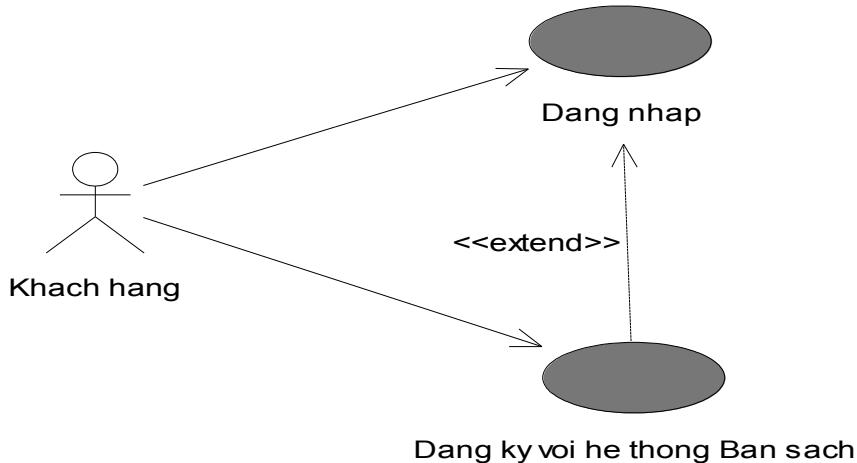
Phần tử mô hình	Ý nghĩa	Cách biểu diễn	Ký hiệu trong biểu đồ
Use case	Biểu diễn một chức năng xác định của hệ thống	Hình ellip chứa tên của use case	
Tác nhân	Là một đối tượng bên ngoài hệ thống tương tác trực tiếp với các use case	Biểu diễn bởi một lớp kiểu actor (hình người tượng trưng)	
Mối quan hệ giữa các use case	Tùy từng dạng quan hệ	Extend và include có dạng các mũi tên đứt nét Generalization có dạng mũi tên tam giác.	
Biên của hệ thống	Tách biệt phần bên trong và bên ngoài hệ thống	Được biểu diễn bởi một hình chữ nhật rỗng.	

Bảng 3.2: Các phần tử mô hình trong biểu đồ use case

3.2.1.3 Ví dụ biểu đồ use case

Dưới đây là một biểu đồ use case ví dụ trong hệ thống Bán sách (Book Shop). Khách hàng sử dụng hệ thống để lựa chọn sách và mua sách. Trước khi mua sách, khách hàng phải đăng nhập. Nếu chưa từng đăng ký thì khách hàng phải qua bước đăng ký. Chức năng đăng ký mở rộng cơ chế kiểm tra người dùng (username và password) trong chức năng đăng nhập nên hai use case này có quan hệ *<extend>*.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



Hình 3.4: Biểu đồ use case ví dụ

3.2.2 Biểu đồ lớp

3.2.2.1 Ý nghĩa

Trong phương pháp hướng đối tượng, *một nhóm đối tượng có chung một số thuộc tính và phương thức tạo thành một lớp*. Mỗi tương tác giữa các đối tượng trong hệ thống sẽ được biểu diễn thông qua mối quan hệ giữa các lớp.

Các lớp (bao gồm cả các thuộc tính và phương thức) cùng với các mối quan hệ sẽ tạo thành biểu đồ lớp. Biểu đồ lớp là một biểu đồ dạng mô hình tĩnh nhằm mô tả hướng nhìn tĩnh về một hệ thống bằng các khái niệm lớp, các thuộc tính, phương thức của lớp và mối quan hệ giữa chúng với nhau.

3.2.2.2 Tập ký hiệu UML cho biểu đồ lớp

Trong phần này, tài liệu sẽ xem xét các vấn đề liên quan đến biểu diễn sơ đồ lớp trong UML. Cuối phần này sẽ là một bảng tổng kết các ký hiệu UML sử dụng trong sơ đồ lớp.

- **Kí hiệu lớp:** trong UML, mỗi lớp được biểu diễn bởi hình chữ nhật gồm 3 phần: tên lớp, các thuộc tính và các phương thức.
- **Thuộc tính:** các thuộc tính trong biểu đồ lớp được biểu diễn theo cấu trúc chung như sau:

phạm_vi tên : kiểu số đối tượng = mặc định (Giá trị giới hạn)

Trong đó:

phạm_vi cho biết phạm vi truy nhập của thuộc tính. Có ba kiểu xác định thuộc tính phổ biến là:

+: thuộc tính kiểu public

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

#: thuộc tính kiểu protected

-: thuộc tính kiểu private.

~: thuộc tính được phép truy nhập tới từ các lớp trong cùng package

Các phạm vi của thuộc tính có thể được biểu diễn dưới dạng ký hiệu (+, #, -, ~) hoặc biểu diễn dưới dạng các từ khoá (public, protected, private).

Tên: là xâu ký tự biểu diễn tên thuộc tính.

kiểu: là kiểu dữ liệu của thuộc tính.

số_đối_tượng: chỉ ra số đối tượng khai báo cho thuộc tính ứng với một

mặc định: là giá trị khởi đầu mặc định (nếu có) của thuộc tính.

Giá_trị_giới_hạn: là giới hạn các giá trị cho thuộc tính (thông tin này không bắt buộc).

Ví dụ một khai báo thuộc tính đầy đủ:

purchaseDate:Date[1] = "01-01-2000" (Saturday)

- **Phương thức (method):** các phương thức trong UML được biểu diễn theo cấu trúc chung như sau [UNG]:

phạm_vi tên(danh_sách_tham_số): kiểu trả_lại { kiểu_phương_thức}

Trong đó:

visibility biểu diễn phạm vi cho phương thức. Giống như đối với thuộc tính, có ba dạng kiểu xác định cơ bản cho phương thức là:

- +: phương thức kiểu public

- #: phương thức kiểu protected

- -: phương thức kiểu private

- ~: phương thức được phép truy nhập tới từ các lớp trong cùng package

tên là xâu ký tự xác định tên của phương thức.

kiểu_ trả_lại: chỉ ra kiểu giá trị trả về của phương thức.

danh_sách_tham_số: biểu diễn danh sách các tham số trong khai báo của phương thức. Mỗi tham số được biểu diễn dưới dạng chung: tên tham số: *kiểu giá trị = giá trị mặc định*.

kiểu_phương_thức: không bắt buộc, cho biết kiểu phương thức. Phương thức có thể nhận một trong các kiểu đặc biệt sau:

abstract: phương thức kiểu trừu tượng

query: phương thức kiểu truy vấn.

Ví dụ một khai báo phương thức cho một lớp:

generatePurchaseList(prodID:int): String

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

• Các kiểu lớp trong UML

UML định nghĩa một số kiểu lớp đặc biệt dựa trên vai trò của nó trong biểu đồ lớp. Ngoài kiểu lớp thông thường đã trình bày ở trên, UML còn định nghĩa một số kiểu lớp bổ sung gồm:

- *Lớp thực thể*: là lớp đại diện cho các thực thể chứa thông tin về các đối tượng xác định nào đó.
- *Lớp biên (lớp giao diện)*: là lớp nằm ở ranh giới giữa hệ thống với môi trường bên ngoài, thực hiện vai trò nhận yêu cầu trực tiếp từ các tác nhân và chuyển các yêu cầu đó cho các lớp bên trong hệ thống.
- *Lớp điều khiển*: thực hiện các chức năng điều khiển hoạt động của hệ thống ứng với các chức năng cụ thể nào đó với một nhóm các lớp biên hoặc lớp thực thể xác định.

STT	Kiểu lớp	Kí hiệu UML
1	<i>Lớp thực thể</i>	
2	<i>Lớp biên (lớp giao diện)</i>	
3	<i>Lớp điều khiển</i>	

Bảng 3.3: Các kiểu lớp trong UML

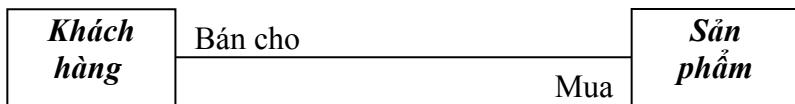
• Các mối quan hệ trong biểu đồ lớp

Giữa các lớp có các dạng quan hệ cơ bản như sau:

- *Quan hệ kết hợp (Association)*: Một kết hợp (association) là một sự nối kết giữa các lớp, cũng có nghĩa là sự nối kết giữa các đối tượng của các lớp này. Trong UML, một quan hệ được xác định nhằm mô tả một tập hợp các liên kết (links), tức là một sự liên quan về ngữ nghĩa (semantic connection) giữa một nhóm các đối tượng được biểu diễn bởi các lớp tương ứng.

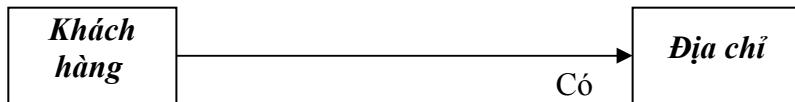
Mặc định, quan hệ kết hợp được biểu diễn bởi đoạn thẳng 2 chiều nối 2 đối tượng và có thể kèm theo ngữ nghĩa của quan hệ tại hai đầu của đoạn thẳng. Xem ví dụ Hình 2.5. Lớp khách hàng có quan hệ kết hợp với lớp sản phẩm. Ngữ nghĩa của quan hệ này thể hiện ở chỗ: khách hàng *mua* sản phẩm, còn sản phẩm *được bán cho* khách hàng.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



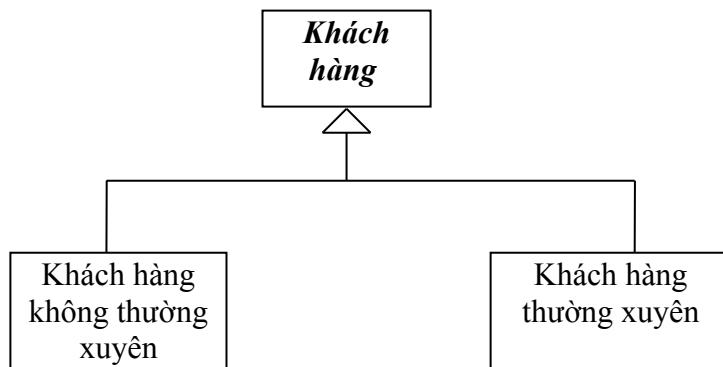
Hình 3.5: Quan hệ kết hợp

Quan hệ kết hợp cũng có thể có dạng một chiều. Xem ví dụ Hình 2.6.



Hình 2.6: Quan hệ kết hợp một chiều

- *Khái quát hóa (Generalization):* Khái quát hóa là mối quan hệ giữa một lớp có các đặc trưng mang tính khái quát cao hơn và một lớp có tính chất đặc biệt hơn. Trong sơ đồ lớp, mối quan hệ khái quát hóa chính là sự kế thừa của một lớp từ lớp khác. Quan hệ khái quát hóa được biểu diễn bằng một mũi tên có tam giác rỗng gắn ở đầu. Xem ví dụ Hình 2.7.



Hình 3.7: Quan hệ khái quát hóa

- *Quan hệ cộng hợp (Aggregation):* là dạng quan hệ mô tả một lớp A là một phần của lớp B và lớp A có thể tồn tại độc lập. Quan hệ cộng hợp được biểu diễn bằng một mũi tên gắn hình thoi rỗng ở đầu hướng về lớp bao hàm. Xem ví dụ Hình 2.8.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

Lớp Hoá đơn là một phần của lớp Khách hàng nhưng đối tượng Hoá đơn vẫn có thể tồn tại độc lập với đối tượng khách hàng.



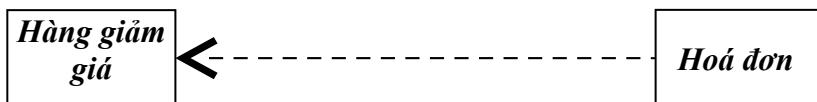
Hình 3.8: Quan hệ cộng hợp

- *Quan hệ gộp (Composition)*: Một quan hệ gộp biểu diễn một quan hệ kiểu tổng thể-bộ phận. Lớp A có quan hệ gộp với lớp B nếu lớp A là một phần của lớp B và sự tồn tại của đối tượng lớp B điều khiển sự tồn tại của đối tượng lớp A. Quan hệ này được biểu diễn bởi một mũi tên gắn hình thoi đặc ở đầu. Xem ví dụ Hình 2.9.



Hình 3.9: Quan hệ gộp

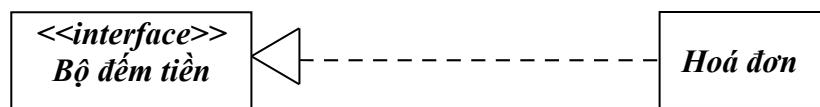
- *Quan hệ phụ thuộc (Dependency)*: Phụ thuộc là mối quan hệ giữa hai lớp đối tượng: một lớp đối tượng A có tính độc lập và một lớp đối tượng B phụ thuộc vào A; một sự thay đổi của A sẽ ảnh hưởng đến lớp phụ thuộc B. Xem ví dụ Hình 2.10.



Hình 3.10: Quan hệ phụ thuộc

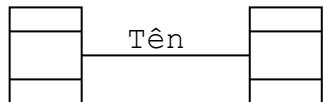
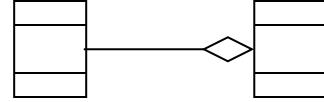
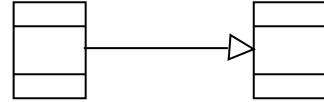
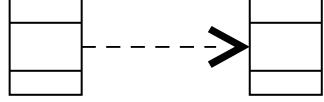
- *Quan hệ thực thi (Realization)*: biểu diễn mối quan hệ ngữ nghĩa giữa các thành phần của biểu đồ lớp, trong đó một thành phần mô tả một công việc dạng hợp đồng và thành phần còn lại thực hiện hợp đồng đó. Thông thường lớp *thực hiện hợp đồng* có thể là các giao diện. Xem ví dụ Hình 2.11.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



Hình 3.11: Quan hệ thực thi

Bảng 2.4 tổng kết các phần tử mô hình UML được sử dụng trong mô hình lớp, ý nghĩa và ký hiệu tương ứng trong các biểu đồ.

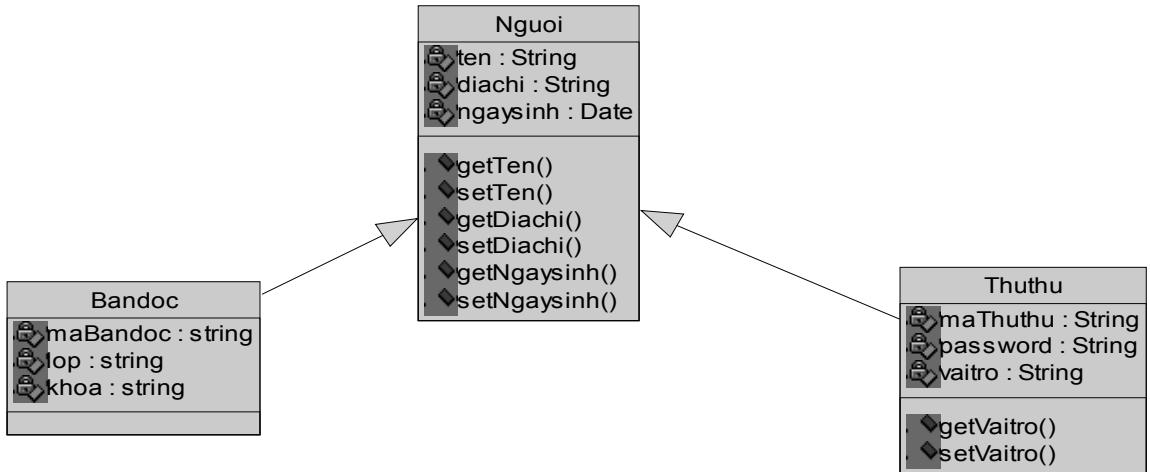
Phần tử mô hình	Ý nghĩa	Cách biểu diễn	Ký hiệu trong biểu đồ
Lớp (class)	Biểu diễn tên lớp, các thuộc tính và phương thức của lớp đó.	Một hình chữ nhật gồm 3 phần tách biệt.	Tên lớp Các thuộc tính Các phương thức
Quan hệ kiểu kết hợp	Biểu diễn quan hệ giữa hai lớp độc lập, có liên quan đến nhau.	Một đường kẻ liền nét (có tên xác định) nối giữa hai lớp.	
Quan hệ gộp	Biểu diễn quan hệ kiểu bộ phận – tổng thể.	Đường kẻ liền nét có hình thoi ở đầu.	
Quan hệ khái quát hóa (kế thừa)	Lớp này thừa hưởng các thuộc tính - phương thức của lớp kia	Mũi tên tam giác.	
Quan hệ phụ thuộc.	Các lớp phụ thuộc lẫn nhau trong hoạt động của hệ thống.	Mũi tên đứt nét.	

Bảng 3.4: Tóm tắt các phần tử mô hình UML trong biểu đồ lớp

3.2.2.3 Ví dụ biểu đồ lớp

Dưới đây là ví dụ một phần của biểu đồ lớp trong hệ thống quản lý thư viện trong đó các lớp Thủ như (người quản lý thư viện) và Bạn đọc kế thừa từ lớp tổng quát là lớp Người.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



Hình 3.12: Biểu đồ lớp

3.2.3 Biểu đồ trạng thái

3.2.3.1 Ý nghĩa

Biểu đồ trạng thái được sử dụng để biểu diễn các trạng thái và sự chuyển tiếp giữa các trạng thái của các đối tượng trong một lớp xác định. Thông thường, mỗi lớp sẽ có một biểu đồ trạng thái (trừ lớp trừu tượng là lớp không có đối tượng). Biểu đồ trạng thái được biểu diễn dưới dạng máy trạng thái hữu hạn với các trạng thái và sự chuyển tiếp giữa các trạng thái đó. Có hai dạng biểu đồ trạng thái:

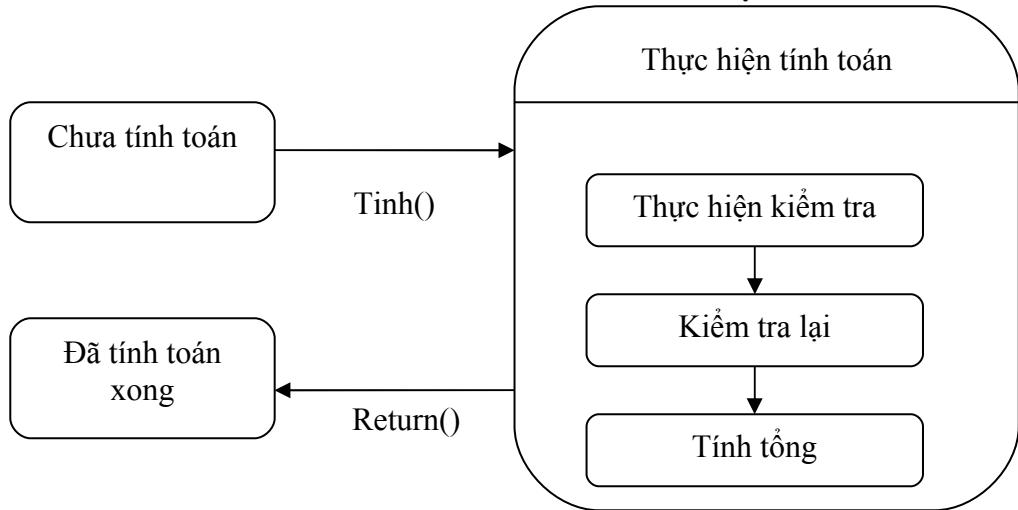
- Biểu đồ trạng thái cho một use case: mô tả các trạng thái và chuyển tiếp trạng thái của một đối tượng thuộc một lớp nào đó trong hoạt động của một use case cụ thể.
- Biểu đồ trạng thái hệ thống mô tả tất cả các trạng thái của một đối tượng trong toàn bộ hoạt động của cả hệ thống.

3.2.3.2 Tập ký hiệu UML cho biểu đồ trạng thái

Các thành phần trong một biểu đồ trạng thái bao gồm:

- **Trạng thái (state).** Bên trong các trạng thái có thể miêu tả các biến trạng thái hoặc các hành động (action) tương ứng với trạng thái đó.
- **Trạng thái con (substate):** là một trạng thái chứa bên trong một trạng thái khác. Trạng thái có nhiều trạng thái con gọi là trạng thái tổ hợp. Xem xét một ví dụ có trạng thái con trong Hình 2.13.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



Hình 3.13: Biểu đồ trạng thái có trạng thái con

- **Trạng thái khởi đầu (initial state):** trạng thái đầu tiên khi kích hoạt đối tượng.
- **Trạng thái kết thúc (final state):** kết thúc vòng đời đối tượng.
- **Các chuyển tiếp (transition):** biểu diễn các chuyển đổi giữa các trạng thái.
- **Sự kiện (event):** sự kiện tác động gây ra sự chuyển đổi trạng thái. Mỗi sự kiện được đi kèm với các điều kiện (guard) và các hành động (action).

Trong biểu đồ trạng thái của UML, một số loại sự kiện sau đây sẽ được xác định:

- **Sự kiện gọi (call event):** Yêu cầu thực hiện một hành động (một phương thức)
- **Sự kiện tín hiệu (signal event):** Gửi thông điệp (chứa các giá trị thuộc tính tham số liên quan) giữa các trạng thái.
- **Sự kiện thời gian (time event):** Biểu diễn quá trình chuyển tiếp theo thời gian, thường kèm theo từ mô tả thời gian cụ thể.

Các phần tử mô hình UML và ký hiệu tương ứng cho biểu đồ trạng thái được tổng kết như trong Bảng 3.5.

Phần tử mô hình	Ý nghĩa	Biểu diễn	Ký hiệu trong biểu đồ
-----------------	---------	-----------	-----------------------

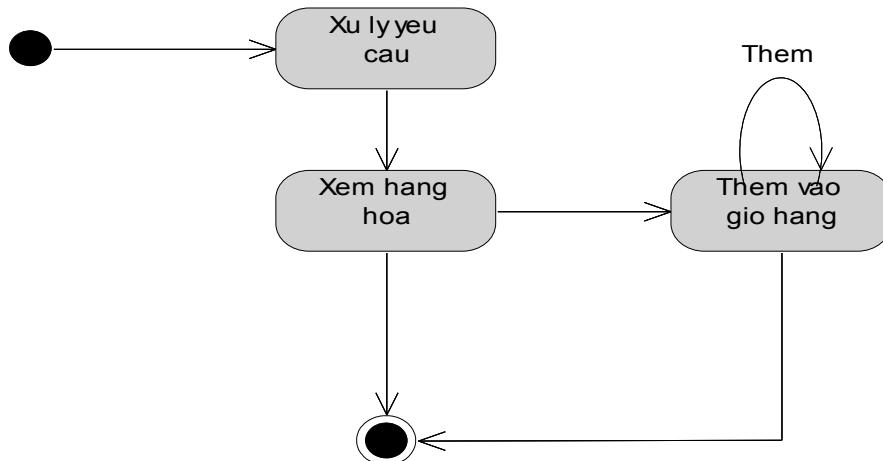
CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

<i>Trạng thái</i>	Biểu diễn một trạng thái của đối tượng trong vòng đời của đối tượng đó.	Hình chữ nhật vòng ở góc, gồm 3 phần: tên, các biến, và các hoạt động.	
<i>Trạng thái khởi đầu</i>	Khởi đầu vòng đời của đối tượng.	Hình tròn đặc	
<i>Trạng thái kết thúc</i>	Kết thúc vòng đời của đối tượng.	Hai hình tròn lồng nhau	
<i>Chuyển tiếp (transition)</i>	Chuyển từ trạng thái này sang trạng thái khác	Mũi tên liền nét với tên gọi là biểu diễn của chuyển tiếp đó.	

Bảng 3.5: Các phần tử mô hình UML trong biểu đồ trạng thái

3.2.3.3 Ví dụ Biểu đồ trạng thái

Để minh họa cho biểu đồ trạng thái, ta xét hệ thống mua hàng đơn giản. Một khách hàng gửi yêu cầu cần tìm mua một số đến hệ thống. Đối tượng hàng hóa được tạo ra và chuyển sang trạng thái *xử lý yêu cầu*. Tùy thuộc vào yêu cầu, đối tượng này sẽ chuyển sang trạng thái *xem hàng hóa*. Với mỗi hàng hóa được lựa chọn, đối tượng sẽ chuyển sang trạng thái *thêm vào giỏ hàng* cho đến khi kết thúc quá trình lựa chọn.



Hình 3.14: Ví dụ biểu đồ trạng thái

3.2.4 Biểu đồ tương tác dạng tuần tự

Các biểu đồ tương tác biểu diễn mối liên hệ giữa các đối tượng trong hệ thống và giữa các đối tượng với các tác nhân bên ngoài. Có hai loại biểu đồ tương tác: Biểu đồ tuần tự và biểu đồ cộng tác.

Ý nghĩa

Biểu đồ tuần tự: Biểu diễn mối quan hệ giữa các đối tượng, giữa các đối tượng và tác nhân theo thứ tự thời gian. Biểu đồ tuần tự nhấn mạnh thứ tự thực hiện của các tương tác.

Tập kí hiệu UML cho biểu đồ tuần tự

Các thành phần cơ bản của một biểu đồ tuần tự là:

- *Các đối tượng (object):* được biểu diễn bởi các hình chữ nhật, bên trong là tên của đối tượng. Cách viết chung của đối tượng là: *tên đối tượng: tên lớp*. Nếu chỉ viết: *tên_lớp* thì có nghĩa là bất cứ đối tượng nào của lớp tương ứng đó. Trong biểu đồ tuần tự, không phải các đối tượng đều xuất hiện ở trên cùng của biểu đồ mà chúng chỉ xuất hiện (về mặt thời gian) khi thực sự tham gia vào tương tác.
- *Các message:* được biểu diễn bằng các mũi tên hướng từ đối tượng gửi sang đối tượng nhận. Tên các message có thể biểu diễn dưới dạng phi hình thức (như các thông tin trong kịch bản) hoặc dưới dạng hình thức (với dạng giống như các phương thức). Biểu đồ tuần tự cho phép có các message từ một đối tượng tới chính bản thân nó.
- Trong biểu đồ tuần tự có thể có nhiều loại message khác nhau tùy theo mục đích sử dụng và tác động của message đến đối tượng. Các dạng message được tổng kết trong Bảng 2.6 dưới đây:

<i>ST</i>	<i>Loại message</i>	<i>Mô tả</i>	<i>Biểu diễn</i>
1	<i>Gọi (call)</i>	Mô tả một lời gọi từ đối tượng này đến đối tượng kia.	Method()
2	<i>Trả về (return)</i>	Trả về giá trị ứng với lời gọi	← — Giá trị trả về — →
3	<i>Gửi (send)</i>	Gửi một tín hiệu tới một đối tượng	Send()
4	<i>Tạo (create)</i>	Tạo một đối tượng	<<create>>

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

5 Huỷ (destroy) Huỷ một đối tượng

<<destroy>>

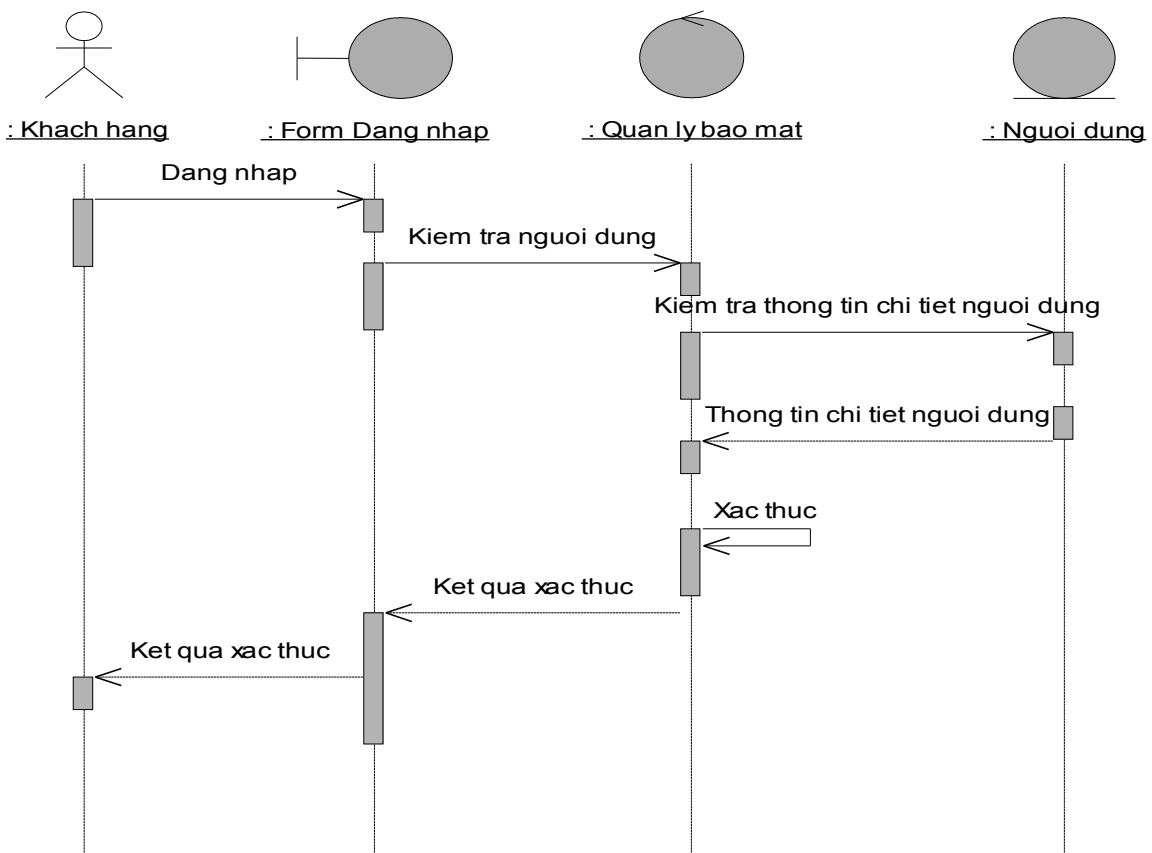


Bảng 3.6: Các dạng message trong biểu đồ tuần tự

- *Đường lifeline:* là một đường kẻ nối dài phía dưới đối tượng, mô tả quá trình của đối tượng trong tương tác thuộc biểu đồ.
- *Chú thích:* biểu đồ tuần tự cũng có thể có chú thích để người đọc dễ dàng hiểu được nội dung của biểu đồ đó.

Ví dụ biểu đồ tương tác dạng tuần tự

Dưới đây là một ví dụ cho biểu đồ tuần tự cho chức năng Đăng nhập trong một hệ thống đơn giản. Các đối tượng tham gia trong tương tác là: tác nhân khách hàng, giao diện đăng nhập, điều khiển quản lý bảo mật và thực thể người dùng.



Hình 3.15: Ví dụ biểu đồ tuần tự

3.2.5 Biểu đồ tương tác dạng cộng tác

Ý nghĩa

Biểu đồ cộng tác: Là biểu đồ tương tác biểu diễn mối quan hệ giữa các đối tượng; giữa các đối tượng và tác nhân nhấn mạnh đến vai trò của các đối tượng trong tương tác.

Biểu đồ cộng tác cũng có các message với nội dung tương tự như trong biểu đồ tuần tự. Tuy nhiên, các đối tượng được đặt một cách tự do trong không gian của biểu đồ và không có đường life line cho mỗi đối tượng. Các message được đánh số thể hiện thứ tự thời gian.

Tập kí hiệu UML cho biểu đồ cộng tác

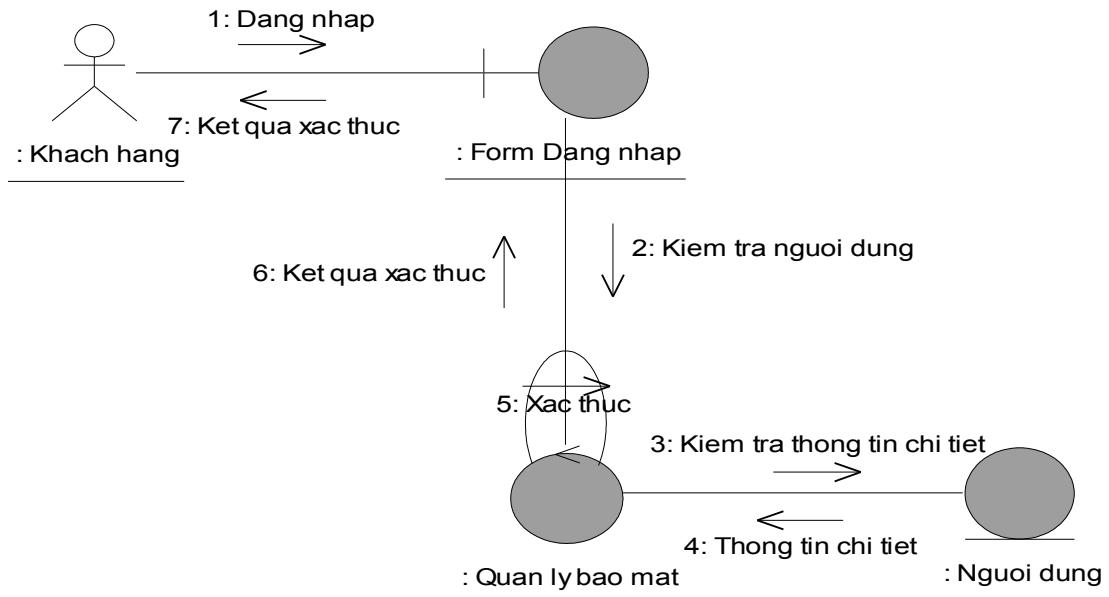
Các thành phần cơ bản của một biểu đồ cộng tác là:

- *Các đối tượng:* được biểu diễn bởi các hình chữ nhật, bên trong là tên của đối tượng. Cách viết chung của đối tượng là: *tên đối tượng: tên lớp*. Trong biểu đồ cộng tác, các đối tượng tham gia tương tác luôn xuất hiện tại một vị trí xác định.
- *Các liên kết:* giữa hai đối tượng có tương tác sẽ có một liên kết nối 2 đối tượng đó. Liên kết này không có chiều.
- *Các message:* được biểu diễn bằng các mũi tên hướng từ đối tượng gửi sang đối tượng nhận bên cạnh liên kết giữa 2 đối tượng đó. Trong biểu đồ cộng tác, các message được đánh số thứ tự theo thứ tự xuất hiện trong kịch bản mô tả use case tương ứng.

Ví dụ biểu đồ cộng tác

Dưới đây là một biểu đồ cộng tác mô tả chức năng đăng nhập trong hệ thống đơn giản. Chi tiết chức năng này hoàn toàn tương tự như đã mô tả trong biểu đồ tuần tự hình 2.15.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



Hình 3.16: Biểu đồ cộng tác

3.2.6 Biểu đồ hoạt động

Ý nghĩa

Biểu đồ hoạt động biểu diễn các hoạt động và sự đồng bộ, chuyển tiếp các hoạt động của hệ thống trong một lớp hoặc kết hợp giữa các lớp với nhau trong một chức năng cụ thể.

Biểu đồ hoạt động có thể được sử dụng cho nhiều mục đích khác nhau, ví dụ như:

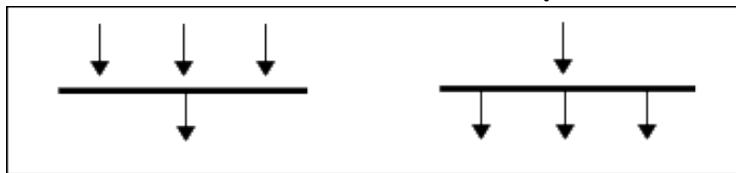
- Để xác định các hành động phải thực hiện trong phạm vi một phương thức.
- Để xác định công việc cụ thể của một đối tượng.
- Để chỉ ra một nhóm hành động liên quan của các đối tượng được thực hiện như thế nào và chúng sẽ ảnh hưởng đến những đối tượng nằm xung quanh.

Tập kí hiệu UML

Các phần tử mô hình UML cho biểu đồ hoạt động bao gồm:

- Hoạt động (Activity)*: là một quy trình được định nghĩa rõ ràng, có thể được thực hiện bởi một hàm hoặc một nhóm đối tượng. Hoạt động được thể hiện bằng hình chữ nhật tròn cạnh.
- Thanh đồng bộ hóa (Synchronisation bar)*: cho phép ta mở ra hoặc là đóng lại các nhánh chạy song song trong tiến trình.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



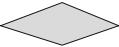
Hình 3.17: Thanh đồng bộ hoá trong biểu đồ động

- *Điều kiện (Guard Condition)*: các biểu thức logic có giá trị hoặc đúng hoặc sai. Điều kiện được thể hiện trong ngoặc vuông, ví dụ: [Customer existing].
- *Các luồng (swimlane)*: Mỗi biểu đồ động có thể biểu diễn sự phối hợp hoạt động trong nhiều lớp khác nhau. Khi đó mỗi lớp được phân tách bởi một luồng (swimlane) riêng biệt. Các luồng này được biểu diễn đơn giản là các ô khác nhau trong biểu đồ.

Các ký hiệu UML cho biểu đồ hoạt động được tổng kết trong Bảng sau:

Phần tử mô hình	Ý nghĩa	Ký hiệu trong biểu đồ
<i>Hoạt động</i>	Mô tả một hoạt động gồm tên hoạt động và đặc tả của nó.	NewActivity
<i>Trạng thái khởi đầu</i>		●
<i>Trạng thái kết thúc</i>		○
<i>Thanh đồng bộ ngang</i>	Mô tả thanh đồng bộ nằm ngang	—
<i>Thanh đồng bộ hoá đọc</i>	Mô tả thanh đồng bộ theo chiều thẳng đứng	

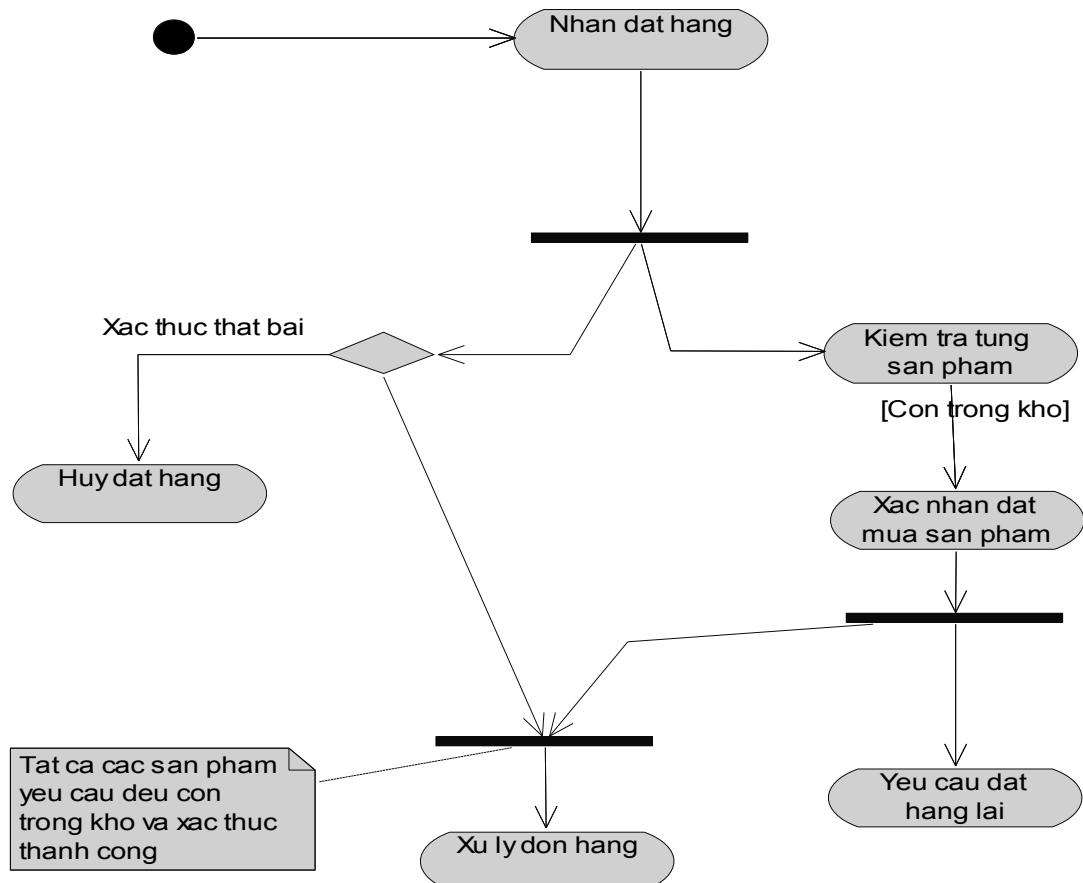
CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

<i>Chuyển tiếp</i>		
<i>Quyết định</i>	Mô tả một lựa chọn điều kiện.	
<i>Các luồng (swimlane)</i>	Phân tách các lớp đối tượng khác nhau tồn tại trong biểu đồ hoạt động	Phân cách nhau bởi một đường kẻ dọc từ trên xuống dưới biểu đồ

Bảng 3.6: Các phần tử của biểu đồ hoạt động

Ví dụ biểu đồ hoạt động

Dưới đây là ví dụ biểu đồ hoạt động mô tả chức năng kiểm tra yêu cầu mua hàng trong hệ thống bán hàng.



Hình 3.18: Biểu đồ hoạt động

3.2.7 Biểu đồ thành phần

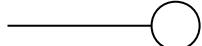
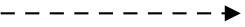
Ý nghĩa

Biểu đồ thành phần được sử dụng để biểu diễn các thành phần phần mềm cấu thành nên hệ thống. Một hệ phần mềm có thể được xây dựng từ đầu bằng cách sử dụng mô hình lớp như đã trình bày trong các phần trước của tài liệu, hoặc cũng có thể được tạo nên từ các thành phần sẵn có.

Mỗi thành phần có thể coi như một phần mềm nhỏ hơn, cung cấp một khối dạng hộp đen trong quá trình xây dựng phần mềm lớn. Nói cách khác, các thành phần là các gói được xây dựng cho quá trình triển khai hệ thống. Các thành phần có thể là các gói ở mức cao như JavaBean, các gói thư viện liên kết động dll, hoặc các phần mềm nhỏ được tạo ra từ các thành phần nhỏ hơn như các lớp và các thư viện chức năng.

Tập ký hiệu UML

Tập ký hiệu UML cho biểu đồ thành phần được tổng kết trong bảng sau:

Phần tử mô hình	Ý nghĩa	Ký hiệu trong biểu đồ
<i>Thành phần</i>	Mô tả một thành phần của biểu đồ, mỗi thành phần có thể chứa nhiều lớp hoặc nhiều chương trình con.	 Component
<i>Giao tiếp</i>	Mô tả giao tiếp gắn với mỗi thành phần. Các thành phần trao đổi thông tin qua các giao tiếp.	
<i>Mối quan hệ phụ thuộc giữa các thành phần</i>	Mối quan hệ giữa các thành phần (nếu có).	
<i>Gói (package)</i>	Được sử dụng để nhóm một số thành phần lại với nhau.	 NewPackage

Bảng 3.7: Các ký hiệu của biểu đồ thành phần

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

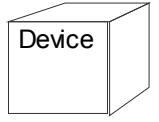
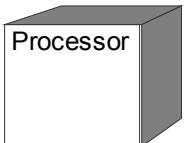
3.2.8 Biểu đồ triển khai hệ thống

Ý nghĩa

Biểu đồ triển khai biểu diễn kiến trúc cài đặt và triển khai hệ thống dưới dạng các nodes và các mối quan hệ giữa các node đó. Thông thường, các nodes được kết nối với nhau thông qua các liên kết truyền thông như các kết nối mạng, liên kết TCP-IP, microwave... và được đánh số theo thứ tự thời gian tương tự như trong biểu đồ cộng tác.

Tập ký hiệu UML cho biểu đồ triển khai

Tập ký hiệu UML cho biểu đồ triển khai hệ thống được biểu diễn trong Bảng sau:

Phần tử mô hình	Ý nghĩa	Ký hiệu trong biểu đồ
Các nodes (hay các thiết bị)	Biểu diễn các thành phần không có bộ vi xử lý trong biểu đồ triển khai hệ thống	
Các bộ xử lý	Biểu diễn các thành phần có bộ vi xử lý trong biểu đồ triển khai hệ thống	
Các liên kết truyền thông	Nối các thành phần của biểu đồ triển khai hệ thống. Thường mô tả một giao thức truyền thông cụ thể.	_____

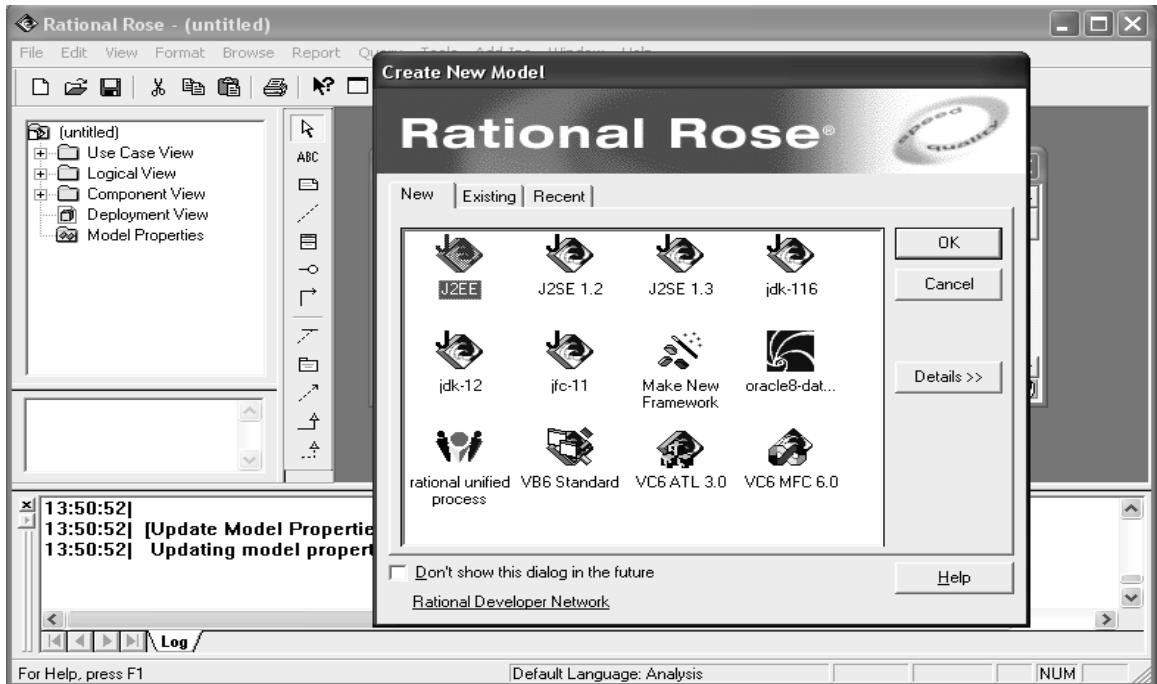
Bảng 3.8: Các ký hiệu của biểu đồ triển khai hệ thống

3.3 GIỚI THIỆU CÔNG CỤ RATIONAL ROSE

Rational Rose là một bộ công cụ được sử dụng cho phát triển các hệ phần mềm hướng đối tượng theo ngôn ngữ mô hình hóa UML. Với chức năng của một bộ công cụ trực quan, Rational Rose cho phép chúng ta tạo, quan sát, sửa đổi và quản lý các biểu đồ. Tập ký hiệu mà Rational Rose cung cấp thống nhất với các ký hiệu trong UML. Ngoài ra, Rational Rose còn cung cấp chức năng hỗ trợ quản lý dự án phát triển phần mềm, cung cấp các thư viện để hỗ trợ sinh khung mã cho hệ thống theo một ngôn ngữ lập trình nào đó.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

Màn hình khởi động của Rational Rose phiên bản 2002 như trong Hình 2.19. Người sử dụng sẽ có thể chọn thư viện dự định sẽ cài đặt hệ thống, Rational Rose sẽ tải về các gói tương ứng trong thư viện đó. Các gói này (cùng các lớp tương ứng) sẽ xuất hiện trong biểu đồ lớp, người sử dụng sẽ tiếp tục phân tích, thiết kế hệ thống của mình dựa trên thư viện đó. Nếu sử dụng Rational Rose để xây dựng hệ thống từ đầu thì người sử dụng nên bỏ qua chức năng này.



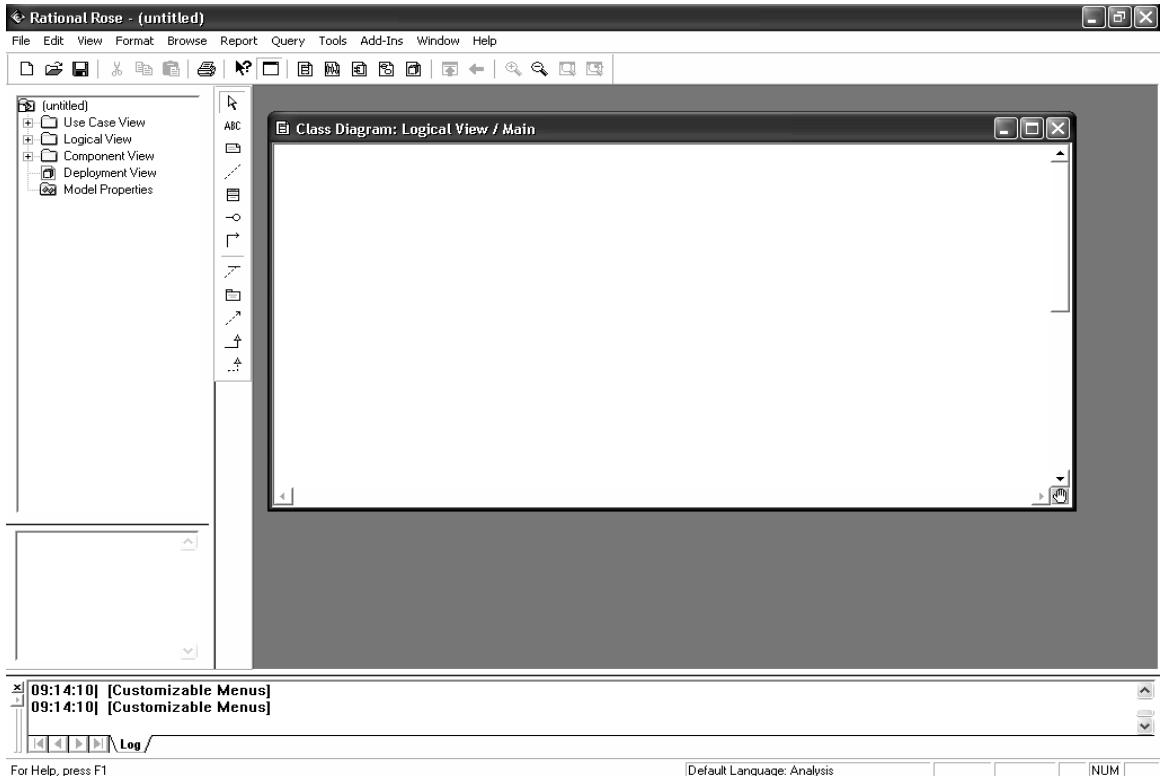
Hình 3.19: Màn hình khởi động Rational Rose

Trong giao diện của Rational Rose (Hình 3.19), cửa sổ phía bên trái là cửa sổ Browser chứa các View (hướng nhìn, quan điểm), trong mỗi View là các mô hình tương ứng của UML. Có thể xem mỗi View là một cách nhìn theo một khía cạnh nào đó của hệ thống.

- *Use Case View*: xem xét khía cạnh chức năng của hệ thống nhìn từ phía các tác nhân bên ngoài
- *Logical View*: xem xét quá trình phân tích và thiết kế logic của hệ thống để thực hiện các chức năng trong Use Case View.
- *Component View*: xem xét khía cạnh tổ chức hệ thống theo các thành phần và mối liên hệ giữa các thành phần đó.
- *Deployment View*: xem xét khía cạnh triển khai hệ thống theo các kiến trúc vật lý.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

Cửa sổ phía bên phải của màn hình Rational Rose là cửa sổ biểu đồ (Diagram Windows) được sử dụng để vẽ các biểu đồ sử dụng các công cụ vẽ tương ứng trong ToolBox. Hầu hết các ký hiệu sử dụng để vẽ biểu đồ trong Rational Rose đều thống nhất với chuẩn UML.

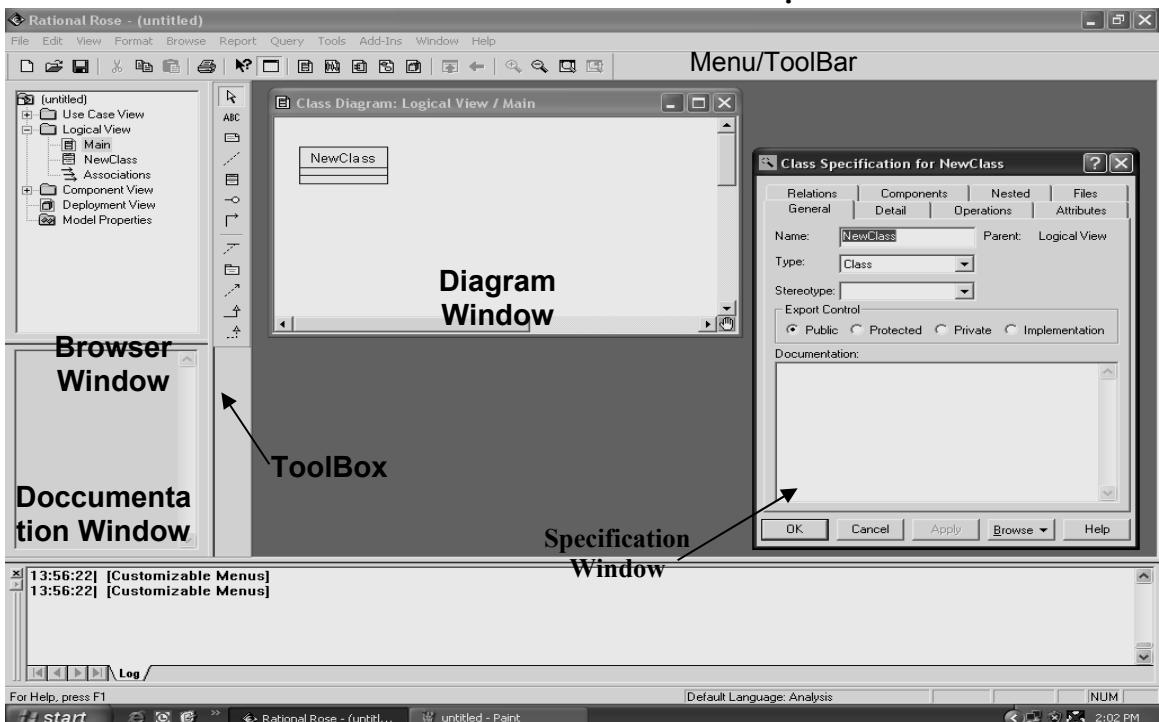


Hình 3.20: Giao diện chính của Rational Rose

Giao diện chính của Rational Rose trong các biểu đồ đều được chia thành các phần như trong Hình 3.21. Ý nghĩa chính của các thành phần này như sau:

- MenuBar vàToolBar chứa các menu và công cụ tương tự như các ứng dụng Windows khác.
- Phần Browser Window cho phép người sử dụng chuyển tiếp nhanh giữa các biểu đồ trong các View.
- Phần Documentation Window dùng để viết các thông tin liên quan đến các phần tử mô hình tương ứng trong biểu đồ. Các thông tin này có thể là các ràng buộc, mục đích, các từ khóa ... liên quan đến phần tử mô hình đó.
- Phần ToolBox chứa các công cụ dùng để vẽ biểu đồ. Ứng với mỗi dạng biểu đồ thì sẽ có một dạng toolbox tương ứng.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM



Hình 3.21: Các thành phần trong giao diện Rational Rose

- Phần Diagram Window là không gian để vẽ và hiệu chỉnh các biểu đồ trong mô hình tương ứng.
- Cửa sổ Specification Window là đặc tả chi tiết của mỗi phần tử mô hình theo các trường thông tin tương ứng với dạng biểu đồ đó.

Vấn đề sử dụng Rational Rose cho các bước cụ thể trong phân tích thiết kế hệ thống sẽ được trình bày chi tiết trong Chương 3 và 4 của tài liệu này.

3.4 KẾT LUẬN

Chương này đã giới thiệu ngôn ngữ mô hình hoá thông nhất UML và công cụ Rational Rose cho phát triển phần mềm hướng đối tượng. Các nội dung chính bao gồm:

- UML ra đời từ sự kết hợp các phương pháp luận phát triển phần mềm hướng đối tượng khác nhau đã có trước đó. UML hiện nay đã được coi là ngôn ngữ mô hình hoá chuẩn cho phát triển các phần mềm hướng đối tượng.
- UML được chia thành nhiều hướng nhìn, mỗi hướng nhìn quan tâm đến hệ thống phần mềm từ một khía cạnh cụ thể.
- Nếu xét theo tính chất mô hình thì UML có hai dạng mô hình chính là mô hình tĩnh và mô hình động. Mỗi mô hình lại bao gồm một nhóm các biểu đồ khác nhau.

CHƯƠNG 3. UML VÀ CÔNG CỤ PHÁT TRIỂN PHẦN MỀM

- Mỗi biểu đồ UML có một tập ký hiệu riêng để biểu diễn các thành phần của biểu đồ đó. Quá trình biểu diễn các biểu đồ cũng phải tuân theo các quy tắc được định nghĩa trong UML.
- Hiện nay có rất nhiều công cụ hỗ trợ phân tích thiết kế hệ thống hướng đối tượng sử dụng UML. Mặc dù Chương này trình bày Rational Rose và chưa cập nhật các công cụ phát triển phần mềm khác. Thực tế, sinh viên khoa CNTT đã sử dụng rộng rãi công cụ VP để phát triển các dự án phần mềm nhóm

<http://www.visual-paradigm.com/download/vpuml.jsp>

BÀI TẬP

1. UML ra đời từ các ngôn ngữ và phương pháp mô hình hóa nào?
2. Hướng nhìn là gì? UML bao gồm các hướng nhìn nào?
3. Liệt kê các biểu đồ của UML và tập ký hiệu UML cho từng biểu đồ đó.
4. Phân biệt mô hình tĩnh và mô hình động trong UML
5. Phân biệt các dạng quan hệ trong biểu đồ lớp như: quan hệ khái quát hóa, quan hệ kết hợp, quan hệ cộng hợp, quan hệ gộp.

CHƯƠNG 4 XÁC ĐỊNH YÊU CẦU

4.1 GIỚI THIỆU

Mục đích của giai đoạn thu thập yêu cầu là:

- Giải thích hoàn cảnh nghiệp vụ: Lý do tại sao phát triển phần mềm này? Khi quyết định xây dựng phần mềm, cần hiểu nghiệp vụ và hiểu các nhà tài trợ cho dự án. Do đó, thu thập yêu cầu giúp nhận biết được nhà tài trợ là ai.
- Mô tả các yêu cầu của hệ thống: Thu thập yêu cầu không những ảnh hưởng đến việc quyết định các chức năng của hệ thống mà còn ảnh hưởng đến việc phát hiện ra các ràng buộc như chi phí phát triển, tài nguyên...

Khi phát triển phần mềm, trước tiên ta phải chắc chắn rằng mình đã hiểu hoàn cảnh nghiệp vụ của hệ thống mới, sau đó sẽ làm việc với nhà đầu tư để đạt được sự chấp thuận về những gì mà hệ thống mới sẽ được xây dựng. Một khi đã hiểu nghiệp vụ và viết được tài liệu về những hiểu biết đó dưới dạng các yêu cầu nghiệp vụ (business requirements), ta cần xem xét những gì mà hệ thống sẽ làm cho người dùng. Quyết định hệ thống có thể làm gì và không thể làm gì sẽ giúp chúng ta tập trung vào việc tạo ra những dòng code cần thiết. Nếu không hiểu một cách tỉ mỉ về các yêu cầu hệ thống (system requirements), ta sẽ lãng phí thời gian vào việc phát triển những dòng code mà ta không được trả tiền để làm.

Các yêu cầu hệ thống thường được chia làm 2 nhóm: *yêu cầu chức năng* và *yêu cầu phi chức năng*. Yêu cầu chức năng là những gì mà hệ thống có thể làm, ví dụ các dịch vụ được cung cấp để đáp ứng những tác động bên ngoài như duyệt các danh mục, đặt chỗ mô hình xe... Các yêu cầu phi chức năng là những gì cần được xác định, ví dụ các trình duyệt Web phía client phải được hỗ trợ, sử dụng video để quảng cáo...

4.2 SỰ RA ĐỜI CỦA MỘT HỆ THỐNG

Mọi hệ thống đều được bắt đầu ở một thời điểm nào đó. Chúng ta có thể may mắn thu thập được tài liệu chi tiết từ khách hàng. Tuy nhiên, điều đó rất khó xảy ra. Là một nhà phát triển, chúng ta phải chuyển đổi tài liệu về các yêu cầu của khách hàng thành bản mô tả hoàn chỉnh về hệ thống được phát triển, theo định dạng chuẩn sao cho khách hàng có thể hiểu.

4.3 USE CASE

Use case được dùng để viết tài liệu về những hiểu biết cách vận hành một nghiệp vụ - mô hình hóa các yêu cầu nghiệp vụ - và để xác định những gì mà hệ thống mới có khả năng làm – mô hình hóa các yêu cầu hệ thống. Chi tiết về use case đã được trình bày khá đầy đủ trong Chương 3

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

4.4 QUAN ĐIỂM NGHIỆP VỤ

Mô hình nghiệp vụ có thể đơn giản chỉ là một biểu đồ lớp, các quan hệ giữa các thực thể nghiệp vụ, và đôi khi được gọi là mô hình miền (domain model). Đối với phần lớn các dự án, chúng ta tạo ra toàn bộ mô hình nghiệp vụ để biểu diễn cách mà các nghiệp vụ được vận hành. Use case không phải là cách duy nhất mô hình hóa một nghiệp vụ, nhưng đó là cách đơn giản bởi vì nó không đòi hỏi một sự am hiểu chuyên sâu về công nghệ, mà chỉ là sự hiểu biết phổ biến và một chút logic để có thể hiểu được. Các cách mô hình phức tạp hơn như mô hình tiến trình nghiệp vụ (business process modeling) và phân tích luồng công việc (workflow analysis) sẽ không được đề cập đến. Mô hình use case bao gồm một số thành phần sau:

- Danh sách actor (kèm theo mô tả).
- Bảng thuật ngữ (glossary).
- Các trường hợp sử dụng use case (có miêu tả chi tiết).
- Biểu đồ giao tiếp (communication diagram)
- Biểu đồ hoạt động (activity diagram)

Trong khía cạnh phát triển phần mềm hướng đối tượng, ta có thể lặp lại những bước để mô hình hóa nghiệp vụ cho đến khi có được một bức tranh đầy đủ về hệ thống.

4.4.1 Xác định các tác nhân nghiệp vụ (business actors)

Việc đầu tiên ta cần làm là xác định các tác nhân nghiệp vụ. Một actor là một người/đối tượng giữ một vai trò nào đó trong nghiệp vụ, một bộ phận hay một hệ thống phần mềm riêng. Việc xác định actor giúp chúng ta xác định được cách mà nghiệp vụ được sử dụng, từ đó chỉ ra được các trường hợp sử dụng. Trong cuộc sống, một actor có thể đóng vai trò khác nhau trong những thời gian khác nhau. Chẳng hạn, một người là một trợ lý trong kho Ô tô cho đến khi hết giờ, nhưng nếu ông ta quyết định thuê một ô tô trước khi về nhà thì ông ấy sẽ trở thành khách hàng. Do vậy trong giai đoạn này của quá trình phát triển, ta nên làm việc với các nhà đầu tư (chính là khách hàng) để tìm hiểu xem các hoạt động nghiệp vụ tương ứng với các actor hoạt động như thế nào và đưa ra những sự thảo luận một cách đơn giản.

Ví dụ: Hệ thống quản lý đăng ký học theo tín chỉ ở trường Đại học

- Văn phòng khoa: xác định các ràng buộc được đăng ký các môn học, mời giảng viên cho các lớp tín chỉ.
- Phòng đào tạo: tạo lớp tín chỉ, xây dựng kế hoạch lớp tín chỉ, ràng buộc được tốt nghiệp các khóa học đối với sinh viên.
- Sinh viên: xem các khóa học trong kỳ tới, đăng ký học, xem lịch học, xem kết quả học tập các môn học.
- Phòng kế toán: dựa trên số tín chỉ sinh viên đăng ký để thu học phí.

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

- Hệ thống lập lịch phòng học: sắp xếp lớp, phòng, ca, kíp theo sĩ số lớp tùy từng hệ, khoa, khóa học, chuyên ngành.
- Hệ thống quản lý sinh viên: phân lớp, quản lý mã sinh viên, hồ sơ học bạ sinh viên, danh sách sinh viên,...
- Hệ thống quản lý kết quả học tập: cập nhật điểm thành phần, danh sách sinh viên không đủ điều kiện dự thi, cập nhật điểm thi, tổng hợp điểm, danh sách sinh viên nhận học bổng,...

4.4.2 Xây dựng Bảng Thuật ngữ của dự án (Project Glossary)

Bảng thuật ngữ là một dạng khác của từ điển dữ liệu (data dictionary) nhằm làm sáng tỏ các thuật ngữ sử dụng cho một miền nào đó để mọi người hiểu được các sản phẩm trong quá trình phát triển phần mềm. Khái niệm từ điển dữ liệu ngụ ý rằng dữ liệu được mô hình hóa một cách độc lập và tách dữ liệu ra khỏi tiến trình làm việc liên quan. Một cách tốt hơn là giữ dữ liệu và tiến trình cùng nhau, đó là một cách tiếp cận phù hợp hơn trong quan điểm hướng đối tượng, do đó ta nên xây dựng những bảng thuật ngữ của dự án. Nó cũng cho phép ta sắp xếp thành các nhóm từ đồng nghĩa mà ta có thể dùng một trong các từ đó.

Mỗi bản ghi trong Bảng thuật ngữ định nghĩa một thuật ngữ có thể ngắn hoặc dài. Tuy nhiên việc mô tả actor thường trở nên chung chung bởi vì nhiều thuật ngữ được áp dụng trong nhiều ngữ cảnh. Do đó ta có thể ghi lại các quan hệ của mỗi thuật ngữ trong quá trình phát triển (actor nghiệp vụ, actor hệ thống..) Sau đây là danh sách các quan hệ mà bạn có thể sử dụng:

- Business actor: actor xuất hiện trong các yêu cầu nghiệp vụ.
- Business object: đối tượng xuất hiện trong các yêu cầu nghiệp vụ.
- System actor: actor xuất hiện trong các yêu cầu hệ thống.
- System object: đối tượng xuất hiện (bên trong hệ thống) trong các yêu cầu hệ thống.
- Analysis object: đối tượng xuất hiện trong mô hình phân tích.
- Deployment artifact: những sản phẩm được triển khai trong hệ thống. Ví dụ một tệp tin.
- Design object: đối tượng xuất hiện trong mô hình thiết kế.
- Design node: máy tính hoặc tiến trình thuộc về kiến trúc hệ thống.
- Design layer: phân rã hệ thống theo chiều ngang.
- Design package: một nhóm các lớp logic, được dùng để tổ chức việc phát triển.

Ở đây, đối tượng được hiểu là thực thể hoặc “dữ liệu và tiến trình đã được đóng gói”. Mỗi loại đối tượng (nghiệp vụ, hệ thống, phân tích, thiết kế) là khá khác nhau. Các thực thể trong Bảng thuật ngữ đều tuân theo phong cách đặt tên lớp. Nếu chúng ta sử dụng

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

cùng một phong cách trong tất cả tài liệu dự án của chúng ta, thì sẽ rõ ràng với người đọc rằng một định nghĩa tồn tại trong thuật ngữ dự án

Ví dụ: Hệ thống đăng ký học theo tín chỉ có một số thuật ngữ sau đây:

STT	Tiếng Anh	Tiếng Việt	Giải thích nội dung
1	Branches	Ngành	Một lĩnh vực đào tạo chia thành một số ngành
2	Classes	Lớp	Là một cách tổ chức một số sinh viên cùng nghành, cùng khóa để theo dõi, quản lý.
3	Credit	Tín chỉ	Được sử dụng để tính khối lượng học tập của SV. Một tín chỉ được quy định bằng 15 tiết học lý thuyết
4	Faculties	Khoa	Một bộ phận trong một trường đào tạo một hay một số ngành học nhất định
5	Majors	Chuyên nghành	Một nghành chia thành một số chuyên nghành
6	Marks	Điểm	Con số đánh giá kết quả thực hiện việc học tập của sinh viên về một môn học nào đó
7	Prerequisite subjects	Môn học điều kiện	Những yêu cầu cần có để có thể học được môn học nêu ra
8	schoolYear	Năm học	Thường gồm 10 tháng, chia làm một số học kỳ
9	Semesters	Học kỳ	Một khoảng thời gian được quy định trong một niên học, Niên học có thể chia thành hai hay ba học kỳ
10	Students	Sinh viên	Thí sinh trúng tuyển vào học trong trường đại học
11	Subjects	Môn học	Gồm các thông tin chi tiết về môn học, số tín chỉ, chuyên nghành, môn điều kiện...
12	Instructor	Giảng viên	Người giảng dạy các môn học
13	timeTables	Thời khóa biểu	Lịch học cho các lớp gồm môn học, giờ học, phòng học, và giảng viên dạy học

4.4.3 Xác định các trường hợp sử dụng nghiệp vụ (Business Use Cases)

Một khi đã xác định được các actor, nhiệm vụ tiếp theo là xác định các trường hợp sử dụng nghiệp vụ. Mỗi trường hợp sử dụng là một phần nhỏ của nghiệp vụ. Không có quy tắc nào để chia nhỏ nghiệp vụ thành các trường hợp sử dụng, mà cần có kinh nghiệm, tư duy logic và cảm nhận chung về nghiệp vụ, ngoài ra còn cần làm việc với các nhà tài trợ. Khi tìm các use case, luôn luôn nằm lòng câu hỏi sau đây: “*Các công việc chính làm cho nghiệp vụ đó hoạt động là gì?*” Nhớ rằng, khi mô hình hóa nghiệp vụ, chúng ta không quan tâm tới cách mà hệ thống mới có thể vận hành, mà phải miêu tả cách mà nghiệp vụ hiện tại đang diễn ra. UML không quy định nội dung của use case. Do đó chúng ta có thể

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

dùng ngôn ngữ tự nhiên, miêu tả trình tự các bước hay ngôn ngữ có cấu trúc (ngôn ngữ tự nhiên với if_then_else và cấu trúc lặp), hoặc bất cứ thứ gì cùng loại. Tuy nhiên, ta nên viết theo trình tự các bước với ngôn ngữ tự nhiên để làm use case rõ ràng và độc lập. Nếu chúng ta sử dụng ngôn ngữ cấu trúc, ta sẽ gặp nguy hiểm khi làm mô tả của chúng ta quá toán học (hướng máy tính) Sau khi đã có danh sách các use case ứng cử, chúng ta có thể liệt kê các bước có liên quan trong mỗi use case.

Ví dụ: Hệ thống đăng ký học theo tín chỉ

- Đăng ký khóa học: sinh viên đăng ký các khóa học trong kỳ tới với phòng đào tạo.
- Hủy khóa học: sinh viên hủy khóa học đã đăng ký với phòng đào tạo.
- Xem điểm: sinh viên xem điểm tổng kết các khóa học của mình.
- Xem kết quả đăng ký học: sinh viên xem danh sách các khóa học đã đăng ký.
- Xem chương trình học: sinh viên xem danh sách các môn học của từng khoa, chuyên ngành, hệ đào tạo
- Đăng ký thi đi: sinh viên đăng ký các khóa học sẽ thi cuối kỳ.
- Đăng ký thi lại: sinh viên đăng ký các khóa học sẽ thi lại.
- Xem lịch thi cá nhân: sinh viên xem lịch thi các khóa học mình đã đăng ký thi.
- Xem lịch giảng dạy: giảng viên xem lịch giảng dạy của mình trong kỳ tới.

Chi tiết hóa các use case:

Use case: Đăng ký khóa học

1. Sinh viên (SV) truy cập vào website nhà trường để đăng ký học
2. Hệ thống yêu cầu SV đăng nhập trước khi đăng ký
3. Hệ thống kiểm tra trong CSDL các thông tin liên quan đến SV này.
 - 3.1. Nếu đúng, SV đề nghị DS các môn học muốn ĐK.
 - 3.1.1. Nếu các yêu cầu đối với các khóa học được thỏa mãn, SV được ghi danh vào các lớp học.
 - 3.1.2. Nếu không, Hệ thống thông báo SV chưa đủ điều kiện học môn học.
 - 3.2. Nếu sai, Hệ thống yêu cầu SV nhập lại tài khoản và mật khẩu truy cập.

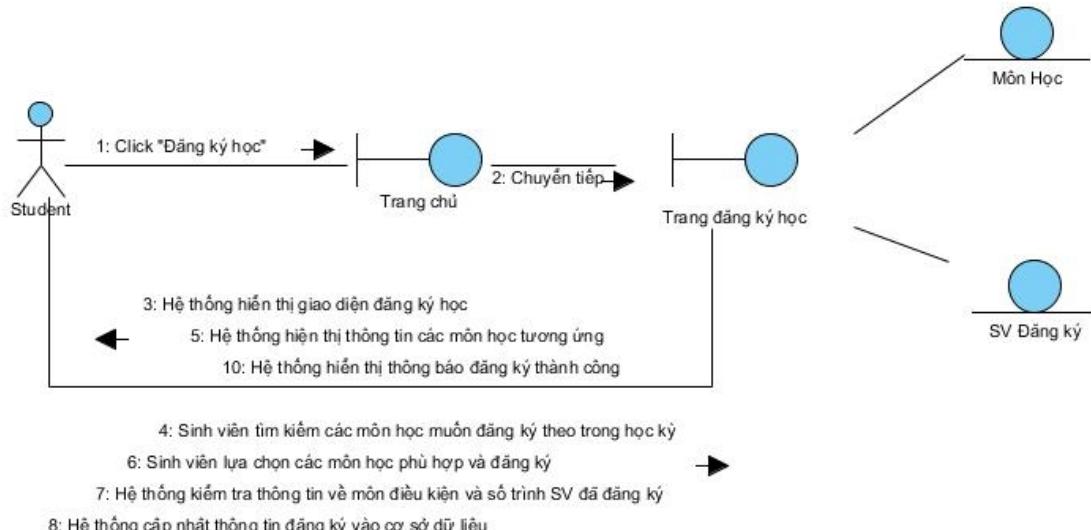
4.4.4 Minh họa các use case bằng biểu đồ giao tiếp (Communication diagram)

Biểu đồ giao tiếp biểu diễn một loạt các tương tác giữa các actor và các đối tượng. Biểu đồ tuân tự tập trung vào các tương tác giữa chúng và thứ tự xảy ra của chúng. Để hạn chế việc sử dụng quá nhiều chi tiết kỹ thuật trong giai đoạn đầu của quá trình phát triển, biểu đồ giao tiếp được dùng để mô hình hóa nghiệp vụ. Chuỗi các tương tác trong biểu đồ có thể không tương ứng hoàn toàn với chuỗi các bước trong miêu tả chi tiết use case. Mỗi tương tác sẽ biểu diễn một cách khái quát về một hoặc nhiều bước. Trong biểu đồ giao

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

tiếp, thông điệp được biểu diễn bằng một mũi tên nhỏ, vẽ dọc theo một kết nối giữa hai đối tượng, với hàm ý rằng nhờ kết nối đó mà bên gửi biết bên nhận để có thể gửi thông điệp.

Ví dụ: Biểu đồ giao tiếp cho use case Sinh viên đăng ký môn học



Hình 2: Biểu đồ giao tiếp

Có thể bạn nghĩ rằng thứ tự của các tương tác nên tương ứng một cách chính xác với thứ tự các bước trong use case chi tiết. Tuy nhiên bởi ngôn ngữ tự nhiên không phải là một dây tuần tự các bước. Do đó nhiều khả năng mỗi thao tác sẽ miêu tả một tổng hợp của một hay nhiều bước. Mặc dù thiếu sự chính xác so với use case nhưng biểu đồ giao tiếp vẫn là cần thiết bởi vì nó làm tăng chi tiết hóa cho các use case và có thể giúp chúng ta có một cái nhìn chi tiết rõ ràng hơn ngay từ đầu.

Khi đó sự tương tác chúng ta giải quyết trong giai đoạn này là không phức tạp, nó hoàn toàn là hợp lý khi đưa ra một biểu đồ giao tiếp cho mỗi use case. Tóm lại, bất kì sự cộng tác nào mà chúng ta miêu tả nên là đường đi chuẩn mực xuyên suốt use case. Khi chúng ta giải quyết use case hệ thống, chúng ta có thể có nhiều cách xác định về các đường đi khác thường (ngoại lệ), nhưng bây giờ chúng ta có thể nên ngụ ý về sự tồn tại của chúng cùng với bản thân mỗi use case.

4.4.5 Minh họa các use case bằng biểu đồ hoạt động (Activity diagram)

Biểu đồ hoạt động chỉ ra sự phụ thuộc giữa các hoạt động khi chúng ta di chuyển từ điểm bắt đầu tới một điểm kế thúc mà ta muốn. Giống như biểu đồ giao tiếp, trong biểu đồ hoạt động, mỗi hoạt động có thể không tương ứng với từng bước trong miêu tả chi tiết của use case. Biểu đồ hoạt động cũng có thể được dùng với nhiều mục đích. Ví dụ, nó có thể được dùng để xây dựng toàn bộ một mô hình nghiệp vụ hoặc viết tài liệu một thuật toán của một đối tượng phần mềm.

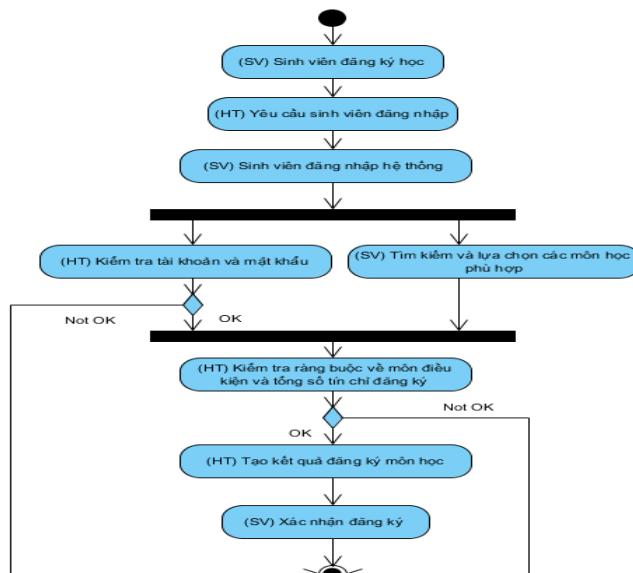
Tổng quan về biểu đồ hoạt động

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

Biểu đồ hoạt động là một đồ thị có hướng, trong đó các nút (đỉnh) là các hoạt động, và các cung là các dịch chuyển

- Hoạt động (activity) là một công việc có thể được xử lý bằng tay như Điền mẫu, hoặc bằng máy như Hiển thị màn hình đăng ký. Một hoạt động được biểu diễn trong biểu đồ bằng một hình chữ nhật tròn góc có mang tên của hoạt động
- Dịch chuyển (Transition) là sự chuyển từ hoạt động này sang hoạt động khác được thể hiện bằng một mũi tên nối giữa hai hoạt động
- Nút khởi đầu thể hiện điểm bắt đầu của hoạt động được ký hiệu bởi một hình tròn đặc
- Nút kết thúc thể hiện điểm kết thúc các hoạt động được ký hiệu hình tròn đặc có viền bao quanh. Tùy trường hợp có thể có một hoặc nhiều nút kết thúc
- Các điều kiện dịch chuyển hoạt động được ký hiệu bởi một hình thoi để thực hiện sự rẽ nhánh các hoạt động
- Thanh đồng bộ hóa (synchronization bars) để mở hay đóng các nhánh thực hiện song song

Ví dụ: Biểu đồ hoạt động cho use case Sinh viên đăng ký môn học



Hình 3: Biểu đồ hoạt động

4.5 QUAN ĐIỂM NHÀ PHÁT TRIỂN

Giai đoạn thứ hai của việc thu thập yêu cầu là mô hình phần mềm mà chúng ta đang định phát triển để cải tiến nghiệp vụ. Ta sẽ sử dụng mô hình use case vì nó dễ tạo và dễ hiểu đối với mọi người. Mô hình use case của hệ thống gồm các thành phần:

- Danh sách các actor (có miêu tả)
- Danh sách các use case (có miêu tả)

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

- Biểu đồ use case
- Chi tiết của use case (gồm cả các yêu cầu phi chức năng có liên quan)
- Một cái nhìn tổng quát về use case
- Các yêu cầu phụ (các yêu cầu hệ thống mà không phù hợp với bất kỳ use case cá biệt nào)
- Bản phác thảo giao diện người dùng
- Bảng thuật ngữ bổ sung
- Các ưu tiên của use case

4.5.1 Xác định các actor của hệ thống

Điều đầu tiên ta cần làm là xác định và miêu tả các actor của hệ thống, với sự giúp đỡ của khách hàng. Các actor ở đây chỉ bao gồm con người (và các hệ thống bên ngoài) có tương tác trực tiếp với hệ thống sẽ xây dựng, chứ không rộng như các actor trong nghiệp vụ.

Ví dụ: Hệ thống đăng ký học theo tín chỉ

- Nhân viên: cập nhật môn học, thông tin về khoa, chuyên ngành, lớp học, điểm thi...
- Sinh viên: xem các khóa học trong kỳ tới, đăng ký học,...
- Giảng viên: đăng ký môn dạy, xem lịch giảng dạy...

4.5.2 Danh sách các use case

Một khi đã có các actor, ta sẽ đi xác định các use case dưới sự trợ giúp của các khách hàng. Mỗi use case phải có một mô tả ngắn gọn về các use case đó.

Ví dụ: Hệ thống đăng ký học theo tín chỉ

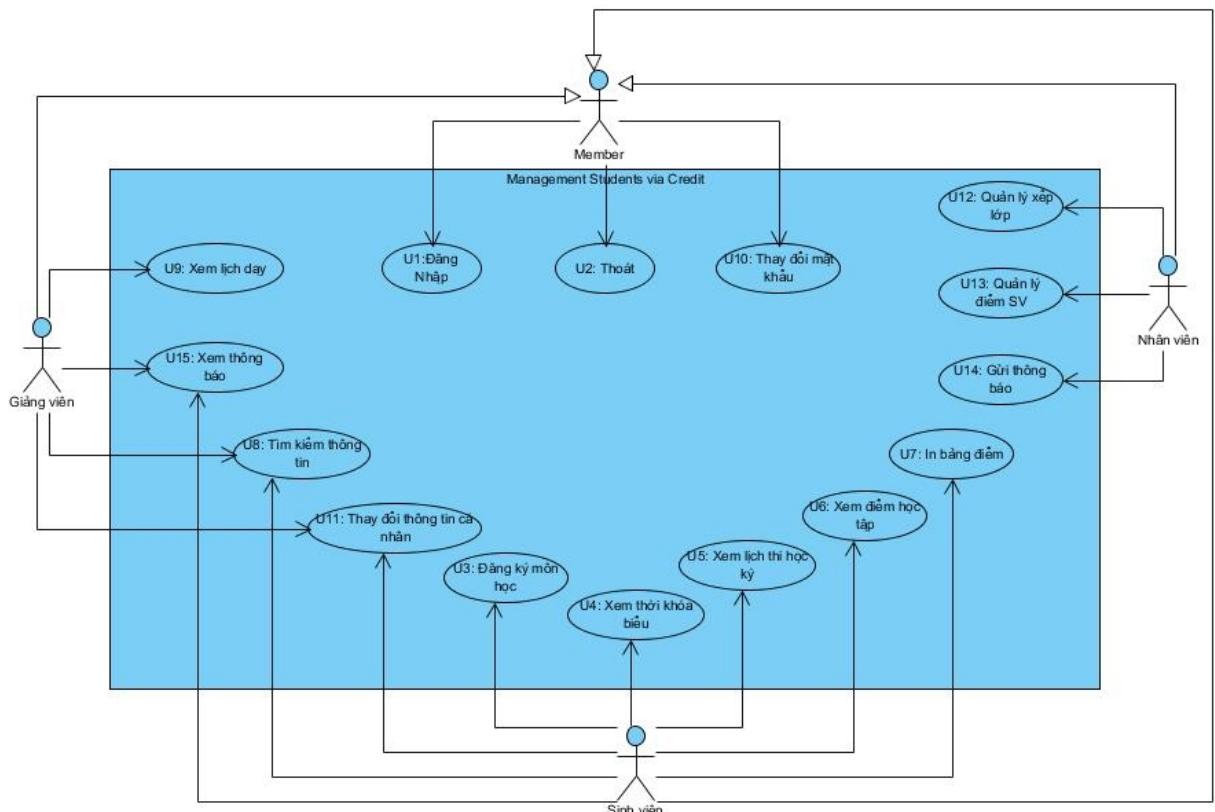
- U1: Đăng nhập: các actor đăng nhập hệ thống.
- U2: Thoát: các actor thoát khỏi hệ thống.
- U3: Đăng ký môn học: sinh viên đăng ký các môn học trong kỳ tới.
- U4: Xem thời khóa biểu: sinh viên xem thời khóa biểu chi tiết các lớp học đã đăng ký của mình
- U5: Xem lịch thi học kỳ: sinh viên xem lịch thi học kỳ các môn đã đăng ký
- U6: Xem điểm học tập: sinh viên xem kết quả học tập của mình.
- U7: In bảng điểm: sinh viên in bảng điểm các khóa học của mình.
- U8: Tìm kiếm thông tin: sinh viên xem danh sách các môn học của từng khoa, chuyên ngành, hệ đào tạo
- U9: Xem lịch giảng dạy: giảng viên xem lịch giảng dạy của mình trong kỳ tới.

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

- U10: Đổi mật khẩu: người dùng thay đổi mật khẩu truy cập hệ thống.
- U11: Thay đổi thông tin cá nhân: Người dùng thay đổi thông tin cá nhân sau khi đăng nhập hệ thống
- U12: Quản lý xếp lớp : Nhân viên phòng đào tạo thực hiện chức năng phân chia và xếp lớp chi các môn học sinh viên đăng ký
- U13: Quản lý điểm sinh viên: Nhân viên phòng đào tạo cập nhật kết quả học tập của sinh viên
- U14: Gửi thông báo: phòng đào tạo gửi các thông báo tới sinh viên.
- U15: Xem thông báo: giảng viên, sinh viên xem thông báo của phòng đào tạo gửi đến

4.5.3 Biểu đồ use case

Use case hệ thống có thể được miêu tả trên một Biểu đồ Use case để chỉ ra những Actor và sự liên kết của chúng với các use case cụ thể. Điều này giúp ta có một cái nhìn tổng thể về hệ thống sẽ được dùng như thế nào. Một Biểu đồ Use case cho Hệ thống quản lý học tập theo tín chỉ được chỉ ra trong sơ đồ sau:



Hình 4: Biểu đồ use case

4.5.4 Quan hệ giữa các actor

Một actor có thể cụ thể hóa actor khác. Ví dụ, actor Người dùng được cụ thể hóa bởi các actor Sinh viên, Giảng viên và Nhân viên Phòng đào tạo. Bất kỳ actor nào cũng có thể

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

được cụ thể hóa (kế thừa hành vi) từ các actor khác. Điều này thêm sức mạnh mô tả tới mô hình use case. Chẳng hạn ta có thể quyết định rằng Sinh viên là một khái niệm trừu tượng mà cần phải được cụ thể hóa như là Thành viên của hệ thống. Điều này sẽ mang lại nhiều lợi ích hơn là những rắc rối phức tạp. Nhớ rằng các nhà đầu tư của bạn sẽ phải hiểu các tài liệu mà bạn giới thiệu, từ mô hình nghiệp vụ đúng cách tới ít nhất là phân tích trạng thái tĩnh. Điều này sẽ giúp đảm bảo rằng bạn hiểu vấn đề chính xác và sẽ chuyển đến nhà đầu tư những gì mà họ thực sự mong muốn.

Ví dụ: Hệ thống đăng ký học theo tín chỉ

- Nhân viên: cập nhật môn học, thông tin về khoa, chuyên ngành, lớp học, điểm thi...
- Sinh viên: xem các khóa học trong kỳ tới, đăng ký học,...
- Giảng viên: đăng ký môn dạy, xem lịch giảng dạy...
- Thành viên: một sinh viên (hoặc giảng viên) mà thông tin cá nhân của họ đã được lưu trong cơ sở dữ liệu. Mỗi thành viên sẽ được cấp mật khẩu để truy cập tài khoản cá nhân của mình
- Phi thành viên: các tác nhân sử dụng hệ thống chưa thực hiện chức năng đăng nhập

4.5.5 Quan hệ giữa các use case

Giữa các actor có quan hệ kế thừa, giữa actor và use case có quan hệ kết hợp, còn giữa các use case có 3 kiểu quan hệ: đặc biệt hóa (specializes), bao gồm (includes) và mở rộng (extends). Chúng cho phép nhóm các use case có liên quan, phân rã use case lớn, dùng lại hành vi và để xác định hành vi không bắt buộc:

- **Specializes:** giống như actor, use case có thể kế thừa từ use case khác. Để tránh những phức tạp liên quan đến việc định nghĩa lại các bước và thêm các bước phụ, ta có thể cụ thể hóa các use case trừu tượng (abstract use case). Use case trừu tượng là use case không có bước nào: nhiệm vụ của nó là nhóm các use case khác.
- **Includes:** một use case có một số bước được cung cấp bởi một use case khác được gọi là bao gồm use case đó. Bao gồm được dùng để rút ra các bước chung cho một số use case, hoặc để chia use case lớn thành các use case nhỏ hơn.
- **Extends:** một use case thêm thông tin vào use case khác được gọi là mở rộng use case đó. Mở rộng cho phép ta thêm các thông tin không bắt buộc, thông thường, các thông tin đó xuất hiện ở cuối use case, nhưng chúng cũng có thể xảy ra ở đầu, hoặc đôi khi là ở giữa.

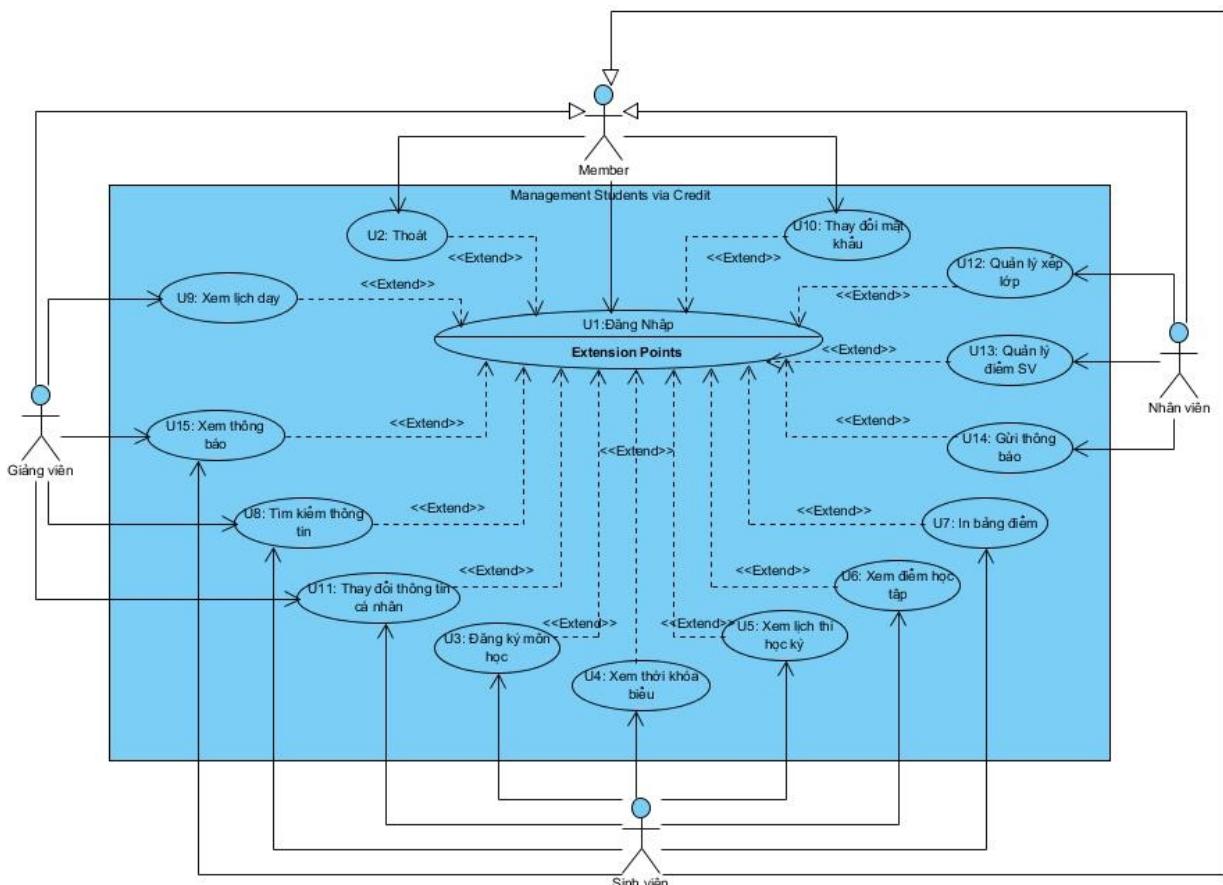
Có một sự khác nhau cơ bản giữa bao gồm và mở rộng: với bao gồm, use case nguồn sẽ không làm việc mà không có use case đích, với mở rộng, use case nguồn sẽ làm việc tốt

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

kết cấu không có use case đích. Nói cách khác, use case được bao gồm có sự tồn tại độc lập, còn use case mở rộng chỉ tồn tại như là một mở rộng.

Chúng ta không thể xác định được chính xác quan hệ giữa các use case trong lần đầu tiên qua mô hình yêu cầu hệ thống. Hơn nữa, việc xác định quan hệ này phụ thuộc vào chúng ta quyết định xem cái gì là thực sự cần thiết và khách hàng của chúng ta chấp nhận nó hay không. Như với các lĩnh vực khác của hướng đối tượng, sẽ có rất nhiều cách phân tích use case thành các mối quan hệ bao gồm, mở rộng và kế thừa. Thật ra không có cách nào là chính xác, vấn đề là chúng ta phải hết sức phát triển một mô hình sao cho phù hợp với cảm nhận của chúng ta và khách hàng.

Ví dụ: Hệ thống đăng ký học tín chỉ



Hình 5: Lược đồ use case tổng quát

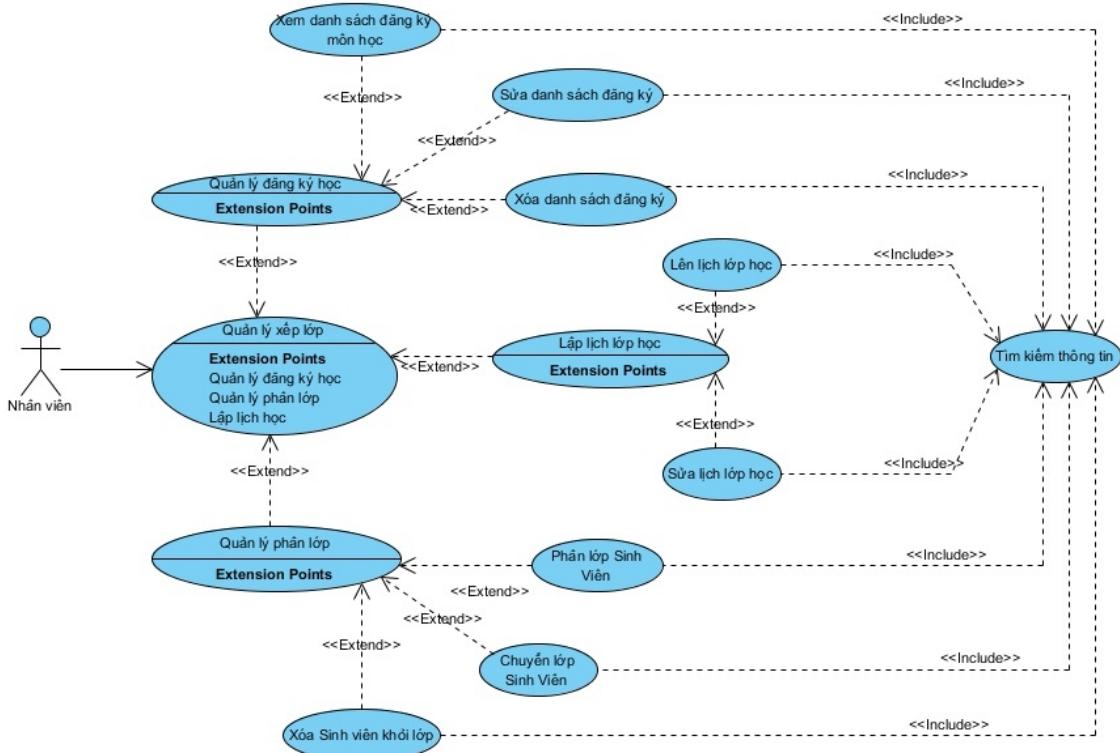
Phân tích mối quan hệ giữa các use case trong Hệ thống quản lý học tập theo tín chỉ:

- Specializes:** Có thể nhận thấy trong lược đồ use case tổng quát (Hình 5) use case Quản lý xếp lớp được phân rã thành các use case: Quản lý Đăng ký học, Quản lý phân lớp, Lập lịch lớp học trong lược đồ use case phân rã (Hình 6). Khi đó use case Quản lý xếp lớp là use case trùu tượng có nhiệm vụ chính là nhóm các use case Quản lý Đăng ký học, Quản lý phân lớp, Lập lịch lớp học thành một use case trùu tượng

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

trên lược đồ use case tổng quát. Khi phân rã use case, use case Quản lý xếp lớp đã được cụ thể hóa (Specializes).

- **Includes:** Use case Xem danh sách đăng ký môn học, Chuyển lớp sinh viên... trước khi thực hiện phải bao gồm thao tác tìm kiếm thông tin cần thao tác. Do vậy các use case đó Includes use case Tìm kiếm thông tin
- **Extends:** Các use case Đăng ký môn học, Xem thời khóa biểu, Xem điểm học tập... là mở rộng của use case Đăng nhập. Người dùng sẽ không thể thực hiện các chức năng đó nếu use case Đăng nhập chưa được thực hiện



Hình 6: Lược đồ phân rã use case Quản lý xếp lớp

4.5.6 Điều kiện trước (preconditions), điều kiện sau (postconditions) và kế thừa (inheritance)

Khi xem xét sự kế thừa giữa các use case, ta phải quan tâm đến việc cụ thể hóa có ảnh hưởng gì đến điều kiện trước và điều kiện sau không (mặc dù các use case chỉ kế thừa từ use case trừu tượng nhưng chúng vẫn có điều kiện trước và điều kiện sau). Sau đây là một số quy tắc:

- Khi một use case cụ thể hóa use case khác, nó kế thừa các điều kiện trước của use case cha như là một điều kiện bắt đầu. Chúng sẽ kết hợp với các điều kiện trước của use case con bởi phép “or”.

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

- Với các điều kiện sau, điểm bắt đầu của use case con là các điều kiện sau của use case cha. Các điều kiện sau do use case con thêm vào sẽ kết hợp với điều kiện sau của use case cha bằng phép “and”.
- Các điều kiện trước và sau được thêm vào bởi use case con sẽ không có ảnh hưởng đến các điều kiện trước và sau của use case cha.

Trong 3 quy tắc trên, quy tắc thứ 3 là hiển nhiên đối với những gì chúng ta đã biết về nguyên lý hướng đối tượng (con không ảnh hưởng đến hành vi của cha). Quy tắc thứ nhất và thứ hai cũng hàm ý rằng nếu use case cha có không có điều kiện trước, use case con cũng phải không có điều kiện trước. Nếu use case sau không có điều kiện sau (không có sự bảo đảm rõ ràng về đâu ra), khi đó use case con có thể có bất cứ điều kiện sau nào. Tóm lại, khi một use case cụ thể hóa use case khác, bạn phải xem xét cẩn thận các điều kiện trước và sau của use case cha.

4.5.7 Khảo sát Use case

Khảo sát Use case là một sự mô tả phi hình thức của một nhóm các use case khớp với nhau như thế nào. Đó là một kiểu tường thuật mà một nhà phát triển có thể đưa ra để dẫn dắt nhà đầu tư xuyên suốt qua một use case diagram. Một khảo sát use case cho phép các nhà đầu tư có sự hiểu thấu đáo hơn về các use case mà không cần bất kỳ sự giới thiệu nào từ nhà phát triển hệ thống.

Ví dụ: Khảo sát use case hệ thống quản lý học tập theo tín chỉ:

- Khi truy cập vào hệ thống đăng ký học theo tín chỉ, yêu cầu người dùng phải đăng nhập vào hệ thống (U1) trước khi có thể sử dụng các chức năng của hệ thống. Sau khi đăng nhập hệ thống, người dùng có thể sửa đổi mật khẩu (U10) tài khoản của mình
- Giảng viên sau khi đăng nhập hệ thống (U1) có thể Tìm kiếm thông tin (U8), xem chi tiết lịch giảng dạy của mình (U9), thay đổi thông tin cá nhân (U11) và nhận thông báo từ phòng đào tạo (U15)
- Sinh viên sau khi đăng nhập vào hệ thống (U1) có các tùy chọn sử dụng chức năng của hệ thống như Tìm kiếm thông tin (U8), Đăng ký môn học trong học kỳ (U3). Tùy vào các ràng buộc đối với môn học về các môn điều kiện, số tín chỉ tích lũy, cán bộ phòng đào tạo có thể xét sinh viên có đủ điều kiện học môn đó hay không và cập nhật lịch học cũng như lịch thi học kỳ (U12). Khi sinh viên không đủ điều kiện học môn nào đó, phòng đào tạo có thể gửi thông báo (U14) về các môn điều kiện còn thiếu tới sinh viên để sinh viên hoàn tất các môn học đó. Sinh viên có thể xem thông báo (U15), xem thời khóa biểu (U4) là lịch học các

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

môn học đã đăng ký mà sinh viên có thể theo học cũng như lịch thi học kỳ các môn đó.

- Phòng đào tạo có nhiệm vụ cập nhật bảng điểm (U13) đối với sinh viên, sinh viên có thể xem kết quả học tập của mình (U6) khi đăng nhập hệ thống (U1). Bên cạnh đó hệ thống còn có các tùy chọn cho sinh viên như In bảng điểm (U7) hay tìm kiếm thông tin (U8)
- Cán bộ Đào tạo sau khi đăng nhập vào hệ thống (U1) có thể cập nhật thông tin quản lý xếp lớp (U12), quản lý điểm sinh viên (U13) và gửi thông báo tới sinh viên (U14)

4.5.8 Các yêu cầu phụ

Đôi khi chúng ta phải kết hợp các yêu cầu phi chức năng với một use case cụ thể. Các yêu cầu phi chức năng mà không phù hợp với use case nào sẽ được ghi lại thành tài liệu về các use case phụ.

Ví dụ: Hệ thống đăng ký học theo tín chỉ

Các yêu cầu phụ

-
- S1. Hệ thống phải đáp ứng việc hàng trăm sinh viên cùng đăng ký các khóa học.
 - S2. Người dùng nên sử dụng trình duyệt FireFox 3.5 (hoặc mới hơn)
 - S3. Thông tin về chương trình đào tạo mỗi chuyên ngành nên hiển thị đầy đủ trên trang web hơn là dưới dạng một tệp tin download

...

4.5.9 Chi tiết của use case

UML không xác định chi tiết các use case nên bao gồm cái gì và chúng nên được sắp xếp như thế nào, một lựa chọn đưa ra ở đây dựa trên sự cảm nhận và kinh nghiệm. Định dạng khi viết chi tiết use case có thể gồm các thành phần:

- Số thứ tự và tên các use case
- Tác nhân chính: là tác nhân thực hiện use case
- Điều kiện trước: điều kiện đảm bảo use case được thực hiện
- Đảm bảo tối thiểu: điều kiện đảm bảo khi xảy ra dị thường (ngoại lệ)
- Điều kiện sau: điều kiện sau khi thực hiện xong use case
- Chuỗi sự kiện chính

- Các đường đi dị thường (ngoại lệ)

Ví dụ: Use case Đăng nhập

Tên Use Case	Đăng nhập
Tác nhân chính	Người dùng hệ thống
Điều kiện trước	Người dùng đã có tài khoản để đăng nhập hệ thống
Đảm bảo tối thiểu	Hệ thống cho phép người dùng đăng nhập lại
Điều kiện sau	Người dùng đăng nhập được vào hệ thống
Chuỗi sự kiện chính	<ol style="list-style-type: none"> Người dùng chọn chức năng đăng nhập trên giao diện chính của hệ thống Hệ thống hiển thị form đăng nhập Người dùng nhập tài khoản và mật khẩu của mình Hệ thống kiểm tra tính hợp lệ của tài khoản và mật khẩu Hệ thống hiển thị giao diện chính tương ứng với actor
Ngoại lệ:	<ol style="list-style-type: none"> Người dùng nhập tài khoản và mật khẩu sai <ol style="list-style-type: none"> Hệ thống thông báo lỗi và yêu cầu nhập lại Tài khoản người dùng đăng nhập không tồn tại <ol style="list-style-type: none"> Hệ thống thông báo lỗi và yêu cầu nhập lại

Use case hệ thống chi tiết hơn nhiều so với use case nghiệp vụ chúng ta đã thảo luận trước đó. Điều này phản ánh thực tế rằng chúng ta hiện đang cố gắng phát triển hệ thống một cách chi tiết hơn, chúng ta muốn xác định chính xác về dịch vụ của hệ thống sẽ cung cấp theo trật tự để xóa đi những sự phỏng đoán về phân tích và thiết kế hệ thống.

Khi viết chi tiết use case, điều quan trọng là chúng ta xác định chức năng của hệ thống nhưng không phải cách chức năng ấy được thể hiện. Ví dụ nếu chúng ta muốn khai báo các bước như “Sinh viên lựa chọn button Chi tiết môn học”, chúng ta nên hạn chế trong thiết kế giao diện người dùng. Trừ phi đó là một yêu cầu bắt buộc, chúng ta nên cố gắng luôn sử dụng các từ ngữ trung lập như Chọn, Thêm, Xóa, Hiển thị...

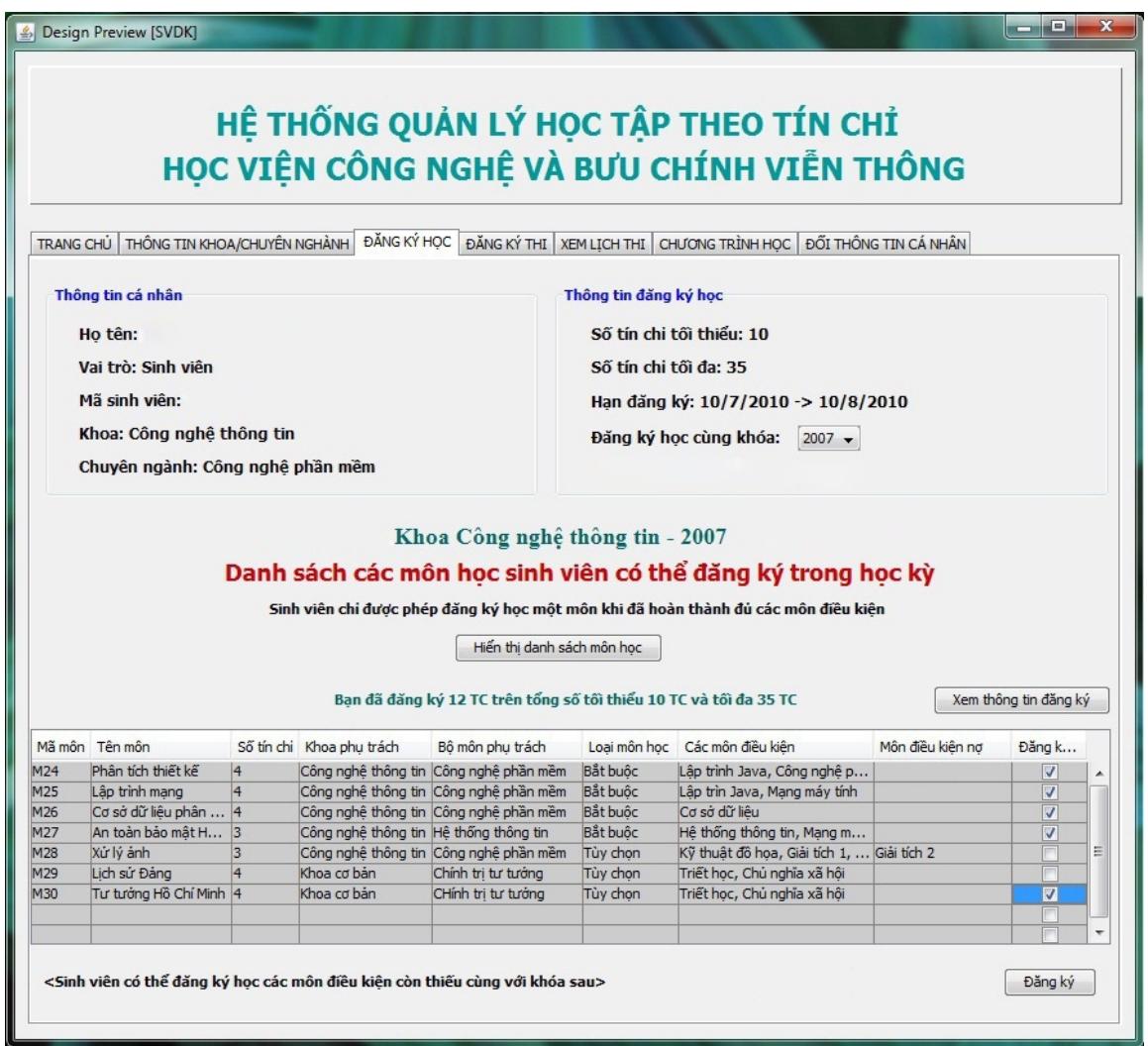
4.5.10 Bản phác họa giao diện người dùng

Suy nghĩ về giao diện người dùng của hệ thống có thể giúp ta tìm ra các use case. Các giao diện có thể được phác họa bởi các nhà tài trợ/khách hàng ở giai đoạn đầu và kết quả được lưu lại thành các bản phác họa giao diện người dùng (user interface sketches). Các use case và giao diện người dùng đều giới thiệu đến những phần trong chức năng của hệ thống, một ý tưởng tốt khi chúng ta duy trì ánh xạ giữa chi tiết các use case và giao diện người dùng một cách rõ ràng, và ánh xạ đó sẽ tồn tại trong suốt quá trình thực thi. Trong mỗi giao diện người dùng, chúng ta nên cung cấp một cửa sổ tương ứng với mỗi use case.

Ví dụ1: Kịch bản Sinh Viên đăng ký môn học

Tên Use Case	Sinh viên đăng ký môn học
Tác nhân chính	Sinh viên
Điều kiện trước	Người dùng đã có tài khoản để đăng nhập hệ thống
Đảm bảo tối thiểu	Hệ thống cho phép sinh viên đăng ký lại

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

Điều kiện sau	SV đăng ký thành công các môn học đủ điều kiện
Chuỗi sự kiện chính	
<ol style="list-style-type: none"> Người dùng chọn chức năng đăng ký học trên giao diện chính của hệ thống Hệ thống hiển thị form đăng ký học cho sinh viên với các thông tin cần thiết Sinh viên chọn khóa học muốn đăng ký học cùng (có thể đăng ký học cùng một số môn học với khóa trước nếu sinh viên phải học lại môn đó) và chọn “Hiển thị danh sách môn học” Hệ thống hiển thị danh sách các môn học mà sinh viên có thể đăng ký học tương ứng với mỗi khóa học cùng Sinh viên lựa chọn các môn học mình muốn đăng ký học và chọn “Đăng ký” Hệ thống kiểm tra thông tin về các môn điều kiện của sinh viên và hiển thị thông báo “Đăng ký môn học thành công” 	
Ngoại lệ:	
<ol style="list-style-type: none"> Sinh viên đăng ký môn học trong khi chưa học đủ các môn điều kiện hoặc còn nợ môn từ các kỳ trước <ol style="list-style-type: none"> Hệ thống thông báo sinh viên không đủ điều kiện học và yêu cầu sinh viên đăng ký lại. 	
 <p>The screenshot shows a Windows application window titled 'Design Preview [SVDK]'. The main title is 'HỆ THỐNG QUẢN LÝ HỌC TẬP THEO TÍN CHỈ HỌC VIỆN CÔNG NGHỆ VÀ BƯU CHÍNH VIỄN THÔNG'. The interface includes a navigation bar with links like TRANG CHỦ, THÔNG TIN KHOA/CHUYÊN NGHÀNH, ĐĂNG KÝ HỌC, ĐĂNG KÝ THI, XEM LỊCH THI, CHƯƠNG TRÌNH HỌC, and ĐỔI THÔNG TIN CÁ NHÂN. Below the navigation bar, there are two sections: 'Thông tin cá nhân' and 'Thông tin đăng ký học'. The 'Thông tin cá nhân' section contains fields for Họ tên (Name), Vai trò (Role: Sinh viên), Mã sinh viên (Student ID), Khoa (Faculty: Công nghệ thông tin), and Chuyên ngành (Major: Công nghệ phần mềm). The 'Thông tin đăng ký học' section contains fields for Số tín chỉ tối thiểu (Minimum credits: 10), Số tín chỉ tối đa (Maximum credits: 35), Hạn đăng ký (Registration deadline: 10/7/2010 -> 10/8/2010), and Đăng ký học cùng khóa (Register together with major: 2007). Below these sections, there is a red header 'Khoa Công nghệ thông tin - 2007' and a red sub-header 'Danh sách các môn học sinh viên có thể đăng ký trong học kỳ'. A note below states 'Sinh viên chỉ được phép đăng ký học một môn khi đã hoàn thành đủ các môn điều kiện'. A button labeled 'Hiển thị danh sách môn học' is present. At the bottom, there is a table titled 'Bạn đã đăng ký 12 TC trên tổng số tối thiểu 10 TC và tối đa 35 TC' with columns for Mã môn (Subject ID), Tên môn (Subject Name), Số tín chỉ (Credits), Khoa phụ trách (Responsible faculty), Bộ môn phụ trách (Responsible department), Loại môn học (Type of subject), Các môn điều kiện (Prerequisites), Môn điều kiện nợ (Mandatory subjects), and Đăng k... (Registration status). The table lists subjects such as M24 (Phân tích thiết kế), M25 (Lập trình mạng), M26 (Cơ sở dữ liệu phân..., etc.). A 'Xem thông tin đăng ký' (View registration information) button is located at the top right of the table area. At the very bottom, there is a note '<Sinh viên có thể đăng ký học các môn điều kiện còn thiếu cùng với khóa sau>' and a 'Đăng ký' (Register) button.</p>	

Hình 7: Giao diện Sinh viên Đăng ký môn học**4.5.10 Xếp mức độ ưu tiên các use case hệ thống**

Sắp xếp các yêu cầu hệ thống theo thứ tự ưu tiên về mặt cài đặt là một ý kiến tốt đặc biệt là trong hoàn cảnh tiến trình phát triển tăng. Sắp xếp thứ tự các use case nghĩa là gắn cho mỗi use case một điểm số và được dùng trong việc lập kế hoạch phát triển và tăng trong tương lai.

Một kỹ thuật xếp ưu tiên là dùng các màu đèn giao thông.

- Use case có mức độ ưu tiên màu xanh phải được cài đặt trong giai đoạn hiện tại.
- Use case có mức độ ưu tiên màu vàng thì không bắt buộc phải được cài đặt trong giai đoạn hiện tại và chỉ được cài đặt khi use case màu xanh đã hoàn thành.
- Use case có mức độ ưu tiên màu đỏ không được cài đặt trong giai đoạn hiện tại.

Trong thực tế, độ ưu tiên của các use case không chỉ phụ thuộc vào tính mong muốn, mà còn phụ thuộc vào kiến trúc hệ thống và việc code use case trong giai đoạn hiện tại. Việc lựa chọn độ ưu tiên cho các use case đòi hỏi nhiều kỹ năng và kinh nghiệm. Cách tốt nhất ta nên đặt những use case dễ dàng hơn đầu tiên bởi chúng sẽ giúp cho chúng ta học thêm nhiều hơn về hệ thống như việc nhắc lại, và tất nhiên sẽ ít rủi ro hơn.

Nếu bạn có thêm thời gian tại thời điểm cuối của giai đoạn hiện tại (sau khi đã kết thúc những use case màu xanh và màu vàng), bạn nên xem lại trạng thái dự án cũng như hoàn thành kế hoạch cho giai đoạn sắp tới (chẳng hạn sắp xếp lại ưu tiên cho những use case chưa được thực thi)

Ví dụ: Hệ thống học theo tín chỉ

- **Green:**

- U1: Đăng nhập
- U3: Đăng ký khóa học
- U4: Xem thời khóa biểu
- U5: Xem lịch thi
- U8: Tìm kiếm thông tin
- U9: Xem lịch giảng dạy
- U14: Gửi thông báo

- **Amber:**

- U2: Thoát
- U11: Thay đổi thông tin cá nhân
- U12: Quản lý xếp lớp
- U13: Quản lý điểm

CHƯƠNG 4. XÁC ĐỊNH YÊU CẦU

- U15: Xem thông báo

- **Red:**

- U6: Xem điểm học tập
- U7: In bảng điểm
- U10: Đổi mật khẩu

4.6 TỔNG KẾT

Trong chương này chúng ta đã đề cập tới

Sự quan trọng của quá trình xác định yêu cầu chức năng (cái mà hệ thống có thể làm) và yêu cầu phi chức năng (Hệ thống vận hành như thế nào, ví dụ như xác định trình duyệt web mà phải được hỗ trợ) trong pha yêu cầu trước khi bắt đầu code

Mô hình hóa ngữ cảnh nghiệp vụ và chức năng hệ thống sử dụng use case nghiệp vụ và các actor xác định

Mô hình hóa yêu cầu hệ thống với mô hình use case hoàn tất gồm có các use case, use case diagram, yêu cầu phi chức năng, phác thảo giao diện người dùng, ưu tiên use case. Mặc dù biểu đồ giao tiếp và biểu đồ hoạt động có thể được cân nhắc tùy chọn trong giai đoạn này, nhưng một mô tả về các thuật ngữ luôn luôn cần thiết

Chương này tập trung một số khía cạnh về thu thập yêu cầu và tất cả nội dung trình bày ở đây đều tuân theo quy trình nào đó. Với mỗi phương pháp luận khác nhau có các tài liệu tương ứng với từng giai đoạn phát triển phần mềm mà phương pháp đó quy định. Vì vậy, khi phát triển phần mềm chúng ta không bắt buộc phải tuân theo đúng các bước như trên.

BÀI TẬP

1. Sinh viên đọc Case study trình bày trong Chương 6 của tài liệu [1] và so sánh danh sách các use case nghiệp vụ và use case hệ thống.
2. Sinh viên làm việc theo nhóm để xây dựng các yêu cầu cho Hệ thống quản lý học theo tín chỉ/Quản lý Thư viện điện tử từ quan điểm nghiệp vụ và hệ thống.
3. Sử dụng Tool VP để hoàn thiện bài tập của mình.

CHƯƠNG 5 PHÂN TÍCH YÊU CẦU

5.1 GIỚI THIỆU

Phân tích một dự án phần mềm là nhằm trả lời câu hỏi hệ thống làm những gì, nhưng không phải chỉ ra cách thức làm những việc đó. Chúng ta cần phải phân rã tập các yêu cầu phức tạp thành các yếu tố chính và mối quan hệ giữa chúng để tìm ra lời giải cho các vấn đề có thể gặp phải. Những công cụ và công nghệ nào cần được sử dụng? Lập lịch cho những mốc thời điểm của dự án, ước lượng ngân sách dự án, những rủi ro có thể gặp phải... Phân tích là giai đoạn đầu tiên giúp ta hiểu được mô hình thực sự của hệ thống.

Một mô hình phân tích thường bao gồm 2 phần: *Phân tích động* và *phân tích tĩnh*. Chúng ta có thể mô tả mô hình phân tích tĩnh bằng Biểu đồ lớp (Class diagram). Mỗi biểu đồ lớp chỉ ra các đối tượng mà hệ thống có thể thao tác và cách thức chúng tác động với nhau. Mô hình phân tích động sử dụng Biểu đồ giao tiếp (communication diagram) để giải thích tính khả thi của mô hình tĩnh. Như trước đây, thay vì sử dụng tất cả những biểu đồ UML phức tạp, chúng ta chỉ cần nhìn hai phần chính ở đây là có thể đạt được mục đích chính trong phát triển. Có 2 phần đầu vào cho phân tích:

- *Mô hình yêu cầu nghiệp vụ* (Business requirements model): Chứa các mô tả của thao tác bằng tay và tự động hóa luồng công việc của phạm vi nghiệp vụ, diễn tả cách thức sử dụng phiên bản hướng nghiệp vụ của actors, usecase, đối tượng, biểu đồ giao tiếp và hoạt động.
- *Mô hình yêu cầu hệ thống* (System requirements model): Thể hiện cái nhìn tổng quát bên ngoài hệ thống, diễn tả như là phiên bản hướng hệ thống của actors, usecases và biểu đồ usecase, toàn cảnh giao diện người dùng, một tự diễn từ vựng đầy đủ hơn và không hướng chức năng.

Những đầu vào cần chuyển đổi thành mô hình của đối tượng sẽ được xử lý bằng triển khai hệ thống, cùng với các thuộc tính và các mối quan hệ giữa chúng. Những đối tượng này sẽ tồn tại bên trong hệ thống của nó hoặc trong lớp biên của hệ thống, có thể truy nhập thông qua một hoặc nhiều giao diện. Hầu hết các đối tượng mà chúng ta thấy được ở giai đoạn này sẽ tương ứng với các đối tượng vật lý hay khái niệm thế giới thực. Khi chúng ta mô hình hệ thống với các đối tượng, chúng ta sẽ đặt chúng thông qua một quá trình xác thực để hỗ trợ một giải pháp.

Tại sao lại phân tích?

Có thể bạn sẽ đặt ra một câu hỏi là tại sao chúng ta cần phải thực hiện phân tích cho các dự án phần mềm trong khi chúng ta là người viết code.

Chúng ta hãy hình dung một ví dụ tương tự như sau. Khi bạn muốn xây một ngôi nhà, bạn cần có một kế hoạch trước khi bạn thực hiện những nhát cuốc đầu tiên. Thông

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

thường, bạn muốn biết làm thế nào nhà sau này giống như ý định mong muốn và đó là lý do tại sao bạn lại thuê kiến trúc sư và kỹ sư.

Phân tích là một phần trong quá trình xây dựng và quản lý dự án. Phân tích dự án giúp ta nắm được các yêu cầu của dự án, xác định được các đối tượng, mối quan hệ giữa chúng. Chúng ta không thể mong đợi có được một sự hiểu biết đầy đủ các vấn đề chỉ từ mô hình yêu cầu nghiệp vụ. Chúng ta không thể tách rời các thao tác công việc tay chân đơn thuần và tự động hóa riêng rẽ trong một luồng công việc. Ví dụ như trong một mô hình xe hơi, nó bao gồm sự tương tác của con người với con người và con người với máy tính. Ngay cả khi chúng ta có một mô hình hệ thống về một trường hợp sử dụng (use case) thì chúng ta cũng có thể chưa hiểu về vấn đề này chắc chắn và đầy đủ. Bởi vì các trường hợp này tập trung vào các thao tác bên ngoài. Use cases xác định sự tương tác giữa các actors và các biến của hệ thống, các hệ thống chính được coi như là một hộp đen chỉ nhìn thấy từ bên ngoài. Mô hình yêu cầu nghiệp vụ (Business requirements modeling) và mô hình yêu cầu hệ thống (Business requirements modeling) vẫn cần phải được hoàn thiện.

Sau khi hoàn thành quá trình phân tích tĩnh, đối tác sẽ xác nhận sự hiểu biết của chúng ta về đối tượng nghiệp vụ là chính xác, trước khi chúng ta tự do thực hiện thiết kế đối tượng. Sau khi phân tích động, chúng ta sẽ có thể xác định được những đối tượng phân tích nào hỗ trợ chức năng hệ thống yêu cầu. Tài liệu này dựa trên mô hình Ripple do O'Docherty đề xuất và khi đó phân tích bao gồm những bước sau đây:

1. Sử dụng mô hình yêu cầu hệ thống để tìm lớp ứng cử mô tả những đối tượng có thể tác động tới hệ thống và biểu diễn chúng bằng biểu đồ lớp.
2. Tìm quan hệ giữa các lớp: kết hợp, kế thừa, tập hợp, thành phần.
3. Tìm thuộc tính cho các lớp: kiểu, tên thuộc tính của lớp.
4. Xem xét use case hệ thống, kiểm tra xem chúng có được hỗ trợ bởi các đối tượng mà chúng ta có hay không, bổ sung các thuộc tính và mối quan hệ giữa chúng và còn các phương thức sẽ được bổ sung khi ta đi đến hiện thực hóa use case.
5. Cập nhật bộ từ vựng và những yêu cầu phi chức năng khi cần thiết, bản thân use case không cần cập nhật mặc dù chúng cũng có thể cần hiệu chỉnh.

Những thao tác được khám phá trong suốt quá trình hiện thực hóa use case sẽ bị bỏ qua trong quá trình thiết kế, nên trong giai đoạn này, chúng ta cố gắng xây dựng lòng tin mà không thiết kế giải pháp. Bạn cần chỉ ra biểu đồ lớp cùng với những thuộc tính của nó cho những người tài trợ của bạn để tìm ra lỗi (những người này hiểu biết về nghiệp vụ hơn bạn). Một thành viên của nhóm nên tổng kết lại những thông tin thể hiện trên biểu đồ lớp để khách hàng có thể xem. Trên thực tế biểu đồ lớp là tương đối dễ dàng để những người không phải là lập trình viên có thể hiểu và điều này sẽ giúp đưa ra những bình luận hữu ích. Quyết định khi nào trình bày biểu đồ lớp tới khách hàng là do bạn và nhóm của bạn, nhưng bạn thường nên làm điều đó khi bạn mắc lỗi và khi bạn cần sửa lỗi đó. Nói

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

chung, việc chỉ ra các biểu đồ giao tiếp và phương thức của đối tượng cho khách hàng không phải là một ý tưởng tốt vì:

- Nó phức tạp và không cần thiết vì bạn vừa giải thích các hành vi động với các usecases hệ thống.
- Chúng bao gồm các đoạn mã mà những người không lập trình thường bị “dị ứng”
- Chúng sẽ bị loại bỏ trước khi thiết kế.

Một số khách hàng, giống như các nhà quản lý kỹ thuật, có thể thích được chỉ ra một vài phân tích động để tăng thêm sự tin cậy của họ. Trong phần còn lại của chương này, chúng ta sẽ xem xét chi tiết các phân tích động và tĩnh tương ứng.

5.2 PHÂN TÍCH TĨNH

Mô hình tĩnh liên quan đến việc phân chia các thành phần logic và vật lý của hệ thống và cách mà chúng kết nối cùng nhau. Nói chung, nó miêu tả cách chúng ta xây dựng (construct) và khởi tạo (initialize) hệ thống như thế nào.

5.2.1. Xác định các lớp

Hầu như không có một công thức chung cho việc phát hiện ra các lớp. Để tìm các lớp là một công việc đòi hỏi trí sáng tạo và cần phải được thực thi với sự trợ giúp của chuyên gia ứng dụng. Vì qui trình phân tích và thiết kế mang tính vòng lặp, nên danh sách các lớp sẽ thay đổi theo thời gian. Tập hợp ban đầu của các lớp tìm ra chưa chắc đã là tập hợp cuối cùng của các lớp sau này sẽ được thực thi và biến đổi thành code. Vì thế, thường người ta hay sử dụng đến khái niệm các *lớp ứng viên* (Candidate Class) để miêu tả tập hợp những lớp đầu tiên được tìm ra cho hệ thống.

Lớp ứng viên thường được chỉ ra bởi các *danh từ* trong các use case. Các danh từ đó mô tả:

- Ngay chính hệ thống , ví dụ “system” hay “iCoot”, với chúng ta thì hệ thống chỉ là một ranh giới cho những nỗ lực phát triển
- Actors, ví dụ “Student” hoặc “Instructor”, trừ trường hợp chúng ta cần lưu trữ thông tin về một actor nội tại như là với Member, chúng ta cần lưu password.
- Boundaries, ví dụ “member applet” hay “head office link”, chúng ta cố gắng nhận dạng các đối tượng liên quan đến nghiệp vụ cùng với các thông tin ưa thích và hành vi. Boundaries là một phần riêng của phần mềm cho phép các tác nhân lấy được các đối tượng.
- Kiểu tầm thường, các kiểu ít quan trọng (ví dụ, String và int); thông thường nó được cung cấp bởi ngôn ngữ được cài đặt hoặc từ thư viện của nó.

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

5.2.2 Quan hệ giữa các lớp

Với một danh sách các lớp ứng viên chúng ta có gắng vẽ các quan hệ giữa chúng. Có 4 loại quan hệ chính sau:

- Ké thừa (Inheritance): một lớp con sẽ kế thừa tất cả các thuộc tính và hành vi của lớp cha của nó.
- Association: các đối tượng của một lớp được liên kết với các đối tượng của lớp khác.
- Aggregation: một loại association mạnh- một lớp được tạo ra bởi một lớp khác.
- Composition: quan hệ aggregation mạnh- đối tượng tổng hợp không thể được dùng chung với các đối tượng khác và nó sẽ mất cùng với thành phần của nó.

Ké thừa là một kiểu quan hệ khác với ba loại kia; kế thừa mô tả một quan hệ trong thời gian biên dịch giữa các lớp trong khi các quan hệ kia mô tả kết nối trong thời gian chạy giữa các đối tượng. Theo chuẩn UML, tất cả các quan hệ trong thời gian chạy đều hình thành từ association. Tuy nhiên, nhiều người sử dụng thuật ngữ “association” với nghĩa là một association thực sự không phải là aggregation hay composition. Lựa chọn quan hệ nào đòi hỏi phải khéo léo, nghĩa là bạn cần sử dụng khả năng trực giác, kinh nghiệm và sự phỏng đoán. Trong quá trình phân tích bạn nên hiểu rõ về tần số xuất hiện của các loại quan hệ:

Association > aggregation > inheritance > composition

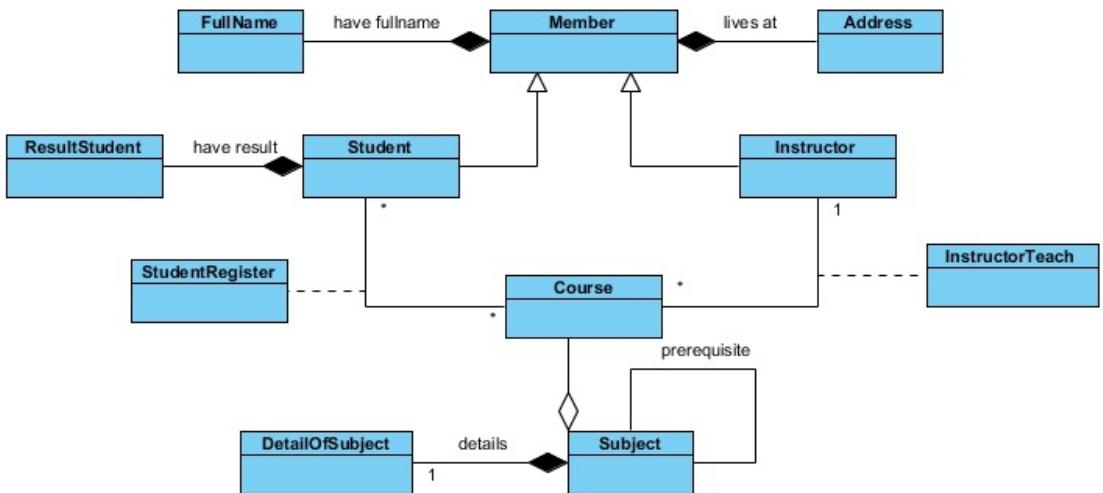
Trong thiết kế và cài đặt thì sự khác nhau giữa association, aggregation và composition có thể khó phát hiện.

5.2.3 Thể hiện biểu đồ lớp và biểu đồ đối tượng

Biểu đồ lớp cho ta thấy những lớp nào tồn tại và quan hệ giữa chúng. (biểu đồ lớp cũng có thể biểu diễn các thuộc tính và hành động, nhưng điều đó đòi hỏi nhiều không gian hơn).

Hình 5.1 biểu diễn một biểu đồ lớp UML cho Hệ thống quản lý học tập theo tín chỉ. Mỗi lớp đều được thể hiện trong một hình chữ nhật với tên lớp bên trong (in đậm). Nếu lớp thuộc loại trừu tượng, tên lớp in nghiêng hay có thể thêm từ khóa *{abstract}* bên trên hoặc bên phải lớp thay cho chữ in nghiêng.

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

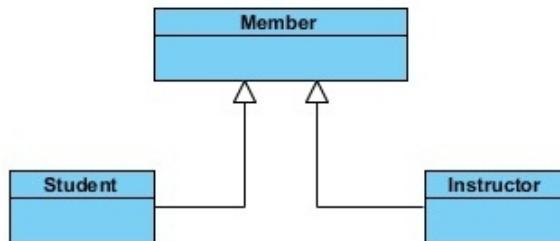


Hình 5.1: Biểu đồ Lớp

Quan hệ giữa các lớp được thể hiện bằng các đường với các chú thích khác nhau. Mặc dù không có kiến thức chuyên về UML, ta cũng có thể dễ dàng lấy ra thông tin từ biểu đồ lớp, chỉ từ các văn bản. Ví dụ, chúng ta có thể thấy rằng “một thành viên có thể có Họ tên và Địa chỉ”, “Mỗi sinh viên sẽ có kết quả học tập của mình”…

5.2.4 Biểu diễn các quan hệ

Xét một số quan hệ trong hệ thống quản lý học tập theo tín chỉ với cách biểu diễn cụ thể như sau:

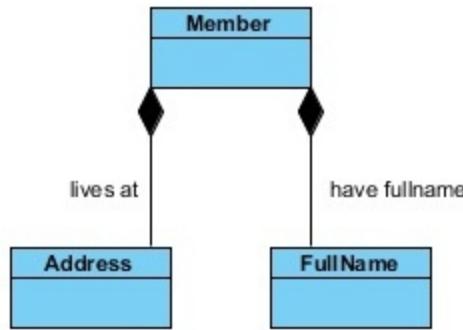


Hình 5.2: Quan hệ kế thừa trong UML

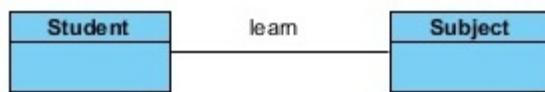


Hình 5.3: Quan hệ aggregation trong UML

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



Hình 5.4: Quan hệ composition trong UML



Hình 5.5: Quan hệ association trong UML

- Quan hệ kế thừa được mô tả trên biểu đồ lớp bởi một đường với đầu mũi tên đặc màu trắng được vẽ từ lớp con hướng tới lớp cha (lớp Student và Instructor là kế thừa lớp Member)
- Quan hệ aggregation trong UML được mô tả bởi một đường nối giữa hai lớp với một đầu có hình oval trắng ở cuối quan hệ (trên lớp tổng thể). Ví dụ, Course là một phần của Subject
- Quan hệ composition trong UML tương tự như quan hệ aggregation, nhưng đầu hình quả trám là màu đen ở cuối, ví dụ FullName và Address luôn là một phần của Member.
- Quan hệ association được mô tả bởi một đường thẳng nối giữa hai lớp có sự liên quan về ngữ nghĩa. Ngữ nghĩa đó có thể được mô tả trên đường nối các quan hệ. Ở đây Sinh viên có quan hệ association với Môn học, Sinh viên sẽ học môn học nào đó. Rõ ràng Môn Học không phải là một thành phần của Sinh viên và ngược lại

Khi triển khai mô hình lớp phân tích, để cho đơn giản nên tạo thông tin theo một chiều.

Multiplicity

Tất cả các quan hệ ngoại trừ kế thừa đều có thể được thể hiện tại cuối liên kết số đối tượng mà được phép tham gia vào mỗi quan hệ.

n: lấy chính xác n

m...n: số nào trong dải từ m đến n

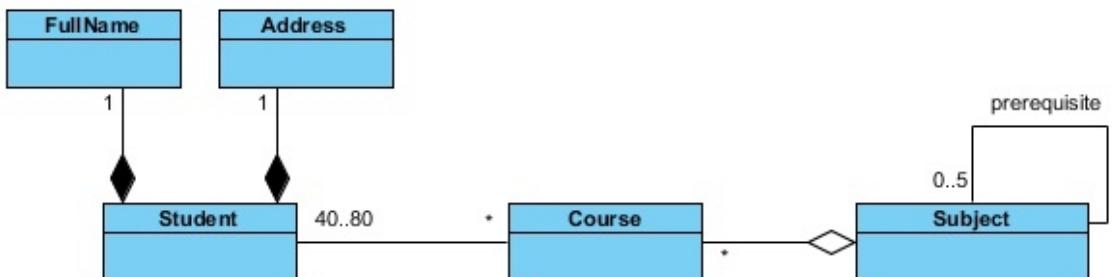
CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

p...* : số nào trong dải từ p đến vô cùng

* : viết nhanh của 0...*

0...1: tùy chọn

Với composition, multiplicity tại cuối composer luôn luôn là 1.



Hình 5.6: Mô tả multiplicity trong UML

Một Student có duy nhất một FullName và một Address

Một Student có thể tham gia học nhiều Course

Một Course chỉ được học bởi tối thiểu 40 Student hoặc tối đa 80 Student;

Một Course sẽ thuộc về một Subject nhất định

Một Subject có thể có nhiều Course (Course 3 đvht hay 4 đvht)

Một Subject có thể không có môn điều kiện nào (Prerequisite Subject)

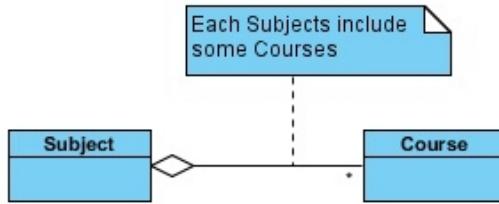
Một Subject có thể có tối đa 5 môn học điều kiện

Quan hệ giữa FullName và Address với Student là composition, FullName và Address là thành phần của Student. Cũng tương tự trong quan hệ giữa Course và Subject là aggregation, course là thành phần của Subject. Sự khác nhau giữa composition và aggregation có thể thấy ở đây là, lớp FullName và Address là thành phần của Student nhưng nó không thể tồn tại độc lập mà không có Student. Sự tồn tại của nó phụ thuộc vào sự tồn tại của lớp Student. Trong khi Course là thành phần của Subject, nhưng sự tồn tại của Subject không quy định sự tồn tại của Course

Association Labels, Roles and comments (các nhãn liên kết, vai trò và lời giải thích cho liên kết)

Tất cả các quan hệ, ngoại trừ ké thừa, đều có thể sinh ra một nhãn liên kết, chỉ ra bản chất của liên kết. Nếu nhãn liên kết không rõ ràng thì một đầu mũi tên đen có thể được sử dụng.

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



Hình 5.7: Các nhãn liên kết, vai trò và giải thích cho liên kết trong UML

Trong hình trên, ta có thể thấy rằng Course là một phần của Subject, một Subject sẽ bao gồm nhiều Course như là thành phần của nó. Cũng như tên liên kết, chúng ta có thể biểu diễn các vai trò (role). Một role mô tả vai trò của đối tượng trong liên kết. nó được biểu diễn gần đối tượng thể hiện vai trò đó.

Theo nguyên tắc, tên liên kết và vai trò liên kết có thể được kết hợp trên cùng một liên kết nhưng nên cân nhắc thay thế để tránh nhầm lẫn lộn xộn. Comment, một phần tùy ý dạng text được đặt trong một biểu tượng trống như trang giấy, dùng để cung cấp thông tin về cái mà nó chú thích.

5.2.5 Thuộc tính

Một thuộc tính là một đặc tính của đối tượng, như là kích cỡ, vị trí, tên, giá , tỉ lệ, hay bất kì điều gì. Trong UML, mỗi thuộc tính có thể được đưa ra cùng kiểu của nó. Kiểu được biểu diễn sau dấu ":" và đặt sau tên thuộc tính.

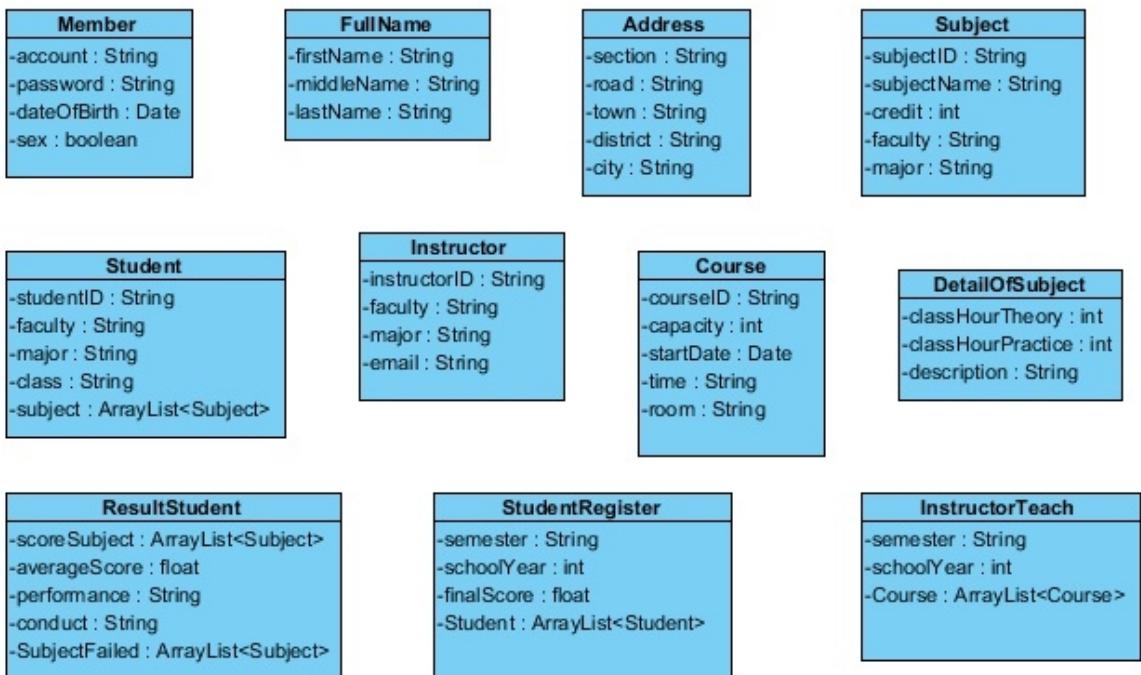


Hình 5.8: Biểu diễn một lớp cơ bản trong UML

Thuộc tính có thể được hiển thị trong biểu đồ lớp bằng cách thêm một ngăn ở dưới tên lớp. Để tiết kiệm không gian, chúng ta có thể viết tài liệu riêng biệt cho chúng thay vì một danh sách thuộc tính, hoàn chỉnh với các mô tả. Nếu chúng ta sử dụng công cụ phát triển phần mềm, chúng ta hy vọng có thể phóng to để nhìn các thuộc tính (và mô tả của chúng) hoặc thu nhỏ để chỉ nhìn tên lớp. Nếu bạn không thể cung cấp các mô tả ngắn gọn cho thuộc tính ở giai đoạn này, có lẽ cần một vài thuộc tính hoặc thậm chí là một lớp riêng của nó.

Hình 5.8 biểu diễn các thuộc tính của một lớp Member trong Hệ thống quản lý học tập theo tín chỉ với các thuộc tính Account, password, dateOfBirth, sex cùng với kiểu của các thuộc tính đó (String, date, Boolean...)

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



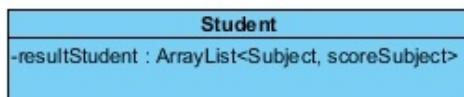
Hình 5.9 Biểu diễn đầy đủ tập các thuộc tính cho phân tích

Thuộc tính hay quan hệ

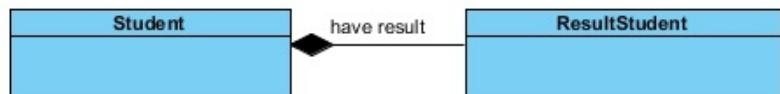
Thông thường chúng ta phải tạo những lựa chọn khác nhau cho mô hình hóa thông tin. Ví dụ, bạn chọn mô hình kết quả học tập của sinh viên như thế nào để phù hợp nhất với hệ thống. Hình 5.10 thể hiện 3 cách khác nhau:

- Thêm một thuộc tính tới Student gọi là resultStudent với kiểu mảng chứa danh sách môn học và kết quả môn học đó của sinh viên
- Quan hệ composition giữa Student và ResultStudent
- Chỉ ra quan hệ aggregation giữa Student và ResultStudent

Lựa chọn 1

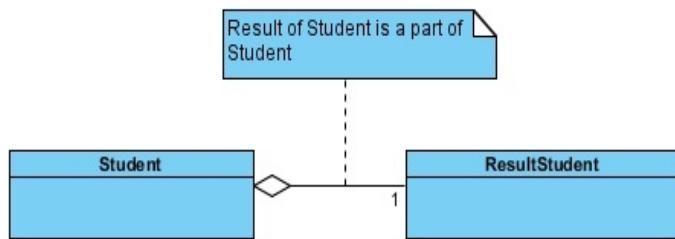


Lựa chọn 2



CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Lựa chọn 3



Hình 5.10: Lựa chọn giữa thuộc tính hay quan hệ

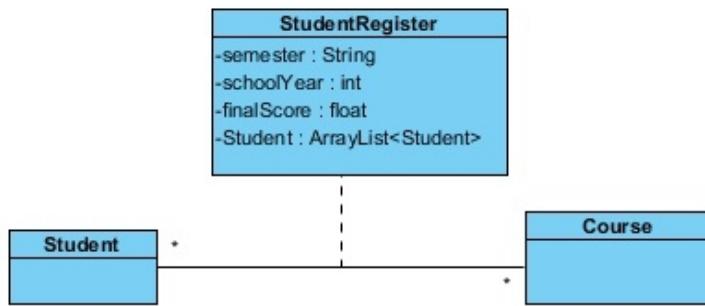
Trong các mô hình này, mô hình nào sẽ được chọn phù hợp với ngữ cảnh nhất?

- **Lựa chọn 1** có thể dẫn đến xây dựng một cơ sở dữ liệu phức tạp cho sinh viên.
- **Lựa chọn 2** khá hợp lý khi ResultStudent là một phần của Sinh viên và sự tồn tại của lớp Student điều khiển sự tồn tại của ResultStudent, ResultStudent chỉ tồn tại khi Student tồn tại
- **Lựa chọn 3** ResultStudent vẫn là một thành phần của Student nhưng dường như không đúng đắn khi cho rằng ResultStudent có thể tồn tại độc lập với Student

Nói chung lựa chọn 2 là tốt nhất trong ngữ cảnh quản lý học tập theo tín chỉ. Một lưu ý khi phân tích cần phải lựa chọn những trường hợp biểu diễn thỏa mãn với hoàn cảnh thực tại nhất nhưng không bao giờ có câu trả lời chính xác. Lời khuyên tốt nhất là không nên lo lắng về mặt triết lý quá nhiều. Thay vì đó, sử dụng những phán đoán chung, những kinh nghiệm, trực giác, lặp lại và mở rộng cho đến khi thực hiện thành công.

5.2.6 Lớp liên kết (association classes)

Một lớp có thể được đính kèm theo một liên kết, trong trường hợp này nó sẽ được gọi là một lớp liên kết. Một lớp liên kết không được nối tới bất kỳ lớp nào của mối quan hệ mà nối tới chính bản thân mối quan hệ. Cũng giống như một lớp bình thường, lớp liên kết có thể có thuộc tính, phương thức và quan hệ khác. Lớp liên kết được sử dụng để bổ sung thêm thông tin cho kết nối. Mỗi kết nối của liên kết gắn liền với một đối tượng của lớp liên kết. Hình 5.11 chỉ ra rằng một lớp Student có thể được kết nối với một số bất kỳ đối tượng Course và ngược lại.



Hình 5.11: Lớp liên kết

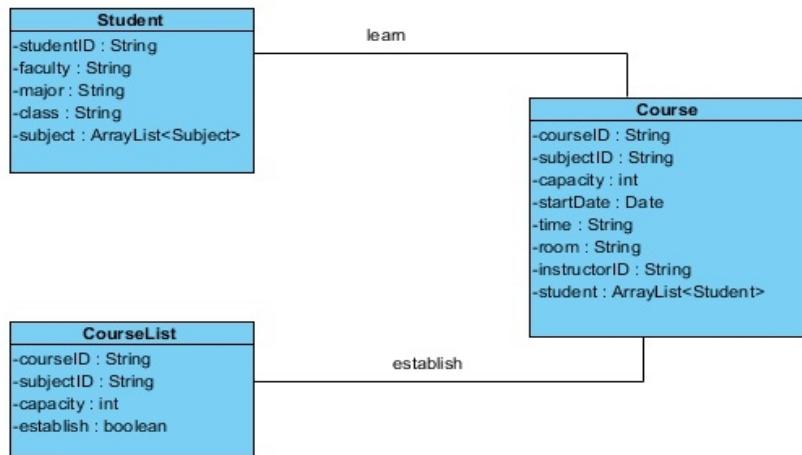
Một lớp liên kết mô tả các thuộc tính và hành động của thực thể và chỉ tồn tại khi sự liên kết tồn tại: các thuộc tính và các hành động không liên quan đến các đối tượng của sự liên kết. Trong ví dụ trên, khi Student đăng ký học một Course, một liên kết mới được tạo trong thời gian chạy giữa Student và Course tương ứng. Khi thiết kế, chúng ta sẽ thay thế các lớp liên kết bởi cái gì cụ thể hơn, bởi vì chúng không hỗ trợ trực tiếp trong phần lớp các ngôn ngữ lập trình. Tuy nhiên chúng rất hữu ích trong quá trình phân tích.

5.2.7 Đối tượng thực và đối tượng không thực (tangible versus intangible objects)

Thông thường, bạn sẽ tìm thấy trong chính mô hình của bạn những đối tượng hữu hình như là một Subject sẽ được sinh viên đăng ký, hay cũng có thể là một đối tượng vô hình như một Course trong CourseList, Course trong danh sách có thể được tạo ra hoặc không tùy theo số lượng sinh viên đăng ký tương ứng với course đó. Các Course trong danh sách là những course có thể được mở, chứ không nhất thiết phải bắt buộc mở. Khi Coure được mở tới sinh viên, course sẽ trở thành một đối tượng hữu hình. Nói chung, có thể có nhiều đối tượng hữu hình trong một đối tượng vô hình.

Có một lỗi phổ biến là ghép cặp đối tượng hữu hình và vô hình như một đối tượng đơn. Ví dụ trong hệ thống quản lý học tập theo tín chỉ, ta sẽ thấy trong suốt quá trình sinh viên đăng ký học các môn học, chúng ta có thể lên kế hoạch dự kiến các lớp học cho các môn học đó để mở lớp. Tuy nhiên danh sách các lớp học khác với một đối tượng lớp học hữu hình được mở và sinh viên tham gia học. Như vậy Course trong CourseList là đối tượng vô hình, nhưng Course mà sinh viên tham gia học là đối tượng hữu hình

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



Hình 5.12: Đối tượng thực và đối tượng không thực

Các thông tin liên quan đến Course List bao gồm:

- CourseID: Mã Course
- SubjectID: Mã môn học tương ứng với Course
- Capacity: số lượng lớp dự kiến
- Establish: biến Boolean cho biết lớp có được tổ chức hay không (có thể có hoặc không)

Các thông tin liên quan đến Course bao gồm:

- CourseID: mã Course
- SubjectID: mã môn học tương ứng với Course
- Capacity: số lượng lớp thực tế
- startDate: ngày bắt đầu học
- Time: thời gian học
- Room: phòng học
- InstructorID: mà giảng viên cho lớp học
- Student: mảng danh sách các sinh viên tham gia học. Số phần tử của mảng chính là giá trị capacity.

5.3 PHÂN TÍCH ĐỘNG

5.3.1 Lý do phân tích động

- Nhằm khẳng định rằng biểu đồ lớp của chúng ta là đã hoàn thiện và chính xác, để chúng ta có thể cố định nó sớm hơn là để muộn: điều này liên quan đến việc thêm, xóa, sửa các lớp, các mối quan hệ, các thuộc tính và các thao tác trong các lớp.
- Để có được tin cậy rằng mô hình hóa của chúng ta đến nay có thể cài đặt thành phần mềm.

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

- Kiểm định các chức năng của giao diện người dùng mà sẽ xuất hiện trong hệ thống cuối cùng. Đây là một ý tưởng tốt để phân quyền truy cập đến hệ thống thành các giao diện độc lập tách biệt theo các dòng use case trước khi đi đến thiết kế chi tiết.

5.3.2 Hiện thực hóa Use Case

Theo Jacobson, phần quan trọng nhất của phân tích động là hiện thực hóa Usecase nghĩa là tạo cho các UseCase của chúng ta hiện thực hơn bằng việc chứng tỏ cách chúng có thể được cài đặt như các đối tượng cộng tác với nhau. Hiện thực hóa bao gồm các bước sau :

- Duyệt qua các use case của hệ thống, mô phỏng các thông điệp gửi giữa các đối tượng và ghi lại các kết quả bằng biểu đồ giao tiếp.
- Đưa các phương thức, hành động trên đối tượng nhận các thông điệp đó.
- Thêm vào các lớp để biểu diễn biên (các giao diện hệ thống) và điều khiển (dành cho các tiến trình nghiệp vụ phức tạp hoặc cho việc tạo và khôi phục lại các đối tượng) khi cần thiết.

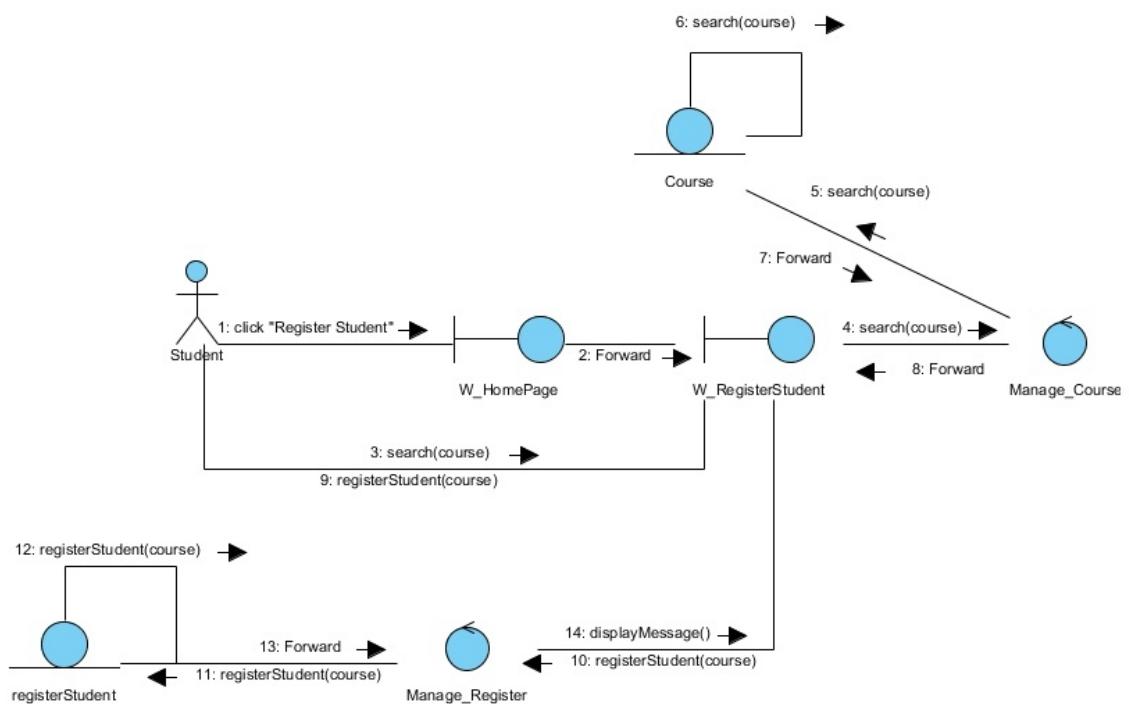
5.3.3 Biểu diễn hiện thực hóa UseCase

Khi chúng ta mô phỏng các thông điệp được gửi đi giữa các đối tượng đang phân tích, chúng ta cần ghi lại các kết quả. Biểu đồ giao tiếp và biểu đồ dây trong UML được thiết kế cho mục đích này. Mặc dù chúng ta có thể ghi lại các thông tin tương tự nhau trên hai biểu đồ này, nhưng biểu đồ giao tiếp tốt hơn cho hiện thực hóa useCase vì nó dễ tạo ra và chú trọng vào các đối tượng và các liên kết của chúng hơn là thứ tự các thông điệp được gửi đi. Biểu đồ giao tiếp mức phân tích có thể chỉ ra :

- Tương tác giữa tác nhân và các lớp biên.
- Các lớp biên tương tác với các đối tượng trong hệ thống.
- Các đối tượng bên trong hệ thống tương tác với các lớp biên hệ thống bên ngoài

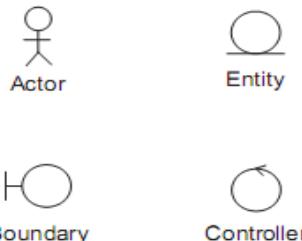
Ta không cần phải chỉ ra bất kì các đối tượng nghiệp vụ nào nằm bên ngoài hệ thống và các tác nhân mà không tương tác trực tiếp với hệ thống. Ví dụ về biểu đồ Communication cho việc sinh viên đăng ký học:

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



Hình 5.13: Biểu đồ giao tiếp cho use case SV đăng ký học

5.3.4 Các lớp biên, lớp thực thể và lớp điều khiển



Actor: Là một người hoặc hệ thống đang tồn tại bên ngoài hệ thống của chúng ta.

Đối tượng biên: một đối tượng nằm tại biên hệ thống, giữa hệ thống và Actors

Đối với tác nhân là 1 hệ thống : Biên cung cấp 1 đường liên kết.

Đối với tác nhân là con người: Biên chính là các giao diện, có nhiệm vụ nhận lệnh, truy vấn, và hiển thị trả về kết quả. Mỗi đối tượng thuộc lớp biên tương đương với một usecase hoặc một nhóm usecase liên hệ với nhau.

Đối tượng thực thể: Một đối tượng nằm bên trong hệ thống, mô tả các khái niệm nghiệp vụ như Customer, car... và chứa các thông tin hữu ích. Thông thường lớp thực thể thường xuất hiện trong biểu đồ giao tiếp và được sử dụng trong suốt quá trình thiết kế.

Đối tượng Điều khiển: Một đối tượng bên trong hệ thống đóng gói một tiến trình phức tạp. Một đối tượng điều khiển là một đối tượng dịch vụ, cung cấp các dịch vụ như: điều khiển tất cả hoặc một phần tiến trình hệ thống, tạo các thực thể mới, triệu hồi các thực thể đang tồn tại. Vì các lớp điều khiển chỉ tiện lợi cho phân tích, chúng ta không hy vọng

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

nhiều lớp này tồn tại trong quá trình thiết kế ngoại trừ đối tượng **home**. Home là đối tượng điều khiển được sử dụng để tạo ra những thực thể mới và truy xuất đến các thực thể đang tồn tại. Vì nó là khái niệm trong sáng nên thường tồn tại trong quá trình thiết kế.

Với cách tiếp cận RUP, ta giữ lại tất cả các lớp điều khiển cho pha thiết kế, hoàn thiện tất cả các phương thức mà chúng ta đã tìm ra trong quá trình phân tích. Với RUP, không có sự phân biệt giữa mô hình phân tích và mô hình thiết kế. Đơn giản là chúng ta sẽ bắt đầu với mô hình phân tích và sẽ thực hiện công việc mở rộng lặp đi lặp lại nhiều lần cho đến khi nó có thể chuyển thành một mô hình thiết kế có thể cài đặt được.

Các đầu ra có giá trị từ việc phân tích là:

- Các đối tượng thực thể tốt với các thuộc tính đã được xác nhận.
- Các đối tượng biên ở mức cao tương ứng với các usecase.
- Sự chắc chắn rằng mô hình chúng ta là đúng
- Đối tượng homes.

5.3.5 Các thành phần của biểu đồ giao tiếp

- Actors: được biểu diễn giống trong biểu đồ UseCase.
- Object: được biểu diễn và gán tên.
- Một đường thẳng giữa các đối tượng chỉ ra message được gửi đến đâu.
- Nhãn: gán tên cho đối tượng và tham số.
- Gán giá trị trả về cho một cái tên, được biểu diễn như sau: n = getnumber();
- Message điều kiện: biểu diễn: [điều kiện]message .
- Lặp: kí hiệu là * và sau là số lần lặp. Ví dụ, *3(có nghĩa là lặp 3 lần).

5.3.6 Thêm các phương thức vào lớp

Mỗi thông điệp trên một biểu đồ giao tiếp tương đương với một phương thức trên một lớp, chính vì vậy chúng ta nên ghi lại các phương thức để có một tập hoàn thiện các hiện thực use case. Các phương thức có thể được biểu diễn trên biểu đồ Lớp như sau :



Hình 5.14: Thêm các phương thức vào lớp

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Dạng tổng quát của cách biểu diễn phương thức trong UML :

opName (paramName1:paramType1, paramName2:paramType2) : ReturnType

Mỗi tên tham số , kiểu tham số và kiểu trả về là không bắt buộc

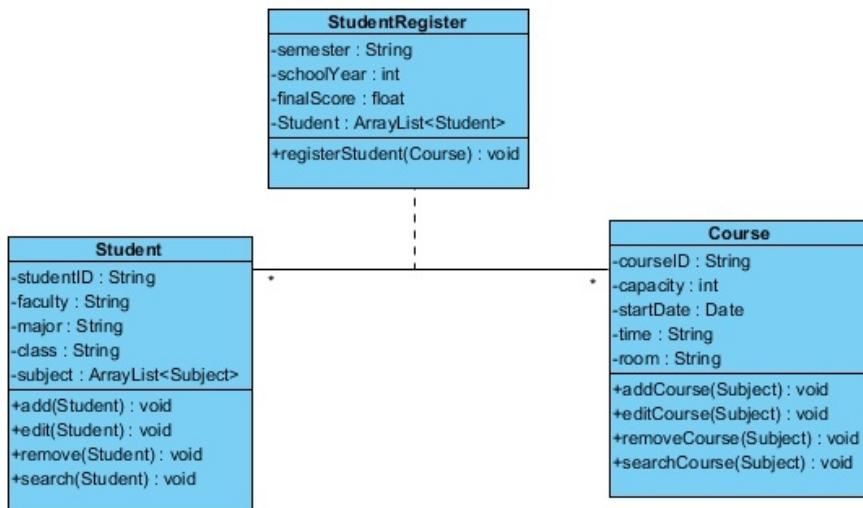
5.3.7 Trách nhiệm (Responsibility)

Bất cứ khi nào bạn thực hiện việc chi tiết hóa đối tượng, bạn cần xem xét đến trách nhiệm của đối tượng đó với hệ thống. Điều này giúp ta tìm và kiểm tra một cách đúng đắn các phương thức và thuộc tính. Khi bạn tìm thấy một vài thông tin hoặc hành vi cần cho hệ thống hãy tự hỏi rằng “đối tượng nào chịu trách nhiệm thực hiện điều này?”. Thông tin cho ta các thuộc tính, hành vi cho ta các phương thức. Có thể không một đối tượng nào chịu trách nhiệm cho hơn một công việc (hoặc một vai trò). Ví dụ, nếu một đối tượng chịu trách nhiệm việc lấy các thông tin chi tiết từ Student và xử lý nó, bạn có thể chọn hai đối tượng: một lớp biên và một lớp điều khiển.

Mang ý tưởng trách nhiệm hơn, hãy chắc chắn rằng không có đối tượng nào chịu trách nhiệm cho nhiều hơn một công việc. Các đối tượng với tập đơn trách nhiệm được gọi là có sự liên kết mạnh, một cái đích đáng ao ước.

Cũng có thể coi đối tượng như khách hàng (yêu cầu các câu hỏi, câu lệnh) hoặc nhà cung cấp (trả lời các câu hỏi và thực hiện các dịch vụ). Sự hợp tác hai chiều ở đây có thể làm cho hệ thống trở nên phức tạp và khó bảo trì hơn. Nhà cung cấp gọi là két dính lồng léo với các khách hàng của nó, ngược lại ta có sự dính kết chặt chẽ.

Một ví dụ đối với hệ thống Quản lý học tập theo tín chỉ.



Hình 5.15: Trách nhiệm phương thức

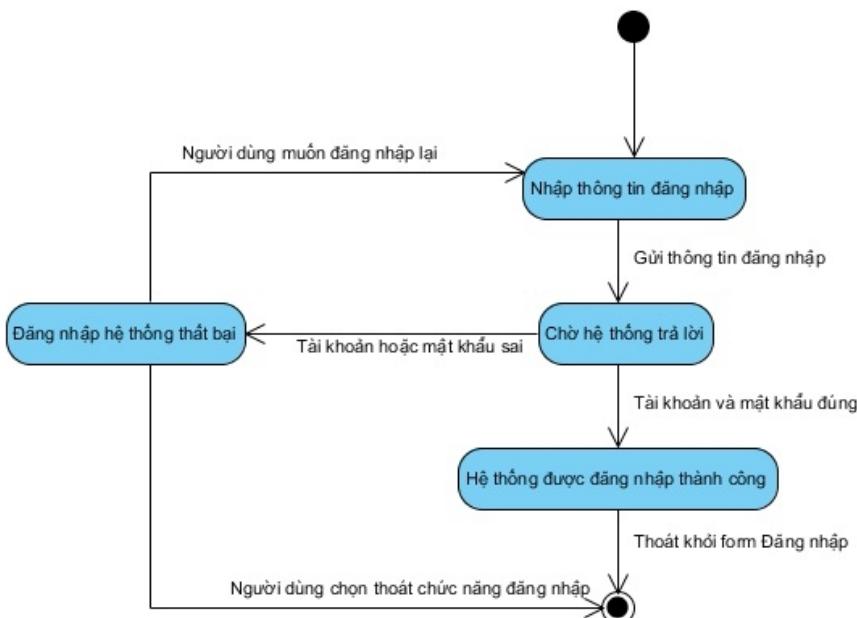
Phương thức **registerStudent(Course)**: thuộc lớp **StudentRegister**, là lớp liên kết (association class) giữa **Student** và **Course**. Mỗi sinh viên sẽ đăng ký **Course** của một **Subject** nào đó, điều kiện đăng ký khóa học dựa trên các ràng buộc về môn điều kiện và số trình còn nợ của sinh viên. Phương thức registerStudent(Course) thuộc lớp

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

StudentRegister có thể đảm bảo tính kết dính chặt chẽ trong thiết kế quan hệ giữa các lớp. Các phương thức thuộc lớp **Student**, **Course** đều gọi đến các thuộc tính đã được khai báo trước đó trong mỗi lớp, không ảnh hưởng đến các lớp khác.

5.3.8. Mô hình hóa trạng thái

Đôi khi, một thực thể sẽ có một vòng đời phức tạp để được chỉ ra trong một biểu đồ trạng thái máy. Ví dụ, hình 5.16 chỉ ra một mô hình vòng đời trạng thái của hệ thống Quản lý học tập theo tín chỉ trong use case Đăng nhập. Trong biểu đồ này, một hộp với các góc tròn chỉ ra một trạng thái, với một nhãn gắn với tên của nó. Một mũi tên chỉ ra một sự chuyển tiếp tới trạng thái khác – nhãn trên mũi tên chỉ ra nguyên nhân gây nên sự chuyển tiếp. Một vòng tròn đen với một mũi tên từ ngoài các điểm của nó vào một trạng thái ban đầu – một trạng thái mà trong đó một đối tượng có thể được sinh ra. Một mũi tên chỉ đến một đường tròn đen chỉ ra nguồn là một trạng thái cuối cùng – một trạng thái mà một đối tượng có thể kết thúc vòng đời.



Hình 5.16: Biểu đồ trạng thái use case Đăng nhập hệ thống

5.4. Tổng kết:

Chương này chúng ta đã đề cập đến:

- Quá trình thực hiện pha phân tích trong quá trình phát triển phần mềm
- Xây dựng mô hình phân tích tĩnh để chỉ ra các đối tượng hướng nghiệp vụ trong hệ thống, cùng với các thuộc tính và quan hệ của nó trên một class diagram
- Xây dựng mô hình phân tích động có thể cải thiện và làm rõ hơn mô hình tĩnh, sử dụng communication diagram, và chúng ta có thể mô hình vòng đời phức tạp của hệ thống sử dụng state machine diagram

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

5.4 CASE STUDY: HỆ QUẢN LÝ ĐĂNG KÝ HỌC THEO TÍN CHỈ

Phân tích là công việc tiếp theo của tiến trình phát triển phần mềm sau khi chúng ta hoàn thành quá trình Thu thập yêu cầu. Để tiến hành phân tích hệ thống chúng ta cần nắm rõ **Input và Output** của quá trình này. Trong đó, Input của quá trình Phân tích là sản phẩm của pha Xác định yêu cầu bao gồm:

- Actor list
- Usecase list
- Usecase details
- Usecase diagram
- Usecase survey
- User interface sketches

Output của pha Phân tích là Class diagram, Communication diagram và State diagram. Quá trình phân tích bao gồm phân tích tĩnh (Static Analysis) và phân tích động (Dynamic Analysis). Trong phân tích tĩnh, chúng ta tiến hành những công việc sau:

1. Xác định các lớp.
2. Xét mối quan hệ giữa các lớp.
3. Tìm các thuộc tính và phương thức của các lớp.
4. Đưa ra biểu đồ lớp và đối tượng.

Đối với phân tích động là đưa ra Communication diagram và State diagram.

Phân tích hệ thống đào tạo theo tín chỉ

5.4.1. Xác định yêu cầu

Trước tiên ta chỉ ra INPUT (*trích trong tài liệu Xác định yêu cầu*) bao gồm:

5.4.1.1. Danh sách Actor

Sinh viên: một người sử dụng có những thông tin được lưu trữ trong CSDL sinh viên, như: mã sinh viên, họ tên, lớp, khóa, ... Mỗi sinh viên phải được cung cấp mật khẩu truy nhập phù hợp với mã sinh viên để có thể truy cập vào hệ thống.

Giảng viên: một người sử dụng có những thông tin được lưu trữ trong CSDL giáo viên, như: mã giáo viên, họ tên, khóa, ... Mỗi giảng viên phải được cung cấp mật khẩu truy nhập phù hợp với mã giảng viên để có thể truy cập vào hệ thống.

Nhân viên phòng đào tạo: Một người chịu trách nhiệm điều hành hệ thống.

5.4.1.2. Danh sách Usecase

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

- U1: Đăng nhập: các actor đăng nhập hệ thống.
- U2: Thoát: các actor thoát khỏi hệ thống.
- U3: Đăng ký môn học: sinh viên đăng ký các môn học trong kỳ tới.
- U4: Xem thời khóa biểu: sinh viên xem thời khóa biểu chi tiết các lớp học đã đăng ký của mình
- U5: Xem lịch thi học kỳ: sinh viên xem lịch thi học kỳ các môn đã đăng ký
- U6: Xem điểm học tập: sinh viên xem kết quả học tập của mình.
- U7: In bảng điểm: sinh viên in bảng điểm các khóa học của mình.
- U8: Tìm kiếm thông tin: sinh viên xem danh sách các môn học của từng khoa, chuyên ngành, hệ đào tạo
- U9: Đăng ký môn dạy: giảng viên đăng ký môn sẽ dạy trong kỳ tới.
- U10: Xem lịch giảng dạy: giảng viên xem lịch giảng dạy của mình trong kỳ tới.
- U11: Đổi mật khẩu: người dùng thay đổi mật khẩu truy cập hệ thống.
- U12: Thay đổi thông tin cá nhân: Người dùng thay đổi thông tin cá nhân sau khi đăng nhập hệ thống
- U13: Quản lý sinh viên: Nhân viên phòng đào tạo thực hiện chức năng quản lý sinh viên với các thao tác cơ bản: Thêm sinh viên, Sửa thông tin sinh viên, Xóa sinh viên.
- U14: Quản lý giảng viên: Nhân viên phòng đào tạo thực hiện chức năng quản lý Giảng viên với các thao tác cơ bản: Thêm Giảng viên, Sửa thông tin giảng viên, Xóa giảng viên.
- U15: Quản lý cơ bản: Nhân viên phòng đào tạo thực hiện các chức năng quản lý cơ bản như Thêm Khoa, Thêm Chuyên Ngành...
- U16: Quản lý đào tạo: Nhân viên phòng đào tạo thực hiện các chức năng quản lý đào tạo như: Quản lý học phần, Quản lý đăng ký học và xếp lớp cho sinh viên, Quản lý đăng ký dạy của giảng viên, Quản lý kết quả học tập của sinh viên.
- U17: Gửi thông báo: phòng đào tạo gửi các thông báo tới sinh viên.
- U18: Xem thông báo: giảng viên, sinh viên xem thông báo của phòng đào tạo gửi đến

5.4.1.3. Khảo sát use case

Khi truy cập vào hệ thống đăng ký học theo tín chỉ, yêu cầu người dùng phải đăng nhập vào hệ thống (U1) trước khi có thể sử dụng các chức năng của hệ thống. Sau khi đăng nhập hệ thống, người dùng có thể sửa đổi mật khẩu (U11) tài khoản của mình

Giảng viên sau khi đăng nhập hệ thống (U1) có thể đăng ký môn dạy và lớp dạy trong học kỳ (U9) và có thể xem lịch giảng dạy của mình (U10)

Sinh viên sau khi đăng nhập vào hệ thống (U1) có các tùy chọn sử dụng chức năng của hệ thống như Tìm kiếm thông tin (U8), Đăng ký môn học trong học kỳ (U3). Tùy vào các

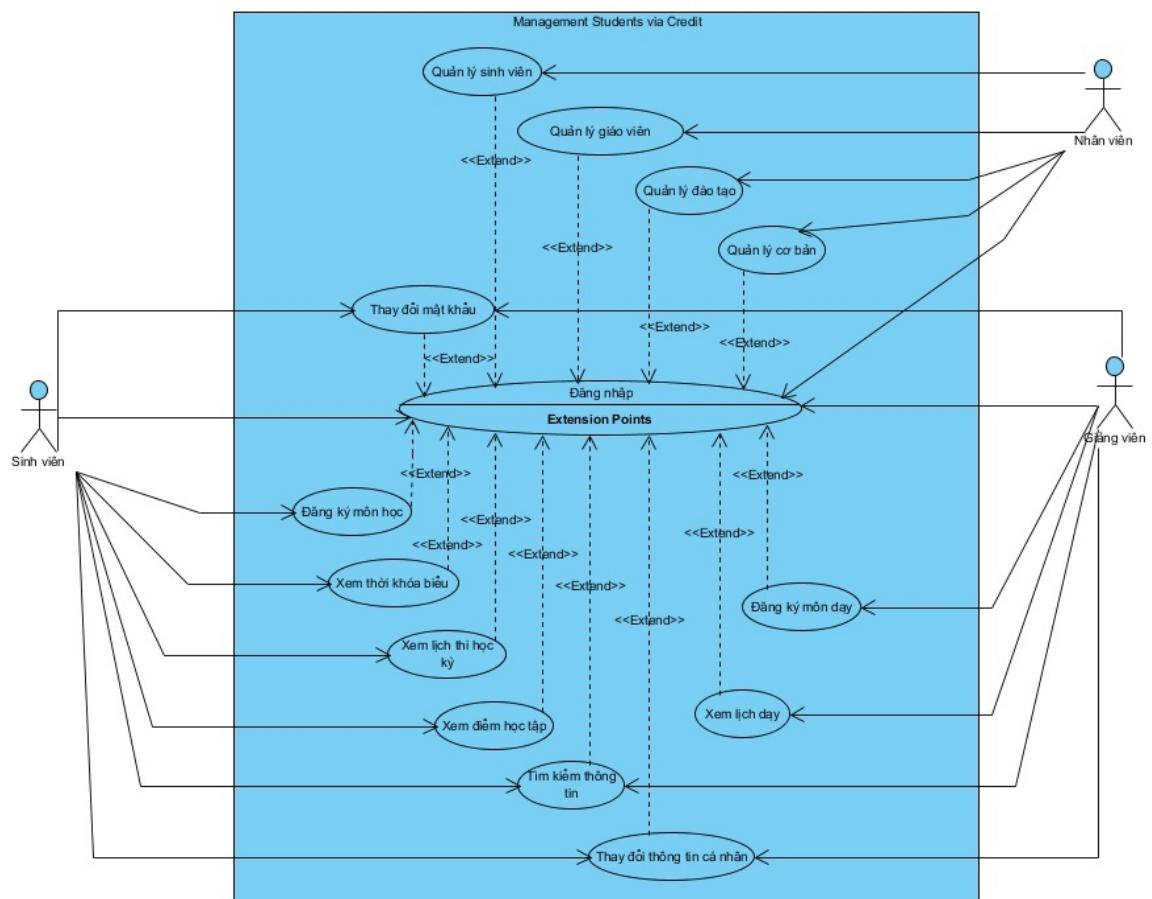
CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

ràng buộc đối với môn học về các môn điều kiện, số tín chỉ tích lũy, cán bộ phòng đào tạo có thể xét sinh viên có đủ điều kiện học môn đó hay không và cập nhật lịch học cũng như lịch thi học kỳ (U16). Khi sinh viên không đủ điều kiện học môn nào đó, phòng đào tạo có thể gửi thông báo (U17) về các môn điều kiện còn thiếu tới sinh viên để sinh viên hoàn tất các môn học đó. Sinh viên có thể xem thông báo (U18), xem thời khóa biểu (U4) là lịch học các môn học đã đăng ký mà sinh viên có thể theo học cũng như lịch thi học kỳ các môn đó.

Phòng đào tạo có nhiệm vụ cập nhật bảng điểm (U16) đối với sinh viên, sinh viên có thể xem kết quả học tập của mình (U6) khi đăng nhập hệ thống (U1). Bên cạnh đó hệ thống còn có các tùy chọn cho sinh viên như In bảng điểm (U7) hay tìm kiếm thông tin (U8) Cán bộ Đào tạo sau khi đăng nhập vào hệ thống (U1) có thể cập nhật thông tin quản lý cơ bản (U15), thông tin về sinh viên (U13), thông tin về giảng viên (U14)

Qua chức năng quản lý đào tạo (U16), cán bộ đào tạo chia lớp học theo Học phần, chuyển đổi học phần cho sinh viên đã đăng kí vào những học phần mà có số lượng sinh viên đăng kí nhỏ hơn điều kiện tối thiểu.

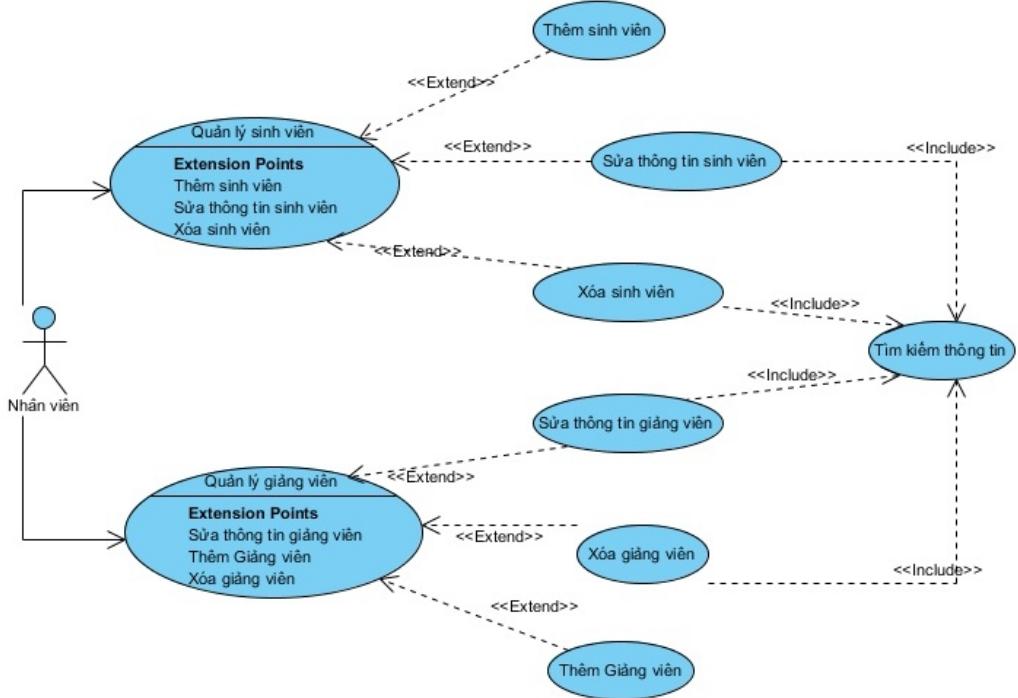
5.4.1.4. Biểu đồ Usecase



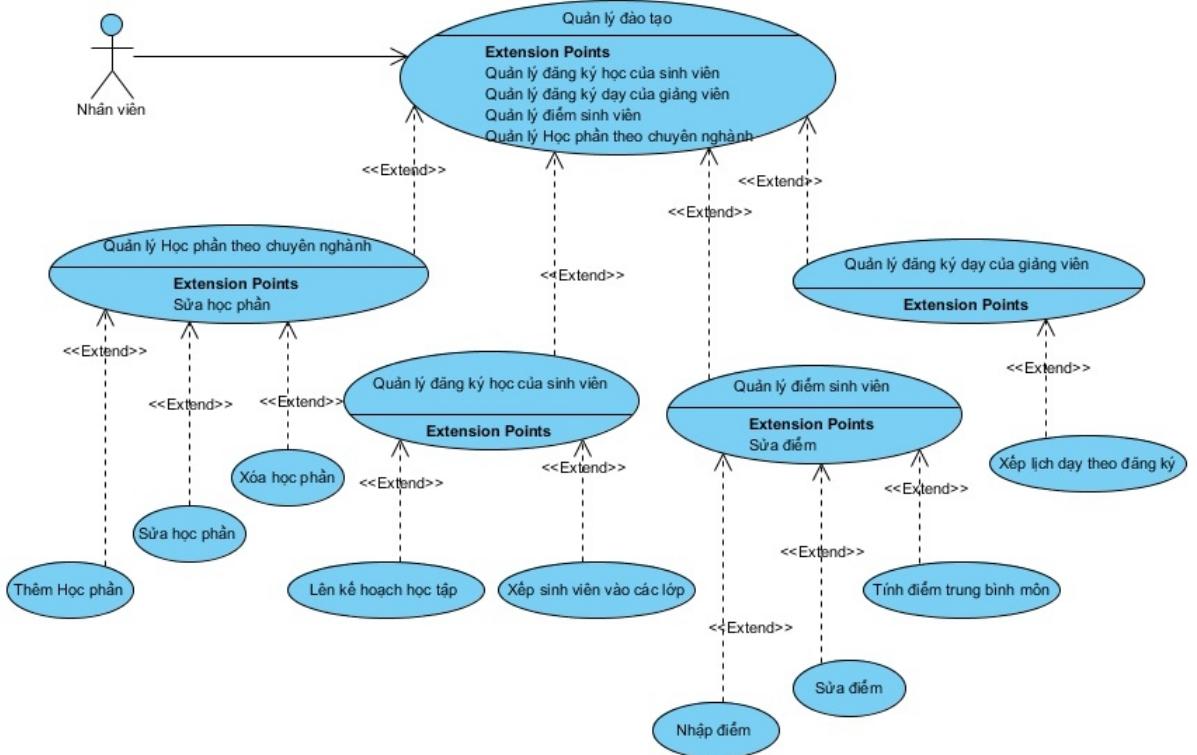
Hình 5.17: Biểu đồ Use case

5.4.1.5. Biểu đồ Use case phân rã

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

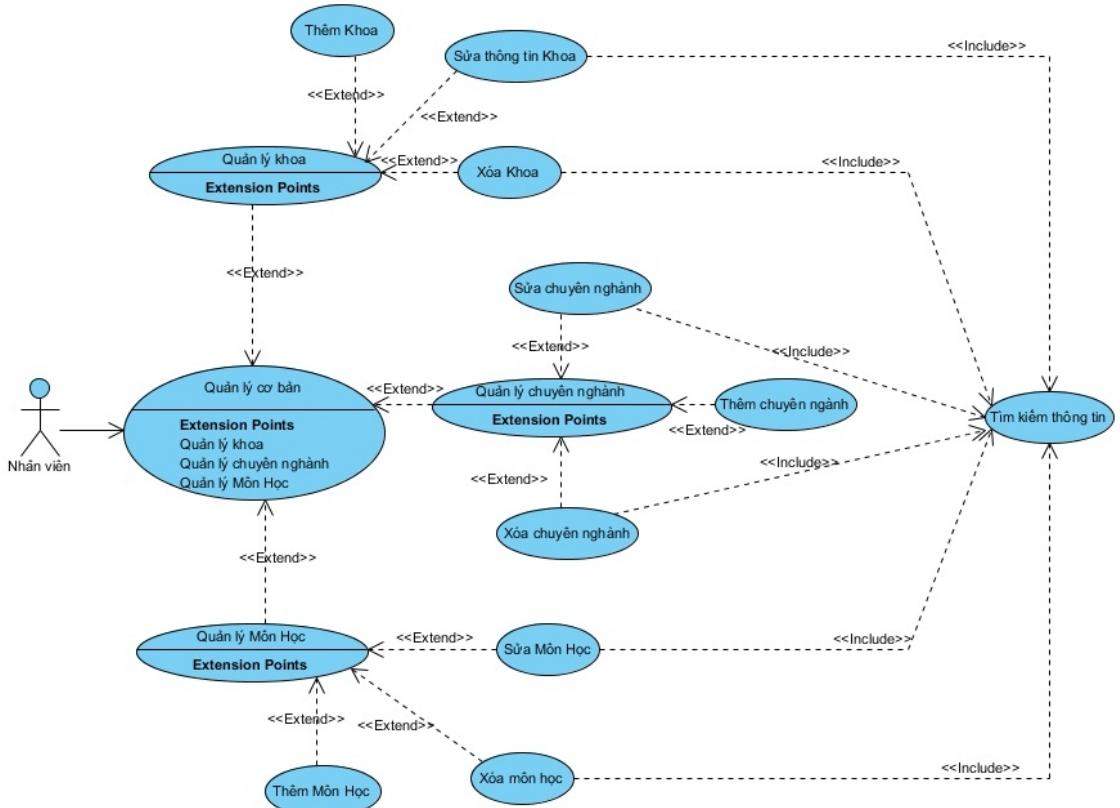


Hình 5.18: Phân rã use-case Quản lý sinh viên – Quản lý giảng viên



Hình 5.19: Phân rã use-case Quản lý đào tạo

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



Hình 5.20: Phân rã use-case Quản lý cơ bản

5.4.1.6. Kịch bản (Scenario) và phác thảo giao diện người dùng

1. Quản lý khoa

1.1. Thêm khoa

Tên Use Case	Thêm Khoa
Tác nhân chính	Cán bộ đào tạo.
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Đã thêm được Khoa
Kích hoạt	Button “Thêm Khoa” trên Form Quản lý Khoa
Chuỗi sự kiện chính	<ol style="list-style-type: none"> Cán bộ đào tạo kích hoạt form Quản lý cơ bản Hệ thống hiển thị ba tùy chọn: Quản lý Khoa, Quản lý Chuyên Nghành, Quản lý Môn học. Cán bộ đào tạo lựa chọn Quản lý Khoa Hệ thống hiển thị form để nhập các thông tin cần thiết về Khoa như Mã Khoa, Tên Khoa, Trưởng Khoa, Mô tả chung Cán bộ đào tạo nhập thông tin về Khoa và chọn button “Thêm Khoa”

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

- | |
|---|
| 6. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
7. Hệ thống thông báo thêm khóa thành công |
|---|

Ngoại lệ:

- 5.1. Hệ thống thông báo **Mã Khoa** đã bị trùng
- 5.1.1. Hệ thống yêu cầu nhập lại **Mã Khoa**.
- 5.1.2. **Cán bộ đào tạo** nhập lại và tiếp tục các bước sau

1.2 Sửa thông tin khoa

Tên Use Case	Sửa thông tin Khoa.
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã đăng nhập vào hệ thống.
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Hiển thị thông tin đã thay đổi về Khoa
Kích hoạt	Button “Sửa Khoa” trên Form Quản lý Khoa

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt **form Quản lý cơ bản**
2. Hệ thống hiện thị ba tùy chọn: Quản lý **Khoa**, Quản lý **Chuyên Nghành**, Quản lý **Môn học**.
3. Cán bộ đào tạo lựa chọn Quản lý **Khoa**
4. Hệ thống hiển thị form để nhập các thông tin cần thiết về **Khoa** như **Mã Khoa**, **Tên Khoa**, **Trưởng Khoa**, **Mô tả chung**
5. Cán bộ đào tạo nhập **Mã Khoa** hoặc **Tên Khoa** và chọn “Tìm kiếm”
6. Hệ thống hiển thị thông tin về **Khoa** tương ứng
7. Cán bộ đào tạo lựa chọn khoa cần sửa và thực hiện sửa thông tin về khoa và chọn button “**Sửa Khoa**”
8. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
9. Hệ thống thông báo quá trình sửa thông tin về **Khoa** thành công

Ngoại lệ

- 7.1. Hệ thống thông báo **Mã Khoa** không tồn tại.
 - 7.1.1. Hệ thống yêu cầu nhập lại **Mã Khoa**.
 - 7.1.2. **Cán bộ đào tạo** nhập lại và tiếp tục các bước sau.

1.3. Xóa thông tin khoa

Tên Use Case	Xóa thông tin Khoa
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống trả về trạng thái ban đầu
Đảm bảo thành công	Đã xóa thông tin
Kích hoạt	Button “ Xóa Khoa ” trên Form Quản lý Khoa

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt **form Quản lý cơ bản**
2. Hệ thống hiện thị ba tùy chọn: Quản lý **Khoa**, Quản lý **Chuyên Nghành**,

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

- Quản lý Môn học.
3. Cán bộ đào tạo lựa chọn “Thay đổi”
 4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Khoa như Mã Khoa, Tên Khoa, Trưởng Khoa, Mô tả chung
 5. Cán bộ đào tạo nhập Mã Khoa hoặc Tên Khoa và chọn “Tìm kiếm”
 6. Hệ thống hiển thị thông tin về Khoa tương ứng
 7. Cán bộ đào tạo lựa chọn khoa cần xóa và chọn button “Xóa Khoa”
 8. Hệ thống hiển thị thông báo xác định sẽ xóa Khoa
 9. Cán Bộ đào tạo chấp nhận xóa Khoa
 10. Hệ thống thông báo xóa Khoa thành công

Ngoại lệ

- 7.1. Hệ thống thông báo Mã Khoa không tồn tại.
 - 7.1.1. Hệ thống yêu cầu nhập lại Mã Khoa
 - 7.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau.

Phác thảo giao diện Quản lý Khoa

The screenshot shows a Windows application window titled "Design Preview [quanLySinhVien1]". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar is a menu bar with tabs: "Quản lý sinh viên", "Quản lý Giảng viên", "Quản lý đào tạo", and "Quản lý cơ bản". Under "Quản lý cơ bản", there is a sub-menu with tabs: "Quản lý Khoa", "Quản lý chuyên ngành", "Quản lý Môn học", and "Quản lý học phần".

Below the menu is a search bar with two input fields: "Tim kiem Khoa" and "Tim kiem theo mã".

The main area contains a table with four columns: "Mã Khoa", "Tên Khoa", "Trưởng Khoa", and "Mô tả". There are five rows in the table, each with empty fields.

Below the table is a section titled "Quản lý thông tin Khoa" with four input fields labeled "Mã Khoa", "Tên Khoa", "Trưởng Khoa", and "Mô tả chung".

At the bottom right of this section are three buttons: "Thêm Khoa", "Sửa Khoa", and "Xóa Khoa".

2. Quản lý Chuyên ngành

2.1 Thêm chuyên ngành

Tên Use Case	Thêm Chuyên Ngành
Tác nhân chính	Cán bộ đào tạo.
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Đã thêm được Chuyên Ngành

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Kích hoạt	Button “Thêm Chuyên Ngành” trên Form Quản lý Chuyên Ngành
Chuỗi sự kiện chính	
<ol style="list-style-type: none"> 1. Cán bộ đào tạo kích hoạt form Quản lý cơ bản 2. Hệ thống hiện thị ba tùy chọn: Quản lý Khoa, Quản lý Chuyên Ngành, Quản lý Môn học, Quản lý Học phần 3. Cán bộ đào tạo lựa chọn Quản lý Chuyên Ngành 4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Chuyên Ngành như Mã chuyên ngành, Tên Chuyên ngành, Trưởng bộ môn, Khoa phụ trách. 5. Cán bộ đào tạo nhập thông tin về Chuyên ngành và chọn button “Thêm Chuyên Ngành” 6. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu 7. Hệ thống thông báo thêm Chuyên Ngành thành công 	
Ngoại lệ: <ol style="list-style-type: none"> 5.1. Hệ thống thông báo Mã Chuyên Ngành đã bị trùng <ol style="list-style-type: none"> 5.1.1. Hệ thống yêu cầu nhập lại Mã Chuyên Ngành. 5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau 	

2.2. Sửa thông tin chuyên ngành

Tên Use Case	Sửa thông tin Chuyên Ngành
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã đăng nhập vào hệ thống.
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Hiển thị thông tin đã thay đổi về Chuyên Ngành
Kích hoạt	Button “Sửa Chuyên Ngành” trên Form Quản lý Chuyên Ngành

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt form **Quản lý cơ bản**
2. Hệ thống hiện thị ba tùy chọn: Quản lý **Khoa**, Quản lý **Chuyên Ngành**, Quản lý **Môn học**, Quản lý **Học phần**
3. Cán bộ đào tạo lựa chọn Quản lý **Chuyên Ngành**
4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Chuyên Ngành như **Mã chuyên ngành**, **Tên Chuyên ngành**, **Trưởng bộ môn**, **Khoa phụ trách**.
5. Cán bộ đào tạo **Mã** hoặc **Tên Chuyên Ngành** và chọn “Tìm kiếm”
6. Hệ thống hiển thị thông tin về **Chuyên Ngành** tương ứng
7. Cán bộ đào tạo lựa chọn **Chuyên Ngành** và thực hiện sửa thông tin về **Chuyên Ngành** và chọn button “Sửa Chuyên Ngành”
8. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
9. Hệ thống thông báo quá trình sửa thông tin về **Chuyên Ngành** thành công

Ngoại lệ

- 7.1. Hệ thống thông báo không tìm thấy **Chuyên Ngành**
 - 7.1.1. Hệ thống yêu cầu nhập lại **Mã Chuyên Ngành**.

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

7.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau

2.3 Xóa chuyên ngành

Tên Use Case	Xóa thông tin Chuyên Ngành
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống trở về trạng thái ban đầu
Đảm bảo thành công	Đã xóa thông tin
Kích hoạt	Button “Xóa Chuyên Ngành” trên Form Quản lý Chuyên Ngành

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt form **Quản lý cơ bản**
2. Hệ thống hiện thị ba tùy chọn: Quản lý **Khoa**, Quản lý **Chuyên Ngành**, Quản lý **Môn học**, Quản lý **Học phần**
3. Cán bộ đào tạo lựa chọn Quản lý **Chuyên Ngành**
4. Hệ thống hiển thị form để nhập các thông tin cần thiết về **Chuyên Ngành** như **Mã chuyên ngành**, **Tên Chuyên ngành**, **Trưởng bộ môn**, **Khoa phụ trách**
5. Cán bộ đào tạo nhập **Mã** hoặc **Tên Chuyên Ngành** và chọn “Tìm kiếm”
6. Hệ thống hiển thị thông tin về **Chuyên Ngành** tương ứng
7. Cán bộ đào tạo lựa chọn **Chuyên ngành** và chọn button “Xóa Chuyên Ngành”
8. Hệ thống hiển thị thông báo khẳng định sẽ xóa **Chuyên Ngành**
9. Hệ thống thông báo xóa **Chuyên Ngành** thành công

Ngoại lệ

- 5.1. Hệ thống thông báo **Mã Chuyên Ngành** đã bị trùng
 - 5.1.1. Hệ thống yêu cầu nhập lại **Mã Chuyên Ngành**.
 - 5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau

Phác thảo giao diện quản lý Chuyên Ngành

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Design Preview [quanLySinhVien1]

Quản lý sinh viên | Quản lý Giảng viên | Quản lý đào tạo | Quản lý cơ bản |

Quản lý Khoa | Quản lý chuyên ngành | Quản lý Môn học | Quản lý học phần |

Tìm kiếm Chuyên Ngành: Tìm kiếm theo mã:

Mã Chuyên Ngành	Tên Chuyên Ngành	Trưởng Bộ môn	Khoa phụ trách	Mô tả

Quản lý thông tin Chuyên Ngành

Mã Chuyên Ngành	<input type="text"/>
Tên Chuyên Ngành	<input type="text"/>
Trưởng Bộ môn	<input type="text"/>
Khoa phụ trách	<input type="text"/>
Khoa phụ trách	<input type="text"/>

3. Quản lý Môn Học

3.1. Thêm Môn học

Tên Use Case	Thêm Môn Học
Tác nhân chính	Cán bộ đào tạo.
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Đã thêm được Môn Học
Kích hoạt	Button “Thêm Môn Học” trên Form Quản lý Môn Học

Chuỗi sự kiện chính

1. **Cán bộ đào tạo** kích hoạt form **Quản lý cơ bản**
2. Hệ thống hiện thị ba tùy chọn: Quản lý **Khoa**, Quản lý **Chuyên Ngành**, Quản lý **Môn học**. Quản lý **Học phần**.
3. Cán bộ đào tạo lựa chọn Quản lý **Môn Học**
4. Hệ thống hiển thị form để nhập các thông tin cần thiết về **Môn Học** như **Mã môn**, **Tên môn**, **Số tín chỉ**, **Số tiết Lý thuyết**, **Số tiết Thực hành**, **Số tiết bài tập**, **Khoa phụ trách**, **Môn điều kiện**.
5. **Cán bộ đào tạo** nhập thông tin về Môn Học và chọn button “Thêm Môn Học”
6. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
7. Hệ thống thông báo thêm **Môn Học** thành công

Ngoại lệ:

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

- | | |
|---|---|
| 5.1. Hệ thống thông báo Mã Môn Học đã bị trùng | 5.1.1. Hệ thống yêu cầu nhập lại Mã Môn Học |
| | 5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau |

Sửa thông tin Môn Học

Tên Use Case	Sửa thông tin Môn Học
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã đăng nhập vào hệ thống.
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Hiển thị thông tin đã thay đổi về Môn Học
Kích hoạt	Button “Sửa Môn Học” trên Form Quản lý Môn Học

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt **form Quản lý cơ bản**
2. Hệ thống hiện thị ba tùy chọn: Quản lý **Khoa**, Quản lý **Chuyên Ngành**, Quản lý Môn học, Quản lý **Học phần**.
3. Cán bộ đào tạo lựa chọn Quản lý **Môn học**
4. Hệ thống hiển thị form tìm kiếm và để nhập các thông tin cần thiết về **Môn học** như **Mã môn**, **Tên môn**, **Số tín chỉ**, **Số tiết Lý thuyết**, **Số tiết Thực hành**, **Số tiết bài tập**, **Khoa phụ trách**, **Môn điều kiện**.
5. Cán bộ đào tạo nhập **Mã** hoặc **Tên Môn học** và chọn “Tim kiếm”
6. Hệ thống hiển thị thông tin về **Môn học** tương ứng
7. **Cán bộ đào tạo** lựa chọn môn học và thực hiện sửa thông tin về Môn học và chọn button “**Sửa Môn học**”
8. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
9. Hệ thống thông báo quá trình sửa thông tin về **Môn học** thành công

Ngoại lệ

- | | |
|--|---|
| 5.1. Hệ thống thông báo không tìm thấy Mã Môn Học | 5.1.1. Hệ thống yêu cầu nhập lại Mã Môn Học |
| | 5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau |

3.3. Xóa thông tin Môn học

Tên Use Case	Xóa thông tin Môn Học
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống trở về trạng thái ban đầu
Đảm bảo thành công	Đã xóa thông tin
Kích hoạt	Button “ Xóa Môn Học ” trên Form Quản lý Môn Học

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt **form Quản lý cơ bản**
2. Hệ thống hiện thị ba tùy chọn: Quản lý **Khoa**, Quản lý **Chuyên Ngành**, Quản lý Môn học, Quản lý **Học phần**.
3. Cán bộ đào tạo lựa chọn Quản lý **Môn học**

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

4. Hệ thống hiển thị form tìm kiếm và để nhập các thông tin cần thiết về Môn học như **Mã môn**, **Tên môn**, **Số tín chỉ**, **Số tiết Lý thuyết**, **Số tiết Thực hành**, **Số tiết bài tập**, **Khoa phụ trách**, **Môn điều kiện**.
5. Cán bộ đào tạo nhập **Mã** hoặc **Tên Môn học** và chọn “Tìm kiếm”
6. Hệ thống hiển thị thông tin về **Môn học** tương ứng
7. Cán bộ đào tạo lựa chọn môn học muốn xóa và chọn button “Xóa Môn học”
8. Hệ thống hiển thị thông báo khi xác định sẽ xóa **Môn học**
9. Hệ thống thông báo xóa **Môn học** thành công

Ngoại lệ

- 5.1. Hệ thống thông báo không tìm thấy **Môn Học**.
 - 5.1.1. Hệ thống yêu cầu nhập lại **Môn Học**
 - 5.1.2. **Cán bộ đào tạo** nhập lại và tiếp tục các bước sau

Phác thảo giao diện Quản lý Môn học

The screenshot shows a Windows application window titled "Design Preview [quanLySinhVien1]". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar is a navigation menu bar with tabs: "Quản lý sinh viên", "Quản lý Giảng viên", "Quản lý đào tạo", "Quản lý cơ bản", "Quản lý Khoa", "Quản lý chuyên ngành", "Quản lý Môn học" (which is the active tab), and "Quản lý học phần".

Below the menu bar is a search bar with two input fields: "Tim kiem Môn học" and "Tim kiem theo mã".

The main area contains a table with columns: Mã Môn..., Tên Môn học, Số tín chỉ, Số tiết LT, Số tiết TH, Số tiết BT, Môn điều kiện, and Thuộc Khoa. There are 8 rows of data in the table.

A large panel below the table is titled "Quản lý thông tin Chuyên Ngành". It contains several input fields and dropdown menus:

- Mã Môn học
- Tên Môn học
- Số tín chỉ
- Số tiết lý thuyết
- Số tiết thực hành
- Số tiết bài tập
- Thuộc Khoa: A dropdown menu currently showing "Công Nghệ Thông Tin".
- Môn điều kiện: A dropdown menu currently showing "Lập trình hướng đối tượng, Công Nghệ Phần mềm".
- <Lựa chọn Môn>: A dropdown menu listing: Công Nghệ Phần mềm, Lập trình hướng đối tượng, Quản lý dự án, and Kỹ thuật đồ họa.

At the bottom of the panel are three buttons: "Thêm Môn", "Sửa Môn", and "Xóa Môn".

4. Quản lý sinh viên

4.1 Thêm sinh viên

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Tên Use Case	Thêm Sinh Viên
Tác nhân chính	Cán bộ đào tạo.
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Đã thêm được Sinh Viên
Kích hoạt	Button “Thêm Sinh viên” trên Form Quản lý Sinh Viên
Chuỗi sự kiện chính	<ol style="list-style-type: none"> 1. Cán bộ đào tạo kích hoạt form Quản lý Sinh viên 2. Hệ thống hiển thị hai tùy chọn “Thêm Sinh Viên” và “Sửa/xóa sinh viên” 3. Cán bộ đào tạo lựa chọn form “Thêm Sinh viên” 4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Sinh viên và các tùy chọn: Tìm kiếm, Thêm Sinh viên, Sửa Sinh viên, Xóa Sinh viên. 5. Cán bộ đào tạo nhập thông tin về Sinh viên gồm có Mã SV, Họ tên, Ngày sinh, Giới tính, Khoa, Chuyên ngành, Khóa, Lớp và chọn button “Thêm Sinh viên” 6. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu 7. Hệ thống thông báo thêm Sinh viên thành công
Ngoại lệ:	<p>1.1. Hệ thống thông báo Mã Sinh viên đã bị trùng</p> <p>1.1.1. Hệ thống yêu cầu nhập lại Mã Sinh viên.</p> <p>1.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau</p>

4.2 Sửa Sinh viên

Tên Use Case	Sửa thông tin Sinh viên
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã đăng nhập vào hệ thống.
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Hiển thị thông tin đã thay đổi về Sinh Viên
Kích hoạt	Button “Sửa Sinh Viên” trên Form Quản lý Sinh Viên
Chuỗi sự kiện chính	<ol style="list-style-type: none"> 1. Cán bộ đào tạo kích hoạt form Quản lý Sinh viên 2. Hệ thống hiển thị hai tùy chọn “Thêm Sinh viên” và “Sửa/xóa Sinh viên” 3. Cán bộ đào tạo lựa chọn form “Sửa/xóa Sinh viên” 4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Sinh viên và các tùy chọn: Tìm kiếm, Thêm Sinh viên, Sửa Sinh viên, Xóa Sinh viên. 5. Cán bộ đào tạo nhập Mã Sinh viên hoặc Họ tên Sinh viên và chọn “Tìm kiếm” 6. Hệ thống kiểm tra thông tin và hiển thị thông tin về Sinh viên cần tìm kiếm gồm có Mã SV, Họ tên, Ngày sinh, Giới tính, Khoa, Chuyên ngành, Khóa, Lớp. 7. Cán bộ đào tạo lựa chọn một Sinh viên cần sửa thông tin 8. Hệ thống hiển thị các thông tin về Sinh viên trên form nhập liệu 9. Cán bộ đào tạo thay đổi thông tin và lựa chọn “Sửa Sinh viên” 10. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

11. Hệ thống thông báo sửa Sinh viên thành công

Ngoại lệ

5.1. Hệ thống thông báo **Mã Sinh viên** không tồn tại.

5.1.1. Hệ thống yêu cầu nhập lại **Mã Sinh viên**.

5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau.

4.3 Xóa thông tin sinh viên

Tên Use Case	Xóa thông tin Sinh viên
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống trở về trạng thái ban đầu
Đảm bảo thành công	Đã xóa thông tin
Kích hoạt	Button “Xóa Sinh Viên” trên Form Quản lý Sinh Viên

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt form **Quản lý Sinh viên**
2. Hệ thống hiển thị hai tùy chọn “Thêm Sinh viên” và “Sửa/xóa Sinh viên”
3. Cán bộ đào tạo lựa chọn form “Sửa/xóa Sinh viên”
4. Hệ thống hiển thị form để nhập các thông tin cần thiết về **Sinh viên** và các tùy chọn: Tìm kiếm, Thêm Sinh viên, Sửa Sinh viên, Xóa Sinh viên.
5. Cán bộ đào tạo nhập **Mã Sinh viên** hoặc **Họ tên Sinh viên** và chọn “Tìm kiếm”
6. Hệ thống kiểm tra thông tin và hiển thị thông tin về **Sinh viên** cần tìm kiếm gồm có **Mã SV, Họ tên, Ngày sinh, Giới tính, Khoa, Chuyên ngành, Khóa, Lớp**.
7. Cán bộ đào tạo lựa chọn **Sinh viên** cần xóa và lựa chọn “Xóa Sinh viên”
8. Hệ thống hiển thị thông báo khẳng định muốn xóa **Sinh viên**
9. Cán bộ đào tạo khẳng định xóa
10. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
11. Hệ thống thông báo xóa **Sinh viên** thành công

Ngoại lệ

5.1. Hệ thống thông báo **Mã Sinh viên** không tồn tại.

5.1.1. Hệ thống yêu cầu nhập lại **Mã Sinh viên**.

5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau.

Phác thảo giao diện quản lý sinh viên:

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Giao diện Thêm Sinh Viên

Giao diện Sửa/xóa Sinh viên

5.Quản lý Giảng viên

5.1. Thêm Giảng viên

Tên Use Case	Thêm Giảng Viên
Tác nhân chính	Cán bộ đào tạo.

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Đã thêm được Giảng Viên
Kích hoạt	Button “Thêm Giảng Viên” trên Form Quản lý Giảng Viên
Chuỗi sự kiện chính	
<ol style="list-style-type: none"> 1. Cán bộ đào tạo kích hoạt form Quản lý Giảng viên 2. Hệ thống hiển thị hai tùy chọn “Thêm Giảng Viên” và “Sửa/xóa Giảng viên” 3. Cán bộ đào tạo lựa chọn form “Thêm Giảng viên” 4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Giảng viên và các tùy chọn: Tìm kiếm, Thêm Giảng viên, Sửa Giảng viên, Xóa Giảng viên. 5. Cán bộ đào tạo nhập thông tin về Giảng viên gồm có Mã GV, Họ tên, Ngày sinh, Giới tính, Khoa, Bộ Môn, Học vị và chọn button “Thêm Giảng viên” 6. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu 7. Hệ thống thông báo thêm Giảng viên thành công 	
Ngoại lệ:	
<ol style="list-style-type: none"> 5.1.Hệ thống thông báo Mã Giảng Viên đã bị trùng <ol style="list-style-type: none"> 5.1.1.Hệ thống yêu cầu nhập lại Mã Giảng viên. 5.1.2.Cán bộ đào tạo nhập lại và tiếp tục các bước sau 	

5.2. Sửa Giảng viên

Tên Use Case	Sửa thông tin Giảng Viên
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã đăng nhập vào hệ thống.
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Hiển thị thông tin đã thay đổi về Giảng Viên
Kích hoạt	Button “Sửa Giảng Viên” trên Form Quản lý Giảng Viên

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt form **Quản lý Giảng viên**
2. Hệ thống hiển thị hai tùy chọn “Thêm Giảng Viên” và “Sửa/xóa Giảng viên”
3. Cán bộ đào tạo lựa chọn form **“Sửa/xóa Giảng viên”**
4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Giảng viên và các tùy chọn: Tìm kiếm, Thêm Giảng viên, Sửa Giảng viên, Xóa Giảng viên.
5. Cán bộ đào tạo nhập **Mã Giảng Viên** hoặc **Họ tên Giảng viên** và chọn “Tìm kiếm”
6. Hệ thống kiểm tra thông tin và hiển thị thông tin về **Giảng viên** cần tìm kiếm gồm có **Mã GV, Họ tên, Ngày sinh, Giới tính, Khoa, Bộ Môn, Học vị**
7. Cán bộ đào tạo lựa chọn một **Giảng viên** cần sửa thông tin
8. Hệ thống hiển thị các thông tin về **Giảng viên** trên form nhập liệu
9. Cán bộ đào tạo thay đổi thông tin và lựa chọn “Sửa Giảng viên”
10. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
11. Hệ thống thông báo sửa Giảng viên thành công

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Ngoại lệ

- 6.1. Hệ thống thông báo **Mã Giảng Viên** không tồn tại.
 - 6.1.1. Hệ thống yêu cầu nhập lại **Mã Giảng Viên**.
 - 6.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau.

5.3. Xóa thông tin Giảng viên

Tên Use Case	Xóa thông tin Giảng viên
Tác nhân chính	Cán bộ đào tạo
Người chịu trách nhiệm	Người quản lý hệ thống
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống trả về trạng thái ban đầu
Đảm bảo thành công	Đã xóa thông tin
Kích hoạt	Button “Xóa Giảng Viên” trên Form Quản lý Giảng Viên

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt form **Quản lý Giảng viên**
2. Hệ thống hiển thị hai tùy chọn “Thêm Giảng Viên” và “Sửa/xóa Giảng viên”
3. Cán bộ đào tạo lựa chọn form **“Sửa/xóa Giảng viên”**
4. Hệ thống hiển thị form để nhập các thông tin cần thiết về Giảng viên và các tùy chọn: Tìm kiếm, Thêm Giảng viên, Sửa Giảng viên, Xóa Giảng viên.
5. Cán bộ đào tạo nhập **Mã Giảng Viên** hoặc **Họ tên Giảng viên** và chọn “Tìm kiếm”
6. Hệ thống kiểm tra thông tin và hiển thị thông tin về **Giảng viên** cần tìm kiếm gồm có **Mã GV**, **Họ tên**, **Ngày sinh**, **Giới tính**, **Khoa**, **Bộ Môn**, **Học vị**.
7. Cán bộ đào tạo lựa chọn **Giảng viên** cần xóa và lựa chọn “Xóa Giảng viên”
8. Hệ thống hiển thị thông báo khẳng định muốn xóa **Giảng viên**
9. Cán bộ đào tạo khẳng định xóa
10. Hệ thống kiểm tra thông tin và lưu vào cơ sở dữ liệu
11. Hệ thống thông báo **Giảng viên** thành công

Ngoại lệ

- 5.1. Hệ thống thông báo **Mã Giảng Viên** không tồn tại.
 - 5.1.1. Hệ thống yêu cầu nhập lại **Mã Giảng Viên**.
 - 5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau.

Phác thảo giao diện quản lý Giảng Viên:

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Design Preview [quanLySinhVien1]

Quản lý sinh viên	Quản lý Giảng viên	Quản lý đào tạo	Quản lý cơ bản
<input type="button" value="Thêm Giảng viên"/> <input type="button" value="Sửa/xóa Giảng viên"/>			
Thông tin Giảng viên			
Thông tin cá nhân		Thông tin giáo vụ	
Họ tên:	<input type="text"/>	Mã GV:	<input type="text"/>
Ngày sinh:	<input type="text"/>	Khoa:	<input type="text"/>
Quê quán:	<input type="text"/>	Bộ môn:	<input type="text"/>
Địa chỉ thường trú:	<input type="text"/>	Học vị:	<input type="text"/>
Số CMT:	<input type="text"/>		
Giới tính	<input type="button" value="Nam"/>		
<input type="button" value="Thêm GV"/>			

Giao diện thêm Giảng Viên

Design Preview [quanLySinhVien1]

Quản lý sinh viên	Quản lý Giảng viên	Quản lý đào tạo	Quản lý cơ bản																																																				
<input type="button" value="Thêm Giảng viên"/> <input type="button" value="Sửa/xóa Giảng viên"/>																																																							
<input type="text"/> Nhập thông tin: <input type="button" value="Tìm kiếm theo"/> <input type="button" value="Mã Giảng Viên"/> <input type="button" value="Tìm kiếm"/>																																																							
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Mã SV</th> <th>Họ Tên</th> <th>Ngày sinh</th> <th>Giới tính</th> <th>Khoa</th> <th>Bộ môn</th> <th>Học vị</th> <th>Xóa</th> </tr> </thead> <tbody> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td><input type="checkbox"/></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td><input type="checkbox"/></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td><input type="checkbox"/></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td><input type="checkbox"/></td></tr> <tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td><input type="checkbox"/></td></tr> </tbody> </table>								Mã SV	Họ Tên	Ngày sinh	Giới tính	Khoa	Bộ môn	Học vị	Xóa								<input type="checkbox"/>								<input type="checkbox"/>								<input type="checkbox"/>								<input type="checkbox"/>								<input type="checkbox"/>
Mã SV	Họ Tên	Ngày sinh	Giới tính	Khoa	Bộ môn	Học vị	Xóa																																																
							<input type="checkbox"/>																																																
							<input type="checkbox"/>																																																
							<input type="checkbox"/>																																																
							<input type="checkbox"/>																																																
							<input type="checkbox"/>																																																
<i><Click đúp vào một giảng viên để thực hiện thao tác sửa thông tin về giảng viên></i>																																																							
<input type="button" value="Xóa SV"/>																																																							
Thông tin Giảng viên																																																							
Thông tin cá nhân		Thông tin giáo vụ																																																					
Họ tên:	<input type="text"/>	Mã GV:	<input type="text"/>																																																				
Ngày sinh:	<input type="text"/>	Khoa:	<input type="text"/>																																																				
Quê quán:	<input type="text"/>	Bộ môn:	<input type="text"/>																																																				
Địa chỉ thường trú:	<input type="text"/>	Học vị:	<input type="text"/>																																																				
Số CMT:	<input type="text"/>																																																						
Giới tính	<input type="button" value="Nam"/>																																																						
<input type="button" value="Sửa thông tin"/>																																																							

Giao diện Sửa/xóa Giảng viên

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

6. Quản lý đào tạo

6.1. Quản lý xếp lớp học

a. Thêm lớp học

Tên Use Case	Thêm Lớp học
Tác nhân chính	Cán bộ đào tạo.
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thêm và quay lui lại bước trước
Đảm bảo thành công	Đã thêm được Lớp học
Kích hoạt	Button “Thêm Lớp học” trên Form Quản lý xếp lớp học
Chuỗi sự kiện chính	
1. Cán bộ đào tạo kích hoạt form Quản lý Xếp lớp học	
2. Hệ thống hiển thị form với các thông tin tùy chọn cần thiết về Khoa, Chuyên Ngành, Học kỳ	
3. Cán bộ đào tạo chọn các thông tin về Khoa, Chuyên Ngành, Học kỳ tương ứng và chọn “Hiển thị”	
4. Hệ thống kiểm tra thông tin và hiển thị các Môn học bắt buộc và tự chọn mà Sinh viên cần học trong học kỳ tương ứng	
5. Cán bộ đào tạo lựa chọn Môn học và điền các thông tin về lớp học tương ứng với môn học đó gồm có Mã môn, Tên môn, Mã lớp, Ngày bắt đầu, Buổi học, Giờ học, Phòng học, Sĩ số và chọn “Thêm lớp học”	
6. Hệ thống kiểm tra thông tin và hiển thị Thêm lớp học thành công	
Ngoại lệ:	
5.1. Hệ thống thông báo Mã Lớp học đã bị trùng	
5.1.1. Hệ thống yêu cầu nhập lại Mã Lớp học .	
5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau	

b.Sửa lớp học

Tên Use Case	Sửa Lớp học
Tác nhân chính	Cán bộ đào tạo.
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống loại bỏ các thông tin đã thay đổi và quay lui lại bước trước
Đảm bảo thành công	Đã sửa được thông tin Lớp học
Kích hoạt	Button “Sửa Lớp học” trên Form Quản lý xếp lớp học
Chuỗi sự kiện chính	
1. Cán bộ đào tạo kích hoạt form Quản lý Xếp lớp học	
2. Hệ thống hiển thị form để tìm kiếm Lớp học theo Môn học tương ứng	
3. Cán bộ đào tạo nhập Tên Môn học (hoặc Mã Môn học) và chọn tìm kiếm	
4. Hệ thống kiểm tra thông tin và hiển thị thông tin về các lớp học tương ứng với môn học đó gồm có Mã môn, Tên môn, Mã lớp, Ngày bắt đầu, Buổi học, Giờ học, Phòng học, Sĩ số .	
5. Cán bộ đào tạo lựa chọn lớp học muốn sửa	
6. Thông tin về lớp học muốn sửa sẽ được hiển thị lên form nhập liệu	
7. Các bộ đào tạo thay đổi thông tin về lớp học và lựa chọn “Sửa lớp học”	

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

8. Hệ thống kiểm tra thông tin và hiển thị quá trình Sửa lớp học thành công

Ngoại lệ:

- 3.1. **Tên (hoặc Mã) Môn học** không tìm thấy trong cơ sở dữ liệu
 - 3.1.1. Hệ thống hiển thị thông báo không tìm thấy và yêu cầu người dùng nhập lại thông tin tìm kiếm
- 3.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau
 - 5.1. Hệ thống thông báo **Mã Lớp học** đã bị trùng
 - 5.1.1. Hệ thống yêu cầu nhập lại **Mã Lớp học**.
 - 5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau

c.Xóa lớp học

Tên Use Case	Xóa Lớp học
Tác nhân chính	Cán bộ đào tạo.
Người chịu trách nhiệm	Người quản lý hệ thống.
Tiền điều kiện	Cán bộ đào tạo đã Đăng nhập vào hệ thống
Đảm bảo tối thiểu	Hệ thống trả về trạng thái ban đầu
Đảm bảo thành công	Đã xóa được Lớp học
Kích hoạt	Button “Xóa Lớp học” trên Form Quản lý xếp lớp học

Chuỗi sự kiện chính

1. Cán bộ đào tạo kích hoạt **form Quản lý Xếp lớp học**
2. Hệ thống hiển thị form để tìm kiếm **Lớp học** theo **Môn học** tương ứng
3. Cán bộ đào tạo nhập **Tên Môn học** (hoặc **Mã Môn học**) và chọn tìm kiếm
4. Hệ thống kiểm tra thông tin và hiển thị thông tin về các **lớp học** tung ứng với môn học đó gồm có **Mã môn**, **Tên môn**, **Mã lớp**, **Ngày bắt đầu**, **Buổi học**, **Giờ học**, **Phòng học**, **Sĩ số**.
5. Cán bộ đào tạo lựa chọn **lớp học** muốn xóa và lựa chọn “Xóa Lớp học”
6. Hệ thống hiển thị thông báo khẳng định muốn xóa **Lớp học** khỏi cơ sở dữ liệu hay không
7. Cán bộ đào tạo đồng ý
8. Hệ thống xóa lớp học khỏi cơ sở dữ liệu và hiển thị thông báo Xóa lớp học thành công.

Ngoại lệ:

- 3.1. **Tên (hoặc Mã) Môn học** không tìm thấy trong cơ sở dữ liệu
 - 3.1.1. Hệ thống hiển thị thông báo không tìm thấy và yêu cầu người dùng nhập lại thông tin tìm kiếm
 - 3.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau

Phác thảo giao diện Quản lý xếp lớp:

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

6.2.

Quản lý lớp học

Tên use case:	Quản lý lớp học
Tác nhân chính	Cán bộ đào tạo
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Cán bộ đào tạo xem thông kê các lớp học thành công
Đảm bảo tối thiểu	Trở lại màn hình lựa chọn tùy chọn cơ bản
Kích hoạt	Người dùng chọn chức năng Quản lý lớp học

Chuỗi sự kiện chính	
1.	Cán bộ đào tạo kích hoạt form Quản lý đăng ký học
2.	Hệ thống hiển thị tùy chọn Quản lý lớp học và Quản lý danh sách lớp học
3.	Cán bộ đào tạo lựa chọn Quản lý lớp học
4.	Hệ thống hiển thị form Quản lý lớp học với các thông tin tùy chọn
5.	Cán bộ đào tạo thay đổi các thông tin tùy chọn về Khoa, Chuyên Nghành, Học Kỳ và chọn “Xem thông tin lớp học”
6.	Hệ thống hiển thị thống kê các lớp học tương ứng với các thông tin gồm có Mã môn, Tên môn, Số học phần, Mã lớp học, Ngày bắt đầu, Buổi học, Giờ học, Phòng học, Số đăng ký .

Ngoại lệ

- 5.1. Cán bộ đào tạo lựa chọn Khoa Quản trị kinh doanh với tùy chọn **Học Kỳ IX** (chỉ có với khoa Công nghệ thông tin)
- 5.1.1. Hệ thống hiển thị thông báo Chọn sai **Học kỳ**
- 5.1.2. Cán bộ đào tạo chọn lại và tiếp tục các bước sau

Phác thảo giao diện Quản lý lớp học
6.3. Quản lý danh sách lớp

Tên use case:	Quản lý lớp học
Tác nhân chính	Cán bộ đào tạo
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Cán bộ đào tạo xem thông kê các lớp học thành công
Đảm bảo tối thiểu	Trở lại màn hình lựa chọn tùy chọn cơ bản
Kích hoạt	Người dùng chọn chức năng Quản lý lớp học
Chuỗi sự kiện chính	
1.	Cán bộ đào tạo kích hoạt form Quản lý đăng ký học
2.	Hệ thống hiển thị tùy chọn Quản lý lớp học và Quản lý danh sách lớp học
3.	Cán bộ đào tạo lựa chọn Quản lý danh sách lớp
4.	Hệ thống hiển thị form Quản lý danh sách lớp
5.	Cán bộ đào tạo tìm kiếm danh sách một lớp nào đó thông qua Mã lớp học
6.	Hệ thống hiển thị thông kê danh sách các Sinh viên đã đăng ký học lớp đó cùng các thông tin liên quan về các môn điều kiện đi kèm
7.	Cán bộ đào tạo lựa chọn xóa đối với những Sinh viên không đạt yêu cầu “Pass” các môn điều kiện và chọn Button “Xóa SV”
8.	Hệ thống hiển thị tùy chọn có chắc chắn xóa Sinh viên khỏi danh sách lớp hay không
9.	Cán bộ đào tạo khẳng định xóa

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

10. Hệ thống tự động gửi thông báo tới các **Sinh viên** bị xóa khỏi danh sách và thông báo đã xóa **Sinh viên** thành công

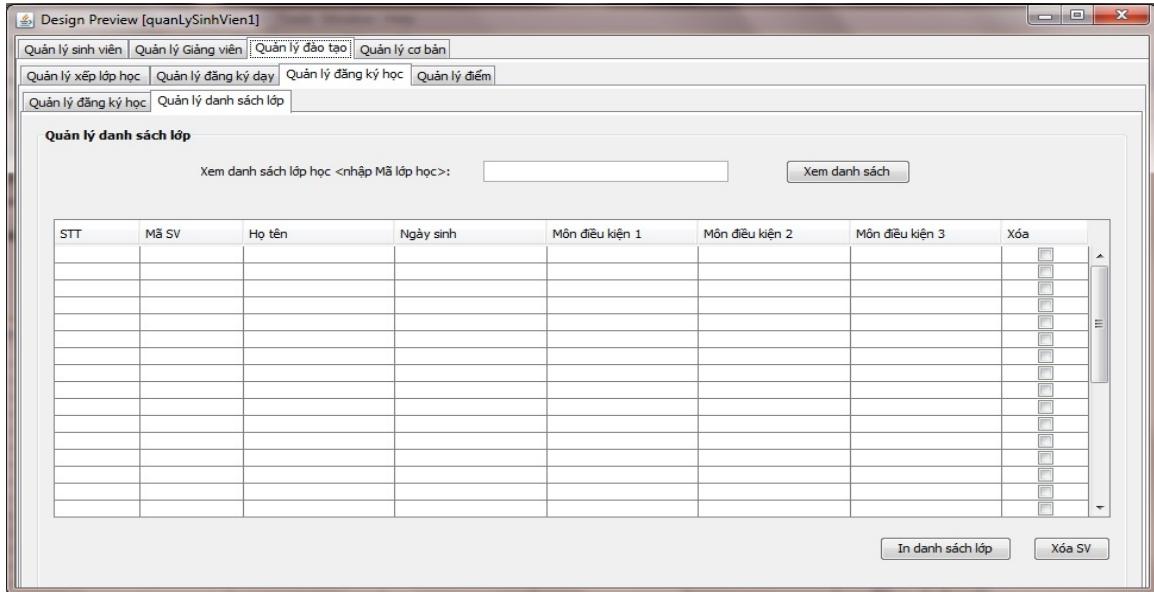
Ngoại lệ

- 5.1. **Mã lớp học** không tìm thấy trong cơ sở dữ liệu

5.1.1. Hệ thống hiển thị thông báo không tìm thấy và yêu cầu người dùng nhập lại thông tin tìm kiếm

5.1.2. Cán bộ đào tạo nhập lại và tiếp tục các bước sau

Phác thảo giao diện Quản lý danh sách lớp

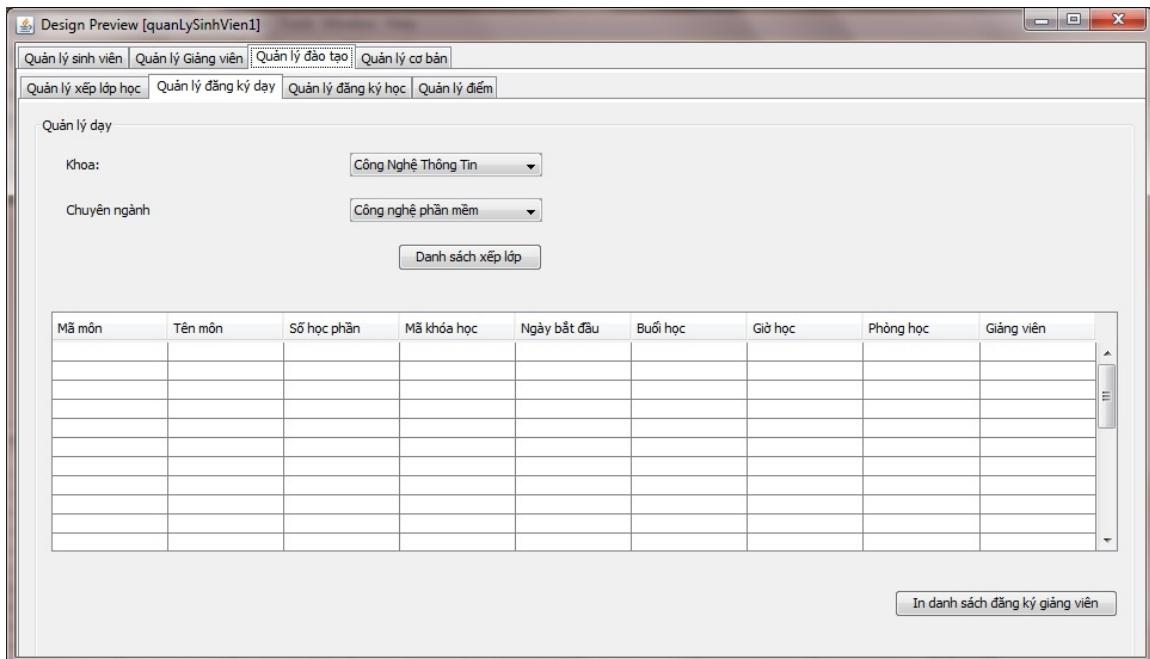


6.4. Quản lý đăng ký dạy

Tên use case:	Quản lý lớp dạy
Tác nhân chính	Cán bộ đào tạo
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Cán bộ đào tạo xem thống kê các lớp học thành công
Đảm bảo tối thiểu	Trở lại màn hình lựa chọn tùy chọn cơ bản
Kích hoạt	Người dùng chọn chức năng Quản lý lớp học
Chuỗi sự kiện chính	
1. Cán bộ đào tạo kích hoạt form Quản lý đăng ký dạy	
2. Hệ thống hiển thị form Quản lý đăng ký dạy với các thông tin tùy chọn	
3. Cán bộ đào tạo thay đổi các thông tin tùy chọn về Khoa , Chuyên Nghành và chọn “Danh sách xếp lớp”	
4. Hệ thống hiển thị thống kê các Lớp học tương ứng với các thông tin gồm có Mã môn , Tên môn , Số học phần , Mã Khóa học , Ngày bắt đầu , Buổi học , Giờ học , Phòng học , Giảng viên .	

Ngoại lệ

- 5.1. Cán bộ đào tạo lựa chọn Khoa Quản trị kinh doanh với tùy chọn **Học Kỳ IX** (chỉ có với khoa Công nghệ thông tin)
- 5.1.1. Hệ thống hiển thị thông báo Chọn sai **Học kỳ**
- 5.1.2. Cán bộ đào tạo chọn lại và tiếp tục các bước sau

Phác thảo giao diện Quản lý đăng ký dạy**7.Giảng viên đăng ký môn dạy**

Tên use case:	Giảng viên đăng ký môn dạy
Ngữ cảnh	Giảng viên đăng ký môn dạy thành công
Tác nhân chính	Giảng viên
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Giảng viên đăng ký thành công môn dạy
Đảm bảo tối thiểu	Trở lại màn hình đăng ký để giảng viên có thể đăng ký lại
Kích hoạt	Người dùng chọn chức năng đăng ký môn dạy

Chuỗi sự kiện chính

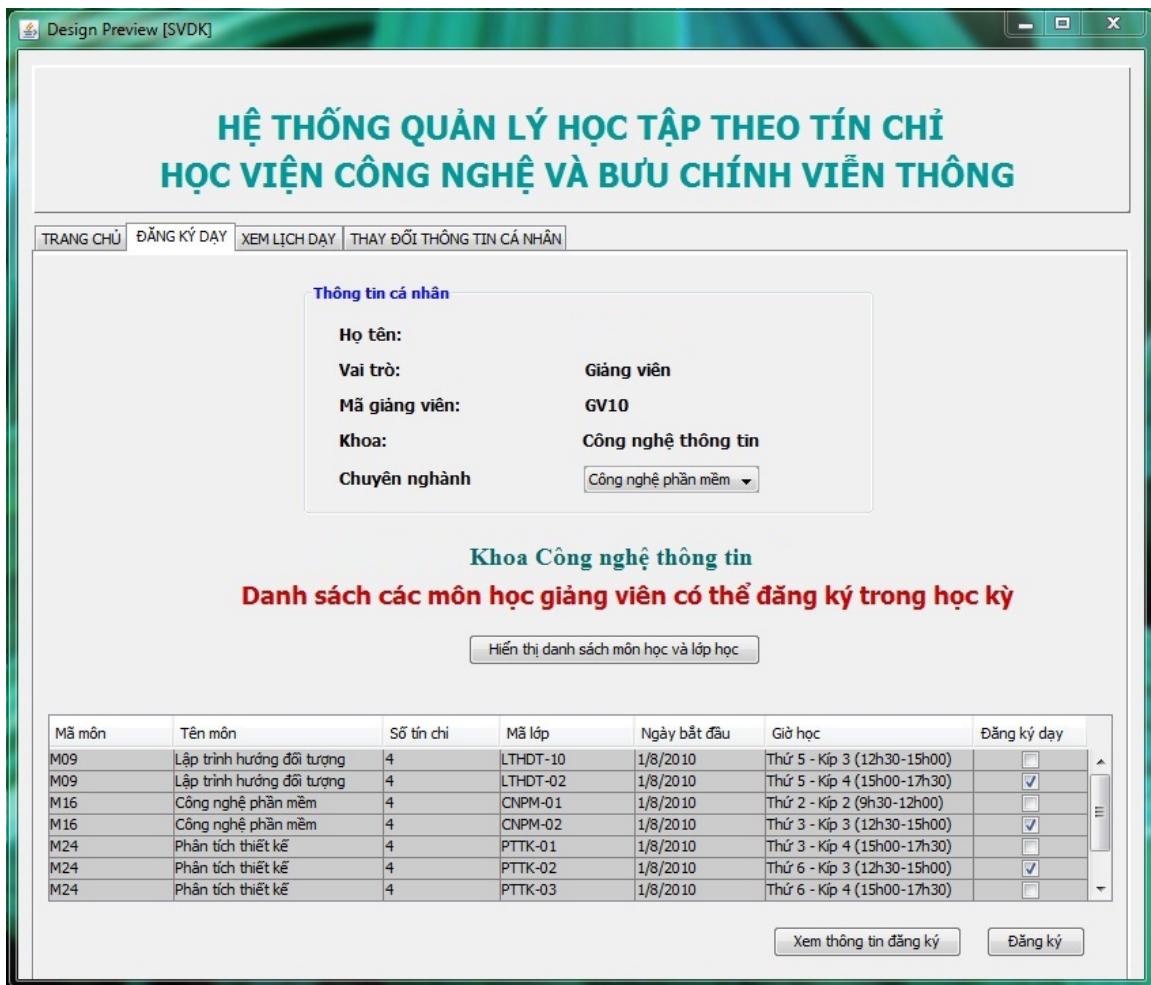
1. Giảng viên kích hoạt **form đăng ký môn dạy**
2. Hệ thống hiển thị danh sách và thông tin chi tiết các **môn** và **khóa học** **giảng viên** có thể đăng ký dạy theo **chuyên ngành**
3. **Giảng viên** lựa chọn các **môn dạy** và **khóa dạy** tương ứng với mỗi môn đó
4. Hệ thống kiểm tra và cập nhật thông tin **giảng viên** đã đăng ký vào cơ sở dữ liệu
5. Hệ thống hiển thị đăng ký thành công
6. Hệ thống thống kê và hiển thị chi tiết thông tin các **môn dạy** và **khóa dạy** mà **giảng viên** đã đăng ký dưới dạng bảng

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Ngoại lệ

- 4.1. Thông tin đăng ký không đủ điều kiện:
 - **Khóa học** đã có **giảng viên** đăng ký trước đó
 - Thời gian một số **môn dạy** giảng viên đăng ký bị trùng nhau
 - Giảng viên (thỉnh giảng) không được dạy môn đăng ký
- 4.2. Hệ thống thông báo đăng ký không thành công và quay lại bước 2

Phác thảo giao diện Giáo viên đăng ký môn dạy



8.Giảng viên xem lịch dạy

Tên use case:	Giảng viên xem lịch dạy
Ngữ cảnh	Giảng viên đăng ký môn dạy thành công
Tác nhân chính	Giảng viên
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Giảng viên đăng ký thành công môn dạy
Đảm bảo tối thiểu	Trở lại màn hình đăng ký để giảng viên có thể đăng kí

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

	lại
Kích hoạt	Người dùng chọn chức năng đăng ký môn dạy
Chuỗi sự kiện chính	<ol style="list-style-type: none"> 1. Giảng viên kích hoạt form Xem lịch dạy 2. Hệ thống hiển thị tùy chọn để Giảng viên lựa chọn Chuyên Ngành 3. Giảng viên lựa chọn Chuyên Ngành tương ứng 4. Hệ thống hiển thị danh sách và thông tin chi tiết các Môn học và Lớp học giảng viên đã đăng ký dạy theo chuyên ngành 5. Giảng viên có thể tùy chọn “Thay đổi” hoặc “In lịch giảng dạy”
Ngoại lệ	

The screenshot shows a Windows application window titled 'Design Preview [SVDK]'. The main title bar reads 'HỆ THỐNG QUẢN LÝ HỌC TẬP THEO TÍN CHỈ HỌC VIỆN CÔNG NGHỆ VÀ BƯU CHÍNH VIỄN THÔNG'. Below the title bar is a menu bar with tabs: TRANG CHỦ, ĐĂNG KÝ DẠY, XEM LỊCH DẠY, THAY ĐỔI THÔNG TIN CÁ NHÂN. The 'XEM LỊCH DẠY' tab is selected.

The main content area contains a 'Thông tin cá nhân' (Personal Information) section with the following details:

Họ tên:	Giảng viên
Vai trò:	Giảng viên
Mã giảng viên:	GV10
Khoa:	Công nghệ thông tin
Chuyên ngành:	Công nghệ phần mềm
Đăng ký dạy các môn khóa:	2007

Below this is a red header 'Thời khóa biểu các môn học và thông tin chi tiết lớp học' (Class schedule and detailed class information). Underneath are two buttons: 'Xem thời khóa biểu' (View class schedule) and 'In thời khóa biểu' (Print class schedule).

The bottom part of the window displays a table titled 'Thời khóa biểu các môn học và thông tin chi tiết lớp học' (Class schedule and detailed class information) with the following data:

Mã môn	Tên môn	Số tín chỉ	Mã lớp	Ngày bắt đầu	Giờ học	Phòng học
M16	Công nghệ phần mềm	4	CNPM-01	1/8/2010	Thứ 2 - Kíp 2 (9h30-12h00)	305A3
M16	Công nghệ phần mềm	4	CNPM-02	1/8/2010	Thứ 3 - Kíp 3 (12h30-15h00)	311A3
M24	Phân tích thiết kế	4	PTTK-01	1/8/2010	Thứ 3 - Kíp 3 (12h30-15h00)	209A3
M24	Phân tích thiết kế	4	PTTK-02	1/8/2010	Thứ 6 - Kíp 3 (12h30-15h00)	309A3
M24	Phân tích thiết kế	4	PTTK-03	1/8/2010	Thứ 6 - Kíp 4 (15h00-17h30)	305A3
M32	Phát triển phần mềm hướng Agent	4	AGENT-01	1/8/2010	Thứ 2 - Kíp 3 (12h30-15h00)	211A3
M32	Phát triển phần mềm hướng Agent	4	AGENT-02	1/8/2010	Thứ 5 - Kíp 4 (15h00-17h30)	305A3

Phác thảo giao diện Giảng viên xem lịch giảng dạy

9.Sinh viên đăng ký môn học

Tên use case:	Sinh viên đăng ký môn học
Tác nhân chính	Sinh viên
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Sinh viên đăng ký thành công môn học
Đảm bảo tối thiểu	Trở lại màn hình đăng ký để sinh viên có thể đăng ký lại

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Kích hoạt	Người dùng chọn chức năng đăng ký môn học
Chuỗi sự kiện chính	
1. Sinh viên kích hoạt form đăng ký môn học	
2. Hệ thống hiển thị danh sách và thông tin chi tiết các khóa học Sinh viên có thể đăng ký học trong học kỳ gồm có thông tin về Mã môn , Tên môn , Số tín chỉ , Loại môn học , Môn điều kiện , Môn nợ .	
3. Sinh viên lựa chọn các môn học và khóa học tương ứng với mỗi môn học	
4. Hệ thống kiểm tra và cập nhật thông tin Sinh viên đã đăng ký vào cơ sở dữ liệu	
5. Hệ thống hiển thị đăng ký thành công	
6. Hệ thống thống kê và hiển thị chi tiết thông tin các môn học và khóa học mà Sinh viên đã đăng ký dưới dạng bảng	
Ngoại lệ	
4.3. Thông tin đăng ký không đủ điều kiện:	
<ul style="list-style-type: none">• Sinh viên chưa học đủ các môn điều kiện của môn học• Môn học đã đủ lớp và đủ số lượng Sinh viên trong 1 lớp.• Sinh viên đăng ký hai khóa học của cùng một môn học• Thời gian một số buổi học môn học Sinh viên đăng ký bị trùng nhau	
4.4. Hệ thống thông báo đăng ký không thành công và quay lại bước 2	

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

**HỆ THỐNG QUẢN LÝ HỌC TẬP THEO TÍN CHỈ
HỌC VIỆN CÔNG NGHỆ VÀ BƯU CHÍNH VIỄN THÔNG**

TRANG CHỦ | THÔNG TIN KHOA/CHUYÊN NGHÀNH | ĐĂNG KÝ HỌC | ĐĂNG KÝ THI | XEM LỊCH THI | CHƯƠNG TRÌNH HỌC | ĐỔI THÔNG TIN CÁ NHÂN

Thông tin cá nhân <p>Họ tên: _____ Vai trò: Sinh viên Mã sinh viên: _____ Khoa: Công nghệ thông tin Chuyên ngành: Công nghệ phần mềm</p>	Thông tin đăng ký học <p>Số tín chỉ tối thiểu: 10 Số tín chỉ tối đa: 35 Hạn đăng ký: 10/7/2010 -> 10/8/2010 Đăng ký học cùng khóa: <input type="button" value="2007"/></p>																																																																								
Khoa Công nghệ thông tin - 2007 Danh sách các môn học sinh viên có thể đăng ký trong học kỳ <small>Sinh viên chỉ được phép đăng ký học một môn khi đã hoàn thành đủ các môn điều kiện</small> <input type="button" value="Hiển thị danh sách môn học"/>																																																																									
Bạn đã đăng ký 12 TC trên tổng số tối thiểu 10 TC và tối đa 35 TC <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>Mã môn</th> <th>Tên môn</th> <th>Số tín chỉ</th> <th>Khoa phụ trách</th> <th>Bộ môn phụ trách</th> <th>Loại môn học</th> <th>Các môn điều kiện</th> <th>Môn điều kiện nợ</th> <th>Đăng k...</th> </tr> </thead> <tbody> <tr> <td>M24</td> <td>Phân tích thiết kế</td> <td>4</td> <td>Công nghệ thông tin</td> <td>Công nghệ phần mềm</td> <td>Bắt buộc</td> <td>Lập trình Java, Công nghệ p...</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>M25</td> <td>Lập trình mạng</td> <td>4</td> <td>Công nghệ thông tin</td> <td>Công nghệ phần mềm</td> <td>Bắt buộc</td> <td>Lập trình Java, Mạng máy tính</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>M26</td> <td>Cơ sở dữ liệu phân ...</td> <td>4</td> <td>Công nghệ thông tin</td> <td>Công nghệ phần mềm</td> <td>Bắt buộc</td> <td>Cơ sở dữ liệu</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>M27</td> <td>An toàn bảo mật H...</td> <td>3</td> <td>Công nghệ thông tin</td> <td>Hệ thống thông tin</td> <td>Bắt buộc</td> <td>Hệ thống thông tin, Mạng m...</td> <td><input checked="" type="checkbox"/></td> <td><input checked="" type="checkbox"/></td> </tr> <tr> <td>M28</td> <td>Xử lý ảnh</td> <td>3</td> <td>Công nghệ thông tin</td> <td>Công nghệ phần mềm</td> <td>Tùy chọn</td> <td>Kỹ thuật đồ họa, Giải tích 1, ...</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>M29</td> <td>Lịch sử Đảng</td> <td>4</td> <td>Khoa cơ bản</td> <td>Chính trị tư tưởng</td> <td>Tùy chọn</td> <td>Giải tích 2, Triết học, Chủ nghĩa xã hội</td> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> <tr> <td>M30</td> <td>Tư tưởng Hồ Chí Minh</td> <td>4</td> <td>Khoa cơ bản</td> <td>Chính trị tư tưởng</td> <td>Tùy chọn</td> <td>Triết học, Chủ nghĩa xã hội</td> <td><input checked="" type="checkbox"/></td> <td><input type="checkbox"/></td> </tr> </tbody> </table>		Mã môn	Tên môn	Số tín chỉ	Khoa phụ trách	Bộ môn phụ trách	Loại môn học	Các môn điều kiện	Môn điều kiện nợ	Đăng k...	M24	Phân tích thiết kế	4	Công nghệ thông tin	Công nghệ phần mềm	Bắt buộc	Lập trình Java, Công nghệ p...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	M25	Lập trình mạng	4	Công nghệ thông tin	Công nghệ phần mềm	Bắt buộc	Lập trình Java, Mạng máy tính	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	M26	Cơ sở dữ liệu phân ...	4	Công nghệ thông tin	Công nghệ phần mềm	Bắt buộc	Cơ sở dữ liệu	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	M27	An toàn bảo mật H...	3	Công nghệ thông tin	Hệ thống thông tin	Bắt buộc	Hệ thống thông tin, Mạng m...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	M28	Xử lý ảnh	3	Công nghệ thông tin	Công nghệ phần mềm	Tùy chọn	Kỹ thuật đồ họa, Giải tích 1, ...	<input type="checkbox"/>	<input type="checkbox"/>	M29	Lịch sử Đảng	4	Khoa cơ bản	Chính trị tư tưởng	Tùy chọn	Giải tích 2, Triết học, Chủ nghĩa xã hội	<input type="checkbox"/>	<input type="checkbox"/>	M30	Tư tưởng Hồ Chí Minh	4	Khoa cơ bản	Chính trị tư tưởng	Tùy chọn	Triết học, Chủ nghĩa xã hội	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Mã môn	Tên môn	Số tín chỉ	Khoa phụ trách	Bộ môn phụ trách	Loại môn học	Các môn điều kiện	Môn điều kiện nợ	Đăng k...																																																																	
M24	Phân tích thiết kế	4	Công nghệ thông tin	Công nghệ phần mềm	Bắt buộc	Lập trình Java, Công nghệ p...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>																																																																	
M25	Lập trình mạng	4	Công nghệ thông tin	Công nghệ phần mềm	Bắt buộc	Lập trình Java, Mạng máy tính	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>																																																																	
M26	Cơ sở dữ liệu phân ...	4	Công nghệ thông tin	Công nghệ phần mềm	Bắt buộc	Cơ sở dữ liệu	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>																																																																	
M27	An toàn bảo mật H...	3	Công nghệ thông tin	Hệ thống thông tin	Bắt buộc	Hệ thống thông tin, Mạng m...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>																																																																	
M28	Xử lý ảnh	3	Công nghệ thông tin	Công nghệ phần mềm	Tùy chọn	Kỹ thuật đồ họa, Giải tích 1, ...	<input type="checkbox"/>	<input type="checkbox"/>																																																																	
M29	Lịch sử Đảng	4	Khoa cơ bản	Chính trị tư tưởng	Tùy chọn	Giải tích 2, Triết học, Chủ nghĩa xã hội	<input type="checkbox"/>	<input type="checkbox"/>																																																																	
M30	Tư tưởng Hồ Chí Minh	4	Khoa cơ bản	Chính trị tư tưởng	Tùy chọn	Triết học, Chủ nghĩa xã hội	<input checked="" type="checkbox"/>	<input type="checkbox"/>																																																																	
<small><Sinh viên có thể đăng ký học các môn điều kiện còn thiếu cùng với khóa sau></small> <input type="button" value="Đăng ký"/>																																																																									

Phác thảo giao diện sinh viên đăng ký môn học

10. Sinh viên xem thời khóa biểu

Tên use case:	Sinh viên xem thời khóa biểu
Tác nhân chính	Sinh viên
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Sinh viên xem thời khóa biểu thành công
Đảm bảo tối thiểu	Trở lại màn hình chính
Kích hoạt	Người dùng chọn chức năng xem thời khóa biểu
Chuỗi sự kiện chính	
1. Sinh viên kích hoạt form Xem thời khóa biểu	
2. Hệ thống hiển thị thời khóa biểu tương ứng với những môn học Sinh viên đã đăng ký học trong học kỳ với các thông tin Mã môn, Tên môn, Số tín chỉ, Mã lớp, Ngày bắt đầu, Giờ học, Phòng học, Giảng viên.	
Ngoại lệ	2.1. Môn Sinh viên đăng ký không được xếp vào thời khóa biểu do Sinh viên chưa trả nợ đủ các môn điều kiện

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Phác thảo giao diện Xem thời khóa biểu

The screenshot shows a software application window titled 'HỆ THỐNG QUẢN LÝ HỌC TẬP THEO TÍN CHỈ HỌC VIỆN CÔNG NGHỆ VÀ BƯU CHÍNH VIỄN THÔNG'. The main menu bar includes 'TRANG CHỦ', 'ĐĂNG KÝ HỌC', 'XEM THỜI KHÓA BIỂU', 'XEM LỊCH THI', 'XEM ĐIỂM HỌC TẬP', and 'THAY ĐỔI THÔNG TIN CÁ NHÂN'. Below the menu is a section for 'Thông tin cá nhân' (Personal Information) with fields for 'Họ tên' (Name), 'Vai trò' (Role - Student), 'Mã sinh viên' (Student ID), 'Khoa: Công nghệ thông tin' (Faculty - Information Technology), 'Chuyên ngành: Công nghệ phần mềm' (Major - Software Engineering), and 'Khóa học: 2007'. A red header 'Thời khóa biểu các môn học và thông tin chi tiết lớp học' (Class Timetable and Detailed Class Information) is followed by two buttons: 'Xem thời khóa biểu' (View Timetable) and 'In thời khóa biểu' (Print Timetable). A table below lists course details:

Mã môn	Tên môn	Số tín chỉ	Mã lớp	Ngày bắt đầu	Giờ học	Phòng học	Giảng viên
M24	Phân tích thiết kế	4	PTTK-02	12/08/2010	Kíp 4 (15h00-17h30)	305A3	Trần Đình Quế
M25	Cơ sở dữ liệu phân tán	4	CSDLPT-02	13/08/2010	Kíp 2 (9h30-12h00)	311A3	Phạm Thế Quế
M27	Quản lý dự án	4	QLDA-01	14/08/2010	Kíp 3 (12h30-15h00)	305A3	Nguyễn Quỳnh Chi
M28	An toàn bảo mật	3	ATBM-02	15/08/2010	Kíp 4 (15h00-17h30)	205A3	Dương Trần Đức
M30	Lập trình mạng	3	LTM-01	11/08/2010	Kíp 2 (9h30-12h00)	305A3	Hà Mạnh Đào
M31	Xử lý ảnh	3	XLA-01	11/08/2010	Kíp 3(12h30-15h00)	305A3	Đỗ Năng Toàn

11. Sinh viên xem lịch thi học kỳ

Tên use case:	Sinh viên xem lịch thi học kỳ
Tác nhân chính	Sinh viên
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Sinh viên xem lịch thi thành công
Đảm bảo tối thiểu	Trở lại màn hình chính
Kích hoạt	Người dùng chọn chức năng xem lịch thi học kỳ
Chuỗi sự kiện chính	
1. Sinh viên kích hoạt form Xem lịch thi học kỳ	
2. Hệ thống hiển thị lịch thi các môn tương ứng với những môn học Sinh viên đã đăng ký học trong học kỳ với các thông tin Mã môn , Tên môn , Ngày thi , Giờ thi , Phòng thi .	
Ngoại lệ	
2.1. Môn Sinh viên theo học không được xếp lịch thi do Sinh viên ko đủ điều kiện thi.	

Phác thảo giao diện Xem lịch thi

12. Sinh viên xem điểm học tập

Tên use case:	Sinh viên xem điểm học tập
Tác nhân chính	Sinh viên
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Sinh viên xem điểm học tập thành công
Đảm bảo tối thiểu	Trở lại màn hình chính
Kích hoạt	Người dùng chọn chức năng xem điểm
Chuỗi sự kiện chính	
	1. Sinh viên kích hoạt form Xem điểm học tập 2. Hệ thống hiển thị các thông tin về kết quả học tập của Sinh viên với các thông tin Tên môn, Số tín chỉ, Điểm lần 1, Điểm lần 1, Điểm Tổng kết.
Ngoại lệ	

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

**HỆ THỐNG QUẢN LÝ HỌC TẬP THEO TÍN CHỈ
HỌC VIỆN CÔNG NGHỆ VÀ BƯỚU CHÍNH VIỄN THÔNG**

TRANG CHỦ | ĐĂNG KÝ HỌC | XEM THỜI KHÓA BIỂU | XEM LỊCH THI | THAY ĐỔI THÔNG TIN CÁ NHÂN | XEM ĐIỂM HỌC TẬP

Thông tin cá nhân

Họ tên:	Khoa: Công nghệ thông tin
Vai trò: Sinh viên	Chuyên ngành: Công nghệ phần mềm
Mã sinh viên:	Khóa học: 2007

Danh sách các môn học và kết quả học tập

Xem bảng điểm | In bảng điểm

Tên môn	Số tín chỉ	Điểm thi lần 1	Điểm thi lần 2	Điểm TK

Điểm trung bình môn hiện tại của sinh viên là ...

13. Người dùng thay đổi thông tin cá nhân

Tên use case:	Người dùng thay đổi thông tin cá nhân
Tác nhân chính	Người dùng hệ thống
Tiền điều kiện	Đăng nhập hệ thống thành công
Đảm bảo thành công	Người dùng thay đổi thông tin thành công
Đảm bảo tối thiểu	Trở lại màn hình chính
Kích hoạt	Người dùng chọn chức năng thay đổi thông tin cá nhân
Chuỗi sự kiện chính	
1. Sinh viên kích hoạt form Thay đổi thông tin cá nhân	
2. Hệ thống hiển thị các thông tin cá nhân chi tiết của người dùng	
3. Người dùng lựa chọn “Thay đổi thông tin”	
4. Hệ thống hiển thị các thông tin chi tiết của người dùng trên form nhập liệu	
5. Người dùng thay đổi thông tin và chọn “Lưu thông tin”	
6. Hệ thống thông báo thông tin đã được thay đổi thành công	
Ngoại lệ	

Phác thảo giao diện thay đổi Thông tin cá nhân

The screenshot shows a software interface for managing student information. At the top, there's a menu bar with links like 'TRANG CHỦ', 'ĐĂNG KÝ HỌC', 'XEM THỜI KHÓA BIỂU', 'XEM LỊCH THI', 'XEM ĐIỂM HỌC TẬP', and 'THAY ĐỔI THÔNG TIN CÁ NHÂN'. The main area is titled 'HỆ THỐNG QUẢN LÝ HỌC TẬP THEO TÍN CHỈ HỌC VIỆN CÔNG NGHỆ VÀ BƯU CHÍNH VIỄN THÔNG'. On the left, there's a preview section labeled 'Thông tin cá nhân' showing current information: Họ tên: [redacted], Ngày sinh: [redacted], Giới tính: Nữ, Vai trò: Sinh viên, Mã sinh viên: D241, Khoa: Công nghệ thông tin, Chuyên ngành: Công nghệ phần mềm, Email: [redacted], Quê quán: [redacted]. Below this is a 'Thay đổi thông tin' section with input fields for the same fields, with 'Giới tính' having a dropdown menu set to 'Nữ'. There are 'Thay đổi thông tin cá nhân' and 'Lưu thông tin' buttons at the bottom.

5.4.2. Phân tích tĩnh (Static Analysis)

5.4.2.1 Xác định lớp (lớp thực thể)

Để xác định các lớp thực thể ta dùng kỹ thuật trích danh từ trong usecase và scenario. Các danh từ thu được là:

Hệ thống học tập tín chỉ, Khoa, Mã Khoa, Tên Khoa, Trưởng Khoa, Chuyên Ngành, Mã chuyên ngành, Tên Chuyên ngành, Trưởng bộ môn, Khoa phụ trách, Môn Học, Mã môn, Tên môn, Số tín chỉ, Số tiết Lý thuyết, Số tiết Thực hành, Số tiết bài tập, Môn điều kiện, Sinh viên, Mã SV, Họ tên, Ngày sinh, Giới tính, Khóa, Lớp, Giảng viên, Mã GV, Họ tên, Ngày sinh, Giới tính, Bộ Môn, Học vị , Mã môn, Tên môn, Mã lớp, Ngày bắt đầu, Buổi học, Giờ học, Phòng học, Sĩ Số, Số đăng ký, thời khóa biểu , lịch thi, kết quả học tập.

Loại bỏ các danh từ nằm ngoài phạm vi mục đích của hệ thống và loại bỏ các danh từ hoặc cụm từ trùng lặp và các danh từ làm thuộc tính của lớp như:

Tên, mã, ngày sinh, địa chỉ, giới tính: là thuộc tính của các lớp Người, Sinh viên, Giáo viên, Người quản lý.

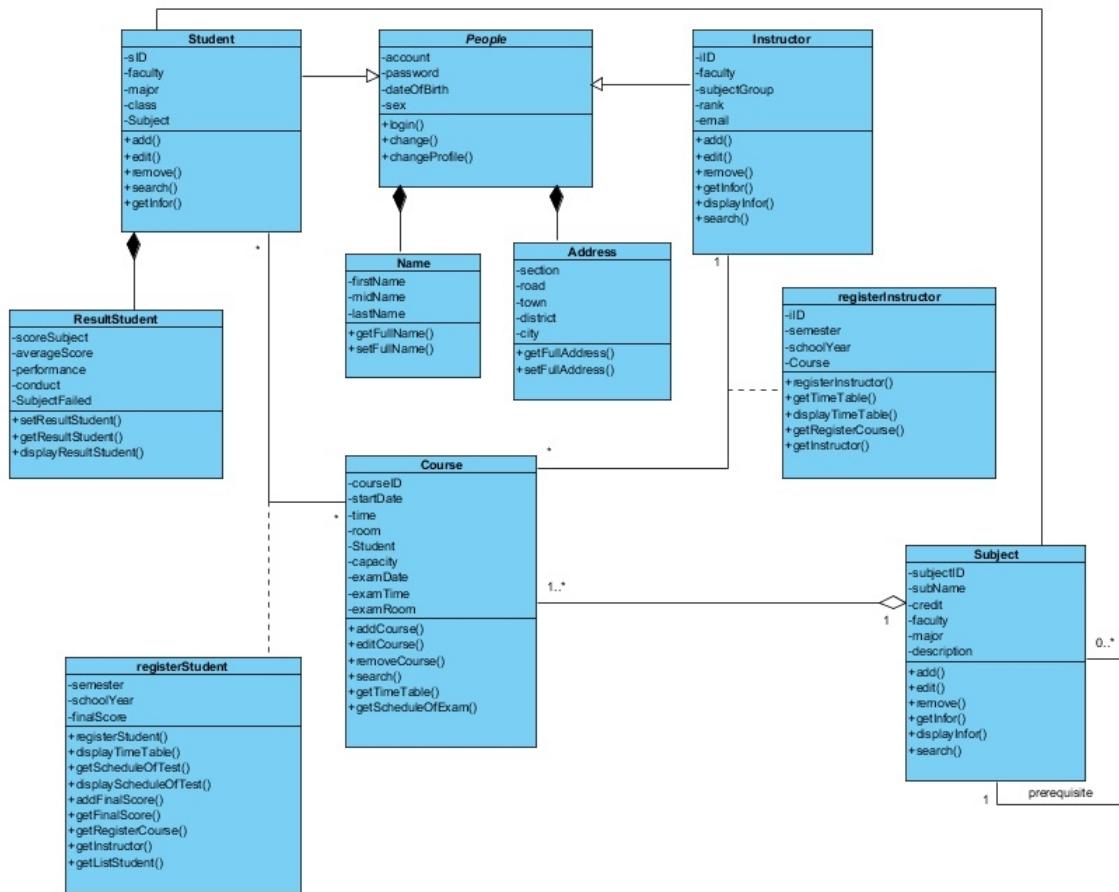
Ngành, khoa: là thuộc tính của Sinh viên, Giáo viên.

Loại môn học, số tiết, học kỳ, số tín chỉ: là thuộc tính của lớp Môn học.

Vậy các danh từ sau là các lớp thực thể: **Người, Họ tên, Địa chỉ, Sinh viên, Giảng viên, Môn học, Đăng ký môn học, Đăng ký môn dạy, Lớp học phần.**

Mối quan hệ giữa các lớp

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



5.4.2.2 Xác định thuộc tính và phương thức cho các lớp thực thể

	Định nghĩa	Chứa các thuộc tính cơ bản của các đối tượng là Người trong hệ thống
People -account : String -password : String -dateOfBirth : Date -sex : bool +login(account, password) : boolean +change(password) : void +changeProfile(account) : void	Thuộc tính	<ul style="list-style-type: none"> ❖ account: tên tài khoản đăng nhập vào hệ thống. ❖ password: mật khẩu đăng nhập hệ thống ❖ dateOfBirth: ngày tháng năm sinh ❖ sex: giới tính: Nam = 1 ; Nữ = 0
	Phương thức	<ul style="list-style-type: none"> ❖ login(account,password): mỗi người sử dụng hệ thống đều phải đăng nhập sử dụng tài khoản và mật khẩu riêng của mình. Phương thức này trả về giá trị True nếu đăng nhập thành công, False nếu đăng nhập không thành công ❖ change(password): người dùng sau khi đăng nhập thành công có thể thực hiện đổi mật khẩu ❖ changeProfile(account): người dùng sau khi đăng nhập thành công có thể thay đổi thông tin cá nhân của mình

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

<pre> Name -firstName : String -midName : String -lastName : String +getFullName() : void +setFullName() : void </pre>	Định nghĩa	Có quan hệ kiểu hợp thành (composition) với lớp Người, việc tách thành lớp Họ tên phục vụ cho việc quản lý và tìm kiếm dễ dàng hơn
	Thuộc tính	❖ firstName, midName, lastName : ba thuộc tính Họ, Đệm, Tên tượng ứng với từng trường trong họ tên đầy đủ
	Phương thức	❖ getFullName() : lấy ra Họ tên đầy đủ ❖ setFullName() : gán các trường thành Họ tên đầy đủ
<pre> Address -section : String -road : String -town : String -district : String -city : String +getFullAddress() : void +setFullAddress() : void </pre>	Định nghĩa	Có quan hệ kiểu hợp thành (composition) với lớp Người, , việc tách thành lớp Địa chỉ phục vụ cho việc quản lý và tìm kiếm dễ dàng hơn
	Thuộc tính	❖ section : số nhà ❖ road : đường/phố ❖ town : phường ❖ district : quận ❖ city : thành phố
	Phương thức	❖ getFullAddress() : lấy ra địa chỉ đầy đủ ❖ setFullAddress() : gán các trường thành địa chỉ đầy đủ
<pre> Student -sID : String -faculty : String -major : String -class : String +add(Student) : void +remove(Student) : void +edit(Student) : void +getInfor(Student) : void +displayInfor(Student) : void +search(Student) : void </pre>	Định nghĩa	Lớp Sinh Viên sẽ kế thừa từ lớp Người và mang đầy đủ thuộc tính của lớp Người
	Thuộc tính	❖ sID : mã sinh viên ❖ faculty : khoa sinh viên theo học ❖ major : chuyên ngành học tập của sinh viên ❖ class : mỗi sinh viên sau khi nhập học sẽ được xếp vào một lớp để quản lý
	Phương thức	❖ add(Student) : thêm sinh viên vào trong cơ sở dữ liệu ❖ edit(Student) : sửa thông tin sinh viên trong cơ sở dữ liệu ❖ remove(Student) : xóa sinh viên khỏi cơ sở dữ liệu ❖ getInfor(Student) : lấy thông tin của sinh viên trong cơ sở dữ liệu ❖ displayInfor(Student) : hiển thị thông tin sinh viên trên giao diện. ❖ search(Student) : tìm kiếm sinh viên
	Định nghĩa	Lớp Giảng Viên sẽ kế thừa từ lớp Người và mang đầy đủ thuộc tính của lớp Người
	Thuộc tính	❖ iID : mã Giảng Viên ❖ faculty : giảng viên thuộc khoa nào ❖ subjectGroup : Bộ môn

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

Instructor - ID -faculty : String -subjectGroup : String -rank : String -email : String + add(Instructor) : void + edit(Instructor) : void + remove(Instructor) : void + getInfor(Instructor) : void + displayInfor(Instructor) : void + search(Instructor) : void		❖ rank : học vị của giảng viên ❖ email : địa chỉ email giảng viên
	Phương thức	❖ add(Instructor) : thêm giảng viên vào trong cơ sở dữ liệu ❖ edit(Instructor) : sửa thông tin giảng viên trong cơ sở dữ liệu ❖ remove(Instructor) : xóa giảng viên khỏi cơ sở dữ liệu ❖ getInfor(Instructor) : lấy thông tin của giảng viên trong cơ sở dữ liệu ❖ displayInfor(Instructor) : Hiển thị thông tin Giảng viên trên giao diện ❖ search(Instructor) : tìm kiếm Giảng Viên
Course -courseID : String -startDate : Date -time : String -room : String -Student : ArrayList<Student> -capacity : int -examDate : Date -examTime : String -examRoom : String + addCourse(Subject) : void + editCourse(Subject) : void + removeCourse(Subject) : void + search(Course) : void + getTimeTable(Course) + getScheduleOfExam(Course)	Định nghĩa	Mỗi sinh viên khi đăng ký học một môn sẽ tương ứng với một Lớp giảng . Tương tự mỗi Giảng Viên cũng có thể đăng ký dạy theo các Lớp giảng phù hợp tương ứng với môn giảng viên đăng ký dạy.
	Thuộc tính	❖ courseID : mã Lớp giảng tương ứng ❖ startDate : ngày bắt đầu học ❖ time : thời gian học ❖ room : phòng học ❖ capacity : sĩ số lớp học ❖ Student : mảng danh sách các sinh viên tham gia lớp giảng ❖ examDate : ngày thi kết thúc môn học ❖ examTime : giờ thi kết thúc môn học ❖ examRoom : phòng thi kết thúc môn học
	Phương thức	❖ addCourse(Subject) : thêm một lớp giảng mới tương ứng với một Môn học. ❖ editCourse(Subject) : sửa thông tin một lớp giảng ❖ removeCourse(Subject) : xóa một lớp giảng trong cơ sở dữ liệu ❖ getInfor(Course) : Lấy thông tin chi tiết của toàn bộ lớp giảng bao gồm môn học, ngày học, giờ học, phòng học ❖ search(Course) : tìm kiếm thông tin lớp giảng ❖ getTimeTable(Course) : lấy thông tin về thời gian và phòng học của lớp học ❖ getScheduleOfExam(Course) : lấy

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

		thông tin về thời gian và phòng thi cuối kỳ
<pre> Subject -subjectID : String -subName : String -credit : int -faculty : String -major : String +addSubject() : void +editSubject() : void +removeSubject() : void +getInfor(Subject) : void +displayInfor(Subject) : void +search(Subject) : void </pre>	<p>Định nghĩa</p> <p>Sinh viên sẽ phải theo học các môn học trong mỗi theo thứ tự trong mỗi học kỳ để tích lũy tín chỉ. Thông tin chi tiết về các môn được thể hiện qua các thuộc tính và phương thức của lớp Môn Học</p> <p>Thuộc tính</p> <ul style="list-style-type: none"> ❖ subjectID: mã môn học ❖ subName: tên môn học ❖ credit: số tín chỉ của môn học ❖ faculty: môn học thuộc khoa nào ❖ major: môn học thuộc chuyên ngành nào <p>Phương thức</p> <ul style="list-style-type: none"> ❖ add(Subject): thêm môn học vào cơ sở dữ liệu ❖ edit(Subject): sửa lại các thông tin về môn học ❖ remove(Subject): xóa môn học khỏi cơ sở dữ liệu ❖ getInfor(Subject): lấy thông tin chi tiết về môn học. ❖ displayInfor(Subject): hiển thị thông tin môn học trên giao diện ❖ search(Subject): tìm kiếm thông tin Môn học 	
<pre> DetailOfSubject -subjectID : String -classHourTheory : String -classHourPractice : String -description : String +getDetailSubject() : void +editDetailOfSubject() : void </pre>	<p>Định nghĩa</p> <p>Vì thuộc tính của Môn học nhiều, trong thực tế không phải lúc nào cũng cần truy cập đến, các thuộc tính như hình thức thi, số tiết lý thuyết, bài tập... sinh viên chỉ quan tâm khi đã đăng ký học khóa học môn đó trong học kỳ, nên tách thêm lớp ChiTietMonHoc</p> <p>Thuộc tính</p> <ul style="list-style-type: none"> ❖ subjectID: mã môn học ❖ classHourTheory: số tiết lý thuyết ❖ classHourPractice: số tiết thực hành <p>Phương thức</p> <ul style="list-style-type: none"> ❖ getDetailSubject(): xem thông tin Chi tiết về Môn học ❖ setDetailOfSubject(): sửa thông tin Chi tiết về Môn học 	
	<p>Định nghĩa</p> <p>Là lớp liên kết giữa lớp Sinh Viên và Khóa học, chứa các phương thức gọi tới thuộc tính của cả hai lớp Sinh Viên và Khóa học.</p> <p>Thuộc tính</p> <ul style="list-style-type: none"> ❖ semester: học kỳ ❖ schoolYear: năm học ❖ finalScore: điểm tổng kết cuối cùng của môn học sinh viên đăng ký 	

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

<pre> registerStudent -semester : String -schoolYear : int -finalScore : float +registerStudent(Course) : void +displayTimeTable(Student) : void +getScheduleOfTest(Subject) : void +displayScheduleOfTest(Subject) : void +addFinalScore(sID, subjectID) : void +getFinalScore(sID, subjectID) : void +getRegisterCourse(Student) : void +getInstructor(Course) : void +getListStudent(Course) : void </pre>	Phương thức	<ul style="list-style-type: none"> ❖ enrollStudent(Course): đăng ký học khóa học của một môn nào đó. Tên sinh viên sẽ được thêm vào danh sách sinh viên của khóa học đó và cập nhật vào cơ sở dữ liệu ❖ displayTimeTable(Student): hiển thị thời khóa biểu học các môn của sinh viên trên giao diện ❖ getScheduleOfTest(Subject): lấy lịch thi của các môn học mà sinh viên đăng ký học trong học kỳ ❖ displayScheduleOfTest(Subject): hiển thị lịch thi học kỳ trên giao diện ❖ addFinalScore(sID,subjected): thêm điểm tổng cuối kỳ của môn học Sinh viên đăng ký ❖ getFinalSocre(sID,subjectID): lấy điểm tổng kết cuối cùng của môn học tương ứng với lớp học Sinh viên đăng ký ❖ getRegisterCourse(Student): lấy những khóa học mà sinh viên đã từng đăng ký học ❖ getListStudent(Course): lấy danh sách những sinh viên trong một lớp học.
<pre> registerInstructor -iID : String -semester : String -schoolYear : int -Course : ArrayList<Course> +registerInstructor(Course) : void +getTimeTable(Instructor) : void +displayTimeTable(Instructor) : void +getRegisterCourse(Instructor) +getInstructor(Course) : void </pre>	Định nghĩa Thuộc tính Phương thức	<p>Là lớp liên kết giữa lớp Giảng Viên và Khóa học, chứa các phương thức gọi tới thuộc tính của cả hai lớp Giảng Viên và Khóa học</p> <ul style="list-style-type: none"> ❖ iID: mã giảng viên ❖ semester: học kỳ ❖ schoolYear: năm học ❖ Course: danh sách các Lớp giảng mà giảng viên đăng ký dạy <ul style="list-style-type: none"> ❖ setInstructor(Course): giảng viên đăng ký dạy Lớp giảng của một môn nào đó. Phương thức này sẽ gán Mã giảng viên vào trong cơ sở dữ liệu tương ứng với trường Giảng viên của mỗi bản ghi về Lớp giảng ❖ getTimeTable(Instructor): lấy thông tin về lịch dạy của các Lớp giảng mà Giảng viên đã đăng ký ❖ displayTimeTable(Instructor): hiển thị lịch dạy của giảng viên trên giao diện ❖ getRegisterCourse(Instructor): lấy danh sách những lớp học mà giảng

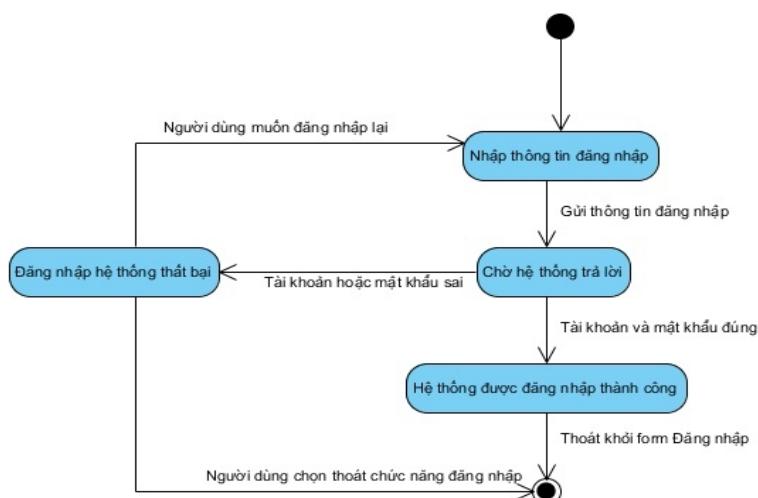
CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

		<p>viên đã đăng ký dạy</p> <ul style="list-style-type: none"> ❖ getInstructor(Course): lấy thông tin về giảng viên của một lớp học
<pre> ResultStudent -scoreSubject : ArrayList<Subject> -averageScore : float -performance : String -conduct : String -SubjectFailed : ArrayList<Subject> +setResultStudent(sID) : void +getResultStudent(sID) : void +displayResultStudent(sID) : void </pre>	Định nghĩa	Vì thuộc tính của sinh viên rất nhiều, trong khi kết quả họ tập là những thuộc tính không phải lúc nào cũng cần truy cập đến trong hệ thống quản lý họa tập theo tín chỉ, do vậy tách ra thành một lớp riêng KetQuaHocTap
	Thuộc tính	<ul style="list-style-type: none"> ❖ scoreSubject: một mảng các môn học sinh viên đã pass và điểm tổng kết tương ứng các môn đó ❖ averageScore: điểm tổng kết trung bình của sinh viên tính đến thời điểm hiện tại ❖ performance: học lực ❖ conduct: hạnh kiểm ❖ SubjectFailed: danh sách các môn sinh viên đã học nhưng chưa pass
	Phương thức	<ul style="list-style-type: none"> ❖ getResultStudent(sID): thống kê kết quả học tập các môn của sinh viên ❖ displayResultStudent(sID): hiển thị bảng điểm học tập của sinh viên trên giao diện

5.4.2.3 Phân tích động

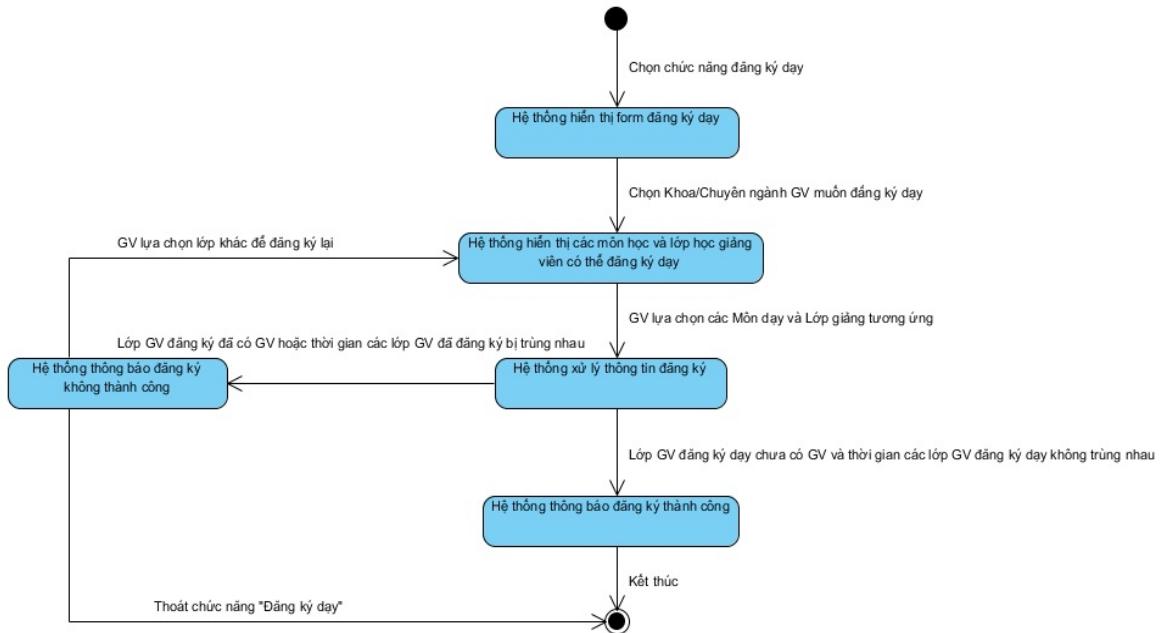
A.Biểu đồ trạng thái:

1. Đăng nhập

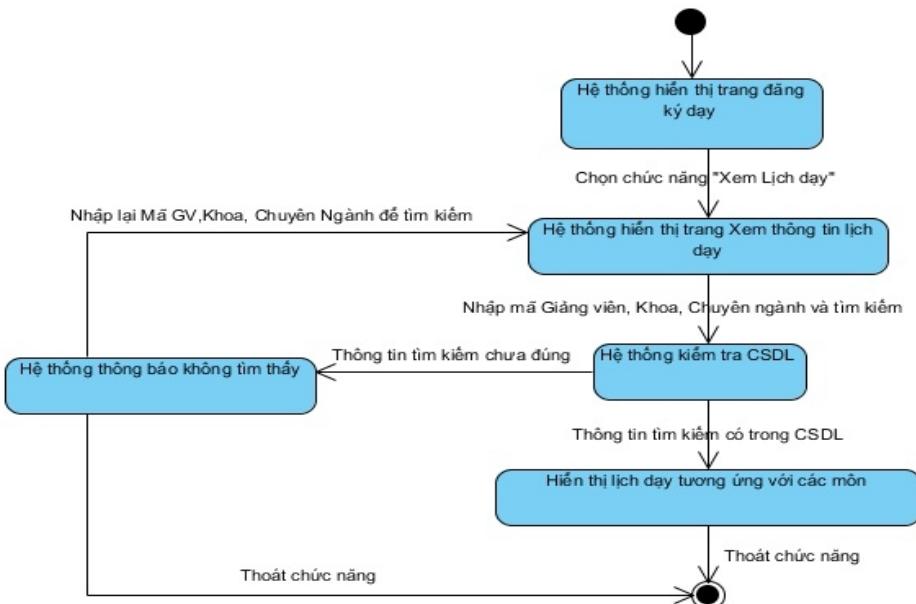


2.Giảng viên đăng ký dạy

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

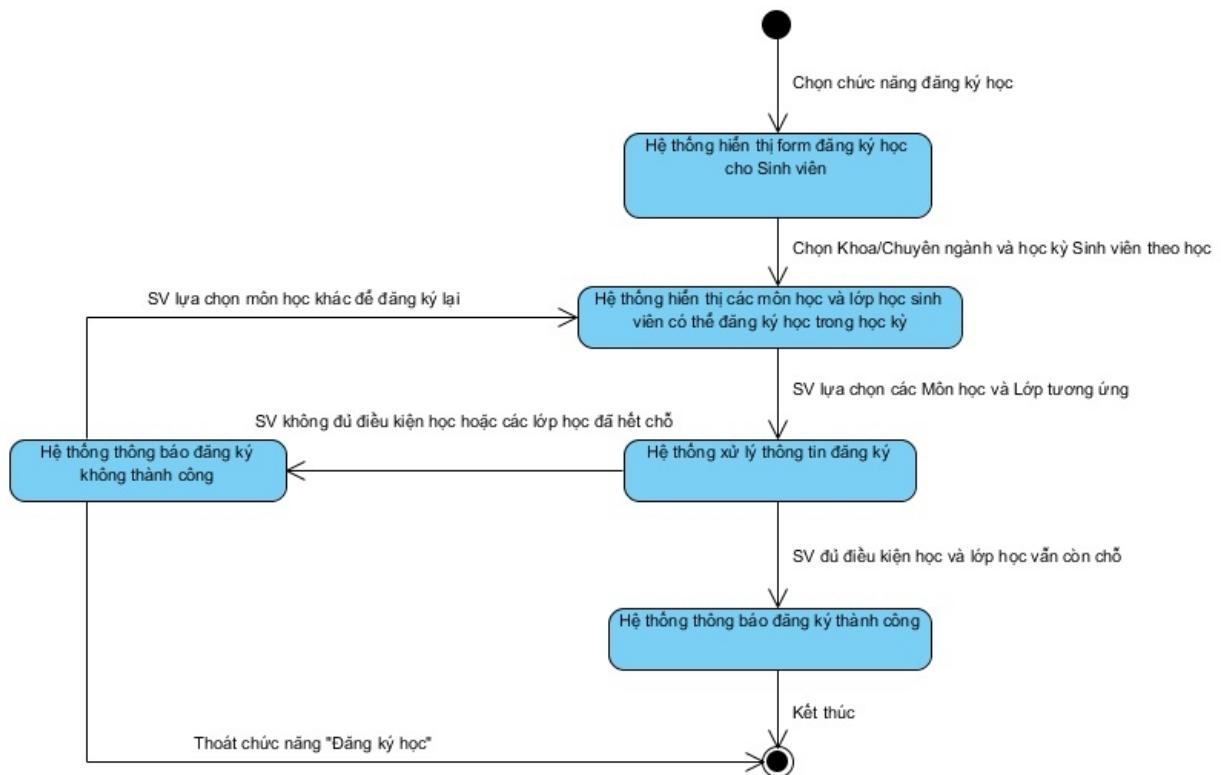


3.Giảng viên xem lịch dạy

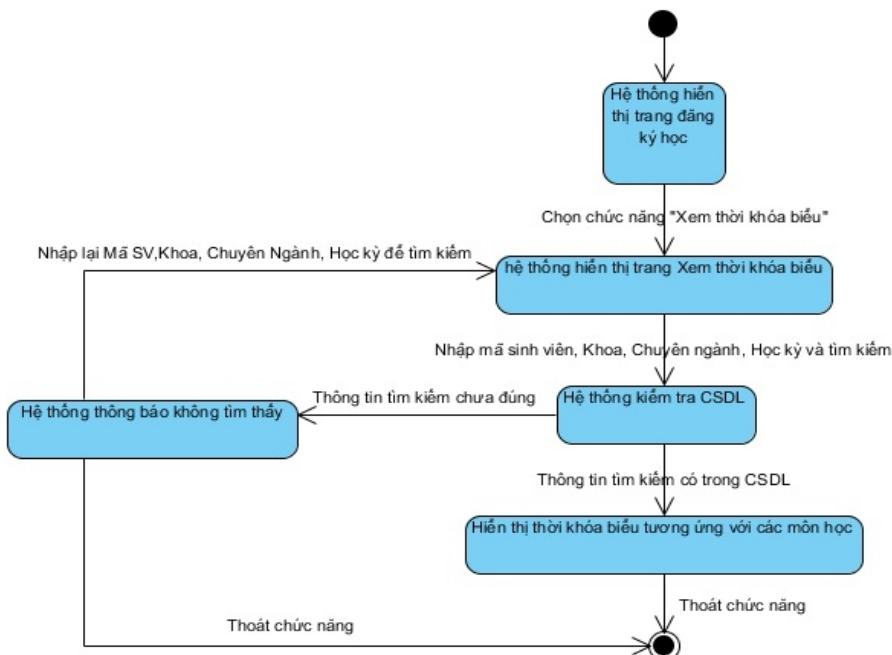


4.Sinh viên đăng ký học

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



5.Sinh viên xem thời khóa biểu

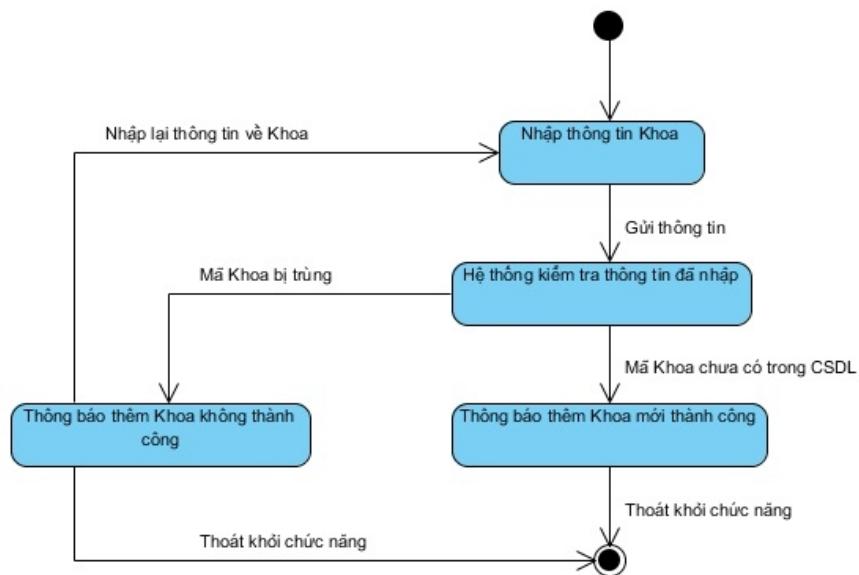


6.Nhân viên Quản lý cơ bản

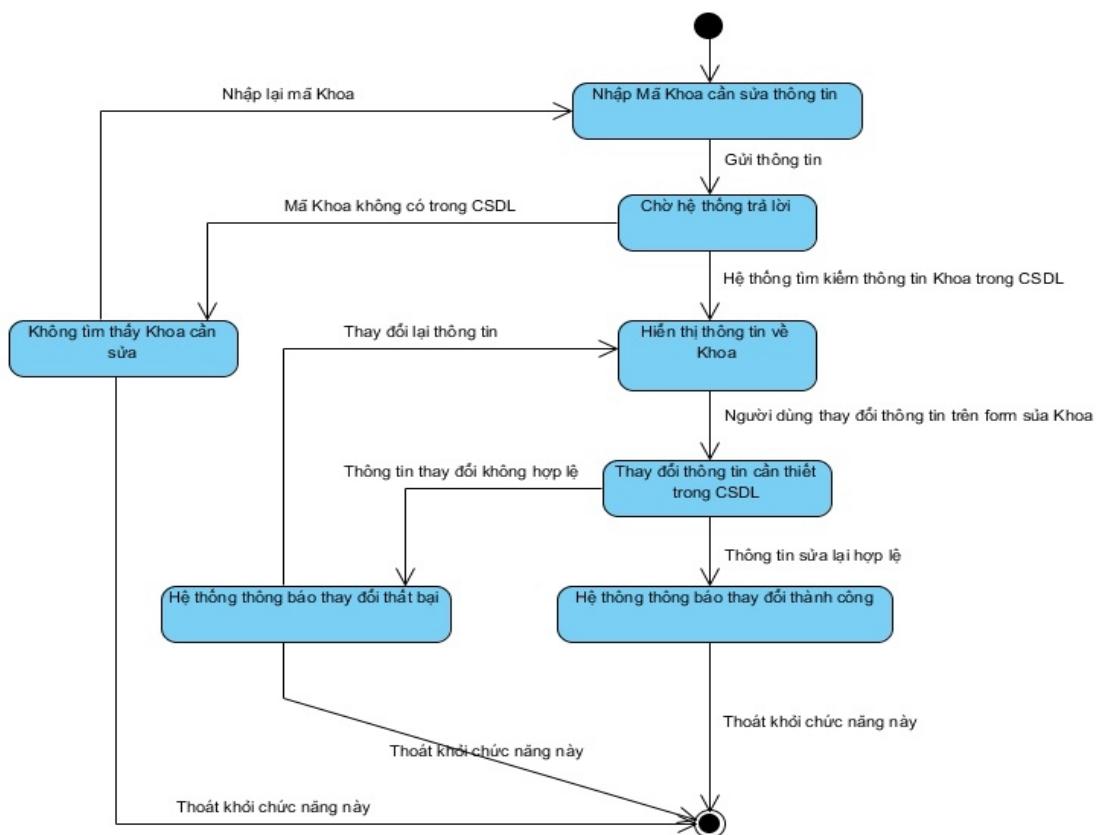
6.1.Quản lý Khoa

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

a. Thêm Khoa



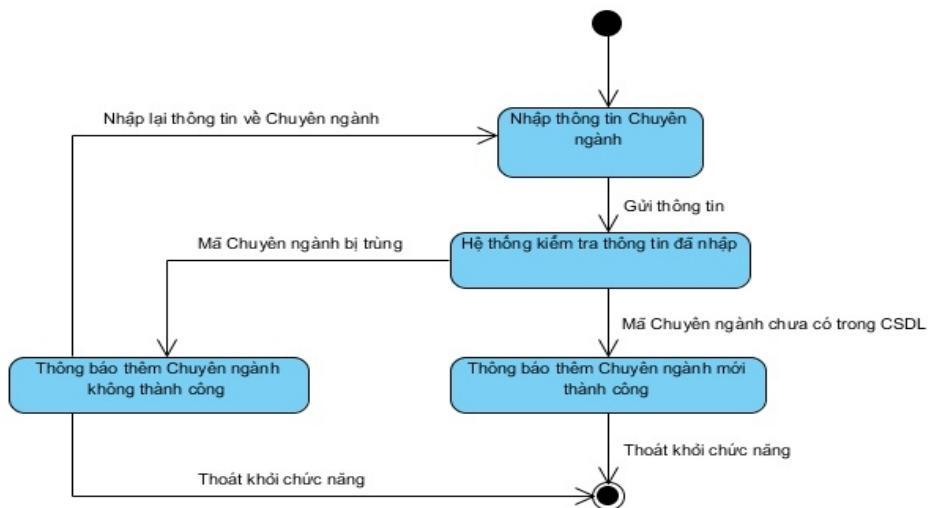
b. Sửa Khoa



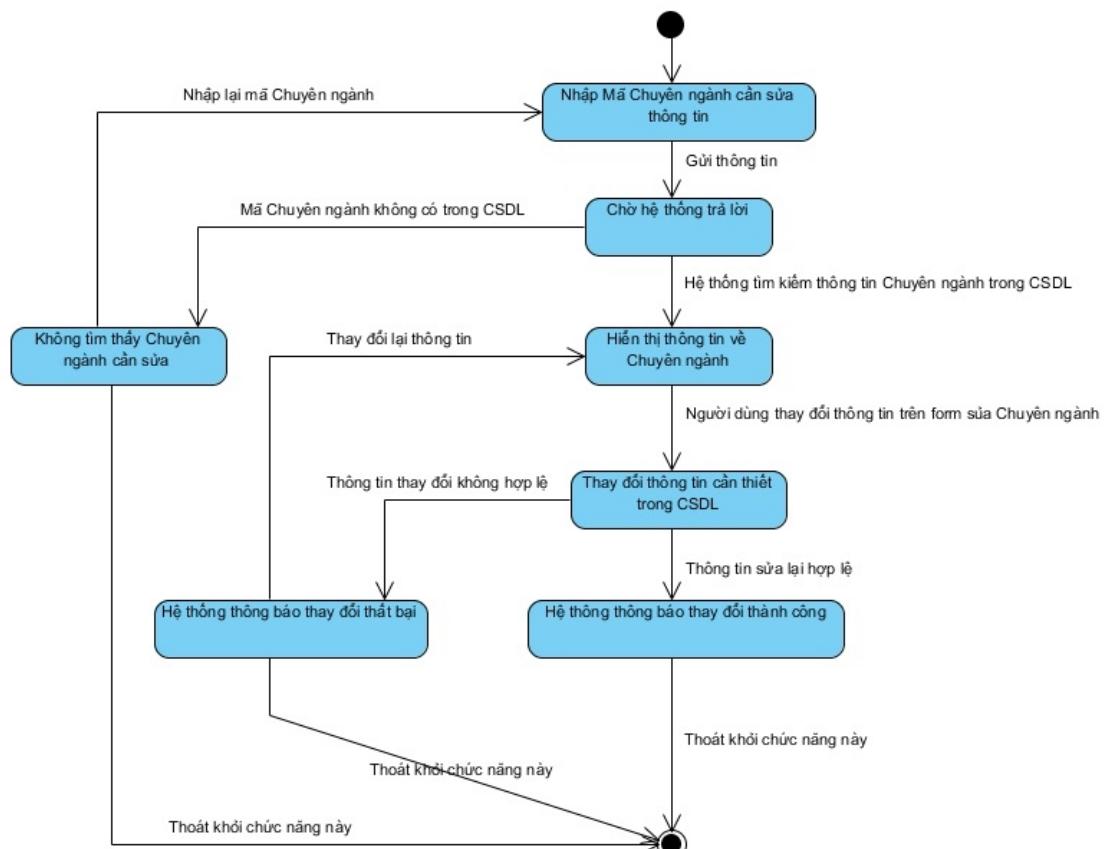
6.2. Quản lý chuyên ngành

a. Thêm Chuyên Ngành

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



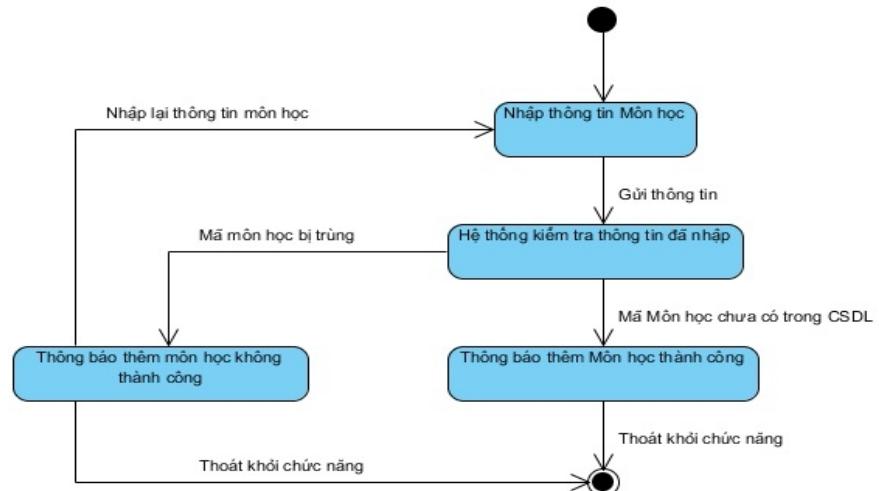
b. Sửa Chuyên Ngành



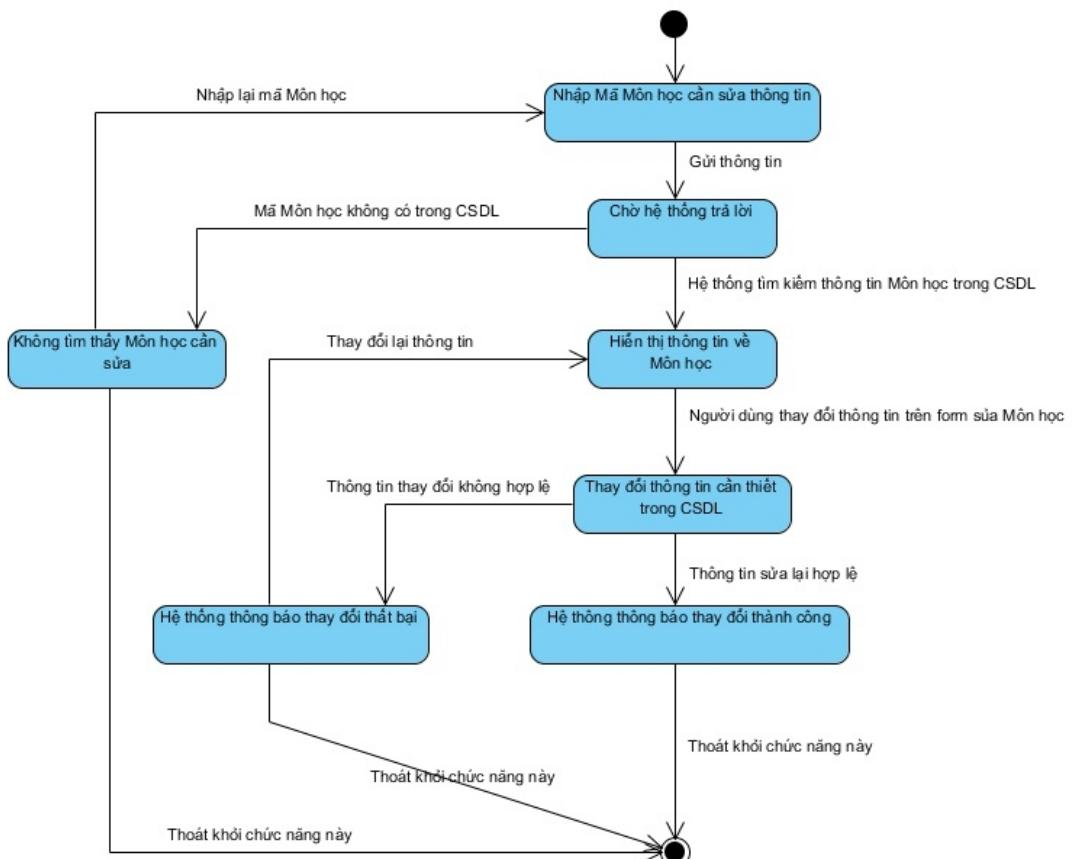
6.3. Quản lý môn học

a. Thêm Môn học

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

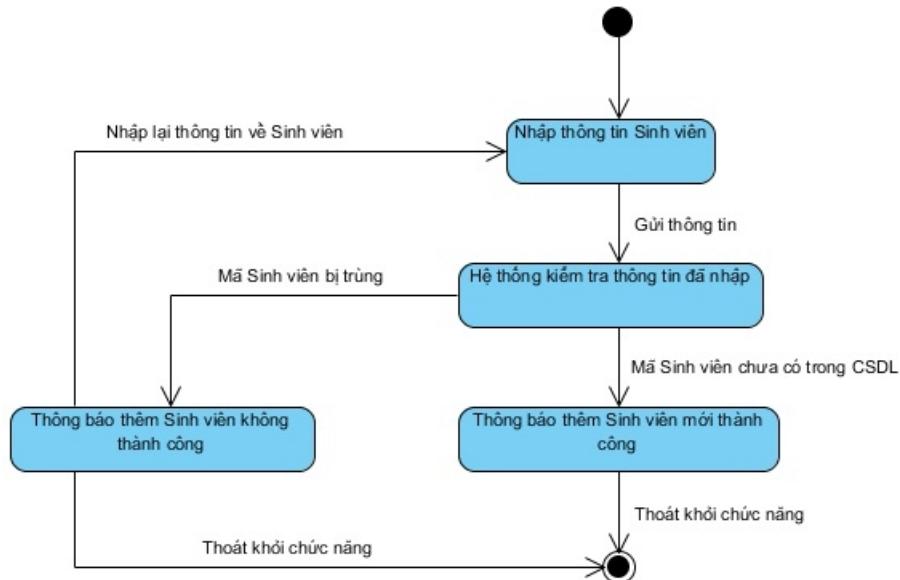


b. Sửa Môn học

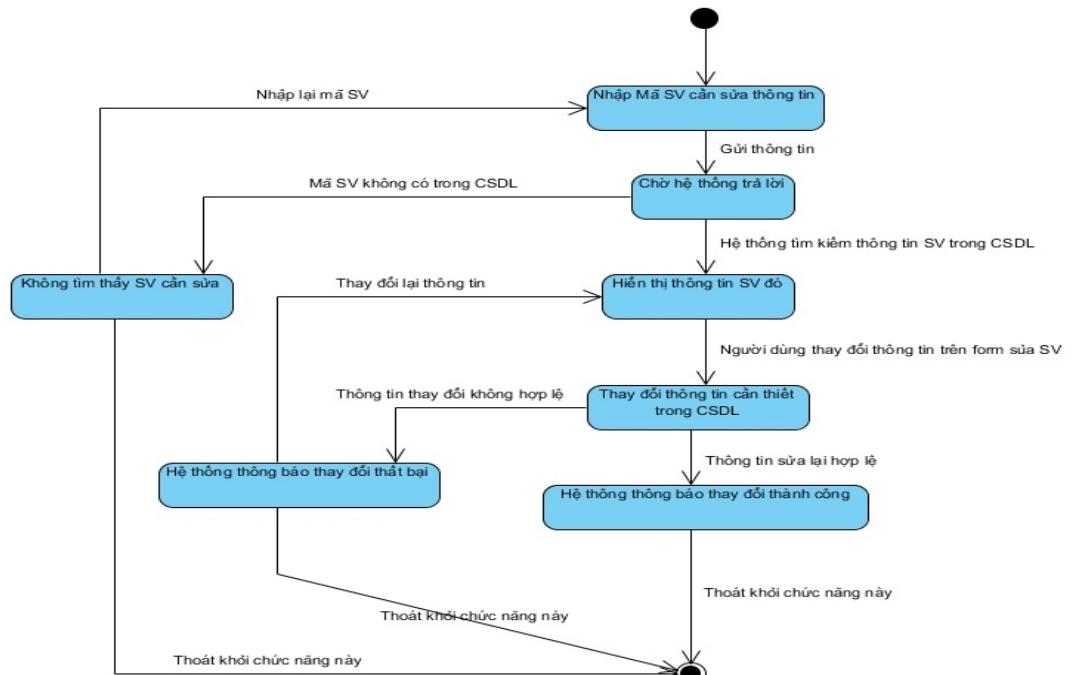


7.Quản lý sinh viên

7.1.Thêm Sinh viên



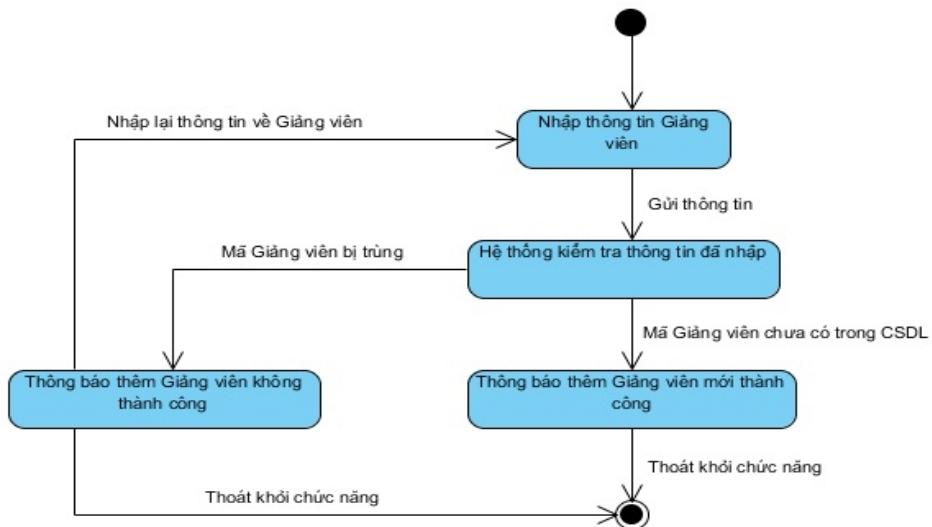
7.2.Sửa Sinh viên



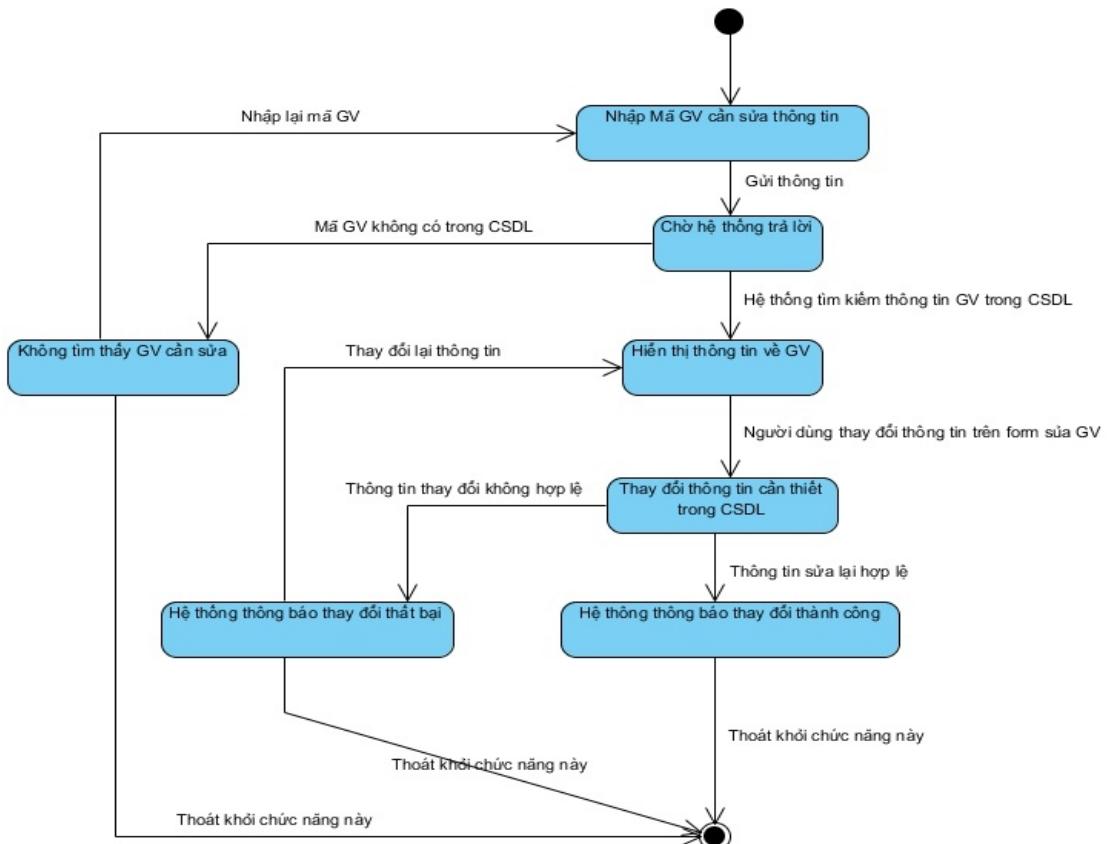
8. Quản lý giáo viên

8.1.Thêm Giảng viên

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



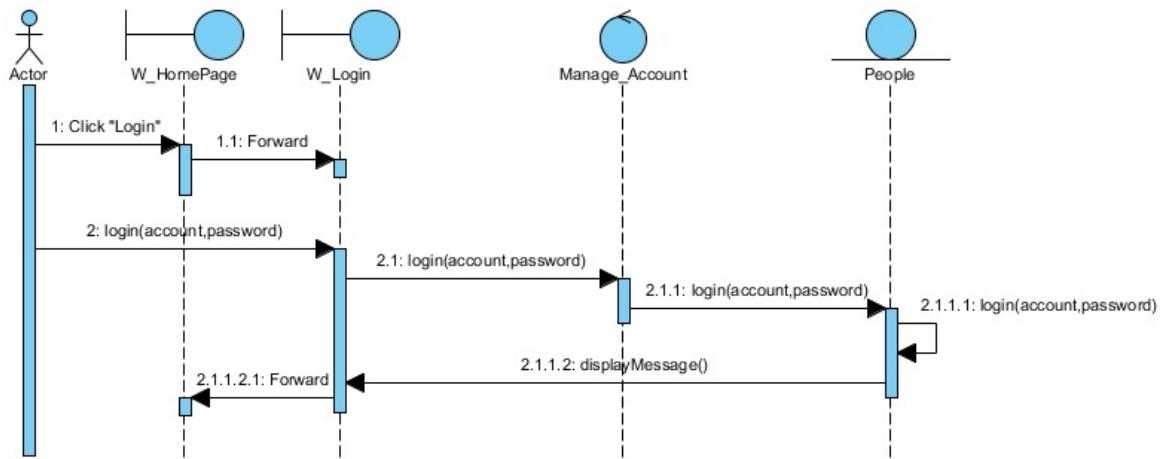
8.2.Sửa giảng viên



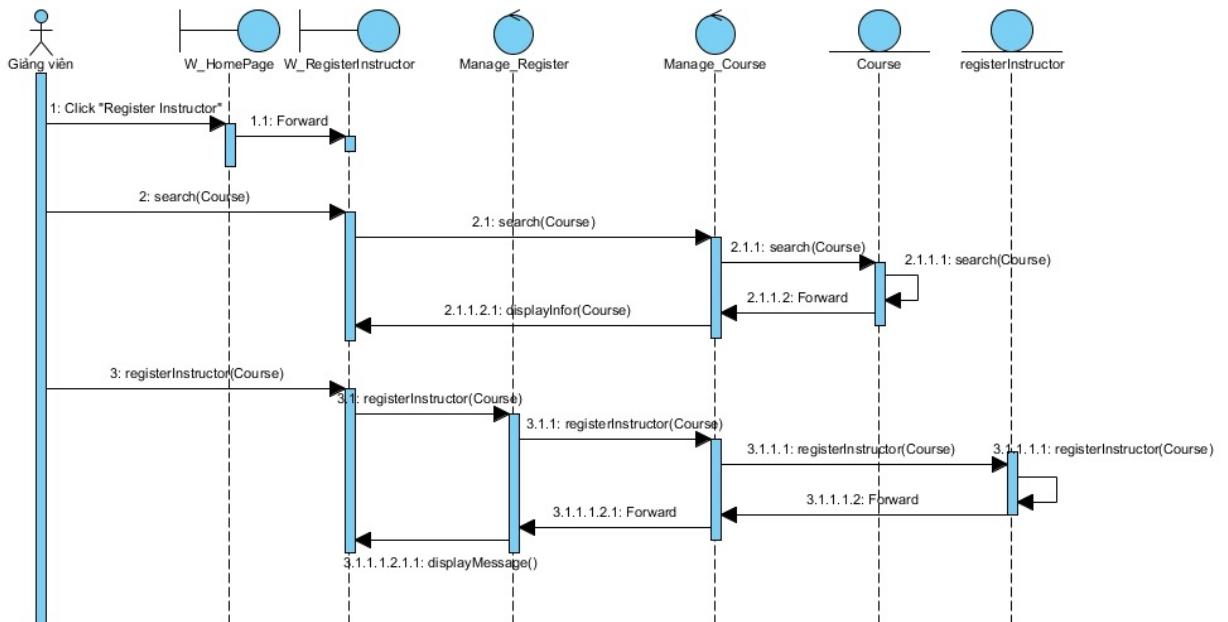
B. Biểu đồ tuần tự

1. Đăng nhập

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

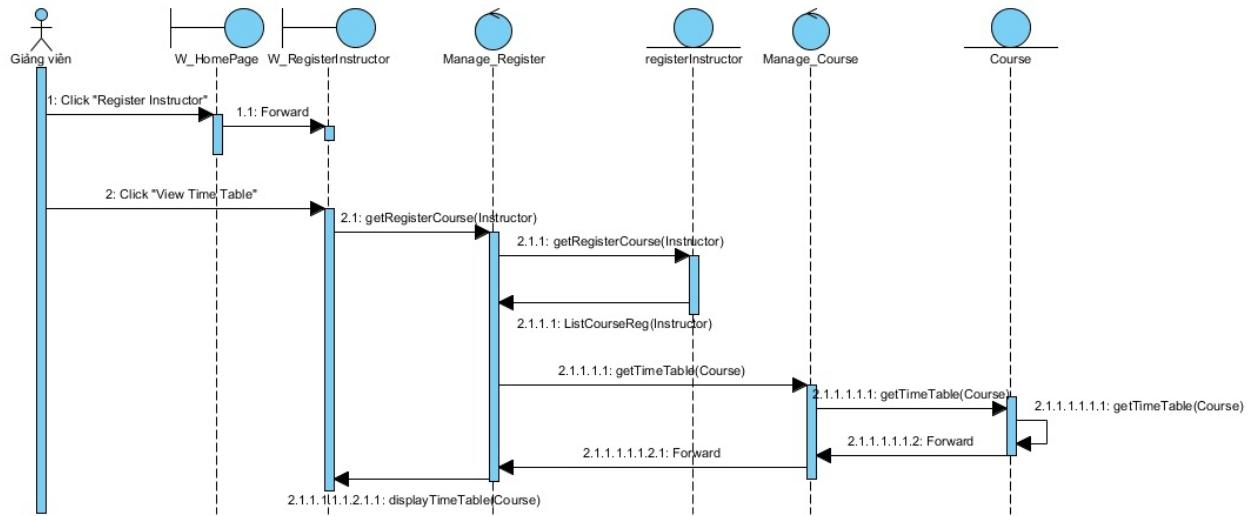


2. Giảng viên đăng ký môn dạy

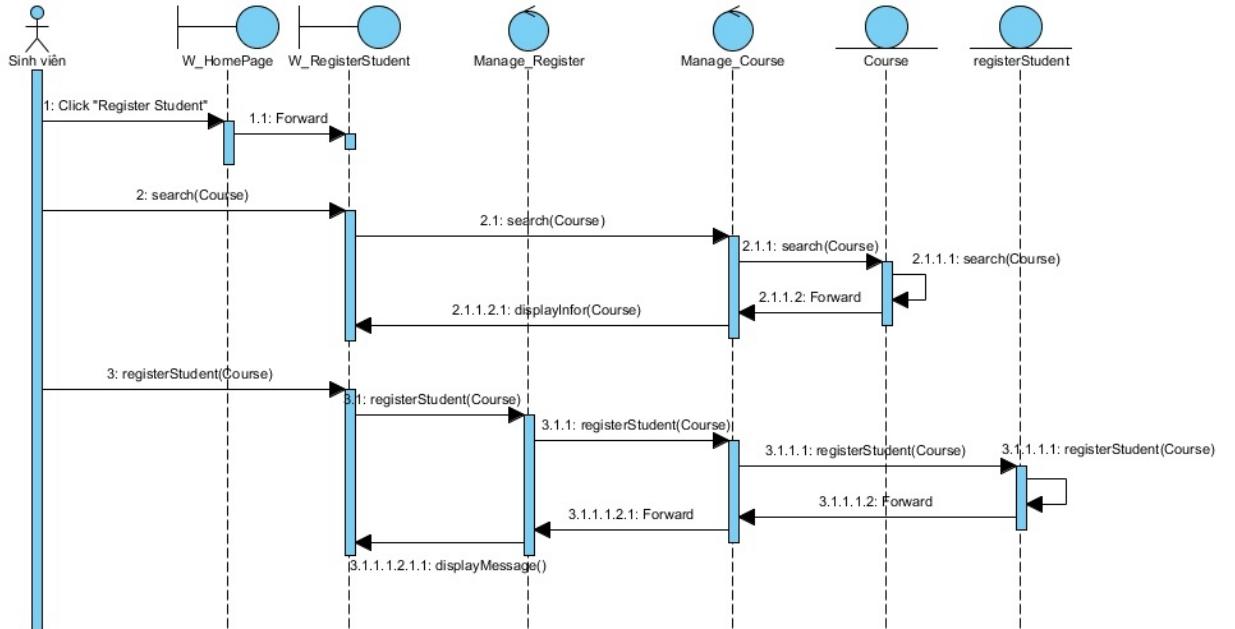


3. Giảng viên xem lịch dạy

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

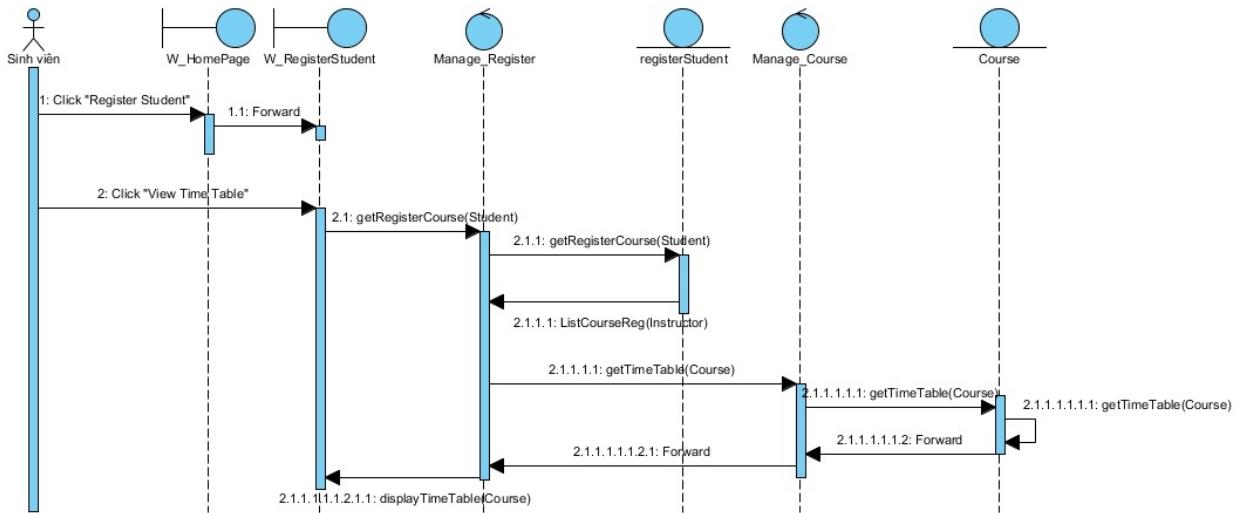


4. Sinh viên đăng ký môn học

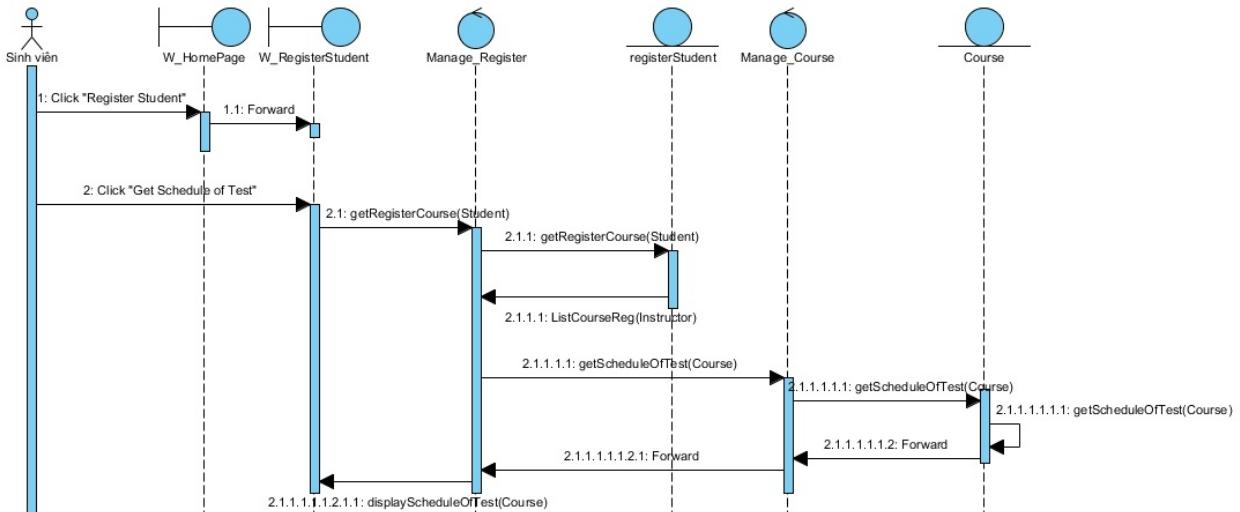


5. Sinh viên xem thời khóa biểu

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

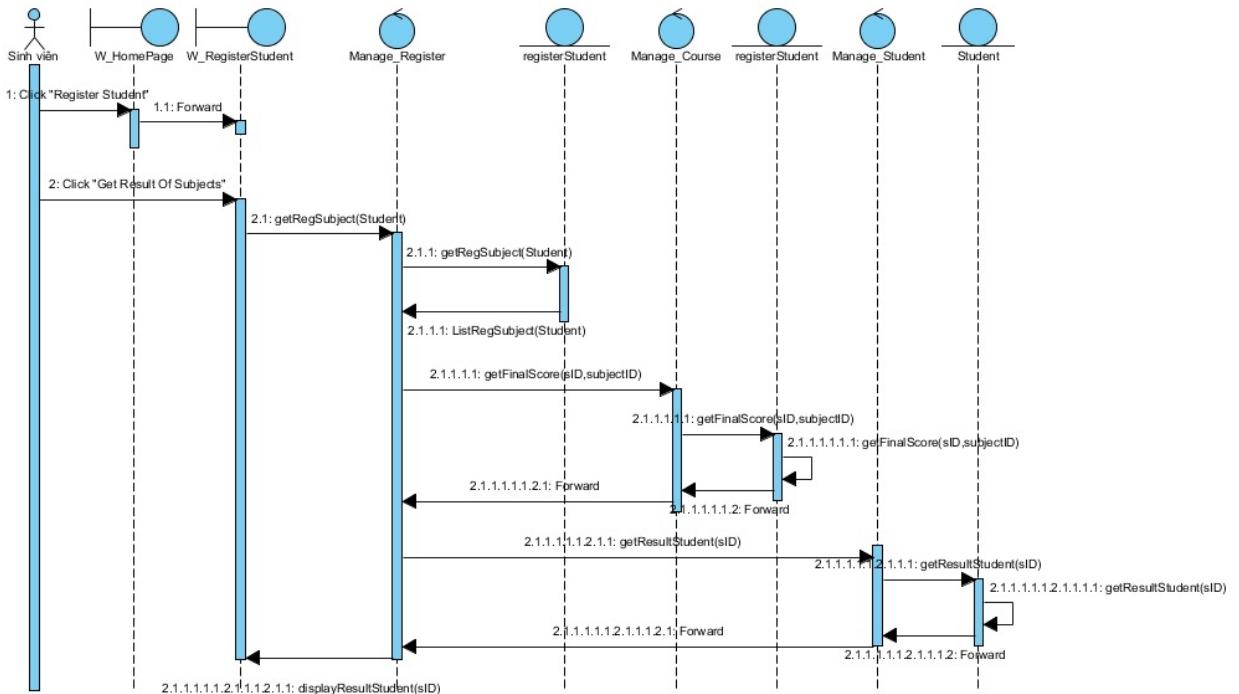


6. Sinh viên xem lịch thi học kỳ

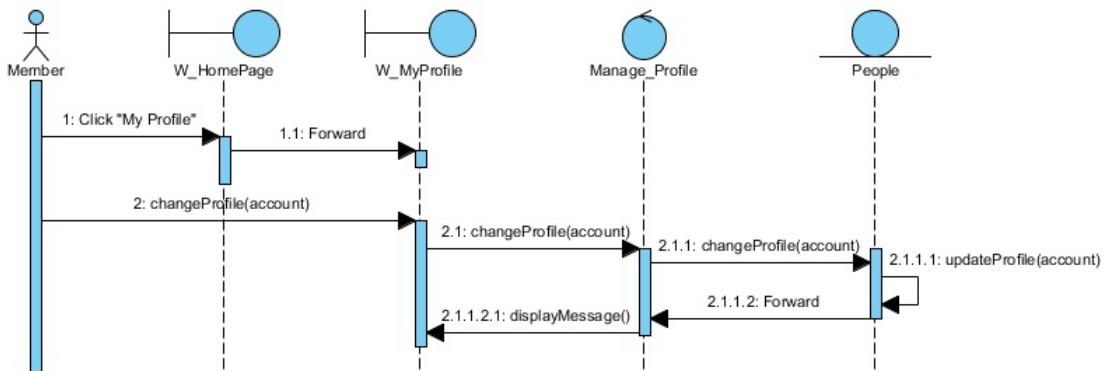


7. Sinh viên xem điểm học tập

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



8.Người dùng thay đổi thông tin cá nhân

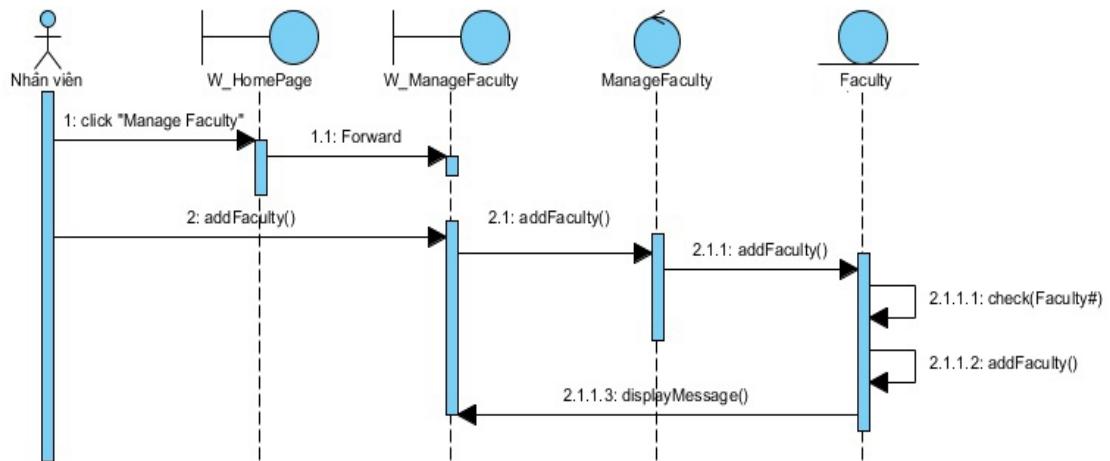


9.Nhân viên quản lý cơ bản

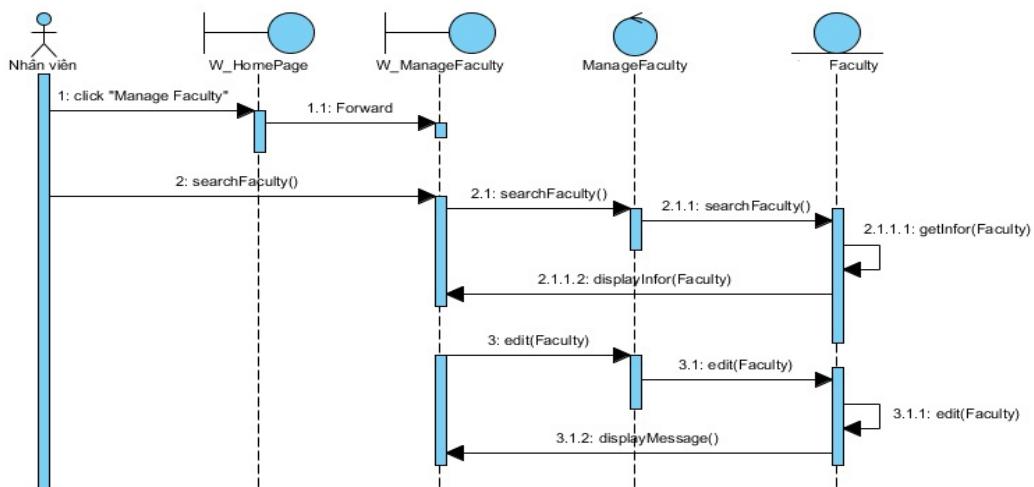
9.1.Quản lý khoa

a. Thêm Khoa

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

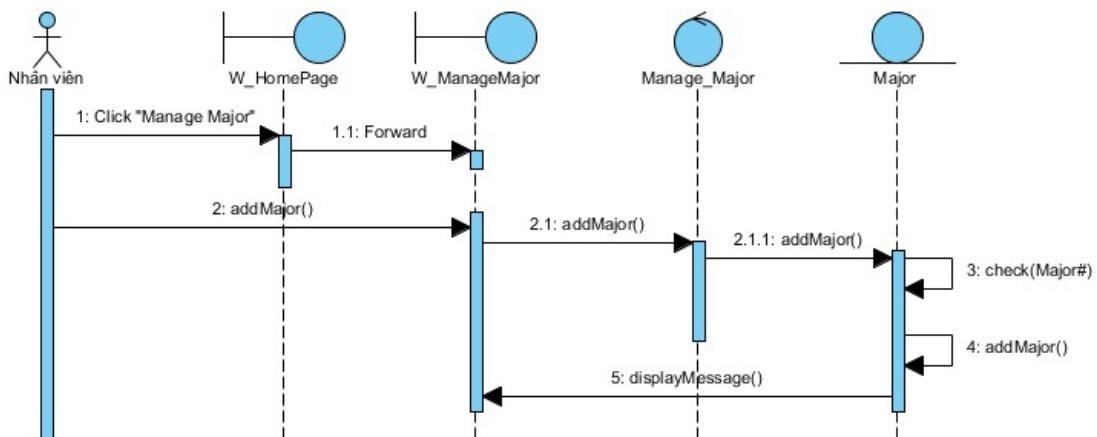


b. Sửa khoa



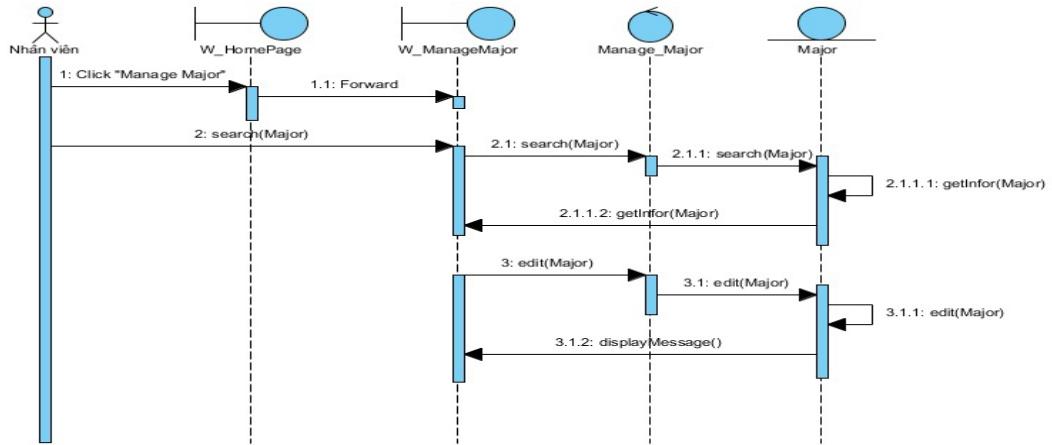
10. Quản lý chuyên ngành

a. Thêm Chuyên ngành



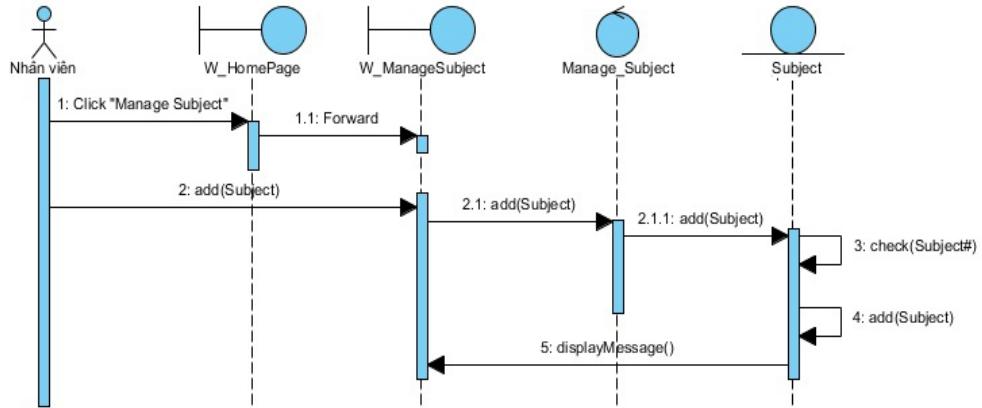
CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

b. Sửa chuyên ngành

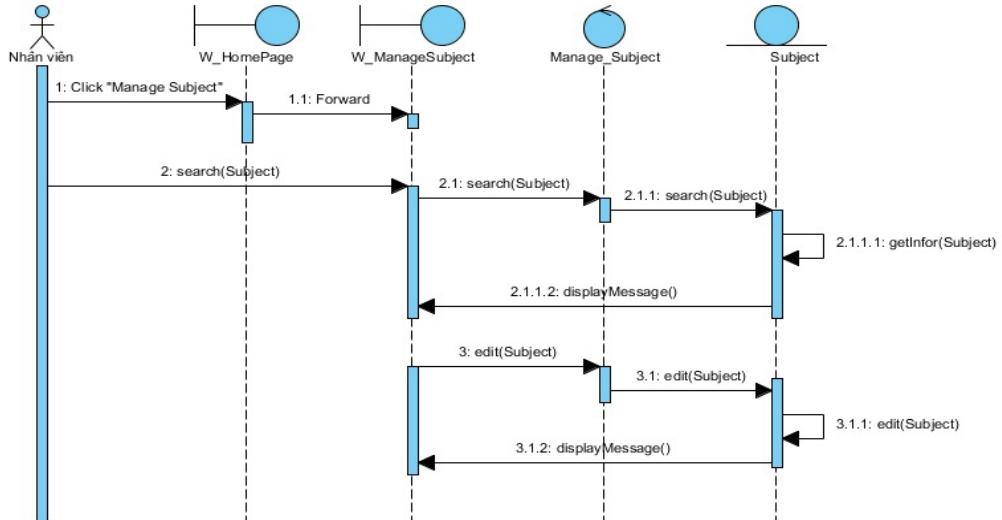


11. Quản lý môn học

Thêm Môn học



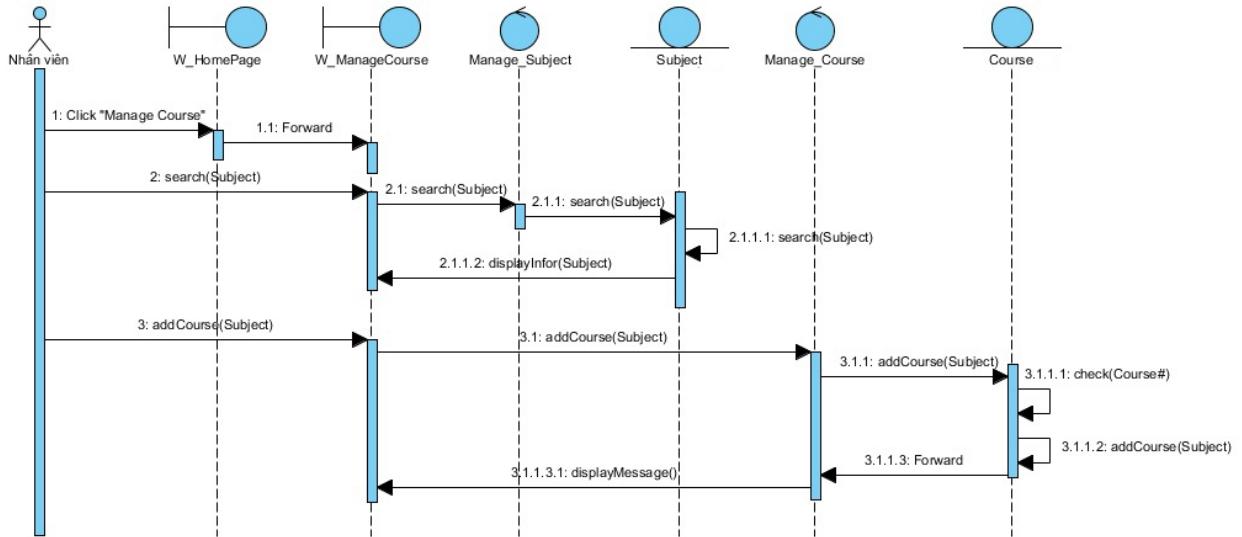
Sửa Môn học



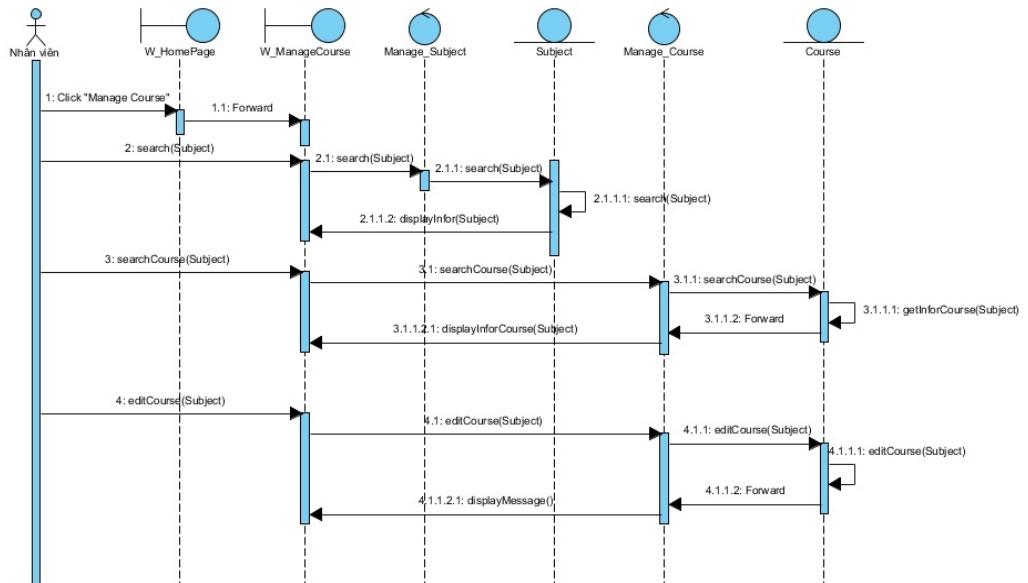
12.Nhân viên quản lý đào tạo

12.1.Quản lý xếp lớp học

Thêm lớp học

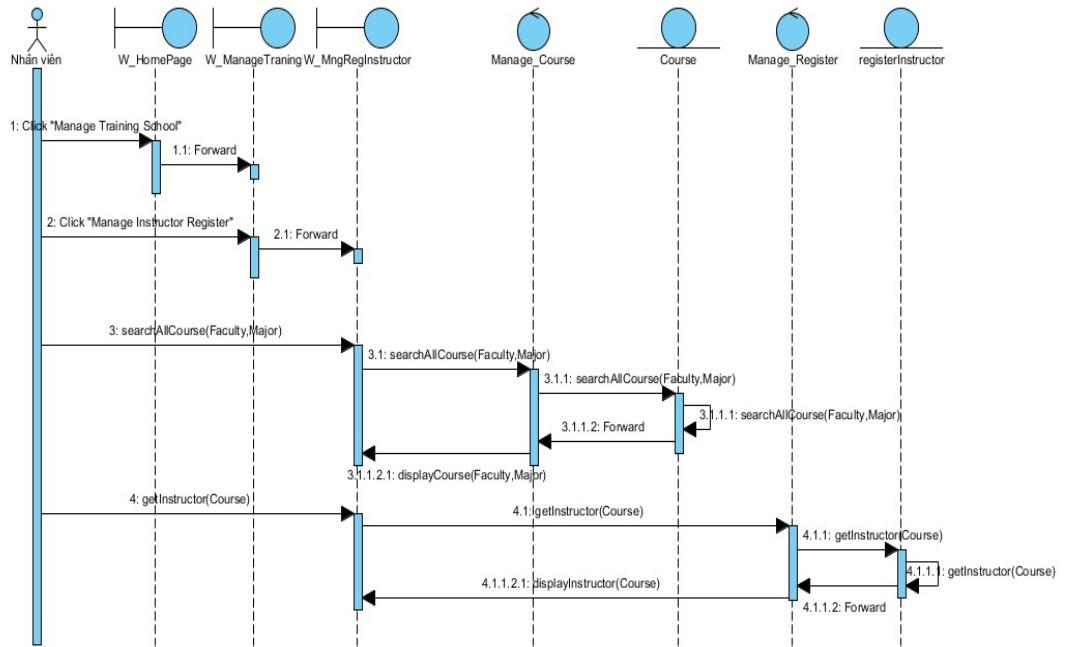


Sửa lớp học

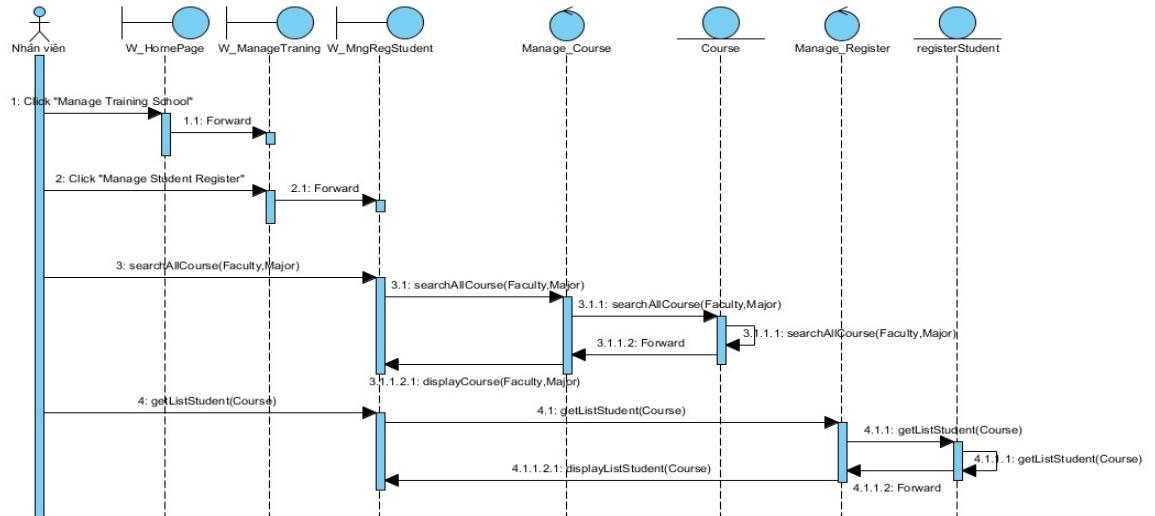


12.2.Quản lý đăng ký dạy của giảng viên

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

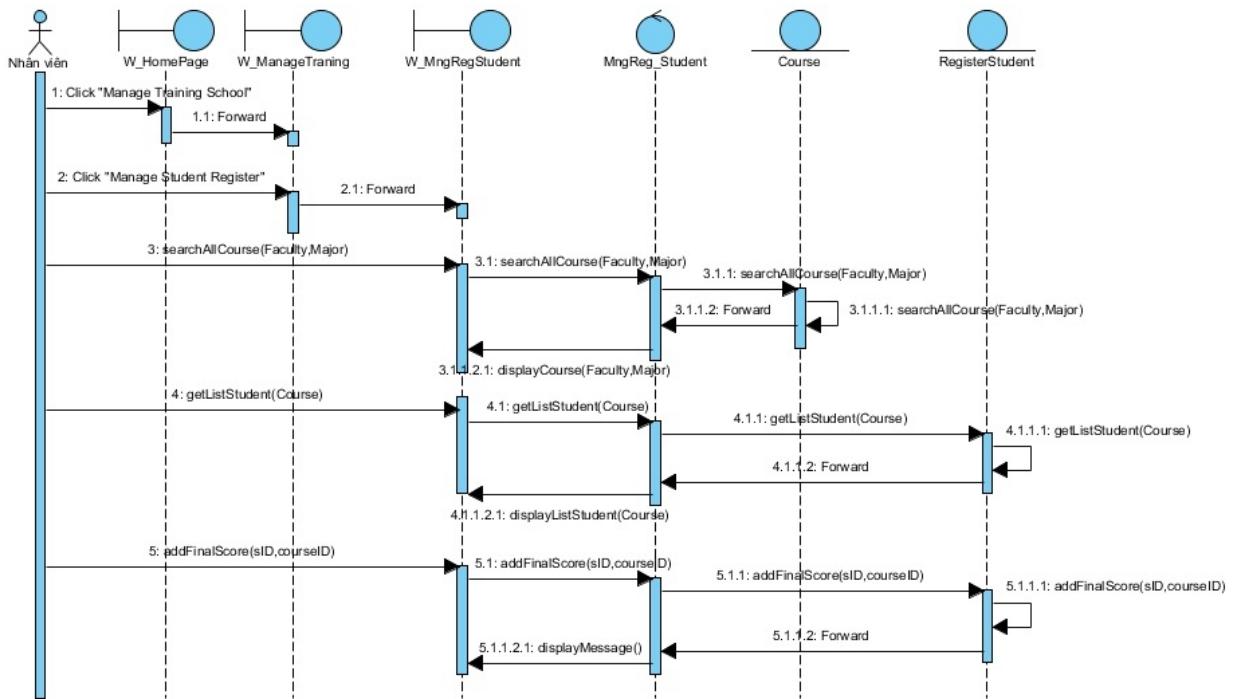


12.3. Quản lý đăng ký học sinh viên



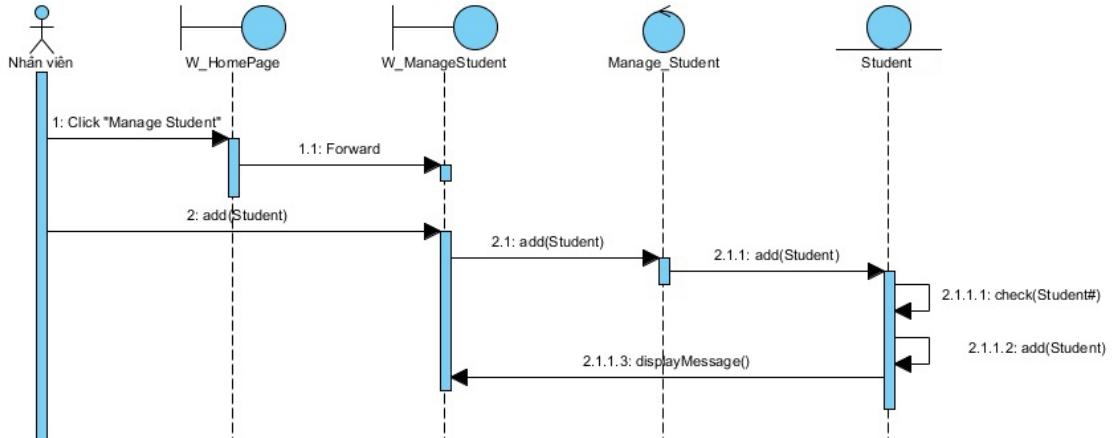
12.4. Quản lý điểm sinh viên

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU



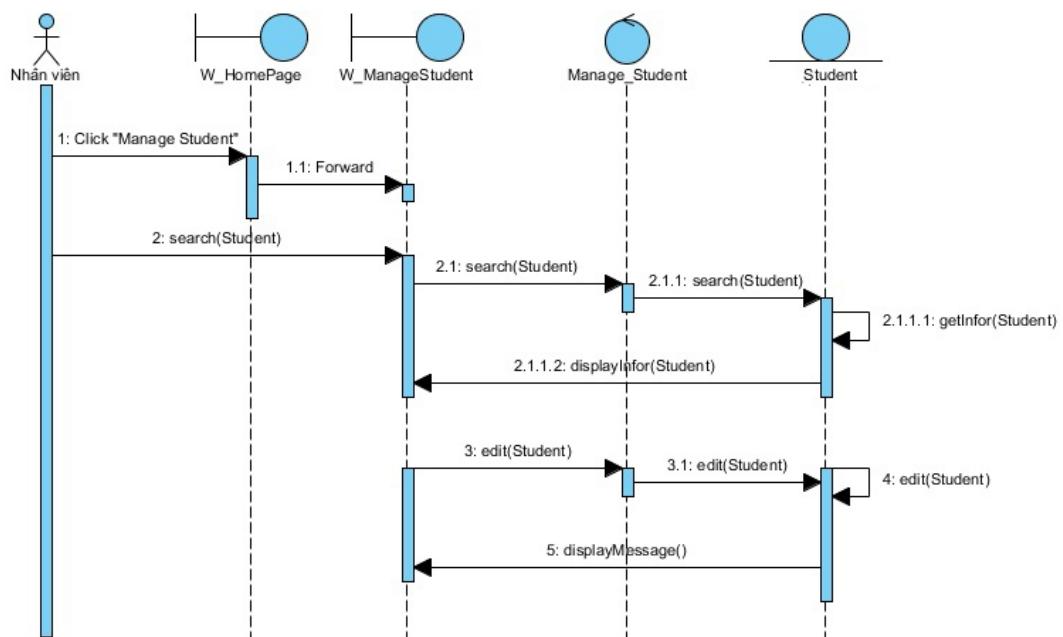
13. Quản lý sinh viên

a. Thêm sinh viên



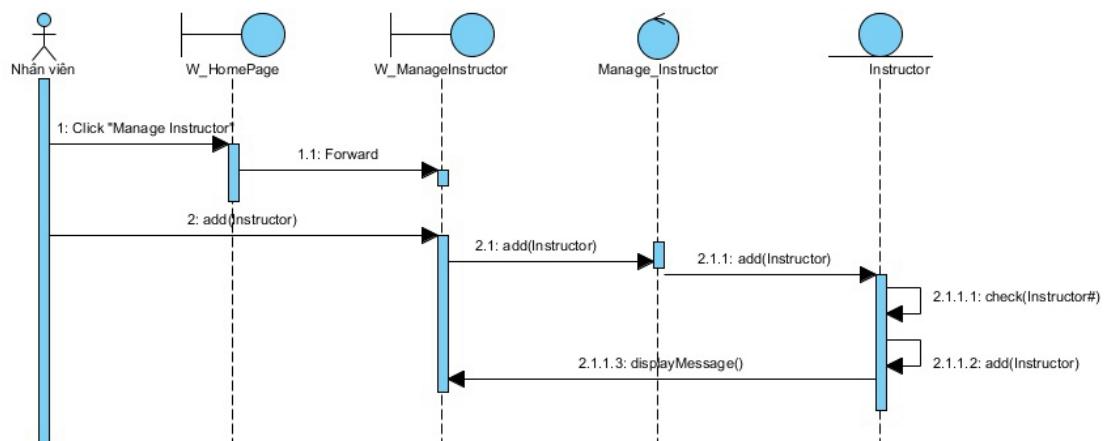
b. Sửa sinh viên

CHƯƠNG 5. PHÂN TÍCH YÊU CẦU

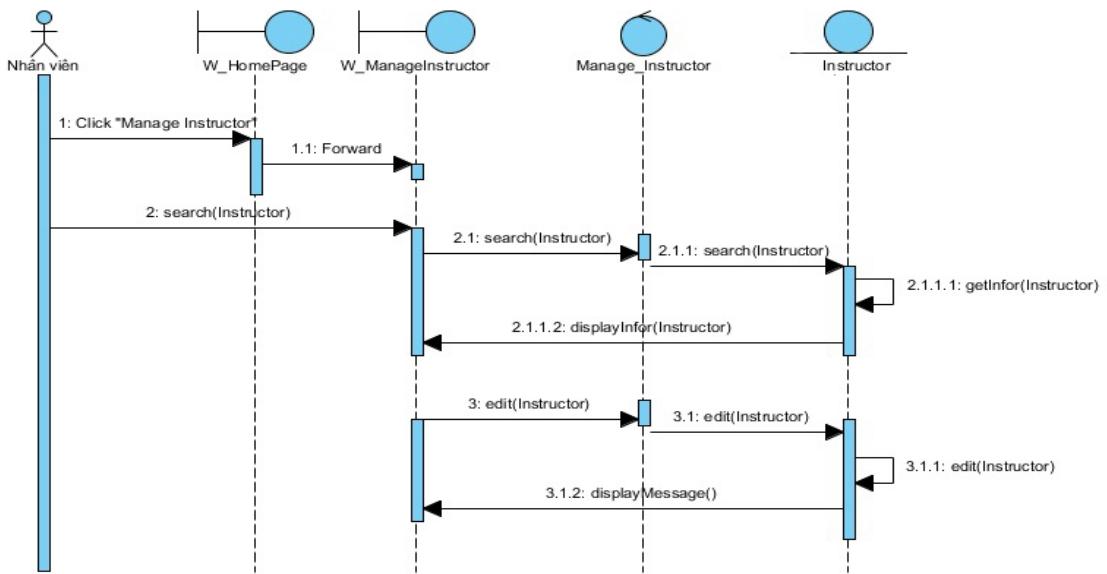


14. Quản lý giáo viên

a. Thêm giảng viên



b. Sửa giảng viên



5.5 KẾT LUẬN

Chương này trình bày các bước trong phân tích hướng đối tượng. Nội dung bao gồm phân tích động và phân tích tĩnh. Những ví dụ minh họa cũng được trình bày để sinh viên hiểu rõ hơn các khái niệm liên quan. Cuối cùng chúng tôi trình bày một case study cho hệ quản lý ghi danh học theo tín chỉ.

BÀI TẬP

1. Hãy chỉ ra mối liên hệ giữa các kịch bản và các biểu đồ trong pha phân tích
2. Hãy hoàn thiện theo nhóm các biểu đồ trong pha phân tích của hệ quản lý học theo tín chỉ.

CHƯƠNG 6 THIẾT KẾ KIẾN TRÚC HỆ THỐNG

6.1 GIỚI THIỆU

Phân tích và thiết kế là những hoạt động cần những ý tưởng rất khác nhau, mặc dù ranh giới giữa hai pha này là không rõ ràng. Ranh giới mờ nhạt này có thể là có ý như trường hợp của RUP, hoặc ngẫu nhiên, là do nhóm phát triển phần mềm kém cỏi. Kinh nghiệm đã chỉ ra rằng tách biệt rõ ràng giữa phân tích và thiết kế là tốt, để chắc chắn rằng vấn đề được hiểu rõ ràng trước khi xem xét các giải pháp. Theo cách hiểu thông thường, phân tích là về nghiên cứu xem xét vấn đề trong khi thiết kế là đưa ra một giải pháp. Hay, nói một cách ngắn gọn, “**Phân tích = what**”; “**Thiết kế = How**”.

Không có một quy tắc chính xác nào cho việc chuyển từ mô hình phân tích sang một mô hình thiết kế. Lý do là do có sự tham gia của con người và tính sáng tạo của việc phát triển phần mềm. Quá trình thiết kế được thúc đẩy bởi sự cần thiết để đưa ra một hệ thống hoàn chỉnh, kinh nghiệm của đội, cơ hội tái sử dụng, sở thích cá nhân... Mỗi người thiết kế khi nghiên cứu sản phẩm dựa trên yêu cầu và phân tích, anh ta có thể bắt đầu với một tờ giấy trắng. Nghĩa là chúng ta không chắc chắn rằng có hay không một sự tương thích cao giữa những đối tượng phân tích và những đối tượng thiết kế, cho tới khi thiết kế dẫn tới một giải pháp hiệu quả.

Trong suốt pha thiết kế, chúng ta phải đưa ra một lựa chọn công nghệ nào đó (ví dụ như: ngôn ngữ lập trình, giao thức và hệ thống quản trị dữ liệu). Chúng ta phải quyết định mức độ tác động của các lựa chọn này trong thiết kế. Những lựa chọn công nghệ sẽ ảnh hưởng tới các thư viện, các pattern và các framework có sẵn của chúng ta và thậm chí lời ghi chú UML chi tiết mà chúng ta sẽ dùng. Rõ ràng rằng thiết kế càng được tổng quát hơn thì chúng ta càng ít gắn bó với một công nghệ cụ thể hơn. Điều này sẽ giảm bớt yêu cầu rằng các nhà phát triển phải thành thạo nhiều công nghệ và điều này sẽ bảo vệ chúng ta khỏi bị lỗi thời với những công nghệ mới. Tuy nhiên, mặt trái của việc thiết kế tổng quát là chúng ta có thể không lợi dụng được lợi ích tối đa từ công nghệ cụ thể mà nó đem lại.

Lịch sử chứng minh rằng nhiều công nghệ xuất hiện và biến mất liên tục hơn những lý thuyết làm cơ sở cho các công nghệ đó. Ví dụ như, các ngôn ngữ lập trình có nhiều như COBOL, Fortran, Pascal, Ada, Modula, PL/I, C, C++, Smalltalk, Eiffel, C# và Java, nhưng việc sử dụng những công nghệ này bị chi phối chỉ bởi hai lý thuyết: lập trình

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

hướng cấu trúc và lập trình hướng đối tượng. Vì vậy, thật có lý khi cho rằng “tổng quát thì an toàn hơn cụ thể”.

6.2. ƯU TIÊN THIẾT KẾ

Vì phát triển phần mềm hướng đối tượng là theo kiểu tăng dần, nên chúng ta không hy vọng có một thiết kế toàn bộ cả hệ thống ngay từ đầu. Vì thế, lúc bắt đầu pha thiết kế, chúng ta cần lên kế hoạch cho những phần của hệ thống mà chúng ta sẽ thiết kế. Ở đây, những mức ưu tiên use case sẽ giúp chúng ta đánh dấu những use case nào cần thiết nhất bằng phương pháp tương tự đèn giao thông trong suốt pha yêu cầu: những use case màu xanh phải được thiết kế đầy đủ ngay; use case màu vàng thì chưa cần thiết kế ngay, nhưng phải được hỗ trợ; use case màu đỏ không phải thiết kế, nhưng chúng vẫn nên được hỗ trợ (“thiết kế” có nghĩa là một cách giải quyết được đưa ra; “hỗ trợ” có nghĩa là một cách giải quyết hợp lý có thể thực hiện được, tức là yêu cầu một số giải pháp dự phòng cho phần mềm của chúng ta).

Trong thực tiễn, chúng ta hay tìm một kiến trúc hệ thống mà sẽ hỗ trợ một giải pháp sao cho có hiệu quả với tất cả các use case. Trong kiến trúc này, chúng ta thực hiện thiết kế chi tiết cho hầu hết các use case quan trọng và thiết kế một phần cho các use case ít quan trọng hơn. Giữa những tăng dần, chúng ta điều chỉnh những mức ưu tiên để sao cho hợp lý.

6.3. CÁC BƯỚC TRONG THIẾT KẾ HỆ THỐNG

Thiết kế có thể được chia làm hai hoạt động riêng biệt: *Thiết kế hệ thống* (còn gọi là Thiết kế kiến trúc hay Thiết kế tổng thể) và *thiết kế hệ thống con* (còn gọi là thiết kế chi tiết). Thiết kế hệ thống bắt chúng ta phải có cái nhìn tổng quát về các tác vụ trước khi chúng ta đi sâu vào thiết kế chi tiết của hoạt động thiết kế hệ thống con (sẽ được giới thiệu ở Chương 8). Tuy nhiên, trong phương pháp hướng đối tượng, chúng ta có thể làm mờ ranh giới đó và có thể xoắn ốc và lặp, nhưng quan niệm có hai hoạt động vẫn thật sự rất cần thiết. Thiết kế hệ thống bao gồm các hoạt động sau đây:

- Lựa chọn hình trạng (topology) mạng: Cách phân tán hệ phần mềm trên các phần cứng và các tiến trình trên mạng.
- Lựa chọn công nghệ: Chọn ngôn ngữ lập trình, Cơ sở dữ liệu, giao thức ...một số quyết định có thể hoãn lại cho tới pha thiết kế sau.
- Thiết kế chính sách tương tranh/dòng thời: tương tranh hay đồng thời có nghĩa là nhiều hoạt động xảy ra cùng một lúc, Ví dụ, đa tiến trình, nhiều người dùng, nhiều máy...phải phối hợp thông qua phần mềm để tránh sự hỗn độn.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

- Thiết kế chính sách bảo mật: Bảo mật có một số khía cạnh và mỗi khía cạnh đều phải được chú tâm và điều khiển một cách thích đáng. Ví dụ, xét dữ liệu về cá nhân một khách hàng: chúng ta phải đảm bảo dữ liệu đó không bị đánh cắp và phải đảm bảo rằng nó không thể tình cờ lộ ra ngoài.
- Lựa chọn sự phân rã hệ thống con: Thông thường, chúng ta không thể làm ra những phần mềm đơn lẻ có khả năng giải quyết tất cả các vấn đề mà thay vào đó, chúng ta cần sản xuất các thành phần phần mềm riêng biệt và đảm bảo các thành phần này giao tiếp với nhau một cách hiệu quả.
- Phân rã hệ thống con thành các tầng hoặc các hệ thống con khác: nói chung, các hệ thống con cần phân rã hơn nữa thành các lớp hoặc các hệ thống con nhỏ hơn trước khi chúng ta tiến hành thiết kế chi tiết.
- Quyết định giao thức: cách thức giao tiếp giữa các máy, các hệ thống con và các lớp như thế nào và chú ý rằng những quyết định giao tiếp thường xảy ra hiệu ứng phụ của các bước khác.

6.4 LỰA CHỌN HÌNH TRẠNG HỆ THỐNG

Hình trạng hệ thống mạng chỉ ra hệ thống được phân rã thành các thành phần logic và vật lý riêng biệt như thế nào. Trong mục này, chúng ta sẽ xem lại lịch sử của các cấu trúc mạng và sau đó, thảo luận các cấu trúc hiện nay như sự khác nhau giữa các client “thin” và “fat” clients, các mạng, điểm khác biệt giữa ứng dụng client-server với các ứng dụng phân tán khác. Chúng ta cũng sẽ học cách sử dụng các lược đồ triển khai trong UML để minh họa các quyết định về cấu trúc của hệ thống.

6.4.1 Lịch sử của các cấu trúc mạng

Hầu hết các hệ thống mạng hiện nay đều có cấu trúc ba tầng. Muốn biết và hiểu cấu trúc ba tầng là gì, chúng ta sẽ xem lại lịch sử phát triển của nó trước đó. Ba thành phần logic của mạng:

- Các chức năng thực hiện việc tương tác với người dùng như tạo các giao diện nhập liệu hay in các báo biểu, thông báo ra màn hình. Các chức năng này được gọi chung là **Dịch vụ giao diện người dùng** (User Interface Service).
- Các chức năng tính toán các dữ liệu, xử lý thông tin theo những quy luật (rule), giải thuật được xác định bởi những vấn đề mà hệ ứng dụng sẽ giải quyết. Các chức năng này được gọi chung là **Dịch vụ nghiệp vụ** (Business Service).
- Trong quá trình tính toán, chương trình ứng dụng cần truy vấn đến các thông tin đã có được lưu trên đĩa cứng hay trong các cơ sở dữ liệu. Cũng như cần thiết phải

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

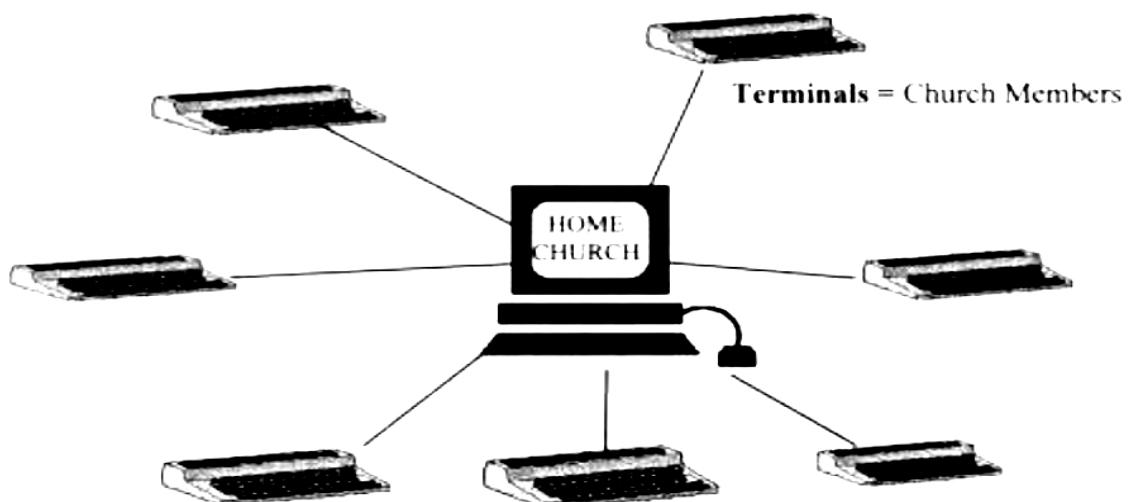
lưu lại các kết quả tính toán cho sử dụng về sau. Các chức năng này được gọi chung là **Dịch vụ lưu trữ dữ liệu** (Data Storage Service).

6.4.2 Kiến trúc đơn tầng

Những năm 1940, máy tính là một thiết bị nguyên khôi lớn, chỉ có khả năng chạy một chương trình tại một thời điểm. Sau đó, nó được cải tiến thành **Mainframe** – có khả năng chạy đồng thời nhiều chương trình, điển hình là chương trình một người dùng hoặc một chương trình trên một đợt (Ví dụ như khi xử lý các hóa đơn điện tử, một đợt bao gồm một tập các dữ liệu giống nhau chạy qua một chương trình theo thứ tự nào đó). Mainframe có khả năng xử lý nhiều chương trình đồng thời bởi vì nó chia thời gian cho CPU chạy mỗi chương trình và mỗi chương trình khi tới lượt sẽ được thực thi.

Lúc đầu, người dùng và người quản trị hệ thống sẽ truy cập vào mainframe thông qua máy điện báo đánh chữ – để gửi một dòng văn bản tới mainframe (dòng văn bản sẽ thể hiện các câu lệnh của chương trình hay dữ liệu của chương trình). Chương trình chạy trên mainframe sẽ gửi lại một hoặc nhiều dòng như một phản hồi. Khi kỹ thuật được cải tiến, máy điện báo đánh chữ được thay thế bằng màn hình, nó sử dụng CRT thay vì giấy cho việc vào ra. Với màn hình, người điều hành có thể chuẩn bị các câu lệnh cũng như dữ liệu trước khi gửi tới mainframe thực thi.

MAINFRAME COMPUTER = CONVENTIONAL CHURCH



Hình 6.1. Mô hình Mainframe – kiến trúc mạng một tầng.

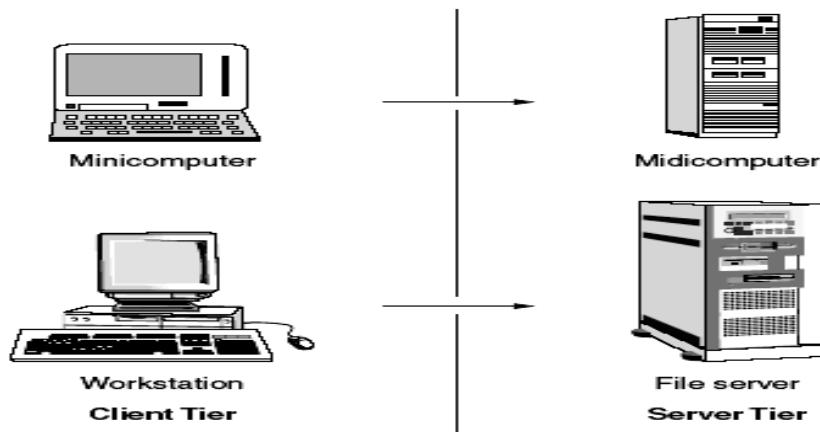
Mô hình mainframe vẫn còn được sử dụng cho các hệ thống nghiệp vụ cỡ lớn, là mô hình kiến trúc một tầng. Điều này có nghĩa là, bất cứ chương trình nào được đưa vào, chỉ có duy nhất một mức hoạt động tính toán chạy trên một máy. Hay nói cách khác, kiến trúc một tầng không có mạng: mặc dù có một dây kim loại mảnh nối từ mỗi thiết bị tới mainframe, nhưng không tạo thành một mạng theo nghĩa hiện nay.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

Ưu điểm chính của mainframe là cài đặt đơn giản, nhưng nhược điểm chính của chúng là chỉ có thể tăng khả năng tính toán bằng cách mua thêm mainframe mới, hoặc nâng cấp cái cũ.

6.4.3 Kiến trúc hai tầng

Thế hệ kiến trúc tiếp theo được phổ biến vào những năm 1970, là **kiến trúc 2 tầng**. Ý tưởng của kiến trúc này là nhằm tăng cường khả năng xử lý trên mỗi máy khách nhằm để cho các máy tính trung tâm không phải thực hiện tất cả các tiến trình. Như vậy, chúng ta có thể thêm hay thay thế các máy khách rẻ hơn các máy tính trung tâm. Các máy khách thường là các máy tính mini hay các workstation, có thể truy nhập vào các máy tính midi hay các file server. Sự kết hợp giữa máy tính mini và máy tính midi được chỉ ra ở đây cũng tương tự như sự kết hợp giữa workstation và file server.



Hình 6.2. Kiến trúc mạng 2 tầng

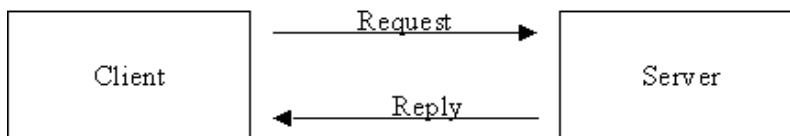
Với kiến trúc 2 tầng, chương trình và dữ liệu phải được chuyển từ máy tính trung tâm sang máy khách. Điều này đòi hỏi phải có sự kết nối nhanh giữa 2 bên và điều này dẫn đến ý tưởng hiện đại về mạng. Mạng là một tập hợp bất kỳ các máy chủ (host) được kết nối với nhau bằng các đường giao tiếp tốc độ cao.

Khi có mạng máy tính, chúng ta có thể thêm các máy trung tâm cũng như các máy khách một cách dễ dàng nhằm nâng cao năng lực tính toán. Các máy khách cũng có thể quản lý chương trình và dữ liệu của nó một cách linh hoạt, mà không cần nhờ vào người quản trị hệ thống. Nếu chúng ta cho phép máy khách lưu trữ chương trình và dữ liệu, thì chúng ta cũng cần giải quyết một số vấn đề: Khi dữ liệu thay đổi và chương trình được nâng cấp, client sẽ bị lỗi. Vì vậy, nên hạn chế lưu trữ chương trình và dữ liệu trên máy khách.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

Kiến trúc hai tầng đã mang lại khả năng đồ họa tinh vi và hệ thống cửa sổ cho máy khách, thay thế cho mô hình chỉ có văn bản của máy điện báo đánh chữ. Kiến trúc hai tầng vẫn được sử dụng rộng rãi hiện nay, chủ yếu cho Unix và file server.

Hoạt động của mô hình hai tầng: Client và Server trao đổi thông tin với nhau dưới dạng các thông điệp (Message). Thông điệp gửi từ Client sang Server gọi là các thông điệp yêu cầu (Request Message) mô tả công việc mà phần Client muốn Server thực hiện.



Hình 6.3. Kiến trúc chương trình Client-Server

Mỗi khi Server nhận được một thông điệp yêu cầu, Server sẽ phân tích yêu cầu, thực thi công việc theo yêu cầu và gửi kết quả về client (nếu có) trong một thông điệp trả lời (Reply Message). Khi vận hành, một máy tính làm Server phục vụ cho nhiều máy tính Client. Mỗi một ứng dụng Client-Server phải xác định một **Giao thức** (Protocol) riêng cho trao đổi thông tin, phối hợp công việc giữa Client và Server. Giao thức qui định một số vấn đề cơ bản sau:

- Khuôn dạng loại thông điệp.
- Số lượng và ý nghĩa của từng loại thông điệp.
- Cách thức bắt tay, đồng bộ hóa tiến trình truyền nhận giữa Client và Server.
-

Thông thường phần client đảm nhận các chức năng về giao diện người dùng, như tạo các form nhập liệu, các thông báo, các báo biểu giao tiếp với người dùng; phần Server đảm nhận các chức năng về lưu trữ dữ liệu. Nhờ đó dễ dàng cho việc bảo trì, chia sẻ, tổng hợp dữ liệu trong toàn bộ công ty hoặc tổ chức. Các chức năng về hoạt động nghiệp vụ có thể được cài đặt ở phần client hoặc ở phần server và khi đó sẽ tạo ra hai loại kiến trúc Client - Server là Fat Client và Thin Client.

Thin Client

Ngày nay, các doanh nghiệp thường sử dụng rộng rãi mô hình client-server. Trong mô hình này, tất cả máy tính để bàn của các nhân viên (gọi là máy client) sẽ được kết nối vào một máy tính trung tâm (máy server). Máy server sẽ đóng vai trò như một kho tài nguyên phục vụ cho tất cả các máy client nối vào nó. Các tài nguyên này thường là dữ liệu, các thiết bị phần cứng và một số dịch vụ mạng khác. Bên cạnh khả năng xử lý của server. Các máy client đều có năng lực xử lý (thường là yếu hơn server) và lưu trữ riêng của nó

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

để phục vụ cho các tác vụ không liên quan đến mạng như soạn thảo văn bản, bảng tính, vẽ hình, ...

Tuy nhiên, trong một doanh nghiệp, các ứng dụng phi mạng của các máy client thường là giống nhau. Ngày nay, vào bất cứ doanh nghiệp nào, bạn cũng đều thấy mỗi máy tính của các nhân viên đều có cài đặt Microsoft Windows, Microsoft Word, Microsoft Excel, các chương trình nghe nhạc, xem phim, ...

Ví dụ, để cài đặt toàn bộ hệ thống gồm có 100 máy tính của một công ty. Các admin phải cài đặt 100 lần giống hệt nhau ở mỗi máy tính. Và để bảo trì, họ cũng phải tiến hành bảo trì từng máy tính một. Đó là một công việc khó khăn và phức tạp. Đa số các nhân viên đều không phải là người rành máy tính, nên đối với họ, việc cài đặt hoặc xử lý các hỏng hóc của máy tính là việc rất khó khăn. Một trong những phản ứng tự nhiên của các nhân viên là “xin lỗi, cho phép tôi gặp quản trị!”. Và điều này đã làm cho công tác quản trị mạng trở thành một cơn ác mộng đối với các doanh nghiệp và đặc biệt là các thành viên quản trị.

Để giải quyết vấn đề “phân tán” này. Người ta đã nghĩ ra một giải pháp tương đối đơn giản. Đó là đặt tất cả các tập tin chương trình lẫn dữ liệu của phần mềm lên máy server. Khi muốn sử dụng phần mềm thì máy client sẽ tải chương trình (và dữ liệu) xuống máy client và sau đó, chương trình (hoặc dữ liệu) này sẽ được thi hành bằng CPU của máy client.

Tuy nhiên, giải pháp này sẽ ảnh hưởng đến sở thích cá nhân của mỗi nhân viên. Các nhân viên thường muốn mỗi phần mềm có một thể hiện riêng (như các Wall-Paper hoặc Screen Saver trên môi trường Windows, hoặc các skin của WinAmp...). Như vậy, làm thế nào có thể quản trị tập trung hệ thống máy tính mà vẫn đảm bảo được tính riêng tư của từng người dùng?

Công nghệ Thin-Client đã được đưa ra nhằm đáp ứng nhu cầu này. Thin-Client có nghĩa là “máy tính của người dùng có cấu hình tối thiểu”. Công nghệ Thin-Client tương tự như công nghệ client/server, điểm khác duy nhất và cũng quan trọng nhất là tất cả năng lực xử lý lẫn lưu trữ của toàn bộ mạng máy tính đều tập trung vào máy server. Mọi máy client đều chỉ còn một màn hình, bàn phím và chuột. Mọi chương trình hay phần mềm của người dùng sẽ được lưu trữ trên máy server và cũng sẽ được thi hành bởi CPU của máy server.

Một câu hỏi này sinh: “Thế thì mỗi người dùng muốn sử dụng một hệ điều hành khác nhau, một phần mềm khác nhau thì làm thế nào?”. Đây chính là điểm thú vị nhất của công nghệ Thin-Client. Mỗi khi có một user mới đăng ký vào hệ thống. Người đó sẽ được “cấp” cho một máy tính ảo. Trên máy tính ảo này, người dùng có thể chọn để sử dụng những phần mềm hoặc hệ điều hành (đã có sẵn trên máy server) mà họ ưa thích. Khi làm việc với chiếc máy tính ảo này, mọi người dùng vẫn cảm thấy là họ đang sở hữu

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

một chiếc máy tính thật của riêng họ với đầy đủ ý nghĩa của nó. Lợi điểm mấu chốt của công nghệ này nằm ở các điểm sau :

Đối với người dùng: Không cần phải cài đặt phần mềm mà chỉ cần đăng ký sử dụng. Tất cả những gì mà bạn cần làm là cắm máy tính của mình vào mạng, mở màn hình lên, tiến hành một số thủ tục đơn giản để đăng ký cấu hình cho “máy tính” của mình (thông thường là thao tác chọn một dòng trong một danh sách). Khi đó, môi trường Windows cùng với các phần mềm Office quen thuộc sẽ ngay lập tức xuất hiện trên màn hình sẵn sàng chờ lệnh của bạn mà không cần phải trải qua hàng giờ liền ngồi cài đặt máy tính.

Tính bảo mật cao hơn (vì toàn bộ thao tác đều được kiểm soát chặt chẽ bởi server). Những kẻ xâm nhập trái phép thì hầu như không có cơ hội lục lọi các thông tin quan trọng được lưu trữ trên các máy người dùng như trước nữa. Bây giờ, những kẻ xâm nhập muốn lục lọi thông tin đều phải đối diện với một khó khăn cực lớn là xâm nhập vào các máy server - vốn đã được bảo vệ cực kỳ nghiêm ngặt so với các máy tính của người dùng. Nguy cơ bị nhiễm virus vào các “máy tính” client gần như là không có.

Đối với chủ doanh nghiệp:

Chi phí thấp hơn: vì chỉ cần mua một máy chủ thay vì phải mua 100 máy cá nhân, cho phép sử dụng một đội ngũ quản trị với ít người hơn. Hơn nữa, việc tiêu thụ điện sẽ thấp hơn rất nhiều. Người ta ước tính rằng công nghệ này có khả năng giúp doanh nghiệp tiết kiệm đến 80% chi phí đầu tư cho CNTT.

Dễ dàng quản trị: chỉ cần cài đặt phần mềm một lần, hơn nữa chi phí mua phần mềm cũng rẻ hơn (mua 100 phần mềm riêng lẻ sẽ đắt hơn rất nhiều so với chỉ mua một phần mềm cho 100 người dùng). Việc bảo trì thiết bị và xử lý sự cố cũng đơn giản hơn (vì chỉ cần bảo trì máy server thay vì phải bảo trì 100 máy con).

Rất tiện lợi khi nâng cấp: chỉ cần nâng cấp một máy server thay vì phải nâng cấp riêng từng máy.

Tiết kiệm được không gian lưu trữ: thay vì mỗi máy tính phải lưu trữ một bản sao của phần mềm thì chỉ cần lưu một bản sao duy nhất trên server.

Bảo vệ trước sự xâm nhập của virus: mọi hoạt động của mạng đều có thể được theo dõi và giám sát tập trung để ngăn chặn ngay từ đầu nguy cơ bị phá hoại bởi các loại virus (thay vì trước kia phải khuyến cáo mỗi người dùng phải tự cài đặt các phần mềm chống virus và có trách nhiệm đối với việc bảo vệ máy tính của mình). Người dùng không có quyền tự ý download và cài đặt những phần mềm nguy hiểm.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

Cho đến nay, vị trí của công nghệ Thin-Client gần như đã được khẳng định. Với một chi phí đầu tư thấp hơn rất nhiều, công nghệ Thin-Client có thể hoàn toàn thay thế được cho mô hình truyền thống mà vẫn đảm bảo được năng lực lẫn tính hiệu quả của hệ thống. Tuy nhiên, mô hình này cũng có một số nhược điểm:

- Tạo ra nhiều thông điệp trao đổi giữa Client và Server sẽ dẫn đến có khả năng làm tăng giao tiếp trên mạng nên dễ tắc nghẽn. Yêu cầu Server phải có cấu hình cao.
- Việc tăng tải trên máy Server vì đồng thời thực hiện các dịch vụ Data Storage và Business Rule sẽ giảm hiệu năng.

Fat Client

Theo mô hình này, các quy tắc nghiệp vụ được cài đặt bên phía Client và phần Server chủ yếu thực hiện chức năng về truy vấn và lưu trữ thông tin. Mô hình này có ưu điểm là tạo ra ít giao thông trên mạng do dữ liệu tạm thời trong quá trình tính toán được lưu trên Client nên có thể giảm tắc nghẽn; yêu cầu Server ít có cấu hình mạnh vì Client được coi như một Thin Server nên giảm đầu tư vào các Server mạnh; có thể làm việc offline do dữ liệu tính toán tạm thời có thể lưu trữ ở Client nên không yêu cầu liên tục kết nối với Server; Server có công suất cao hơn, tăng số Client mà Server có thể hỗ trợ trong cùng một thời điểm.

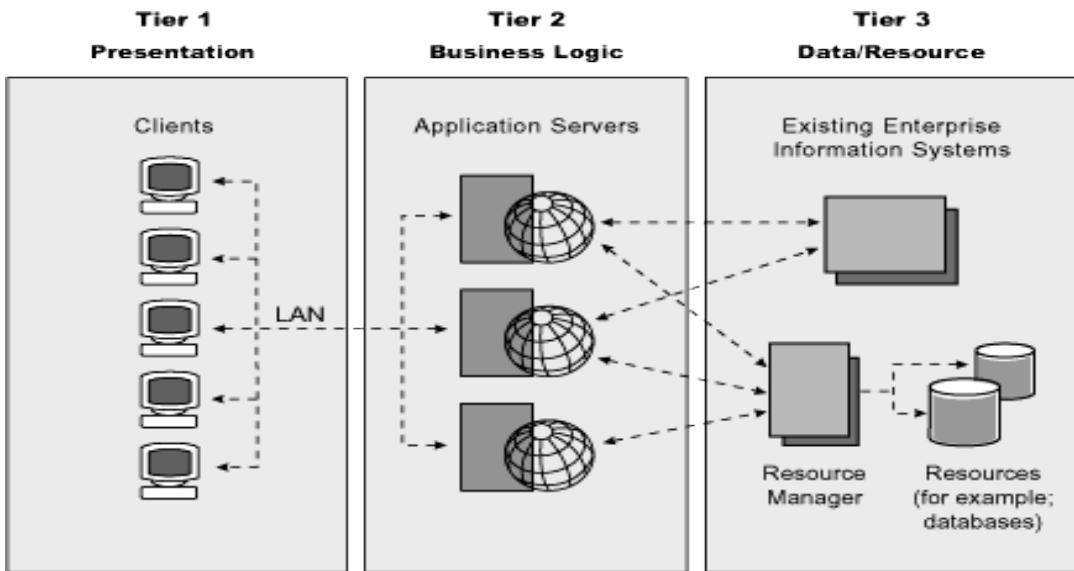
Tuy nhiên, nó có một số nhược điểm là chi phí đầu tư phần cứng cho máy Client cao, số lượng Client nhiều và do đó chi phí đầu tư tăng đáng kể, khó nâng cấp vì phải cài đặt lại toàn bộ máy Client khi muốn nâng cấp hệ thống.

6.4.4 Kiến trúc ba tầng

Kiến trúc 3 tầng (Hình 6.4) trở nên khá phổ biến vào những năm 1990 bằng cách chia hệ thống thành 3 phần: giao diện người dùng, chương trình logic và dữ liệu. Trong một hệ thống 3 tầng, mỗi chương trình bao gồm ít nhất 3 máy:

- Tầng giao diện người dùng (**User Interface**) hay Tầng client thể hiện giao diện của người sử dụng, ở đây người dùng có thể nhập dữ liệu và xem kết quả.
- Tầng ứng dụng (**Application Server, Business Rule**) được biết như tầng nghiệp vụ logic hay tầng dịch vụ để chạy mã chương trình đa luồng.
- Tầng dữ liệu (**Database Server, Data Storage**) nhằm lưu trữ dữ liệu và cung cấp cơ chế an toàn cho việc truy nhập đồng thời, với sự giúp đỡ của hệ quản trị cơ sở dữ liệu.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG



Hình 6.4. Cấu trúc mạng 3 tầng

Các ưu điểm của kiến trúc 3 tầng

- Bất kì hệ thống lớn nào cũng đều bao gồm 3 phần: cơ chế quản lý hiệu năng và bảo mật của dữ liệu; logic chương trình và giao diện người dùng. Bằng việc chia hệ thống thành các phần như trên, người lập trình có thể thực hiện công việc một cách đơn giản hơn.
- Sử dụng máy tính một cách hiệu quả: Tùy theo từng tầng chúng ta sẽ sử dụng các máy tính cho phù hợp. Chẳng hạn như: Chạy giao diện người dùng là một nhiệm vụ đơn giản không đòi hỏi máy tính là mainframe hay file server; việc thực thi logic chương trình yêu cầu sử dụng CPU, bộ nhớ, nhưng không đòi hỏi dung lượng đĩa quá lớn, vì vậy có thể sử dụng máy tính server; quản lý dữ liệu yêu cầu nhiều về khả năng tính toán, và dung lượng đĩa, do đó có thể sử dụng máy server hay mainframe.
- Cải thiện hiệu năng: có thể nhân rộng các máy ở lớp dữ liệu và lớp giữa để lan truyền tải tính toán (cân bằng tải), mỗi lớp được chuyên môn hóa và như vậy sẽ dễ dàng tối ưu hóa.
- Nâng cao tính bảo mật: Thường thì hệ thống 3 tầng sẽ được phát triển cho các máy client chạy thông qua mạng internet. Vì vậy, chúng ta phải có cơ chế bảo mật để bảo vệ máy chủ, chương trình và dữ liệu. Với cấu trúc 3 tầng, chúng ta có thể đặt cơ chế bảo mật ở tầng giữa nhằm tránh những tấn công vô tình hay cố ý từ bên ngoài. Tầng dữ liệu ở sau tầng giữa, do đó chúng ta không cần phải bảo mật cho phần cứng hay sự giao tiếp của chúng. Điều này giúp tầng dữ liệu chạy với tốc độ cao và dễ thao tác.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

- Đối với trường hợp chúng ta có một mainframe lưu trữ và xử lý dữ liệu trong nhiều năm, khi có sự cố chúng ta không muốn phải vứt bỏ tất cả và làm lại từ đầu. Nhưng phải giải quyết vấn đề đó như thế nào? Cấu trúc 3 tầng và mạng là phương án thích hợp nhất. Trong đó sử dụng tầng giữa làm đơn vị trung gian khi client kết nối với mainframe, hoặc khi server kết nối tới client.
- Tính linh hoạt: Được thể hiện rõ qua việc chúng ta có thể thêm hoặc bớt các máy tính trong hệ thống nếu hệ thống đó được thiết kế theo cấu trúc 3 tầng. Ví dụ: Khi phần logic được thiết kế đúng, chúng ta có thể phát triển nó theo cấu trúc một tầng, sau đó phát triển lên thành 2 tầng, 3 tầng tùy theo yêu cầu.
- Đa dạng kiểu dáng máy client: máy tính ở tầng client chỉ thực hiện nhận đầu vào và hiển thị kết quả trên màn hình, do đó chúng ta có thể sử dụng các thiết bị với các giao diện khác nhau như: máy tính cá nhân, PDAs, mobile-phone.... Khi đó, tầng giữa và tầng dữ liệu vẫn làm việc như nhau, không thay đổi.

Với tất cả những ưu điểm trên, kiến trúc ba tầng nên được sử dụng khi thiết kế các hệ thống dù có kích cỡ nhỏ hay lớn.

6.4.5 Máy tính cá nhân

Những năm 1970, những người say mê máy tính bắt đầu chế tạo máy tính cá nhân dùng ở nhà. Máy tính cá nhân đầu tiên được ra đời ở IBM đầu những năm 1980. Cuối những năm 1980, máy tính cá nhân trở nên hữu ích hơn khi chúng được kết nối với nhau thông qua mạng, tới server hay mainframe và điều này đã dẫn tới sự ra đời của kiến trúc hai tầng. Trong kiến trúc này, máy tính cá nhân có thể thực thi những nhiệm vụ phức tạp như soạn thảo và tạo tài liệu trong khi các máy tính trong mạng cung cấp e-mail, cho phép truy cập tới dữ liệu.

Từ năm 1990, PC trở nên phổ biến hơn do khả năng thực tế, giá cả thấp và có thể mang công việc về làm việc ở nhà. Trong khi đó, các máy trạm có chi phí cao, tốc độ thấp nên không có sự cạnh tranh trong thị trường máy tính cá nhân này.

6.4.6 Mạng máy tính

Giữa những năm 1990, những chiếc máy tính lớn như Unix và máy tính cá nhân PCs cạnh tranh nhau vị trí trong lĩnh vực máy tính, việc duy trì giá bán của PCs trở thành một vấn đề khó khăn. Như đã trình bày ở trên, nếu bạn đưa cho ai đó một chiếc máy cùng với ổ đĩa cứng, thì dữ liệu và chương trình trong ổ đĩa đó sẽ khác với trung tâm dữ liệu và chương trình. Điều này dẫn đến lỗi như dữ liệu của khách hàng quá hạn, chi phí tăng quá nhiều và tất cả máy tính của khách hàng sẽ trả lại trung tâm quản trị để cài đặt một phiên bản mới của phần mềm email hoặc bộ office...

Tất cả những điều trên đã tạo điều kiện cho mạng máy tính ra đời. Một mạng máy tính sẽ chứa tất cả dữ liệu và chương trình đáp ứng đầy đủ các yêu cầu của nhiều trung

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

tâm lớn. Điều này cũng yêu cầu một mạng máy tính có tốc độ nhanh (từ 10 đến 100 megabit trên giây). Mạng máy tính là một ý tưởng tuyệt vời. Thực sự không có gì ngạc nhiên, vì máy trạm với UNIX đã có nhiều thành công đáng kể sau một vài năm sử dụng và cũng không phá vỡ sự toàn vẹn của PCs.

Mạng máy tính được những người yêu thích Java sử dụng để phát triển Java một cách rộng khắp. Các máy Unix và các chương trình Java có thể chạy trong Unix giúp cho Java và Unix xuất hiện ở mọi nơi. Tuy nhiên mạng máy tính cũng vấp phải một số khó khăn là không có băng thông rộng và rất chậm. Để vượt ra khỏi PCs, những người sử dụng ở nhà cần bộ lướt web để có thể truy nhập vào kiến trúc ba tầng.

6.4.5. Internet và World Wide Web

Giữa những năm 80, nhiều nhà nghiên cứu và nhân viên chính phủ đã tạo ra một mạng diện rộng trên toàn thế giới mà ngày nay gọi là Internet. Internet được nghiên cứu và ứng dụng đầu tiên ở Mỹ. Internet được đặc trưng bởi sự truy nhập miễn phí tới các máy chủ trung tâm, nó cho phép máy tính và con người có thể xác định vị trí của các máy tính khác thông qua địa chỉ internet được xác định bởi một chuỗi thập phân (ví dụ 100.99.88.32), và nó được mô hình hóa sang mã ASCII (ví dụ, www.nowherecars.com). Dưới việc mô hình hóa như vậy, internet phát triển một giao thức bậc thấp gọi là TCP/IP: nó có thể hiểu là một giao thức giúp sử dụng các khả năng của internet một cách dễ dàng.

Vào đầu những năm 1990, khi mà internet đã được thiết lập một cách vững chắc trong cộng đồng nghiên cứu, Tim Berners-Lee làm việc trong một viện nghiên cứu CERN đã đưa ra ý kiến sử dụng các siêu liên kết (hyperlinks) từ văn bản này tới văn bản khác trong môi trường internet. Ông đã phát minh ra ngôn ngữ thể hiện tài liệu HTML và giao thức HTTP để lấy tài liệu qua siêu liên kết từ bất cứ máy nào. Vị trí của một văn bản có thể được đặc tả bởi một Uniform Resource Indication (URI) dưới dạng <http://www.nowherecar.com/index.html>.

Phát minh của Berners-Lee đã làm thay đổi World Wide Web. Bởi giữa những năm 90, người nào muốn tạo văn bản cho người khác đọc có thể triển khai trên Web server và ai muốn đọc có thể chạy lướt web browser trên máy khách của họ. Ngày nay sự khác biệt giữa Internet và World Wide Web gần như không có. Bạn có thể nghe thấy một số thuật ngữ có thể được sử dụng để thay thế nhau như Internet, Net, World Wide Web, Web, Siêu xa lộ thông tin.

6.4.6. Intranets

Internet có hai vấn đề lớn: chậm và không an toàn. Vấn đề sẽ nghiêm trọng nếu thông tin được truyền cách xa hàng ngàn km để tới đích. Việc không an toàn của internet xuất hiện khi có nhiều người có thể truy nhập và xem thông tin một cách trái phép. Thông thường, lỗi này có thể khắc phục bằng cách mã hóa thông tin, tuy nhiên những người không được phép vẫn có thể giải mã để biết được thông tin bên trong.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

Những vấn đề trên tạo tiền đề cho intranet ra đời, intranet có nghĩa là “mini Internet” hoạt động dưới sự bảo vệ giúp cách biệt với môi trường mạng bên ngoài. Intranet cũng có nghĩa là “internet inside”. Một mạng intranet sẽ được điều khiển và quản lý bởi một tổ chức hoặc chính phủ. Từ nội mạng intranet chúng ta có thể truy nhập ra môi trường mạng bên ngoài mà không cần phải lo lắng đến vấn đề bảo mật. Với mạng intranet toàn cầu, thông tin có thể được truyền với khoảng cách rất lớn.

Vấn đề lớn nhất của intranet là thực hiện bảo mật khi nhân viên truy nhập ra mạng internet bên ngoài (qua email, thu thập thông tin hoặc truy cập các website). Để làm được điều đó, chúng ta cần sử dụng một công cụ mà được gọi là Internet firewall. Một firewall là một phần mềm cho phép các máy tính trong mạng intranet truy cập một số địa chỉ TCP/IP, nó cũng làm cho các địa chỉ IP trong mạng intranet không hiển thị với bên ngoài. Internet firewall cũng có thể thực hiện các chức năng khác. Ví dụ, chúng có thể làm cho máy tính của nhân viên chỉ có thể sử dụng duy nhất một Web browser để truy nhập Internet – nó sẽ giúp hiển thị những chương trình có tính phá hoại từ một liên kết bí mật bên ngoài, chúng ta có thể sử dụng Internet firewall để ngăn chặn hacker và virus.

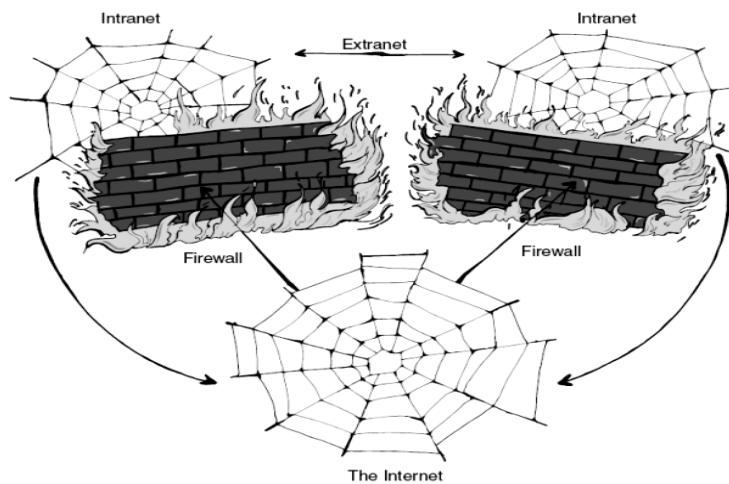
6.4.7. Các mạng extranet và các mạng riêng ảo

Vậy làm thế nào chúng ta khai thác được thành quả và các lợi ích bảo mật của các mạng intranet cho việc truyền thông trong mô hình nghiệp vụ? Chúng ta có thể sử dụng một mạng extranet, một kiểu kết nối an toàn giữa một hoặc nhiều mạng intranet. Thuật ngữ extranet là một thuật ngữ đa nghĩa trên intranet, viết tắt cho cụm từ mạng “intranet bên ngoài”. Cách dễ dàng nhất để tạo một extranet là chạy một phần của phần mềm trên các tường lửa mạng Internet ở phần rìa của mỗi mạng intranet, nó làm 2 việc sau:

- Cho phép thông tin chuyển từ tường lửa này đến tường lửa kia.
- Sử dụng việc mã hóa mạnh để bảo vệ thông tin khi nó truyền trên mạng Internet.

Một extranet cũng được coi như là một mạng riêng ảo (VPN: Virtual Private Network). Dạng đơn giản nhất của VPN, thậm chí còn trước cả ý tưởng về một mạng extranet, là một công nhân quay số đến mạng LAN chung của họ từ nhà bằng cách sử dụng phần mềm chuyên biệt.

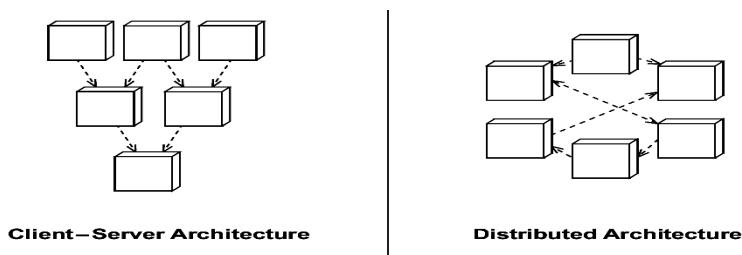
CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG



Hình 6.5. Một ví dụ về việc kết hợp các kiểu mạng khác nhau

6.4.8 Kiến trúc Client – Server và kiến trúc phân tán

Bất cứ khi nào chúng ta kết nối với nhiều máy hoặc nhiều hệ thống phần mềm, chúng ta phải chọn giữa hai loại client – server hoặc phân tán, như được trình bày ở Hình 6.6. Mặc dù xuất phát từ thời mainframe, “client – server” có nghĩa là chúng ta có một lượng lớn các client, đơn giản gửi yêu cầu đến một server đa luồng lớn xử lý các yêu cầu đó. Ngược lại, kiến trúc phân tán (hay ngang hàng) được đặc trưng bằng một tập các peer tự chủ, truyền thông nhau theo bất kỳ một hướng nào khi có nhu cầu phát sinh.



Hình 6.6. Kiến trúc Client – Server và phân tán

Ví dụ quen thuộc nhất của kiến trúc client – server là mô hình thương mại điện tử: trình duyệt web của các khách hàng phát đi các yêu cầu để kết nối với các máy chủ Web và khi đó máy chủ web phát đi các lệnh đến các hệ thống đầu cuối. Phần lớn các hệ thống hai tầng và ba tầng là mô hình client – server.

Một ví dụ hay cho kiến trúc phân tán là khi một công việc tính toán đồ sộ trải rộng ra trên nhiều máy tính trên Internet. Nếu chúng ta có một lượng rất lớn dữ liệu hoặc tính toán mà cần phải thực hiện và chúng ta có thể chia dữ liệu hoặc tính toán đó, thì chúng ta có thể phân tán bài toán trên nhiều máy độc lập.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

Một ví dụ là SETI@home, một tổ chức phi lợi nhuận tìm kiếm các tín hiệu radio lạ ngoài trái đất. Việc tìm kiếm thông tin từ ngoài vũ trụ bắt đầu khi một dự án của NASA với những kính thiên văn radio quét các khoảng trời và ghi lại bất cứ thứ gì đi qua. Thông tin này được xử lý để thấy liệu chúng có chưa dấu hiệu của sóng radio được truyền từ người ngoài hành tinh không. Dự án SETI đã bị hủy (vì thiếu ngân sách hay thiếu niềm tin) và được đổi tên thành SETI@home. Những người nhiệt tình phát triển SETI@home cùng lấy dữ liệu radio giống nhau và phân tán nó đến các máy trên mạng Internet: mỗi máy cá nhân phân tích dữ liệu của nó sử dụng phần mềm được chạy như một màn hình chờ; bất kì kết quả thú vị nào cũng sẽ được gửi trở lại máy chủ trung tâm để phân tích kỹ hơn (xem thêm tại website của SETI@home, setiathome.ssl.berkeley.edu).

Mặc dù SETI@home có một kho dữ liệu tập trung, nó vẫn là một kiến trúc phân tán vì phần lớn các xử lý được thực hiện bởi các điểm độc lập. Ý tưởng gộp một số lượng lớn các máy lại cùng nhau nhằm giải quyết một bài toán phức tạp cũng được sử dụng để tìm các số nguyên tố và nghiên cứu bệnh ung thư. Ý tưởng cơ bản bây giờ được nghiên cứu dưới tiêu đề *tính toán lưới* (Grid Computing).

Các thuật ngữ “client – server” và “distributed” (hay “peer to peer”) cũng được sử dụng để mô tả các kiến trúc phần mềm, không phụ thuộc vào việc phần mềm được triển khai như thế nào trên các máy hoặc trên các mạng. Một ví dụ thú vị đó là các đối tượng chạy trong một chương trình: thường thì chúng ta viết các đối tượng như các server mà có thể được sử dụng lại trong các ngữ cảnh khác nhau với những đối tượng client khác nhau; nhưng với những ứng dụng đặc biệt, chúng ta có thể viết một nhóm các đối tượng cùng cộng tác với nhau theo kiểu phân tán.

Các liên kết truyền thông mạng có khuynh hướng là trao đổi hai chiều, nghĩa là mặc dù các liên kết có thể ban đầu được mở bởi client nhưng server vẫn có thể gửi thông tin đến client (client có nhận hay không phụ thuộc người thiết kế của phần mềm client). Vì thế, sự khác nhau giữa client – server và phân tán chỉ là một khía cạnh tạo ra, được sử dụng bởi những người thiết kế để cấu trúc giải pháp của họ theo cách này hay cách khác.

Nói chung, các kiến trúc client –server phát triển dễ hơn, nhưng chúng không đem lại hiệu năng theo lý thuyết (ví dụ, client thường rỗi rải khi server đang xử lý một trong những yêu cầu của nó). Các kiến trúc phân tán thường thì khó hơn để phát triển nhưng chúng có thể đem lại hiệu quả tốt hơn. Thường thì việc lựa chọn kiến trúc rất tự nhiên từ hệ thống. Ví dụ tạo một phiên mua bán là một quá trình xử lý theo từng bước (khách hàng hỏi các chi tiết về sản phẩm, người bán cung cấp các chi tiết sản phẩm, khách hàng chọn mua sản phẩm, người bán đưa ra mẫu đơn để mua, khách hàng điền vào mẫu đơn và gửi đi...) và như vậy đó là một tương tác kiểu client –server. Ngược lại, bộ mô phỏng chuyến bay nhiều người dùng yêu cầu máy tính của phi công thực thi công

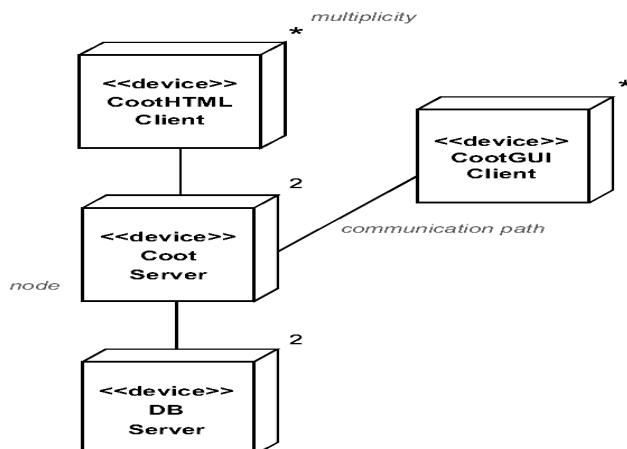
CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

việc theo thời gian thực biểu diễn qua buồng lái và việc truyền thông duy nhất cần thiết là phát tin định kì về vị trí hiện thời của máy bay. Vì thế hệ thống mô phỏng chuyến bay nhiều người dùng đương nhiên là kiến trúc phân tán.

6.4.9. Biểu diễn cấu hình mạng trong UML

Các kiến trúc hệ thống có thể được mô tả trong UML bằng một biểu đồ triển khai (xem Hình 6.7). Biểu đồ triển khai đơn giản này chỉ nêu ra các nút, đường truyền thông và vô số các điểm khác. Mỗi nút trong biểu đồ này thể hiện một máy trạm (thể hiện với từ khóa UML <<device>>). Một đường truyền thông chỉ ra rằng hai nút có thể liên lạc với nhau theo một cách nào đó. Các nút có daaus * thể hiện có rất nhiều nút có thể tồn tại trong thời gian chạy: vì thế trong biểu đồ, chúng ta có các nút lặp lại (CootServer và DBServer) và các nút nhân lên nhiều lần (CootHTMLClient và CootGUIClient).

Các biểu đồ triển khai giống như các biểu đồ class và biểu đồ đối tượng, trong đó chúng có thể nêu ra các kiến trúc có thể (kiểu nốt) và các kiến trúc thực sự (các thể hiện nốt). Khi chỉ ra các thể hiện nút, giống như các đối tượng trong biểu đồ đối tượng, nhãn nút có dạng tên:Kiểu, và nên được gạch chân.



Hình 6.7: Biểu đồ triển khai cơ bản cho Coot.

Trong Hình 6.7, tầng dữ liệu iCoot bao gồm 2 server cơ sở dữ liệu (gọi là DBServer). Có hai nút như thế cung cấp lượng lớn dữ liệu và độ tin cậy. Tầng giữa, liên lạc với tầng dữ liệu bao gồm hai máy server (CootServer), lại được gấp đôi với mục đích cho sự tin cậy và khả năng truyền dữ liệu. Mỗi CootServer có thể được truy nhập đồng thời bởi bất cứ điểm CootHTML-Client nào. Cuối cùng chúng ta cũng có thể cung cấp sự truy nhập từ các nút CootGUIClient.

6.5 THIẾT KẾ CHO TƯƠNG TRANH

Hầu hết các hệ thống, đặc biệt là hệ thống mạng có nhiều hoạt động xảy ra cùng một lúc; có nghĩa là, chúng là những hệ thống tương tranh. Có những sự liên quan khi thiết kế hệ thống như một thể thống nhất và cũng khiến những hoạt động tiến trình riêng lẻ như một phần

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

của hệ thống. Mặc dù sẽ dễ dàng hơn nhiều để phát triển hệ thống nếu chúng ta có thể dựa vào tất cả người dùng và các tiến trình để thiết lập một hàng đợi có thứ tự, trong thực tế chúng ta phải xáo trộn thành thứ tự bằng nỗ lực lập trình của chúng ta.

Tương tranh nói về những vấn đề, nhiều lần:

- Làm sao để đảm bảo rằng thông tin được cập nhật đầy đủ trước khi bất kỳ ai có thể thao tác trên cập nhật đó; ví dụ; dừng bất kỳ sự truy nhập chi tiết của một mẫu ô tô mới tới khi tất cả các chi tiết đã được thêm vào.
- Làm sao để đảm bảo rằng thông tin không được cập nhật trong khi nó đang được đọc; ví dụ không xóa một mẫu ô tô trong khi các chi tiết của nó đang được xem.

Ở mức độ thấp, sự giải quyết cơ sở dữ liệu và điều khiển luồng được dùng để bảo vệ dữ liệu bên trong các tiến trình riêng lẻ là một ví dụ. Ở mức cao, chúng ta cần sử dụng các quy tắc hệ thống và quy tắc nghiệp vụ để điều khiển các hành vi tương tranh.

Ở mức độ thấp chúng ta sử dụng các transaction để xử lý. Một transaction được hiểu là một đơn vị công việc, được sử dụng để ngăn chặn nhiều truy nhập cở sở dữ liệu. Thông tin trong cơ sở dữ liệu được chuyển từ một trạng thái quán này sang một trạng thái quán khác, dữ liệu không bị cập nhật một phần mà chỉ có thể cập nhật tất cả hoặc không cập nhật gì cả. Có hai giải pháp xử lý tương tranh là pessimistic và optimistic. Với pessimistic concurrency, cơ sở dữ liệu được đảm bảo rằng không một transaction nào có thể thực thi xung đột truy nhập trong khi một transaction đang thực thi, thường liên quan đến CSDL quan hệ. Cơ sở dữ liệu quan hệ: dữ liệu được lưu trữ trong các bảng dữ liệu gọi là các thực thể, giữa các thực thể này có mối liên hệ với nhau gọi là các quan hệ, mỗi quan hệ có các thuộc tính, trong đó có một thuộc tính là khóa chính. Các hệ quản trị hỗ trợ cơ sở dữ liệu quan hệ như: MS SQL server, Oracle, MySQL... VỚI optimistic concurrency, các transaction đều được yêu cầu, khi một transaction yêu cầu cơ sở dữ liệu kiểm tra để không một transaction nào khác có thể thực thi xung đột trong cùng một thời gian, thường liên quan đến cơ sở dữ liệu hướng đối tượng. Cơ sở dữ liệu hướng đối tượng: dữ liệu cũng được lưu trữ trong các bản dữ liệu nhưng các bảng có bổ sung thêm các tính năng hướng đối tượng như lưu trữ thêm các hành vi, nhằm thể hiện hành vi của đối tượng. Mỗi bảng xem như một lớp dữ liệu, một dòng dữ liệu trong bảng là một đối tượng.

Cách tiếp cận dễ nhất để tương tranh là hạn chế hệ thống hoặc nêu các quy tắc nghiệp vụ mở rộng, đặc biệt nếu kinh nghiệm người dùng không giảm đáng kể. Ví dụ, cho iCoot, hơn là cố gắng đối phó với việc cập nhật danh mục ô tô trong khi khách hàng đang truy nhập nó, chúng ta có thể cập nhật danh mục tại một cơ sở dữ liệu riêng biệt và chuyển đổi cơ sở dữ liệu mỗi ngày một lần; đó là cách các hệ thống mạng con có thể áp dụng để danh mục là chỉ đọc, nó khiến cho việc viết mã trở nên dễ dàng hơn nhiều. (Kinh nghiệm người dùng hơi giảm – trong số ít trường hợp họ sẽ phải thông báo rằng mẫu ô tô

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

họ vừa có đề lưu trữ đã được ngừng – nhưng chúng ta chọn cách chấp nhận chúng). Đó là một ràng buộc nhân tạo trên hệ thống.

Quy tắc nghiệp vụ cũng có thể khiến cuộc sống của người phát triển dễ dàng hơn. Quả thực, quy tắc nghiệp vụ đôi khi được đặt ra vì có cách đơn giản không **off-the-shelf** để giải quyết vấn đề. Ví dụ, xem xét một hệ thống mua bán vé hòa nhạc. Khách hàng Fred tới một văn phòng đặt vé tại Paris cùng thời điểm với khách hàng Beryl tới một văn phòng đặt vé khác tại New York. Cả hai đều quyết định rằng họ muốn một vé tại cùng một buổi hòa nhạc tại London. Thật không may, chỉ còn lại một vé duy nhất. Làm thế nào để chúng ta quyết định bán vé cho ai? Cả hai sẽ cùng hỏi “Có còn vé không?” Cả hai nhân viên bán vé sẽ cùng kiểm tra trên hệ thống và sẽ cùng trả lời “Còn vé”. Bây giờ chúng ta có một tranh chấp: người khách hàng đầu tiên nói “Đồng ý, tôi sẽ mua nó” sẽ thắng, phụ thuộc vào khả năng của nhân viên và sự trễ mạng từ Paris và New York tới nơi đặt máy chủ thực sự.

Trong kịch bản vé nói trên, chúng ta phải đảm bảo, ít nhất là chúng ta không tình cờ bán hai vé khi chỉ còn một ghế duy nhất. Đó là một vấn đề tương tranh mức tiến trình, bởi nó có thể được điều khiển bởi tiến trình máy chủ (những yêu cầu được phân loại: thứ nhất đến để mua vé, thứ hai đưa ra một thông điệp lỗi). Nhưng nó có thể không đủ tốt từ thái độ nghiệp vụ vì chúng ta kết thúc bằng một khách hàng bức tức là người được trả lời rằng còn một vé mà chỉ ngay đó vài giây đã không còn gì cả.

Để tránh những khách hàng bức tức, chúng ta có thể đưa ra một quy tắc nghiệp vụ mở rộng: khi một nhân viên truy vấn về số vé còn lại, nếu chỉ còn một vé, nó tạm thời được lưu trữ - việc đặt vé cho tới khi nhân viên hủy bỏ yêu cầu hoặc việc đặt vé hết thời hạn (ví dụ, việc đặt vé tạm thời được phép kéo dài 10' nếu nhân viên không hủy bỏ nó trước). Với quy tắc nghiệp vụ mới này, chúng ta có thể đảm bảo rằng chỉ khách hàng đầu tiên đưa ra yêu cầu tới máy chủ vé được trả lời rằng còn lại một vé (máy chủ vé có thể tích hợp truy vấn và lưu trữ bên trong một dịch vụ nghiệp vụ đơn)

Giả sử việc đặt vé cho Fred thành công. Nhân viên đang phục vụ Fred có 10' để thuyết phục anh ta chi trả cho vé đã lưu trữ, hoặc nhân viên có thể hủy bỏ trong thời gian đó nếu Fred thay đổi ý định. (Nhân viên cũng sẽ hủy bỏ nếu phương thức chi trả của Fred lỗi, đó là một lý do cho việc có lưu trữ tạm thời tại nơi đầu tiên). Beryl lúc này nghĩ rằng vé của buổi hòa nhạc đã được bán hết trước khi cô ấy tới cửa hàng đặt vé, vì thế cô ấy không có lý do gì để tức giận với nhà cung cấp vì đã cung cấp thông tin sai lệch. (Nếu cuối cùng Fred không mua vé và sau đó Beryl biết rằng vé chưa bán hết, cô ấy có thể chỉ đơn giản nói rằng “Ai đó đã hủy bỏ”)

Sự xem xét chi tiết của vấn đề tương tranh và cách giải quyết chúng được đề cập ở những phần sau của cuốn sách này. Trong bất cứ trường hợp nào, thực sự không có sự thay thế nào cho việc ngồi xuống và suy nghĩ kỹ về nó. Bây giờ, chúng ta rút ra những nhận xét:

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

- Để nhìn và cảm nhận về một hệ thống tương tranh được thiết kế tốt không khó so với phiên bản một người dùng.
- Dịch vụ nghiệp vụ của chúng ta cùng dành cho cả trường tương tranh và đơn người dùng.
- Để tạo ra một đối tượng nghiệp vụ tương tranh an toàn, chỉ cần thêm vào các thông điệp và đối tượng được cung cấp, vì thế, thông điệp nghiệp vụ (và những thuộc tính kết hợp) có thể được thiết kế đơn lẻ.

Nếu bạn nghĩ về một trường hợp tương tranh có thể là lý do làm khó hệ thống của bạn, đừng tiến tới việc triển khai cho tới khi bạn có thể đảm bảo rằng trường hợp đó không là một vấn đề lớn. Hãy chắc chắn với hệ thống của bạn bởi vì thực tế sẽ khó khăn hơn.

6.6 AN TOÀN THIẾT KẾ

Như chúng ta đã biết, vấn đề bảo mật là một vấn đề quan trọng, và luôn luôn được quan tâm một cách đặc biệt trong hầu hết các hệ thống. Do đó, việc xem xét chi tiết về bảo mật cũng rất được quan tâm, và có riêng những quyển sách giới thiệu rất chi tiết về nó. Phần này chỉ trình bày tóm tắt quan về vấn đề này

6.6.1 Các khía cạnh bảo mật

Một hệ thống được gọi là an toàn khi hệ thống được bảo vệ khỏi sự lạm dụng, dù là vô tình hay cố ý. Bảo mật là thuật ngữ rộng hơn rất nhiều, nó có thể được chia thành năm khía cạnh :

- Privacy (sự riêng tư): Thông tin có thể được che dấu, và chỉ ở trạng thái sẵn sàng với những người dùng được phép tác động vào nó (Xem hoặc chỉnh sửa). đảm bảo cho người sử dụng khai thác tài nguyên của hệ thống theo đúng chức năng, nhiệm vụ đã được phân cấp, ngăn chặn được sự truy nhập thông tin bất hợp pháp
- Authentication (xác thực): Cân biết nơi mà mỗi phần thông tin được gửi đến để quyết định xem phần thông tin đó có đáng tin cậy hay không
- Irrefutability (tính không thể bác bỏ được): ngược lại với authentication, irrefutability nhằm đảm bảo rằng người tạo ra thông tin không thể phủ nhận rằng họ chính là người tạo ra nó. Điều này hữu ích cho chúng ta khi có bất kì sai sót xảy ra
- Integrity (tính toàn vẹn): đảm bảo rằng thông tin không bị mất mát trên đường đi, bảo đảm sự nhất quán của dữ liệu trong hệ thống. Các biện pháp đưa ra ngăn chặn được việc thay đổi bất hợp pháp hoặc phá hoại dữ liệu

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

- Safety (tính an toàn): phải điều khiển việc truy cập tài nguyên (như máy móc kỹ thuật, quy trình, cơ sở dữ liệu và các file) . Tính an toàn cũng được hiểu như là quyền hạn

Bốn khía cạnh đầu tiên có thể được thực hiện bằng cách sử dụng mã hóa số. Riêng tính an toàn thì phức tạp hơn nhiều. Thông thường, khi một đoạn code đang được thực thi, hệ điều hành sẽ áp dụng một số loại điều khiển mà đoạn code có thể thực hiện được; điển hình như điều khiển truy cập các file, các thư mục và các chương trình khác. Các hệ điều hành có lỗi hoặc lỗ thủng.Thêm vào đó, sự điều khiển của hệ điều hành là không thể thay đổi được.

Nếu hệ thống của chúng ta điều khiển qua mạng, thì vấn đề an toàn càng trở nên quan trọng . Bởi vì , qua mạng, các hacker có thể cố gắng chiếm đoạt chương trình đang chạy trên máy của ta nhằm mục đích phá hoại.

Một số biện pháp bảo mật trong hệ thống thông tin:

Thiết lập quy tắc quản lý: Mỗi tổ chức cần có những quy tắc quản lý của riêng mình về bảo mật hệ thống thông tin trong hệ thống. Có thể chia các quy tắc quản lý thành một số phần :

- Quy tắc quản lý đối với hệ thống máy chủ
 - Quy tắc quản lý đối với hệ thống máy trạm
 - Quy tắc quản lý đối với việc trao đổi thông tin giữa các bộ phận trong hệ thống, giữa hệ thống máy tính và người sử dụng, giữa các thành phần của hệ thống và các tác nhân bên ngoài.
1. An toàn thiết bị
 - Lựa chọn các thiết bị lưu trữ có độ tin cậy cao để đảm bảo an toàn cho dữ liệu. Phân loại dữ liệu theo các mức độ quan trọng khác nhau để có chiến lược mua sắm thiết bị hoặc xây dựng kế hoạch sao lưu dữ liệu hợp lý.
 - Sử dụng các hệ thống cung cấp, phân phối và bảo vệ nguồn điện một cách hợp lý .
 - Tuân thủ chế độ bảo trì định kỳ đối với các thiết bị

Thiết lập biện pháp bảo mật

Cơ chế bảo mật một hệ thống thể hiện qua quy chế bảo mật trong hệ thống, sự phân cấp quyền hạn, chức năng của người sử dụng trong hệ thống đối với dữ liệu và quy trình kiểm soát công tác quản trị hệ thống. Các biện pháp bảo mật bao gồm:

- Bảo mật vật lý đối với hệ thống : hình thức bảo mật vật lý khá đa dạng, từ khóa cứng, hệ thống báo động cho đến hạn chế sử dụng thiết bị. Ví dụ như loại bỏ đĩa mềm khỏi các máy trạm thông thường là biện pháp được nhiều cơ quan áp dụng.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

- Các biện pháp hành chính như nhận dạng khi vào văn phòng, đăng nhập hệ thống hoặc cấm cài đặt phần mềm, hay sử dụng các phần mềm không phù hợp với hệ thống.
 - Mật khẩu là một biện pháp phổ biến và khá hiệu quả . Tuy nhiên mật khẩu không phải là biện pháp an toàn tuyệt đối. Mật khẩu vẫn có thể mất cắp sau một thời gian sử dụng.
 - Bảo mật dữ liệu bằng mật mã tức là biến đổi dữ liệu từ dạng nhiều người dễ dàng đọc được, hiểu được sang dạng khó nhận biết.
 - Xây dựng bức tường lửa, tức là tạo hệ thống bao gồm phần cứng và phần mềm đặt giữa hệ thống và môi trường bên ngoài như Internet. Thông thường, tường lửa có chức năng ngăn chặn những thâm nhập trái phép (không nằm trong danh mục được phép truy nhập) hoặc lọc bỏ, cho phép gửi hay không gửi các gói tin.

6.6.2 Mã hóa và giải mã

Đây là một số ý tưởng được đưa ra về cách mã hóa và giải mã số có thể được sử dụng để cung cấp tính riêng tư (privacy), tính xác thực (authentication), tính không thể bác bỏ (irrefutability), và tính toàn vẹn (integrity). Đầu tiên, ý tưởng cơ bản là mã hóa thông tin, nghĩa là hòa trộn nó để vô hiệu hóa bất cứ ai muốn ăn cắp nó. Để làm được điều này, phương thức hòa trộn phải có thể đảo ngược được và nó phải được người nhận mong đợi biết. Ngược lại của việc mã hóa là giải mã. Sự thành công của một kỹ thuật mã hóa được tóm tắt lại bởi 2 điều :

- Độ khó để phá được mã
- Độ an toàn khi phân phát khóa đến người nhận mong đợi

Việc mã hóa số và giải mã số là mã hóa thông tin số giữa các máy tính, nơi mà phần mềm có thể làm tất cả các công việc khó khăn. Mức độ của mã hóa số thường thể hiện ở độ dài bit: 128 bit mã hóa được coi như là nhỏ nhất hiện nay, và 1024 bit mã hóa là có thể thực hiện được.

Những khóa số được dựa trên các số quan trọng, do đó việc bẻ khóa sẽ khó khăn bởi vì nó bao gồm việc thử để tìm các nhân tố chính của khóa, đây là một tiến trình khó khăn. Bản thân các khóa được phân tán và sử dụng các chứng chỉ số. Với sự giúp đỡ của một chứng chỉ đáng tin cậy, xác thực khóa, chúng ta sẽ ko bị đánh lừa bởi các hacker.

Dưới đây là những chỉ số để tạo ra bốn khía cạnh bảo mật xác định trước đây có thể được thực thi sử dụng các mật mã:

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

- Privacy (sự riêng tư): Trong mã hóa và giải mã số, sự phân tán khóa an toàn được hoàn thành nhờ sử dụng các cặp khóa public/private, các chứng chỉ và các chứng chỉ tin cậy.
- Authentication (sự xác thực): Điều này dựa vào việc xác minh ra nguồn gốc của khóa, sử dụng chứng chỉ và chứng chỉ tin cậy. Nói một cách đại khái, nếu chúng ta có thể giải mã thành công thông tin và và chúng ta biết nguồn gốc của khóa, chúng ta biết rằng thông tin phải tới từ cùng một nơi với khóa.
- Irrefutability (Tính ko thể bác bỏ được): Vì độ tin cậy dựa trên việc chứng minh nguồn gốc của khóa, nên khi chúng ta đã chứng thực một phần của thông tin, thì thông tin của chúng ta sẽ ko thể bác bỏ được.
- Integrity (Tính toàn vẹn): Đầu tiên, chúng ta mã hóa thông tin và giải mã thông tin tới client. Sau đó, client giải mã phiên bản đã mã hóa và so sánh kết quả với phiên bản đã giải mã- rõ ràng, 2 bản này nên khớp với nhau (sự thay đổi ngẫu nhiên của chúng là rất nhỏ). Do đó, chúng ta có thể tin rằng chúng ta đã nhận đúng thông tin.

Có hai vấn đề với việc kiểm tra tính toàn vẹn: đầu tiên, chúng ta gửi thông tin đã giải mã , sau đó chúng ta gửi thông tin lần hai (phiên bản mã hóa quan trọng như phiên bản đã được giải mã). Vấn đề đầu tiên có thể khắc phục bằng cách gửi tất cả mọi thứ qua một đường truyền đảm bảo. Vấn đề thứ hai có thể khắc phục bằng một sự tối ưu dựa trên các tập san thông điệp – một tập san là một chuỗi nhỏ các bit sinh ra từ một phần của thông tin, sử dụng thuật toán ko thể đảo ngược được. Không cần đi sâu vào chi tiết, chúng ta kết thúc việc gửi các tập san mã hóa, nhưng ko mã hóa thông tin. Kết quả cuối cùng là chúng ta đạt được mức độ tương tự so với kiểm tra mức độ toàn vẹn, nhưng thông tin chỉ được gửi một lần.(Theo các bản ghi, một tập san mã hóa được gọi là chữ ký số).

6.6.3 Nhũng quy luật chung về an toàn

Khi thiết kế một hệ thống bảo mật tốt, ta buộc phải ngồi và suy nghĩ một cách nghiêm túc để chắc chắn rằng không ai có thể xâm nhập bất hợp pháp vào hệ thống. Sau đây là một vài điều đáng lưu ý để bảo vệ hệ thống của bạn:

- Ngăn chặn việc xâm nhập máy chủ khi không được phép.
- Các thông tin nhạy cảm như: chi tiết về ý tưởng kinh doanh hay chiến lược kinh doanh, các hồ sơ cá nhân, chi tiết về số thẻ tín dụng mà bạn đang dùng, các thông tin liên quan đến an ninh quốc gia....Các thông tin này phải được đảm bảo không bị truyền ra ngoài.
- Đảm bảo thông tin khi đi ra bên ngoài không bị “ nghe lén “ trong quá trình truyền và chỉ có đúng người nhận mới có thể đọc được.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

- Bảo vệ mật mã (password) của khách hàng và nhân viên,nó không chỉ đơn thuần là chính sách bảo mật mà nó còn phải có tính riêng tư cao.
- Bảo vệ tài nguyên hệ thống của server
- Bảo vệ tài nguyên hệ thống của client: bảo vệ client chống lại các truy cập bất hợp pháp đến tài nguyên và chống lại sự phá hoại của chúng (vì muốn cung cấp 1 chất lượng dịch vụ tốt và vì không muốn bị kiện ra tòa do 1 vài điều sai lầm).

Ngoài ra,chúng ta có thể thuê “ethical hackers” (hacker đạo đức), hay những người có chuyên môn để mở xé hệ thống của chúng ta nhằm kiểm tra và nó sẽ giúp ta không bị quá bất ngờ khi một lỗi nào đó xuất hiện.

Trên chỉ là một số các lưu ý khi xây dựng các chính sách bảo mật cho hệ thống, khi nhắc đến khía cạnh security thì ta nên biết một số thuật ngữ liên quan đến cách tấn công và hiểu cách tấn công vào hệ thống để có thể đưa ra các chính sách hợp lý.

6.7 PHÂN RÃ PHẦN MỀM

Trong bất kỳ nghiệp vụ lớn nào, khó có thể gộp tất cả các thực thể và các tiến trình nghiệp vụ vào trong một hệ thống phần mềm đơn lẻ vì điều đó sẽ quá phức tạp và khó sử dụng. Chúng ta có thể hoặc nên chia nhỏ phần mềm thành các hệ thống, rồi chia thành các hệ thống con tự trị nếu cần thiết và chia thành các tầng và có thể coi như các hệ thống con.

6.7.1 Hệ thống và hệ thống con

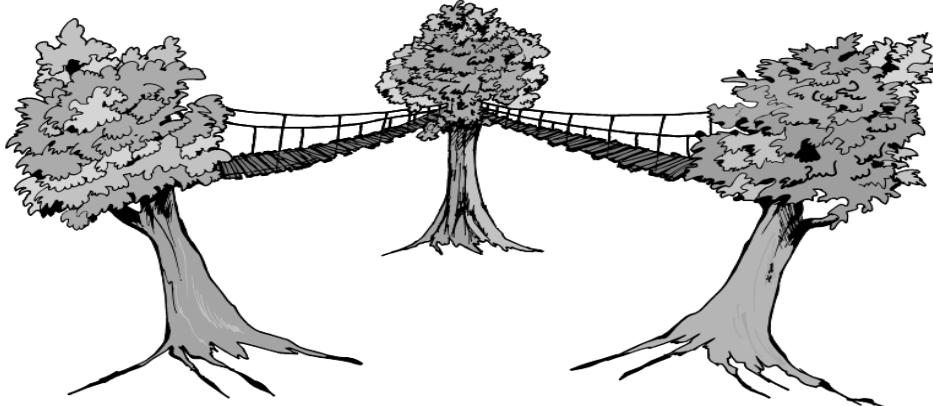
Chúng ta hãy xem khái niệm đơn giản “Khách hàng”. Khái niệm này được nhìn theo những cách khác nhau bởi các bộ phận trong một tổ chức lớn. Nếu chúng ta có gộp chúng lại với nhau thành một hệ phần mềm đơn lẻ hỗ trợ tất cả những bộ phận đó thì “Khách hàng” sẽ có hàng trăm thuộc tính và hàng trăm các thao tác. Thật là điều kinh khủng.

Thay vào đó, nghiệp vụ như vậy nên có một số hệ thống tách biệt, mỗi hệ thống được cài đặt bởi các nhóm phát triển khác nhau để việc sử dụng lại các đối tượng không phù hợp được giảm bớt. Khi đó, thông tin nào cần được chuyền giữa các hệ thống phải được truyền theo một cách xác định, được điều khiển qua những giao diện xác định trước. Để giảm độ phức tạp hơn nữa, mỗi hệ thống nên được chia nhỏ hơn nữa thành các hệ thống con riêng biệt.

Hình 6.8 biểu diễn những hệ thống của một công ty như những cái cây độc lập trong một khu rừng. Bên dưới mỗi cây là cơ sở dữ liệu của những thông tin mà những hệ thống cần truy nhập và tại đỉnh là giao diện của người dùng. Truyền thông xảy ra dọc theo con đường chập hẹp là cầu treo từ logic nghiệp vụ này đến logic nghiệp vụ khác theo giao diện định sẵn. Mặc dù mỗi hệ thống đều có dữ liệu độc lập riêng, chúng ta vẫn có

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

thể sử dụng hệ quản trị cơ sở dữ liệu (DBMS) để triển khai bởi vì BDMS sẽ quản lý nhiều cơ sở dữ liệu.



Hình 6.8: Kết hợp đa hệ thống

6.7.2 Các tầng

Bên trong một hệ thống phần mềm, ta thường sử dụng code nhiều tầng. Mỗi tầng là một tập hợp các đối tượng cộng tác với nhau phụ thuộc vào những khả năng được cung cấp bởi các lớp dưới. Ví dụ, thư viện hệ thống Unix cung cấp truy cập tới các tiện ích của hệ điều hành thấp hơn qua các lớp các chức năng của C.

Những lớp này giúp giảm sự phức tạp bằng cách chia nhỏ phần cài đặt thành nhiều đoạn có khả năng dễ quản lý hơn. Những tầng này cũng tăng cơ hội tái sử dụng, bởi mỗi tầng được viết một cách độc lập với các tầng ở trên. Bertrand Meyer từng nói [1]:

“Một hệ thống phần mềm thực thụ, thậm chí một phần mềm nhỏ theo như chuẩn ngày nay, đều liên quan đến rất nhiều lĩnh vực mà không thể đảm bảo độ chính xác của nó nếu xử lý tất cả các thành phần và các tính chất trên cùng một mức. Thay vào đó, cách tiếp cận theo tầng là cần thiết trong đó mỗi tầng dựa vào các tầng thấp hơn.”

Thông thường thì không chú ý tới tổng số tầng, nhưng nói chung tầng trên cùng thường biểu diễn giao diện người dùng, tầng dưới cùng thường biểu diễn hệ điều hành, hoặc là tầng vật lý kết nối mạng. Các tầng có thể là “mở” (tầng trên gọi một số đối tượng của tầng dưới nhưng không hoàn toàn giấu chúng) hoặc “đóng” (đóng gói hoàn toàn các tầng dưới nghĩa là các đối tượng tầng dưới ẩn với tầng trên). Không có luật nào qui định tầng nào nên để mở hay đóng. Nói chung, tầng đóng yêu cầu nhiều code hơn và chạy chậm hơn so với tầng mở vì có nhiều thông tin copy và chuyển đổi cần được thực thi). Ngược lại, tầng mở thì thường không an toàn vì tầng dưới không được bảo vệ và khó để bảo trì vì các tầng đều phụ thuộc nhiều hơn một tầng trên nó.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

Chúng ta có thể trao đổi giữa các tầng mà không ảnh hưởng đến các đoạn code khác. Chúng ta có thể vứt bỏ tầng cao nhất và thay thế bằng cái khác. Chúng ta có thể thay đổi tầng đóng trung gian bằng tầng khác có cùng interface mà không ảnh hưởng đến các tầng ở trên. Thường thì, các tầng được thực thi lại tại các lớp dưới khi ta thay đổi hệ thống, hoặc tại đỉnh khi ta thay đổi giao diện người dùng tới thiết bị mới.

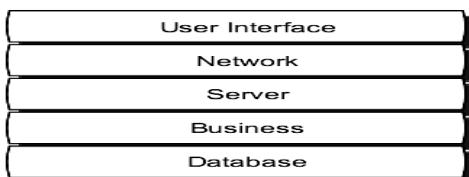
Các lớp cho hệ thống đơn tầng (Single-Tier System):



Hình 6.10. Các lớp trong hệ thống đơn tầng

Hình 6.10 biểu diễn một lược đồ phân lớp đơn giản có thể được thực thi trong hệ thống đơn tầng. Tại tầng đáy, chúng ta có tầng cơ sở dữ liệu, công việc của nó là chuyển dữ liệu đi và về giữa hệ quản trị cơ sở dữ liệu và tầng Business. Giả thiết rằng phần lớn các ứng dụng đều có yêu cầu lưu trữ dữ liệu, dữ liệu không bị mất đi nếu hệ thống bị tắt do bất kỳ nguyên nhân gì. Nếu chúng ta có một hệ thống đơn giản lưu trữ dữ liệu dưới dạng các file thì lớp cơ sở dữ liệu sẽ là một hệ thống file thay vì hệ thống quản trị cơ sở dữ liệu. Phía trên của lớp cơ sở dữ liệu là lớp nghiệp vụ (business) là lớp bao gồm những đối tượng thực thể và những đối tượng hỗ trợ thực thi. Cuối cùng, phía trên của lớp nghiệp vụ là lớp giao diện người dùng chứa các đối tượng mà công việc của nó là mô tả những lựa chọn có giá trị cho người, đưa các lệnh và dữ liệu đến tầng nghiệp vụ và hiển thị dữ liệu được trả về từ lớp nghiệp vụ.

Các tầng cho hệ thống hai và ba lớp:



Hình 6.11. Các tầng trong hệ thống hai hoặc ba lớp

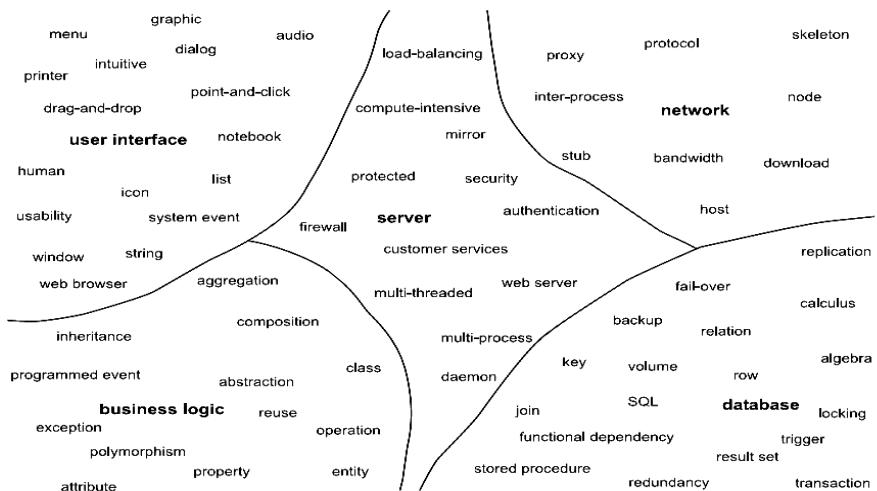
Các hệ thống 2 tầng và 3 tầng thực thi trên mạng lấy thông tin từ giao diện người dùng chuyển tới lớp nghiệp vụ chạy trên máy chủ. Chúng ta giải quyết vấn đề này bằng cách triển khai nhiều hơn hai lớp. Lớp Network chứa các đối tượng giúp cho việc chuyển tới giao diện người dùng (user interface). Lớp giao diện người dùng có thể truy nhập trực tiếp đến các đối tượng máy chủ. Lớp Server chứa các đối tượng sử dụng tập các dịch vụ nghiệp vụ của lớp Business. Trong hệ thống hai tầng thì lớp cơ sở dữ liệu ở trên cùng

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

máy với lớp server và lớp business. Trong hệ thống ba tầng, lớp cơ sở dữ liệu trải rộng ra trên mạng nhưng chi tiết được ẩn bởi DBMS.

Nếu chúng ta dùng dạng HTML để lấy từ client tới tầng trung gian (hoặc server), thì giao diện người dùng hay vị trí mạng ít rõ ràng hơn. Trong trường hợp này thì giao diện người dùng là một phần trên client (HTML page và HTML form) và một phần trên server (như servlet và JSPs).

Lớp translation



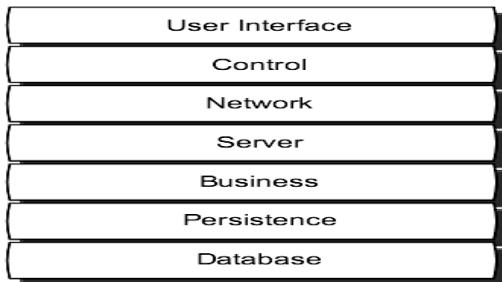
Hình 6.12. Các mối liên quan khác nhau trong một hệ thống lớn

Như mô tả trong hình 6.12, các lớp khác nhau sẽ có các trọng tâm khác nhau. Ví dụ, khi thiết kế giao diện người dùng, chúng ta thường đề cập đến menu, dialog, notebook... Về network ta thường lo lắng về giao thức (protocol), băng thông (bandwidth)... Khi chúng ta đến server thì chúng ta liên quan tới bảo mật (security), đa luồng (multi-thread). Trong lớp Business, phần này chúng ta quan tâm nhất đến trừu tượng, các thuộc tính, các thực thi, đa hình, tái sử dụng và các nguyên tắc khác của mô hình hướng đối tượng. Cuối cùng, ở lớp cơ sở dữ liệu, thường thì ta giải quyết với key, bảng, SQL, các phụ thuộc hàm và tất cả các khía cạnh khác của cơ sở dữ liệu. Nếu chúng ta có kết nối những phần này trực tiếp với nhau, ta sẽ dẫn tới việc quá phức tạp và quá nhiều kết nối (kết nối vũng chắc, khi thực thi của 1 đối tượng liên quan đến 1 đối tượng khác sẽ làm code trở nên khó bảo trì).

Chúng ta có thể giảm sự phức tạp và kết nối bằng cách thêm các lớp hoạt động giống như một người phiên dịch. “Translation layer” hữu dụng cho việc chuyển đổi lớp nghiệp vụ (trong trường hợp đơn tầng) hoặc lớp mạng (trong trường hợp đa tầng) thành các chức năng nhỏ được yêu cầu bởi người dùng đầu cuối như các lớp, thường liên quan giống như “controller”. Controller quản lý kết nối giao diện người dùng với các phần còn

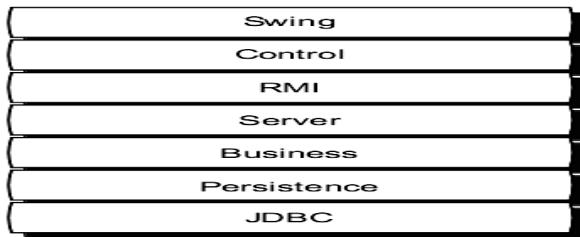
CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

lại của hệ thống. (điều này phù hợp với ý tưởng của Jacobson về controller). Lớp translation thông thường khác còn được gọi là “persistence layer” nó nằm giữa lớp nghiệp vụ và lớp cơ sở dữ liệu: xóa đi các phụ thuộc vào các cơ chế lưu trữ đang dùng của lớp nghiệp vụ, điều đó làm cho thay đổi các cơ chế lưu trữ sau này trở nên dễ dàng hơn (ví dụ như từ file sang DBMS). Hình 6.13 mô tả hệ thống đa tầng với các lớp control và persistence thêm vào



Hình 6.13. Các lớp chuyển đổi trong một hệ thống đa tầng

Java Layers: Applet plus RMI



Hình 6.14. Các lớp Applet RMI

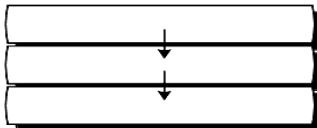
Để mô tả một java client với GUI phù hợp, hình 6.14 biểu diễn tập đầy đủ các lớp của hệ thống 3 tầng với RMI applet. RMI là giao thực mạng của java. Trong mô tả này, lớp giao diện người dùng sử dụng thư viện Swing (thư viện portable của các thành phần GUI của Java). Phía dưới lớp giao diện người dùng là lớp điều khiển chứa tất cả các code cho việc truy cập các dịch vụ nghiệp vụ, đoạn code này khác với việc phải được ẩn trong những đối tượng giao diện người dùng và được thực thi lại cho tất cả các giao diện mới mà ta thêm vào sau (ví dụ như mobile phone). 95% lớp mạng được cung cấp qua RMI Framework, ta phải tuân theo một vài luật đơn giản trong lớp điều khiển và lớp server để tạo các đối tượng server có khả năng truy cập từ bất kỳ client nào.

Lớp Server, lớp business và lớp persistence giống những lớp được mô tả trong những phần trước. Tiếp theo lớp cơ sở dữ liệu được cung cấp bởi thư viện Java Database Connectivity (JDBC). JDBC cho phép chúng ta truy cập vào bất kỳ cơ sở dữ liệu quan hệ nào sử dụng dynamic SQL hoặc precompiled SQL. Vì chú ý đến cả lớp persistence, nên

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

ta có thể thay thế JDBC với một cơ sở dữ liệu hướng đối tượng hoặc hệ thống file mà không ảnh hưởng đến các đối tượng nghiệp vụ.

Luồng truyền tin trong các tầng



Hình 6.15. Luồng thông tin chuyển từ trên xuống dưới trong hệ thống phân lớp

Trong một hệ thống lớn, mỗi lớp là một client của lớp bên dưới. Do vậy ta muốn xem xét những bản tin được gửi từ tầng trên xuống tầng dưới như hình 6.15. Mỗi bản tin (message) là một câu hỏi (ví dụ như `getAddress`) hoặc một câu lệnh (hướng dẫn làm gì đó ví dụ `setAddress`).

Nhiều câu lệnh gửi đi trong một tầng sẽ có ảnh hưởng đến thông tin được quản lý bởi chính tầng đó – mặt khác sẽ có tahy đổi trong việc đưa ra các câu lệnh. Nhưng điều gì sẽ xảy ra nếu lớp trên cần biết thông tin nào vừa thay đổi? Ví dụ, lớp trên phải là giao diện người dùng cần để cập nhật hiển thị của nó với những thông tin mới. Ta có hai lựa chọn:

- Thêm tri thức cho lớp trên về những câu lệnh nào thay đổi thông tin gì
- Lớp dưới gửi các bản tin tới lớp trên bất kỳ khi nào có thông tin thay đổi.

Vấn đề với lựa chọn đầu tiên, về mặt logic nó sẽ làm cho lớp trên bị tràn ngập các tri thức của lớp dưới và điều này khiến cho việc code lớp này trên sẽ phức tạp hơn và lớp trên sẽ kết nối chặt chẽ với lớp dưới. Vấn đề với lựa chọn thứ 2 là lớp dưới phải biết về lớp trên nên nó biết những đối tượng nào để gửi các bản tin, do đó, lớp dưới sẽ bị tràn ngập các tri thức về lớp trên, điều đó làm nó phức tạp hơn (về lý thuyết lớp dưới không nên bị ràng buộc bởi lớp trên).

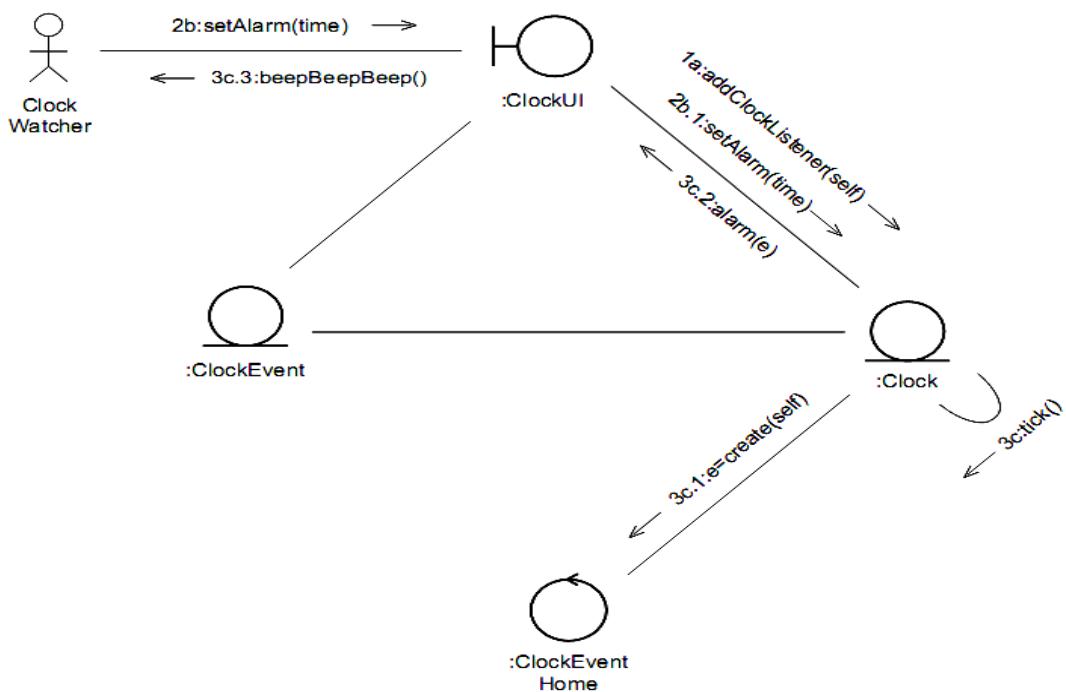
Sự kiện (Event)

Vậy, có một cách nào đó để một lớp có thể thông báo cho lớp trên khi có việc gì đó xảy ra mà không làm tăng độ phức tạp hay ràng buộc theo các hướng không? Câu trả lời là có, đó là “event”. Một *event source* phát ra khi nào có gì đó xảy ra (*event*) và nói chi tiết về sự kiện đấy tới bất kỳ ai muốn nghe (*event listener*). Một event phải là 1 *attribute event*, chỉ ra 1 thay đổi trong giá trị của những thuộc tính của một trong các *event source*, hoặc nó có thể là 1 *pure event* không liên quan đến bất kỳ thuộc tính nào. Ví dụ, xem xét 1 thực thể Clock: nó có thể quảng bá 1 *attribute event* khi thời gian thay đổi từng giây và nó có thể quảng bá một *pure event* khi cảnh báo kết thúc.

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

Khi “quảng bá một event” đưa ra, chúng ta sẽ chờ *event source* phát hiện ra event xảy ra và thông báo cho tất cả những ai đang lắng nghe: theo cách này thì *event source* sẽ không bị làm phức tạp hóa bởi những người lắng nghe (*listener*) và các *event listener* sẽ không bị làm phức tạp hóa bởi khi nào các event có thể xảy ra. Trong lược đồ lop, ta có thể sử dụng *event source* tại mỗi lớp để quảng bá các event tới *listener* tại các lớp trên, do đó đạt được việc đặt các tri thức tại đúng chỗ và giảm thiểu các ràng buộc.

Nhưng làm thế nào để chúng ta thực thi ý tưởng sử dụng message? Giống như bước đầu tiên để hiểu được quá trình hoạt động của nó, hình 6.7 là một ví dụ kết hợp giữa Clock – *event source* và một ClockUI – *event listener*. Trong biểu đồ, tên “self” là chú giải UML cho “đối tượng hiện thời” (nó giống như *this* trong Java).



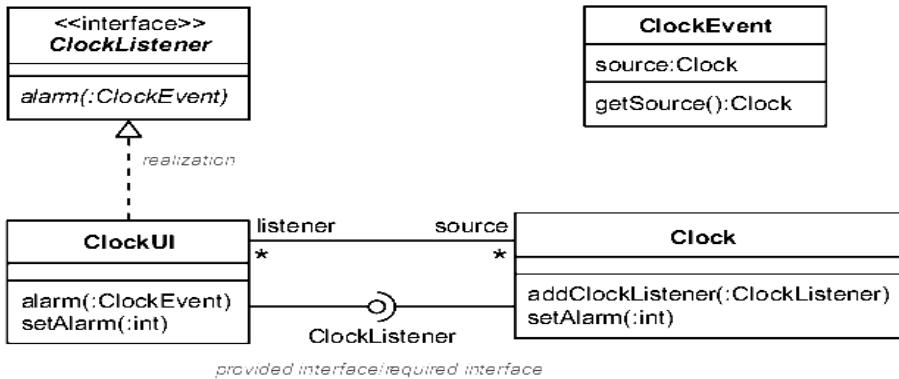
Hình 6.17. Biểu đồ cộng tác thể hiện cách các sự kiện hoạt động

Đóng vai trò như một phần của tiến trình khởi tạo, ClockUI gửi cho Clock bản tin *addClockListener*. Tiếp đó, ClockWatcher sẽ đặt cảnh báo vào ClockUI, nó sẽ gửi các cài đặt cảnh báo cho Clock. Clock tự gửi các bản tin dấu định kỳ, cuối cùng, nó sẽ phát hiện ra thời gian cảnh báo kết thúc. Khi việc này xảy ra, Clock tạo ra 1 *ClockEvent*, với thông tin về event (trong trường hợp này, thông tin duy nhất là nguồn của event). Sau đó, Clock gửi bản tin cảnh báo tới ClockUI với *ClockEvent* như một tham số. (các đối tượng event ghi lại *event source*, nếu ClockUI nghe thấy hơn một clock thì nó sẽ phát hiện ra cái nào vừa phát ra cảnh báo). Cuối cùng ClockUI cảnh báo tại ClockWatcher.

Bạn có thể băn khoăn tại sao sau các số trong các biểu đồ kết nối để các chữ cái sau nó. Nguyên nhân là ta có 3 chuỗi độc lập trong scenario này: thêm clock listener, cài

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

đặt thời gian và cảnh báo.(miễn là “thêm clock listener” xảy ra đầu tiên, 2 việc sau có thể xảy ra nhiều lần tùy thứ tự). Clock phải chạy 1 cách độc lập với ClockUI và Clock thì không được đánh dấu. Tên thì giống như là phần của số tuần tự UML chỉ ra bản tin trong câu hỏi phụ thuộc vào những bản tin cùng tên, nhưng độc lập với những cái khác. Chúng ta vẫn có thể biểu diễn sự sắp xếp các bản tin độc lập, bằng cách sử dụng cẩn thận các con số: ví dụ, nó ẩn đi là bản tin 2b xảy ra trước bản tin 3c trong scenario, nó cũng sẽ ẩn đi bản tin số 99x và 99x xảy ra cùng thời điểm. Biểu đồ lớp cho alarm clock được biểu diễn trong hình 6.18:



Hình 6.18. Các sự kiện như một biểu đồ lớp

Từ biểu đồ ta có thể nhìn thấy **ClockEvent** có 1 “getter” cho `source` attribute. **ClockUI** cũng có 1 bản tin cho phép **ClockWatcher** cài đặt cảnh báo –`setAlarm` – và các bản tin khác dành cho việc phát hiện các alarm event – `alarm`. Cuối cùng, lớp **Clock** có bản tin dành cho việc cài đặt cảnh báo – `setAlarm` và 1 bản tin cho việc đăng ký 1 listenner – `addClockListener`.

Vì ngôn ngữ hướng đối tượng thường không có cơ chế quảng bá thực sự nên **Clock** trong hình 6.18 phải duy trì một danh sách các listenner và gửi một bản tin tới từng listenner khi có event xảy ra. Các listenner, phải chắc chắn rằng chúng lên danh sách cho các event. Nó có thể xảy ra, **Clock** bây giờ đã được kết nối tới **ClockUI**, đó là điều ta đã cố gắng tránh ở bước đầu tiên. Chúng ta có thể giải quyết vấn đề này bằng cách đưa ra một lớp trừu tượng – **ClockListener**, lớp này chỉ liệt kê danh sách những bản tin được yêu cầu để phát hiện ra **Clock event**. Miễn là **ClockUI** kế thừa từ **ClockListener**, ta có thể đăng ký **ClockUI** với **Clock** và **ClockUI** sẽ có thể nhận được bản tin cảnh báo. Do đó mặc dù **Clock** được kết nối tới **ClockListener**, nó không thể kết nối tới **ClockUI** (**ClockListener** nằm ở cùng lớp với **Clock**, trong khi **ClockUI** truyền 1 cách độc lập trong lớp trên).

Một *interface* biểu diễn bởi từ khóa `<<interface>>` là một lớp trừu tượng - lớp không có những phương thức cụ thể và không có thuộc tính. Bởi vì những *interface* rất hữu dụng, cho những kết định rõ với việc giảm thiểu các ràng buộc, Có vài ký hiệu UML đặc biệt được dùng trong hình 6.18, mũi tên với đầu trăng chỉ sự kế thừa, cho

CHƯƠNG 6. THIẾT KẾ KIẾN TRÚC HỆ THỐNG

trường hợp đặc biệt, superclass là 1 interface. Ký hiệu ClockListener, dán nhãn interface được cung cấp/interface được yêu cầu, cho phép chúng ta chỉ ra rằng một lớp dùng một lớp khác qua một interface.

Luồng thông điệp sử dụng sự kiện



Hình 6.19. Luồng điều khiển sử dụng các sự kiện

Hình 6.19 chỉ ra các bản tin truyền đi như thế nào khi sử dụng các lớp. Thông thường các bản tin được biểu diễn chuyển xuống qua các lớp, trong khi các bản tin event thì đi lên (bản tin event được biểu diễn là các mũi tên đứt để chỉ ra các đối tượng lớp dưới không có tri thức của người nhận). Event thường được dùng trong đoạn mã client, bởi vì chúng là cách thuận lợi cho việc giữ các giao diện người dùng cập nhật thông tin từ lớp dưới. Các event hiếm khi được dùng bên phía server, bởi vì đoạn mã server phục vụ đa luồng, nó lập trình điều khiển event khá phức tạp (ví dụ, có khả năng làm tăng tắc nghẽn) và bởi vì các event không nên quảng bá qua mạng, chúng ta không muốn server tắc nghẽn khi nó gửi bản tin event tới rất nhiều client, một vài trong số đó có thể bị hỏng hoặc khó liên lạc. Do đó, cần tránh lập trình điều khiển event trên server.

6.8 KẾT LUẬN

Trong chương này chúng ta đã xem xét:

- Các bước liên quan đến thiết kế hệ thống và cách phân rã hệ thống thành các thành phần logic và vật lý và đặc biệt xem xét hình trạng mạng
- Cách biểu diễn kiến trúc bằng biểu đồ triển khai trong UML
- Những vấn đề đồng thời này sinh trong mạng: làm thế nào đảm bảo rằng thông tin được cập nhật hoàn toàn trước khi người sử dụng tác động đến nó và làm thế nào đảm bảo thông tin không được cập nhật khi đang đọc.
- Làm thế nào phần mềm có thể phân rã thành nhiều hệ thống, hệ thống con và các tầng

BÀI TẬP

Hãy trình bày kiến trúc phân tầng cho bài tập nhóm về quản lý học theo tín chỉ và chỉ ra cách chọn lựa của mình

CHƯƠNG 7 LỰA CHỌN CÔNG NGHỆ

7.1 GIỚI THIỆU

Mặc dù chúng ta có bộ tài liệu yêu cầu và phân tích đầy đủ và ngay cả biểu đồ kiến trúc ban đầu nhưng vẫn chưa đưa ra lựa chọn nào về công nghệ cài đặt mà chúng ta sẽ sử dụng. Việc lựa chọn công nghệ có vai trò vô cùng quan trọng trong một quy trình sản xuất phần mềm. Thời gian trì hoãn việc chọn công nghệ càng lâu, thì càng ít cơ hội để khai thác những điểm tốt của những công nghệ đó. Quyết định chọn lựa công nghệ trong giai đoạn này của tiến trình phát triển (trước thiết kế chi tiết) là một sự kết hợp tốt. Khi chúng ta đã quyết định chọn công nghệ rồi, chúng ta sẽ hiểu sâu sắc các chọn lựa để trở thành trung thành định hướng với mỗi chức năng duy nhất của một công nghệ cụ thể.

7.2 CÔNG NGHỆ TẦNG CLIENT

Chúng ta hãy nhìn phần mềm chạy trên các client trong hệ đa tầng. Chúng ta có 2 sự chọn lựa chính: chúng ta có thể thực thi một chương trình ứng dụng hoặc một trình duyệt web. Các loại ứng dụng thực thi có thể bao gồm:

- Truyền thông giữa người với người: email, chat...
- Truyền file hoặc trao đổi file.
- Đăng nhập từ xa.
- Những ứng dụng đặc quyền

Client kết nối máy chủ bằng một trình duyệt web có thể sử dụng những công nghệ sau:

- Dạng HTML.
- JavaScript.
- Plug-ins
- Điều khiển ActiveX.
- Java Applets.

Tất cả các công nghệ trên sử dụng một vài loại giao thức, ví dụ IMAP cho email, AIM cho thư tín trực tiếp và HTTP/CGI cho dạng thức HTML, để truyền thông với ít nhất 1 máy khác (mail server, messaging server hoặc Web Server).

Client ứng dụng có nhược điểm là chúng yêu cầu cài đặt phần mềm trên máy client trước khi sử dụng. Tuy nhiên, với một vài mục đích (desktop publishing), chúng là một lựa chọn tốt. Một trình duyệt web, một ứng dụng trên chính nó, đặc biệt hữu ích cho

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

phần mềm client bởi vì nó có thể nâng cao bằng cách phát triển đa tầng để thực thi theo cách ko ngờ tới bởi người phát triển trên chính trình duyệt của nó. Ví dụ, một trình duyệt web có thể sử dụng để kiểm tra chi tiết tài khoản ngân hàng sử dụng một Java Applet. Và tất cả được thực hiện mà ko cần cài đặt trước trên máy khách. Mỗi công nghệ trình duyệt đều có ưu điểm và nhược điểm. Ví dụ:

- HTML thường hỗ trợ nhiều và rộng, nhưng dạng thức HTML còn nguyên thủy và chưa tự động xác nhận trên client.
- JavaScript cho phép lập trình trên client (ví dụ, xác nhận dữ liệu trên dạng thức HTML), nhưng JavaScript được phiên dịch (bởi vậy sẽ chậm hơn khi biên dịch code), sẽ ko rõ ràng (trong hướng đối tượng) và với các trình duyệt khác nhau sẽ cung cấp các mức độ khác nhau.
- Java là một ngôn ngữ hướng đối tượng đơn giản, rõ ràng, và là giải pháp tiềm năng tốt nhất. Java cũng cung cấp một phương thức an toàn bởi vì nó ngăn cấm truy cập tài nguyên trên máy cụ bộ mà ko có sự xác nhận rõ ràng và chi tiết. Đến chừng mực nào đó, tại thời điểm ghi vào hầu hết các trình duyệt web chỉ hỗ trợ phiên bản cũ của Java – để có một phiên bản chức năng đầy đủ, bạn cần cài đặt thêm java plug in từ Sun hoặc mua một PC với Java đã được cài đặt trước.
- Theo nguyên lý, Java là sự lựa chọn tốt nhất để thực thi client trên một hệ thống 3 tầng. Bởi vì sự thiếu hỗ trợ trình duyệt web, hầu hết các developer chọn lựa dạng thức HTML. Rất nhiều trình duyệt web có host lớn hơn client ứng dụng, bao gồm truyền file, email và tin tức USENET.
- Hầu hết các công nghệ quan trọng đều được di chuyển vào các thiết bị mới hơn, như PDA hoặc mobile phone. TCP/IP được sử dụng rộng rãi, thậm chí trên cả những thiết bị client nhỏ- nó cũng làm việc qua kết nối ko dây. Do đó, di chuyển một hệ thống 3 tầng từ một loại thiết bị mới hơn là bình thường với việc thiết kế lại giao diện người dùng để cho chúng nhỏ hơn và riêng tư hơn.

7.3 GIAO THỨC GIỮA TẦNG CLIENT VÀ TẦNG GIỮA

Phần mềm client, khi chạy như một ứng dụng hoặc trong một trình duyệt web, phải truyền thông với một server sử dụng một vài giao thức. Hầu hết các giao thức đều được phân lớp: tại đây, chúng ta có giao thức mức thấp như là TCP/IP và trên đỉnh chúng ta xây dựng một giao thức khác, đặc trưng với những nhiệm vụ cụ thể. Ví dụ, trên đỉnh của TCP/IP, ta có thể dùng giao thức SSL để mã hóa và giải mã thông tin với mục đích riêng tư và toàn vẹn thông tin. Trên đỉnh của SSL, chúng ta có thể chạy secure HTTP, một giao thức bảo mật cho phép 1 client tới yêu cầu một tài liệu bằng URI và trả lại nội dung của tài liệu.

Các giao thức thường sử dụng được chia làm 2 loại: chuyên dụng và chung. Giao thức chuyên dụng gồm:

- IMAP (email).
- AIM (AOL instant messaging).
- NNTP (USENET news).
- HTTP/CGI (HTML form).
- FTP(truyền file)
- Telnet (đăng nhập từ xa).

Giao thức chung (có khả năng thực hiện nhiều nhiệm vụ) bao gồm:

- TCP/IP (mức thấp của tầng giao vận, cũng được hiểu như sockets).
- JRMP (để truyền thông Java- tới -Java).
- IIOP (dùng cho truyền thông với CORBA, tương tự như RMI nhưng là một ngôn ngữ đa thực thi).

Developer thường sử dụng một sự trùu tượng hóa mức cao, sự giúp đỡ của thời gian thực thi hệ thống và thư viện. Ví dụ, lập trình viên RMI và CORBA chỉ gửi thông điệp tới đối tượng – code RMI và CORBA thực thi tất cả các hành động đóng gói và mở gói vô hình, lập trình viên của dạng thức HTML đơn giản chỉ thiết kế layout trên form của họ - kỹ thuật CGI sắp xếp dạng thức dữ liệu đã thực thi trên server.

7.4 CÔNG NGHỆ TẦNG GIỮA

Ứng dụng của Server là một phần của đoạn code đa luồng đặc trưng, thiết kế cho một thông lượng lớn (có khả năng xử lý hàng nghìn, hoặc hàng triệu client đồng thời). Một ứng dụng Server lắng nghe trên một vài cổng (điểm kết nối) chờ client kết nối.

Với một số loại phần mềm, vị trí của server tương tự như client. Trên client, chúng ta thực thi ứng dụng riêng lẻ hoặc code với máy chủ dựa trên trình duyệt web. Trên tầng server (tầng giữa), chúng ta có thể chạy ứng dụng riêng lẻ hoặc chạy một trình duyệt web và đặt code vào trong đó. Các ứng dụng riêng lẻ gồm:

- Thư, tin nhắn, tin tức và chat server.
- FTP server.
- Telnet server.
- RMI Registry (a look-up RMI objects).
- CORBA naming service (Một kỹ thuật tìm kiếm với đối tượng của CORBA).
- Java Naming and Directory Interface (JNDI) server (Một cách thức để anh xạ các tên chung để có thể sử dụng thay cho một RMI registry, một Thiết bị đặt tên CORBA, hay một đăng ký người dùng).

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

- Proprietary server (ví dụ, một tiến trình hosting đối tượng CORBA hoặc RMI, một EJB client, một .Net client).
Server code có thể được dùng máy chủ là một web server bao gồm:
 - Java Server Pages (JSPs), để xây dựng các trang Web.
 - Active Server Pages (ASPs), tương tự như JSPs nhưng việc coding được thực hiện trên Visual Basic chứ ko phải là Java.
 - CGI scripts (chúng là những file đã phiên dịch, được viết trên các ngôn ngữ như PERL, hoặc các chương trình có thể thực thi được).
 - Servlets (các đối tượng của Java server có thể được truy cập bởi Java applets, JSPs hoặc dạng thức HTML).

RMI registries and CORBA naming services cho phép RMI và CORBA client tìm đối tượng server bằng tên. CGI Script là file nguyên bản được viết trong một vài ngôn ngữ lệnh như PERL hoặc có thể thực hiện được, biên dịch một ngôn ngữ lập trình theo cách thông thường. JSP là những file text bao gồm HTML nguyên sơ đặt rải rác với java code. Khi sự yêu cầu đầu tiên của JSP được biên dịch tới servlet: HTML nguyên sơ sẽ được thay thế với các câu lệnh in và java code theo đúng nguyên văn. Servlet được biên dịch và yêu cầu. JSP thường được sử dụng để cá nhân hóa các trang web- bằng cách chèn một dãy câu lệnh cho khách hàng hiện tại đang loggin, ví dụ, họ có thể được yêu cầu chính xác hoặc qua servlet. ASP tương tự JSP nhưng sử dụng cho công nghệ của Microsoft, vì vậy chúng không linh động.

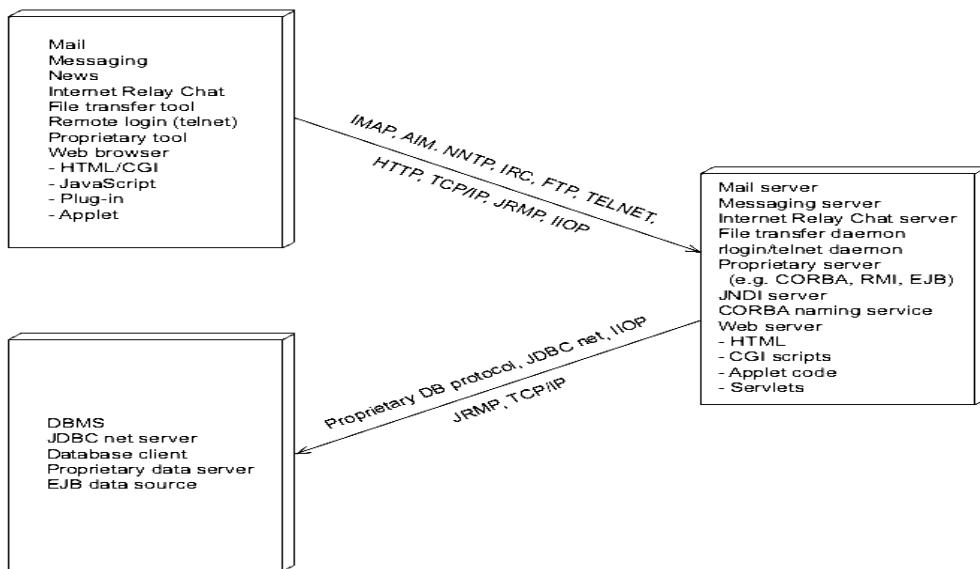
7.5 CÔNG NGHỆ TẦNG GIỮA ĐẾN TẦNG DỮ LIỆU

Có một số cách truy cập tầng dữ liệu:

- Bao gồm database- client code trên tầng giữa để ta có thể truy cập một DBMS chạy trên tầng dữ liệu. Với java, chúng ta có thể thực hiện điều này với sự giúp đỡ của kỹ thuật java database connectivaty (JDBC).
- Truyền thông với tầng dữ liệu sử dụng bất kỳ công nghệ tầng client tới tầng giữa nào đã được thảo luận. Sau đó, tầng dữ liệu sẽ được quan tâm, tầng giữa chỉ là một tầng client khác.
- Làm một cái gì đó đặc quyền, như truy cập một server đang chạy trên một máy tầng dữ liệu hoặc chạy một vài đoạn code trực tiếp trên tầng giữa (với một cấu hình 2 tầng).
- Truy cập tầng dữ liệu sử dụng một vài giao thức ko phải TCP/IP (chúng ta sẽ chỉ thường xuyên thực hiện để truy cập một hệ thống kế thừa).
- Chứa EJB client code trên server tầng giữa, sau đó truy cập tầng dữ liệu qua EJB.

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

- Chứa .Net client code trong server tầng giữa. mạng .Net là đối thủ của Microsoft với EJB framework.



Hình 7.1: Tổng quan về các công nghệ trên 3 tầng

7.6 CÁC CÔNG NGHỆ KHÁC

7.6.1 Tổng quan về J2EE

Mở đầu

J2EE là một platform để phát triển những ứng dụng phân tán. J2EE bao gồm những phần sau (được xem như là J2EE Framework):

- J2EE Platform - một platform chuẩn để hosting các ứng dụng J2EE.
- Reference Implementation - một application server hỗ trợ chuẩn mới nhất của J2EE, ngoại trừ những tiêu điểm của nó là những đặc tính mới trong phiên bản chuẩn của J2EE, đây chưa phải là một sản phẩm hoàn chỉnh.
- Compatibility Test Suite - một công cụ để kiểm tra xem một application server có tương thích với chuẩn J2EE hay không. Thiếu cái này thì mỗi nhà cung cấp có thể hiểu và từ đó phát triển chuẩn J2EE theo những hướng khác nhau mà như thế thì làm giảm đi thé mạnh của J2EE platform là “write once, run anywhere”.
- Application Programming Model (APM) Blueprint – một mô hình ứng dụng tham khảo, được cung cấp để minh họa cách phát triển ứng dụng dùng J2EE.

Phát triển chương trình dùng J2EE

J2EE Framework cho phép phát triển những ứng dụng phân tán bằng cách cung cấp một tập các dịch vụ cơ bản như quản lý giao dịch, kiểm tra an toàn, quản lý trạng thái, quản lý

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

tài nguyên. Các máy chủ ứng dụng đều cung cấp những dịch vụ cơ bản này của J2EE Framework.

Các thành phần của J2EE

J2EE được xây dựng trên một mô hình thành phần container. Bốn thành phần cốt lõi cung cấp môi trường cho các thành phần khác của J2EE thông qua các API. Những thành phần cốt lõi này liên quan đến bốn kiểu container được hỗ trợ trong J2EE bao gồm, Application Client, Applet, Web và EJB:

- Java Application: thành phần này là 1 chương trình chạy bên trong Application Client container. Application Client container cung cấp những API hỗ trợ cho messaging, gọi từ xa, kết nối dữ liệu và lookup service. Application Client container đòi hỏi những API sau: J2SE, JMS, JNDI, RIM-IIOP và JDBC. Container này được hỗ trợ bởi nhà cung cấp application server.
- Applet – Applet thành phần là java applet chạy bên trong Applet container, chính là web browser có hỗ trợ công nghệ Java. Applet phải hỗ trợ J2SE API.
- Servlet và JSP – đây là Web-based component chạy ở bên trong Web container, được hỗ trợ bởi Web Server. Web container là một môi trường run-time cho servlet và jsp. Web Container phải hỗ trợ những API sau: J2SE, JMS, JNDI, JTA, JavaMail, JAF, RIM-IIOP và JDBC. Serlet và JSP cung cấp một cơ chế cho việc chuẩn bị, xử lý, định dạng nội dung động.
- Enterprise JavaBean (EJB) – EJB là thành phần nghiệp vụ chạy bên trong EJB container. EJB là phần nhân, cốt lõi của ứng dụng J2EE. EJB container cung cấp các dịch vụ quản lý giao dịch, bảo mật, quản lý trạng thái, quay vòng tài nguyên. EJB container phải hỗ trợ những API sau: J2SE, JMS, JNDI, JTA, JavaMail, JAF, RIM-IIOP và JDBC.

7.6.2 Giao thức tầng Client và tầng giữa

Mô hình TCP/IP được sử dụng kết hợp với một số giao thức khác như FTP, Telnet, IMAP...TCP/IP là một tập các giao thức xác định các qui tắc cũng như định dạng cho việc truyền thông này. TCP/IP hiện là giao thức mạng được sử dụng rộng rãi nhất trên thế giới. Các lý do khiến giao thức mạng này trở nên ngày càng thông dụng là:

- Tính có thể định tuyến và mở rộng được.
- Tính mở.
- Là một chuẩn đã được kiểm nghiệm, mang tính ổn định.
- Đã trở thành bộ giao thức sử dụng cho mạng Internet.

Các giao thức TCP/IP và Mô hình OSI

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

Bộ giao thức TCP/IP xây dựng dựa trên mô hình mạng do DoD phát triển, được gọi là mô hình DoD.

Các lớp trong mô hình OSI và DoD

Thay vì một mô hình gồm bảy lớp, mô hình DoD chỉ có bốn lớp. Lớp Truy nhập mạng (Network Access) dùng để miêu tả khuôn thức vật lý của mạng cũng như các thông điệp sẽ được định dạng như thế nào trong quá trình truyền dữ liệu. Lớp Internet có nhiệm vụ chuyển phát các gói dữ liệu trong liên mạng và Lớp Giao vận (Transport) sẽ thực hiện các công việc kiểm tra lỗi để đảm bảo sự chuyển phát tin cậy. Tất cả các chức năng mạng khác được thực hiện trong lớp ứng dụng (Application). Các giao thức trong bộ giao thức TCP/IP sẽ tương xứng với một lớp cụ thể của mô hình DoD và cũng có thể ánh xạ sang mô hình OSI. Dưới đây chúng ta sẽ tóm tắt mỗi giao thức TCP/IP và chỉ rõ mối quan hệ của nó với các lớp trong cả hai mô hình DoD và OSI.

Chức năng của các giao thức TCP/IP độc lập với kiến trúc mạng ở mức thấp. Dưới đây là tóm lược về chức năng cụ thể của các giao thức trong bộ giao thức TCP/IP.

Các giao thức lớp ứng dụng

- FTP (File Transfer Protocol): Giao thức truyền tệp

FTP cung cấp một phương thức chung để truyền tệp trong một liên mạng. Nó có thể bao gồm các tính năng bảo mật tệp thông qua sử dụng một cặp tên/mật khẩu để xác thực. Nó có thể cho phép chuyển tệp giữa các hệ thống máy tính khác nhau.

- TFTP (Trivial File Transfer Protocol): Giao thức truyền tệp đơn giản

Tương tự như FTP, cho phép truyền tệp giữa một host và một máy chủ FTP (FTP server). Tuy nhiên, giao thức này không bao gồm việc xác thực người sử dụng và dùng UDP chứ không phải là TCP làm giao thức giao vận.

- HTTP (Hypertext Transport Protocol): Giao thức truyền tệp siêu văn bản

Các trình duyệt Web và máy chủ Web sử dụng giao thức này để trao đổi các tệp (ví dụ các trang Web) qua mạng toàn cầu WWW hay intranet. Chúng ta có thể coi HTTP là giao thức yêu cầu và trả lời thông tin. Nó thường sử dụng để yêu cầu trả gửi trả các tài liệu Web. Ngoài ra, HTTP cũng được sử dụng để làm giao thức truyền thông giữa các tác tử (agent) sử dụng các giao thức TCP/IP khác.

- SMTP (Simple Mail Transfer Protocol): Giao thức chuyển thư đơn giản

Đây là giao thức được sử dụng để định tuyến các thư điện tử trong một liên mạng. Các ứng dụng thư điện tử sẽ cung cấp giao diện để truyền thông với SMTP và máy chủ thư điện tử.

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

Các giao thức khác: Bộ giao thức TCP/IP còn có nhiều giao thức khác ở lớp ứng dụng để đáp ứng cho một số dịch vụ cụ thể khác. Trong số những giao thức này có thể kể đến:

- Telnet: Giao thức điều khiển từ xa.
- NFS (Network File System): Hệ thống tệp tin máy chủ.
- TCP (Transmission Control Protocol): Giao thức kiểm soát truyền thông tin. Giao thức này cung cấp các dịch vụ hướng kết nối (connection-oriented) và thực hiện các công việc như kiểm soát thứ tự của các gói tin, việc đánh địa chỉ các dịch vụ cũng như các chức năng kiểm tra lỗi.
- UDP (User Data Protocol): Giao thức gói dữ liệu người dùng. Giao thức này cũng hoạt động ở lớp giao vận, giống như giao thức TCP. Tuy nhiên, đây không phải là giao thức hướng kết nối và làm phát sinh ít phụ phí hơn TCP. Do ít phụ phí hơn, nó truyền dữ liệu nhanh hơn, nhưng cũng ít tin cậy hơn.
- DNS (Domain Name System): Hệ thống tên miền. Đây là hệ thống được phân tán trong liên mạng để cung cấp việc phân giải tên/địa chỉ. Ví dụ, tên miền internet.vdc.com.vn sẽ được phân giải thành một địa chỉ cụ thể là 203.162.1.181.
- IP (Internet Protocol): Giao thức Internet. Đây là giao thức chính trong bộ giao thức TCP/IP. Đây là một giao thức phi kết nối (connectionless) có chức năng ra các quyết định trong việc định tuyến trong một liên mạng dựa vào các thông tin nó nhận được từ ARP. Sử dụng địa chỉ IP để xác định một mạng cụ thể trong một liên mạng, nó cũng kiểm soát các vấn đề liên quan tới địa chỉ IP khi thực hiện định tuyến.
- ICMP (Internet Control Message Protocol): Giao thức kiểm soát thông điệp Internet.

Giao thức này kết hợp chặt chẽ với giao thức IP trong việc cung cấp các thông tin kiểm soát và báo lỗi trong quá trình di chuyển các gói dữ liệu trong một liên mạng.

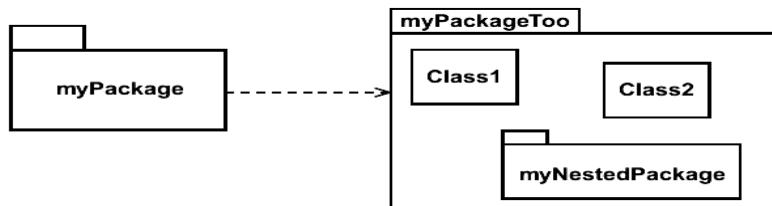
- IGMP (Internet Group Multicast Protocol): Giao thức nhóm Multicast Internet
Giao thức này xác định các nhóm Multicast. Tất cả các thành viên của một nhóm có thể nhận các thông điệp phát quảng bá dành cho nhóm đó (gọi là các thông điệp multicast). Một nhóm Multicast có thể gồm các thiết bị trong cùng một mạng hay thuộc các mạng khác nhau.
- OSPF (Open Shortest Path First): Giao thức tìm đường ngắn nhất trước. Là giao thức tìm lưa chọn tuyến đường trong phương pháp định tuyến trạng thái liên kết. Giao thức này hiệu quả hơn RIP trong việc cập nhật thông tin cho bảng định tuyến, đặc biệt trong các mạng có qui mô lớn.

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

- ARP (Adress Resolution Protocol): Giao thức phân giải địa chỉ. Giao thức này cung cấp địa chỉ Internet hoàn chỉnh bằng cách kết hợp một địa chỉ mạng (lôgic) với một địa chỉ vật lý cụ thể. Nó làm việc cùng với các giao thức khác để cung cấp việc phân giải địa chỉ/tên logic.
- RARP và BOOTP. Cả hai giao thức này đều được sử dụng để xác định địa chỉ IP của một thiết bị từ địa chỉ MAC đã biết. BOOTP là một bước cài tiền đối với RARP và hiện được sử dụng nhiều hơn RARP. Các máy tính khi khởi động sẽ sử dụng giao thức này để lấy được địa chỉ IP của chúng từ một máy chủ BOOTP trên mạng. Giao thức BOOTP sẽ giám sát các gói tin yêu cầu địa chỉ do các host gửi tới máy chủ BOOTP sẽ được trả lời.
- DHCP (Dynamic Host Configuration Protocol): Giao thức cấu hình chủ động DHCP đơn giản hoá việc quản lý địa chỉ IP trong một mạng. Các máy chủ DHCP duy trì một danh sách gồm các địa chỉ chưa được sử dụng và các địa chỉ đã được gán và truyền thông tin về địa chỉ tới các host yêu cầu. DHCP gồm hai thành phần sau đây:
 - Một giao thức để phân phối các tham số cấu hình IP từ máy chủ DHCP tới host.
 - Một giao thức xác định các địa chỉ IP sẽ được gán cho các host như thế nào.

7.7 CÁC GÓI UML

Khái niệm gói (package) trong UML cho phép chúng ta nhóm các lớp liên quan nhau (xem Chương 3). Một gói có thể chứa các lớp hay gói khác. Hình 7.5 chỉ ra sự phụ thuộc (mũi tên đứt) từ một gói đến một gói khác và nó ám chỉ rằng gói nguồn sử dụng một phần bên trong gói đích.



Một gói có thể sử dụng để biểu diễn:

- Một tầng
- Hệ thống con
- Thư viện dùng lại
- Khung (framework)
- Các lớp cản triển khai cùng nhau
- ...

CHƯƠNG 7. LỰA CHỌN CÔNG NGHỆ

Từ quan điểm lập trình, các gói có thể ánh xạ thành các cấu trúc như Java package hoặc C++ namespace. Hãy nhớ rằng, các gói là khái niệm trong thời gian biên dịch: chúng giúp ta tổ chức mã cho tiện lợi để phát triển, triển khai và bảo trì.

Nhiều người phát triển sử dụng biểu đồ gói để chỉ các tầng. Tuy nhiên, cách tiếp cận này có một số vấn đề. Một là các tầng được chọn trước khi quyết định cách tổ chức mã nguồn thành các gói và như vậy tổ chức sau đó có thể khác. Hai là biểu đồ gói không cho phép chúng ta chỉ ra thông tin quan trọng; ví dụ, sự tồn tại của HTTPCGILayer trong iCoot là quan trọng nhưng không ánh xạ đến gói nào mà chúng ta có thể cài đặt hay mượn từ thư viện.

Biểu đồ gói cũng không tốt khi thể hiện phân rã thành hệ thành các hệ thống con và khi đó biểu đồ triển khai nên được sử dụng.

7.8 KẾT LUẬN

Chương này đã trình bày một số vấn đề liên quan đến công nghệ và lựa chọn công nghệ:

- Các công nghệ chủ yếu cho client và server
- Các giao thức middleware có thể sử dụng để nối client và server
- Các lựa chọn công nghệ quen thuộc cho hệ thống mạng
- Các gói UML có thể sử dụng thế nào phân cụm các lớp liên quan trong biểu đồ triển khai

BÀI TẬP

1. Hãy trình bày so sánh về những đặc điểm giữa hai công nghệ J2EE và .NET
2. Nêu lên quyết định của nhóm về sử dụng công nghệ nào cho bài tập của nhóm của mình

CHƯƠNG 8 THIẾT KẾ CÁC HỆ THỐNG CON

8.1 GIỚI THIỆU

Do quy mô của việc thiết kế các hệ thống con và sự sáng tạo là duy nhất đối với mỗi dự án, nên chúng ta không thể hy vọng viết ra hướng dẫn từng bước thiết kế hệ đa tầng hướng đối tượng chuyên nghiệp. Vì vậy, chương này tập trung trình bày tổng quan những hoạt động chủ yếu mà khi bạn đặt nó ở vị trí trung tâm trong thực tế thiết kế thì mọi thứ sẽ đâu vào đó.

Trước tiên chúng ta cần xem xét tầng nghiệp vụ được thiết kế như thế nào? Điều này thường liên quan đến việc phải quyết định xem đối tượng sẽ được đặt ở tầng nào, chúng được kết nối như thế nào và giao diện của chúng sẽ là gì. Chúng ta phải biến đổi mô hình lớp hướng nghiệp vụ mà chúng ta đã phát triển trong pha phân tích thành các lớp có thể cài đặt theo ngôn ngữ lập trình mà chúng ta đã chọn. Trong tài liệu này, thiết kế được trình bày ít nhiều độc lập với ngôn ngữ và các công nghệ khác nhưng để tiện minh họa chúng tôi nghiên về java và cơ sở dữ liệu quan hệ.

Việc thiết kế hệ thống con liên quan tới việc chuyển đổi mô hình phân tích theo khái niệm thành các lớp cài đặt được theo chiến lược đặt ra trong mô hình thiết kế hệ thống. Theo nguyên lý thiết kế hệ thống, thiết kế hệ thống con có thể tiến hành như sau:

1. Thiết kế các lớp và trường của tầng nghiệp vụ bằng cách sử dụng mô hình lớp phân tích. Tầng nghiệp vụ bao gồm các thực thể trong miền bài toán và các lớp hỗ trợ khác nếu cần.
2. Quyết định cách lưu trữ dữ liệu lâu dài và thiết kế khuôn dạng lưu trữ. Dữ liệu lưu trữ lâu dài nghĩa là không bị mất đi khi hệ thống ngưng hoạt động.
3. Xác định giao diện cuối cùng liên quan tới bản vẽ phác thảo được đưa ra trong suốt pha phân tích.
4. Duyệt lại các use case hệ thống cùng đối chiếu thiết kế giao diện người dùng, chú ý đến các dịch vụ nghiệp vụ phải được hỗ trợ bởi lớp trung gian. Các dịch vụ nghiệp vụ là những câu hỏi và lệnh mà client có thể gửi đến server như “mua một quyển sách” hoặc “đặt trước một mô hình xe hơi”.
5. Phát triển các dịch vụ thành các đối tượng server với những thông điệp có sẵn trên mạng. Các đối tượng Server cài đặt các dịch vụ nghiệp vụ bằng cách sử dụng tầng nghiệp vụ.
6. Hoàn thành điều khiển tương tranh và an toàn cho các tiến trình. Điều khiển tương tranh nghĩa là sử dụng những nguyên tắc nghiệp vụ để điều khiển quyền truy

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

cập hệ thống: tên người sử dụng, password, đặt vé trước khi mua....An toàn nghĩa là phải chắc rằng dữ liệu trong quá trình xử lý không sai sót và các hoạt động song song phải có sự phối hợp tốt.

8.2 ÁNH XẠ TỪ MÔ HÌNH LỚP PHÂN TÍCH SANG MÔ HÌNH LỚP THIẾT KẾ

Khi đi từ phân tích sang thiết kế, một số lớp sẽ bị loại bỏ(ví dụ như những lớp điều khiển) và một số lớp khác sẽ được thêm vào (như các lớp để thực thi đa nhiệm). Người thiết kế sẽ tự quyết định thiết kế đối tượng nghiệp vụ, biên và và homes thành mã cài đặt. Thật ra, điều này cũng tùy thuộc cách tiếp cận mô hình tiến trình mà dự án sử dụng. Trong mỗi lớp thiết kế mà chúng ta đưa ra, chúng ta cần chọn tên và kiểu câu các trường. Thông thường, các trường được dẫn xuất từ các thuộc tính và các liên kết được tìm thấy trong suốt quá trình phân tích. Cũng như các thuộc tính và liên kết, chúng ta cần xét tới kế thừa. Quan hệ kế thừa không nhất thiết phải được ánh xạ vào thành cái gì mới mà chỉ cần quyết định giữ hay không giữ chúng. Nó có thể có hay không có tính kế thừa trong hệ thống nhưng thường nó được đưa vào trong pha thiết kế hơn là phân tích.

8.2.1 Ánh xạ các phương thức

Cho đến nay, phương thức được đưa ra chỉ đơn thuần như một cách ghi lại hiện thực hóa các use case và có thể xem như là hiệu ứng phụ để kiểm định các lớp phân tích hỗ trợ cài đặt. Các phương thức phân tích này nên bỏ qua trong thiết kế. Như vậy, phương thức thiết kế đến từ đâu? Khi chuyển đến thiết kế, chúng ta thường gắn với các thuật ngữ “phương thức” và “thông điệp” trong lập trình (“message” và “method”) hơn là thuật ngữ thao tác trong UML (“operation”). Đối với đa số các đối tượng, không quan tâm đến tầng chứa nó, các thông điệp sẽ được thêm vào bởi một số lý do sau đây:

- Để cho phép các đối tượng client đọc hoặc thay đổi các giá trị của các trường.
- Để cho phép đối tượng client truy nhập dữ liệu dẫn xuất (ví dụ, như một message để đọc bán kính của đường tròn, chúng ta cũng muốn có thể được đường kính).
- Do kinh nghiệm hay trực giác nói rằng thêm vào là có ích.
- Bởi vì khung hay mẫu mà chúng ta định sử dụng đòi hỏi phải có các thông điệp nào đó.

Thêm vào đó, khi chúng ta thiết kế dịch vụ nghiệp vụ cho tầng giữa, chúng ta tìm ra những thông điệp cho các đối tượng server. Khi chúng ta chỉ ra được dịch vụ nghiệp vụ có thể được thỏa mãn bằng cách sử dụng các đối tượng nghiệp vụ, chúng ta có khả năng đổi đầu với nhiều thông điệp hơn. Tóm lại, khi ánh xạ các lớp phân tích, các thuộc tính và các quan hệ sang bản sao thiết kế của chúng, các thông điệp sẽ bắt đầu xuất hiện từ mọi hướng.

8.2.2 Các kiểu biến

Khi giới thiệu một trường, chúng ta cần quyết định nó thuộc kiểu nào: kiểu cơ bản hay kiểu lớp. Cho nhiều mục đích, chúng ta có thể tự hạn chế các kiểu sau đây:

- Kiểu cơ bản và kiểu lớp đơn giản mà chúng ta hy vọng tìm được trong mọi ngôn ngữ lập trình hướng đối tượng (ví dụ: int, float, boolean, String, List.....).
- Những lớp mà chính chúng ta định thiết kế.
- Những lớp từ các mẫu và khung mà chúng ta chọn sử dụng.

8.2.3 Phạm vi của các trường

Cũng như cung cấp các tên và kiểu của trường, chúng ta phải khai báo phạm vi của chúng. Phạm vi của chúng chỉ rõ các đoạn code nào cho phép đọc hay thay đổi giá trị. Bốn cách truy nhập sau là đủ cho nhiều mục đích:

- Private (trong UML là dấu -): chỉ sử dụng trong phạm vi lớp định nghĩa.
- Package (trong UML dấu ~): dùng trong phạm vi lớp định nghĩa và tất cả những class cùng gói.
- Protected (trong UML dấu #): dùng trong phạm vi lớp định nghĩa, tất cả những lớp cùng gói và tất cả những lớp con của lớp định nghĩa (dù trong hay ngoài gói).
- Public ((trong UML dấu +): dùng bất cứ ở đâu.

Thông thường, nếu một ngôn ngữ cho phép, người phát triển sẽ tạo ra trường private: một phần là do lợi ích của việc đóng gói, điều này sẽ cho compiler các cơ hội tối ưu hơn. Đôi khi người phát triển sẽ tạo ra trường protected để người phát triển những lớp con có nhiều cơ hội để sửa đổi các hành vi của lớp cha (mặc dù việc làm này làm tăng sự gắn bó giữa lớp cha và lớp con).

Các trường với package là ý tưởng tồi ví chúng có thể được truy nhập bởi mẫu code trong gói mà không biết gì về sở hữu đối tượng. Đôi khi vì lý do thực tế như hiệu năng, người phát triển sẽ cho truy nhặt kiểu package nhưng chỉ với ngôn ngữ lập trình nào có cách chỉ cho đọc giá trị (ví dụ, từ khóa final trong java).

Phạm vi cũng có thể áp dụng với thông điệp. Khi đó, thông điệp có thể là:

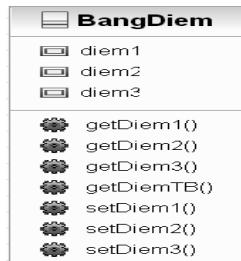
- Public, nếu nó là phần của giao diện của package
- Package, nếu nó là mã cài đặt được sử dụng bởi chính lớp đó và bởi lớp trong cùng package
- Protected, nếu nó là mã cài đặt được sử dụng bởi chính lớp đó, lớp con và các lớp trong cùng package
- Private, nếu nó là mã cài đặt để sử dụng chỉ bởi lớp đó

Ngoại trừ java không phải ngôn ngữ nào cũng hỗ trợ các kiểu truy nhập trên.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

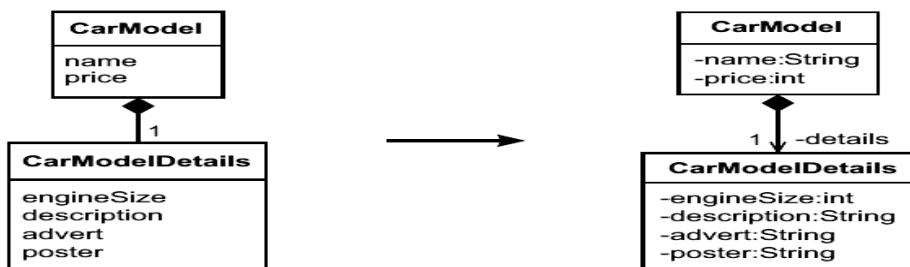
8.2.4 Các toán tử truy nhập

Có hai toán tử truy nhập: get trả về giá trị của trường và set thiết lập giá trị mới của trường. Các toán tử này làm dễ dàng cho việc bảo trì hơn và trình biên dịch dễ tối ưu hơn.



8.2.5 Ánh xạ các lớp, thuộc tính và hợp thành

Trong UML, một số ký hiệu giống nhau được sử dụng cả cho biểu đồ lớp phân tích và biểu đồ lớp thiết kế. Tuy nhiên, bạn sẽ thấy chính bạn sử dụng nhiều ký hiệu sẵn có trong biểu đồ thiết kế của bạn hơn.



Hình 8.1 biêt diêt môt môt đoạn cùa biêt đồ lớp phân tích từ hệ quản lý thuê xe đưốc chuyêñ đổi sang môt biêt đồ lớp thiết kế. Khi ánh xạ sang biêt đồ lớp thiết kế, ba vấn đề chính đâ đưốc giải quyết: các lớp phải đưốc cài đặt, các kiểu thuộc tính, và cách ánh xạ hợp thành. Trong trường hợp này, quyết định phải đưốc tạo ra để giữ lại cả hai các lớp phân tích mà khôñ hỗ trợ các lớp phụ thuât (ngoài các lớp String). Các thuộc tính truy nhập trường vẫn đưốc private là thích hợp.

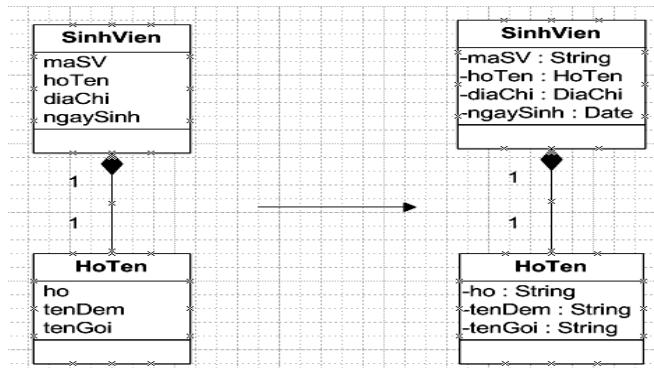
Một thuộc tính tùy chọn sẽ đưốc ánh xạ theo cùng cách với sự khác biệt duy nhất là trong thời gian chạy, trường tương ứng lấy giá trị null. Các quan hệ hợp thành cần suy nghĩ hơn. Trong khi phân tích, hợp thành là song hướng: bắt đầu từ môt trong hai đối tượng, chúng ta có thể dễ dàng tìm thấy các đối tượng ở đầu kia. Tuy nhiên, khi chuyêñ sang thiết kế, chúng ta phải đối phó với môt thực tế là các trường chỉ có thể hướng theo môt chiều (tại thời gian chạy, chúng ta có thể lấy từ trường từ môt đối tượng tới đối tượng ở kia, nhưng khôñ thể lấy lại). Vì vậy, chúng ta cần quyết định xem chúng ta có muốn có môt trường ở môt đầu cùa môt quan hệ hoặc chỉ ở môt đầu và nếu vậy thì đầu nàò.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Chúng ta mong quan hệ hợp thành ban đầu là từ cái khởi tạo đến cái được hợp thành. Điều này phản ánh một thực tế là cái khởi tạo là chủ sở hữu của mỗi quan hệ. Nó cũng phản ánh một thực tế là một hợp thành tương tự như một thuộc tính: bình thường, đối tượng khởi tạo sử dụng dịch vụ của các thuộc tính hoặc các đối tượng được hợp thành, nhưng không phải ngược lại. Hãy nhớ rằng điều này không tuyệt đối: nếu chúng ta cảm thấy nó phù hợp với hệ thống cụ thể của chúng ta có thể đặt một trường bên trong đối tượng được hợp hay bên trong cả khởi tạo và hợp thành.

Sau khi đã quyết định hướng của hợp, chúng ta có thể thêm một đầu mũi tên cho biểu đồ lớp để chỉ ra cách hợp thành ánh xạ tới một trường (Xem hình 8.1). Trong thời gian chạy, các thông điệp chỉ có thể theo hướng mũi tên. Chúng ta có thể bỏ hợp thành phần và biểu diễn detail:CarModelDetails bên trong CarModel.

Ví dụ Hệ quản lý học theo tín chỉ



8.2.6 Ánh xạ các kiểu kết hợp khác

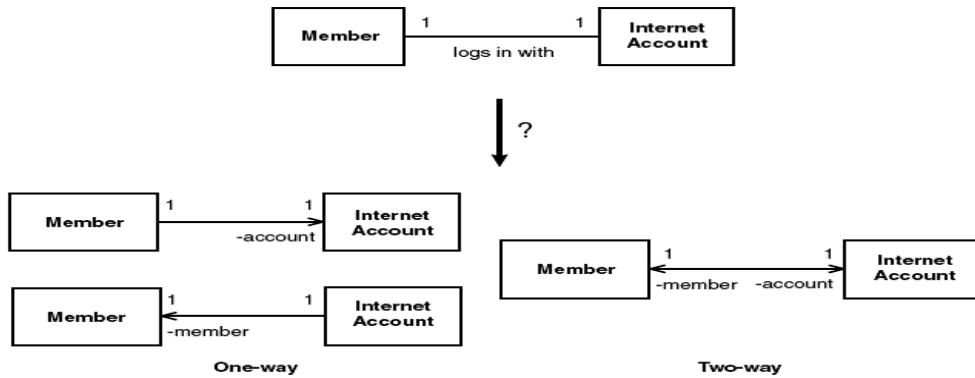
Phần này chúng ta sẽ xem xét các kiểu kết hợp khác. Nhắc lại rằng có ba loại kết hợp: association, aggregation và composition. Cho mục đích ánh xạ, chúng ta không cần phân biệt giữa aggregation và association vì các ngôn ngữ lập trình hướng đối tượng không có sự phân biệt này. Vì vậy, thuật ngữ kết hợp sẽ sử dụng cho phần còn lại của trình bày này.

Đa số kết hợp mà chúng ta quyết định giữ lại từ phân tích, cùng với bất kỳ hợp nào mà chúng ta thêm vào khi thiết kế thì chỉ như các trường trong các đối tượng. Một số có thể là trường của lớp. Vì các trường chỉ cho phép chuyển theo một hướng nên chúng ta cần phải quyết định xem chúng ta muốn đi theo cả hai hướng hay một hướng. Cách chúng ta cài đặt kết hợp phụ thuộc vào tính nhiều ở mỗi điểm cuối: 1-1, 1-n hay m-m.

Kết hợp kiểu 1-1

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Hãy xem phần biểu đồ lớp phân tích ở Hình 8.2. Có ba cách cài đặt kết hợp: đặt trường account vào trong Member, trả tới InternetAccount; cũng có thể đặt một trường gọi member vào trong InternetAccount trả tới Member; hoặc kết hợp cả hai.

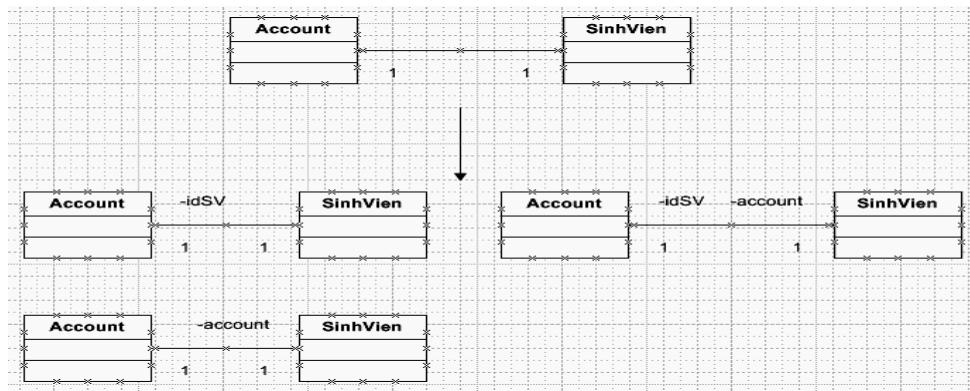


Hình 8.2: Ánh xạ kết hợp 1-1

Nếu theo cách hai (two-way), chúng ta cần thêm code để làm chắc chắn rằng sự kết hợp không thay đổi cách đi. Ví dụ, không có ý nghĩa khi thay đổi một trường member của InternetAccount để trả tới Member khác nếu chúng ta không thông báo cho Member ban đầu rằng account của nó đã thay đổi (có thể giá trị null). Tất nhiên có thể thực hiện sự đồng bộ hóa này bằng cách sử dụng các phương thức thích hợp trong Member và InternetAccount nhưng hơi vụng về. Vì vậy, phần lớn chúng ta chọn one-way. Trong trường hợp này, nó có vẻ thích hợp để Member tham chiếu tới InternetAccount nhưng không phải là ngược lại.

Nếu chúng ta lựa chọn cách khác, miễn là chúng ta có một cơ sở dữ liệu quan hệ bên dưới tầng nghiệp vụ của chúng ta, chúng ta có thể lấy được những thông tin bổ sung ở thời gian chạy. Ví dụ, nếu chúng ta chọn đặt một trường vào trong Member, thì InternetAccount có thể hỏi cơ sở dữ liệu Member của nó ở đâu.

Ví dụ

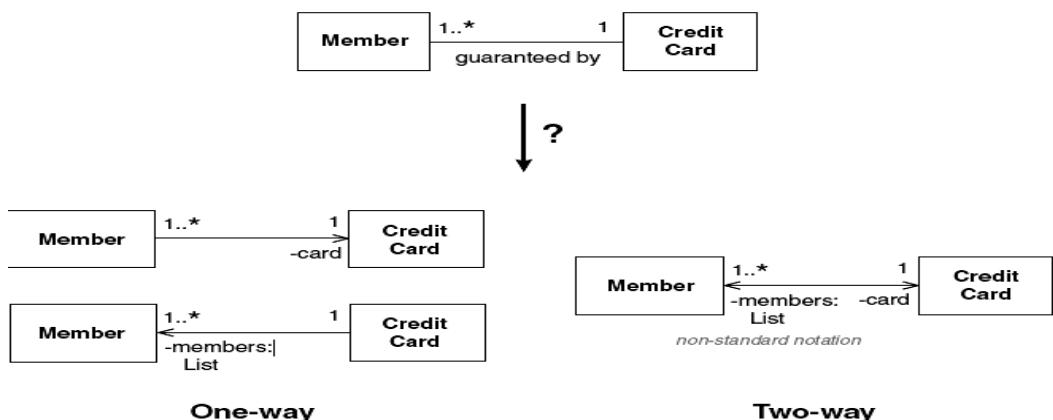


CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Hình 8.5: Ánh xạ kết hợp 1-1

Kết hợp kiểu 1-n

Hình 8.6 biểu diễn một sự kết hợp 1-n. Với trường hợp này, chúng ta phải quyết định có nên đặt một trường trong một đầu hoặc cả hai. Những đối số tương tự cung cấp như trong trường hợp 1-1. Như trước, chúng ta có thể quyết định rằng có một giải pháp tự nhiên, nhưng chúng ta vẫn tự do sử dụng kỹ năng và quyết định của chúng ta như những người thiết kế đã làm điều đó.



Hình 6.8: Ánh xạ hợp 1-n

Không giống như trường hợp 1-1, nếu chúng ta quyết định đặt một trường vào cuối thời gian này, chúng ta phải được chuẩn bị để lưu trữ nhiều hơn một đối tượng liên quan. Ví dụ, nếu chúng ta thêm một trường thành viên tới CreditCard, trường sẽ phải lưu một hoặc nhiều các đối tượng thành viên. Như vậy CreditCard sẽ cần phải sử dụng một số loại llop collection để tổ chức các llop về đến các đối tượng thành viên của nó (hoặc có lẽ là một mảng, tùy theo sở thích các nhân).

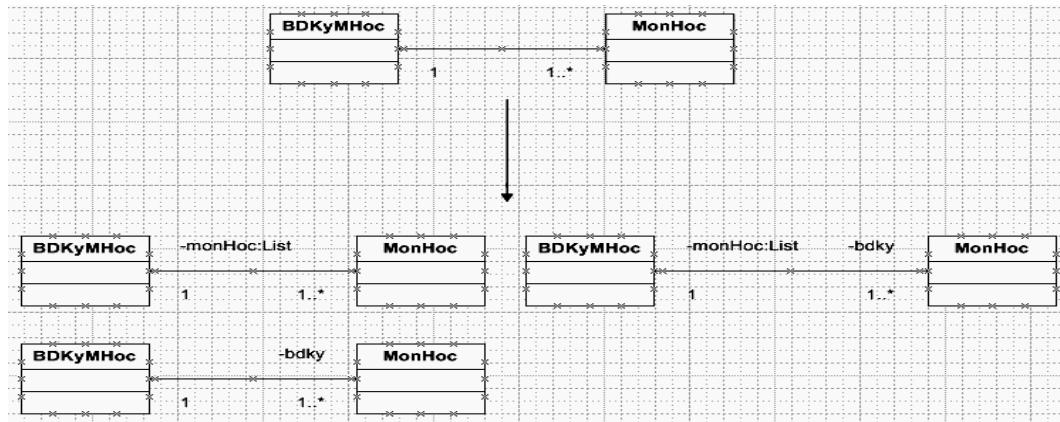
Trong hình 10.3, một List đã được sử dụng. Chúng ta có một cái khó xử lý ở đây: lựa chọn tự nhiên của collection giống như Set; tuy vậy, khi bạn thêm một đối tượng tới một Set, phần lớn sự thực thi sẽ kiểm tra xem các đối tượng đã có, để nó không xuất hiện 2 lần, khả năng một hành động tốn kém.

Bởi thế, nếu bạn có thể chắc rằng bạn sẽ không gán thêm một đối tượng hai lần, bạn sẽ sử dụng một Bag. Nếu bạn không luôn đảm bảo, xem xét việc sử dụng một Bag và thực hiện kiểm tra trùng lặp bằng tay trong các trường hợp đặc biệt. Nếu ngôn ngữ thực hiện của bạn không cung cấp một cái j đó như một Bag, như Java không cung cấp, một List sẽ làm, đặc biệt nếu các bản ghi được giữ theo thứ tự, cái làm nhanh hơn việc tìm kiếm.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

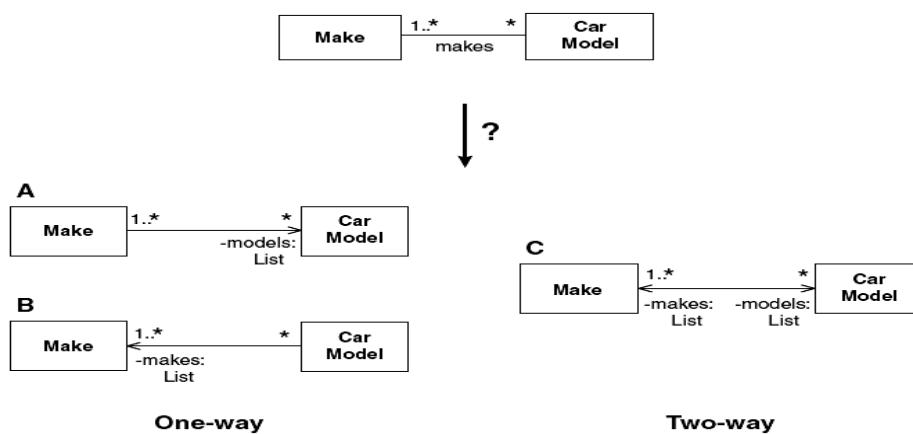
Ký hiệu thẻ hiện trong hình 10.3, liên quan tới các trường giá trị collection, ko thuộc về UML. Mặc dù chúng ta có thể chắc chắn biểu diễn một trường như một tập hợp thích hợp với một sự biểu diễn vai trò tính chất và trường tên, :List là ko chuẩn. Những cách khác để hiển thị quan hệ trong UML là phức tạp hơn và ít thông tin – phiên bản dc hiển thị ở đây là nhỏ gọn nó cho phép chúng ta rút ra các mối quan hệ tối kiều của đối tượng bên trong collection.

Ví dụ áp dụng



Hình 8.7: Kết hợp n – n.

Đây là trường hợp phức tạp nhất. Xem xét ví dụ biểu diễn trong hình 10.4. Ở đây, chúng ta có một Make mà có thể tạo ra nhiều số của các đối tượng CarModel, trong khi mỗi CarModel có thể tạo bởi một hoặc nhiều đối tượng Make. Mỗi cái của 3 ánh xạ có thể là nhãn A, B hoặc C – Danh sách các đối tượng đã dc lựa chọn lại, thời gian này, cho phép chúng ta đặt các đối tượng Make theo thứ tự quan trọng.



Hình 8.8: Ánh xạ hợp n-m

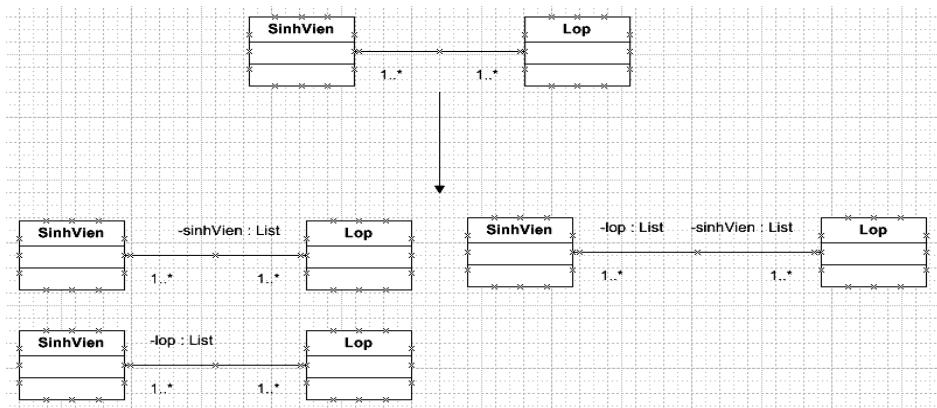
CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Cũng như rất nhiều sự kết hợp n-n, có thực sự là ko có chủ sở hữu của sự kết hợp trong trường hợp này. Nếu chúng ta quyết định, dựa trên miền vấn đề của chúng ta, mà chúng ta có khả năng bắt đầu với một Make và sau đó chuyển hướng tới các đối tượng CarModel của nó, lựa chọn A sẽ làm việc tốt.

Ngược lại, nếu chúng ta quyết định chúng ta thích hơn bắt đầu với một CarModel, B sẽ là lựa chọn tốt hơn. Nếu, tuy vậy, chúng ta quyết định rằng chúng ta có vẻ phù hợp giống như bắt đầu với một Make hoặc một CarModel, chúng ta thực sự lựa chọn C.

Với C, vấn đề đồng bộ hóa là vấn đề nguy hiểm hơn các trường hợp các trường hợp 1 – 1 và 1 – n: chúng ta sẽ phải tìm thông qua collections để tìm các đối tượng, chứ ko chỉ các đối tượng truy cập trực tiếp. Trong trường hợp như vậy, nó có thể giới thiệu một lớp association để thỏa mãn với sự phức tạp, như mô tả tiếp theo.

Ví dụ



Các lớp liên kết

Chúng ta lần đầu tiên gặp phải các lớp association trong phần phân tích. Một lớp association là cần thiết nếu có dữ liệu có liên quan tới một association. Đoạn của lược đồ lớp phân tích ở đinh hình 10.5 biểu diễn một ví dụ của một lớp association.

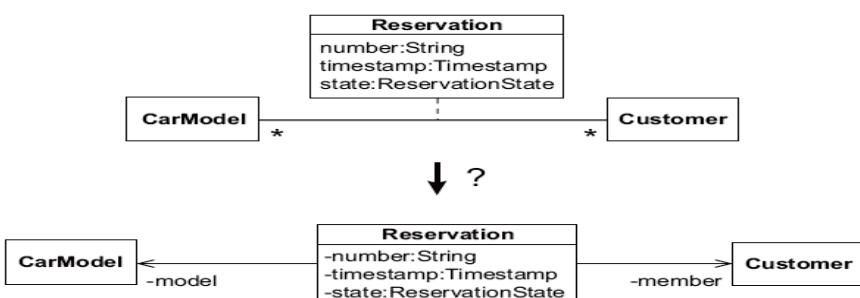


Figure 10.5: Mapping an association class

Từ lược đồ này, chúng ta biết các đối tượng khách hàng có thể đăng ký bất kỳ số của các đối tượng CarModel và để các đối tượng CarModel có thể đăng ký bởi bất kỳ số của

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

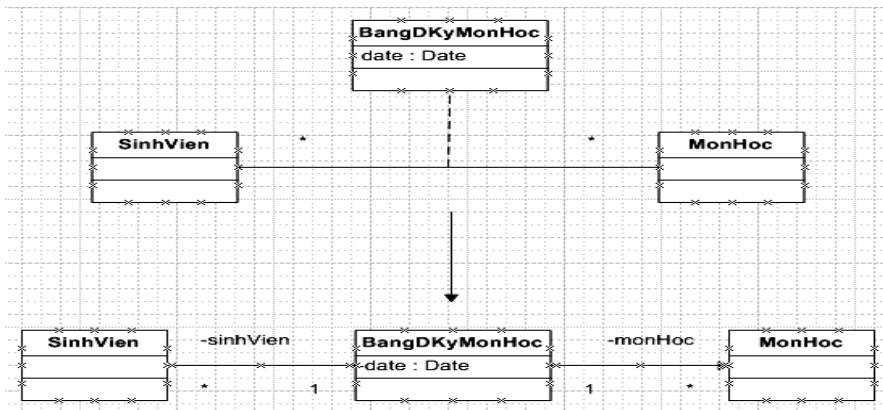
các đối tượng khách hàng; nhưng, cho mỗi liên kết Reservation (dành riêng), chúng ta cần ghi lại số dành riêng, time-stamp và trạng thái, dữ liệu này thuộc về Customer hoặc CarModel, hơn nữa nó thuộc về association của nó. Bởi thế, chúng ta sẽ giới thiệu Reservation như một lớp Association, với các thuộc tính thích hợp.

Khi chúng ta đến việc thiết kế một lớp association, điều đơn giản nhất để làm là tạo một lớp thiết kế với các trường của tất cả các thuộc tính, và 2 trường thêm để trỏ tới các đối tượng associated, như biểu diễn ở giữa của hình 10.5. Sau đó, chúng ta có thể tạo một trường của lớp association cho mỗi liên kết.

Nếu chúng ta muốn có thể chuyển hướng từ một CarModel tới các đối tượng Reservation của nó, chúng ta sẽ phải thêm một trường tới CarModel, với tất cả các vấn đề đồng bộ hóa. Một tham số tương tự cung cấp tới Customer. Như một sự lựa chọn, chúng ta có thể cung cấp một message findAllReservation- For(:CarModel) trên lớp ReservationHome. Một home cho phép chúng ta tạo và tìm kiếm các đối tượng của một lớp cụ thể, thông thường bằng việc nhúng vào trong db bên dưới, mỗi đối tượng home là một singleton: chúng ta việc code để đảm bảo rằng chỉ một trường của home tồn tại.

Một lớp association các tổng quát nhất của việc ánh xạ association từ phân tích tới thiết kế. Vì vậy chúng ta có thể, nếu chúng ta muốn, giới thiệu một lớp association để biểu diễn mọi association từ biểu đồ lớp phân tích của chúng ta. Cái này sẽ giảm bớt vấn đề đồng bộ hóa như chúng ta đã thấy. Mặc dù vài tool phát sinh mã thực hiện hoạt động theo cách này, bạn có thể cảm thấy rằng nó là quá nhiều rắc rối nếu bạn đang việc code bằng tay.

Ví dụ áp dụng

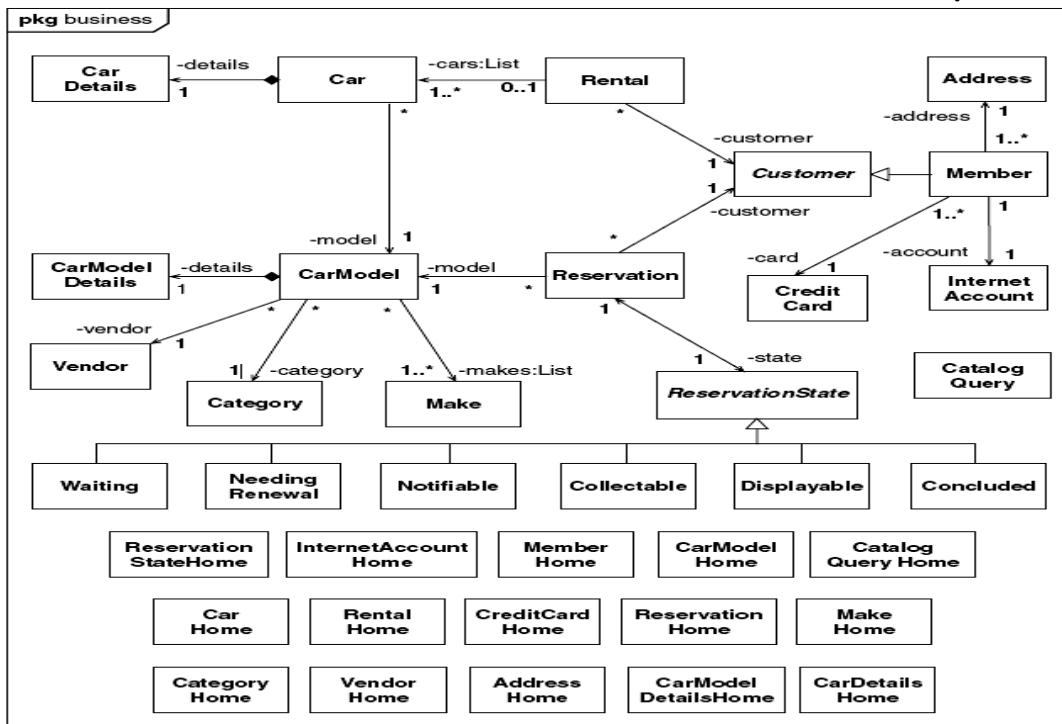


Case Study

iCoot BusinessLayer class diagram

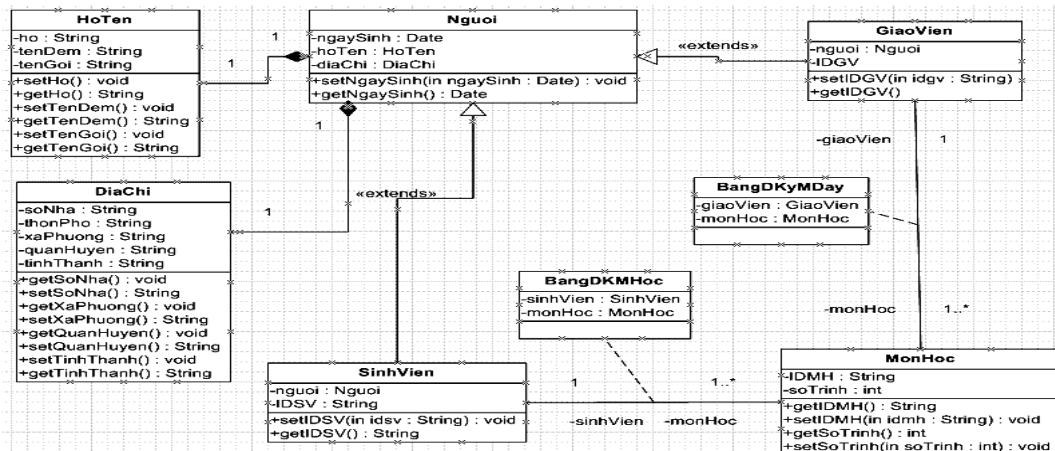
Case study: lược đồ lớp iCoot BusinessLayer đã được ánh xạ vào trong các lớp thiết kế để có thể code ngay lập tức trong bất kỳ ngôn ngữ lập trình hướng đối tượng nào.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON



Hình 8.10: Các lớp tầng nghiệp vụ cho iCoot

Ví dụ áp dụng



8.2.7 Định danh phổ quát

Hầu hết các đối tượng nghiệp vụ, ở vài thời điểm trong cuộc sống của chúng, cần được gọi ra bằng Key. Một khóa là một giá trị thuộc tính, hoặc một sự kết hợp của các giá trị, đó là duy nhất tới mỗi trường. Ví dụ, một acc ngân hàng là duy nhất bởi sự kết hợp của số tài khoản và mã phân loại

Xử lý các loại khác nhau của các khóa trong một hệ thống phần mềm có thể nặng nề. Bởi thế, xem xét giá trị của nó để đưa vào một số duy nhất để phân biệt mỗi đối tượng nghiệp vụ từ các đối tượng khác nhau của vài class. Cái này giúp đồng bộ hóa sao chép của các đối tượng, để kiểm tra hoạt động của chúng như chúng ta đi vòng một mạng

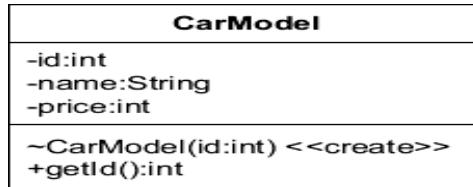
CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

(của vũ trụ) và để xử lý các khóa thống nhất và hiệu quả (thông qua các đối tượng via, ví dụ). Nhận dạng phổ thông là hữu ích khi các đối tượng và các liên kết đã được lưu trữ trong một db – nhận dạng phổ thông nắm giữ nơi của một địa chỉ bộ nhớ khi hệ thống tắt.

Một kiểu phù hợp cho một nhận dạng phổ biến là một integer. Kiểu đặc trưng của integer phụ thuộc vào có bao nhiêu các đối tượng duy nhất của mỗi lớp bạn muốn cung cấp: một integer 16 bit (như trong java) cho phép 65 ngàn trường độc nhất, một số 32 bit (int trong java) cho phép 4 nghìn tỷ trường độc nhất. Nếu 4 tỷ trường cung cấp ko đủ, có 64 bit số nguyên thay thế (long) cái cho phép một số vô tận số của các trường (khoảng 18×10^{30}).

Nếu bạn sử dụng một trường độc nhất, nó sẽ thiết lập trong suốt construction của đối tượng và nó sẽ cố định còn lại về sau. Hình 10.7 minh họa sự thi hành của một trường duy nhập cho lớp CarModel. Từ khóa UML <<create>> đã sử dụng để biểu thị rằng các hoạt động CarModel tạo các trường của CarModel.

Cái này là thực hành tốt trong UML, ko có từ khóa, người đọc sẽ phải giả định rằng một hoạt động với tên giống như lớp của một constructor nhưng thực tế là tồn tại ở tất cả, chưa nói đến các tên của chúng là luật, là đặc trưng ngôn ngữ. Nó có thể phổ biến, nhưng nó vẫn là ngôn ngữ đặc trưng. Bởi thế, trong cuốn sách này, mặc dù giới hạn constructor là sử dụng như tốc ký cho “hành động sử dụng để tạo trường của lớp”, từ khóa keyword <<create>> vẫn sử dụng trong lược đồ.



Hình 8.7: Implementing a universal identifier

Thông thường, constructor sẽ ko public: cái này sẽ cho phép người phát triển điều khiển cái nhận dạng chung thực tế để lấy cái đã gán, sử dụng home cho ví dụ. Nếu constructor là public, bất kỳ client nào có thể tạo một CarModel sử dụng một từ định danh để sử dụng ở nơi khác. Trong hình 10.7, constructor có gói truy cập, để các đối tượng có thể tạo bởi một lớp CarModelHome sống trong cùng một gói.

Áp dụng



8.3 XỬ LÝ LUU TRỮ VỚI CƠ SỞ DỮ LIỆU QUAN HỆ

8.3.1 Hệ quản trị cơ sở dữ liệu

Trong bốn thập kỷ qua, nhiều biến của DBMS đã được phát minh ra – ví dụ như indexed file, hierarchical, network và object-oriented. Đó luôn luôn có sự khác biệt giữa các ngôn ngữ lập trình mà chúng ta sử dụng để viết mã của chúng ta và cách chúng ta truy cập dữ liệu trong database. Trước đó chúng ta cần phải thực hiện một vài kiểu mapping run time giữa các hệ thống phần mềm và database. Một vài công cụ (chẳng hạn như việc triển khai EJB) sẽ tạo code mapping cho chúng ta. Dù thế nào, thậm chí nếu chúng ta muốn sử dụng các công cụ như vậy cho dự án của chúng ta vẫn cần phải hiểu rõ các nguyên tắc để sử dụng chúng hiệu quả (và để hiểu rõ các nguyên tắc để sử dụng chúng hiệu quả (và để hiểu được những message runtime error)). Một cơ sở dữ liệu quan hệ thường được sử dụng để lưu trữ dữ liệu trong một hệ thống đa tầng Internet. The mô hình quan hệ đã được chọn ở đây vì:

- Relational databases là cái chung nhất. Mặc dù Object oriented database vẫn tồn tại, chúng được sử dụng rộng rãi ít hơn nhiều, đặc biệt là trong thế giới kinh doanh, nhưng mà chúng ta không thể bỏ qua chúng vì sự đơn giản
- Tất cả relational database hỗ trợ mô hình lõi tương tự dựa trên toán học chắc chắn. Điều này làm cho chúng duy nhất trong số các loại đa dạng của DBMS
- Java cung cấp một tool hỗ trợ cho kết nối tới relational databases đó là JDBC(java Database Connectivity)

8.3.2 Mô hình quan hệ

Mô hình quan hệ là một mô hình toán học mà có độ tin cậy và dễ hiểu. Trong mô hình quan hệ không giống mô hình mạng và mô hình phân cấp, nó không có các liên kết vật lý. Tất cả dữ liệu được chứa trong các bảng, các cột. Dữ liệu trên 2 bảng được quan hệ với nhau thông qua các cột thay cho liên kết vật lý.

Theo mô hình hướng đối tượng-sẽ trở thành xu hướng công nghệ cơ sở dữ liệu, cần phải chuyển đổi mô hình quan hệ thành mô hình hướng đối tượng để nâng cao năng suất và tính linh hoạt. Chuyển đổi này bao gồm dịch lược đồ, chuyển đổi dữ liệu và chuyển đổi chương trình. Dịch Schema liên quan đến việc tái thiết lập bản đồ và ngữ nghĩa của các giản đồ quan hệ vào hướng đối tượng schema. Dữ liệu chuyển đổi liên quan tới việc xếp dỡ tập quan hệ vào các file tuần tự và tải lại chúng tới các file lớp hướng đối tượng.. Các phương pháp toàn vẹn của relational database bằng cách lập các mapping dữ liệu tương đương.

Một mô hình quan hệ và mô hình hướng đối tượng, ngôn ngữ lập trình có thể kết hợp để mang lại một hiệu quả đáng ngạc nhiên OO-DBMS cho nhiều ứng dụng. Mặc dù một mô hình hướng đối tượng có thể ánh xạ dễ dàng tới một giản đồ quan hệ, nó vẫn khó

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

khăn để biết trước rằng một vài ánh xạ sẽ là hiệu quả nhất cho một vài phần mềm đặc biệt(với khía cạnh để hiệu quả lưu trữ và tốc độ truy cập). Trước đó bạn có thể thử với mmot vài công cụ mapping để tìm ra công cụ mapping nào là tốt nhất cho hệ thống của bạn. Ở đây chúng ta sẽ nhìn vào một mapping hợp lí: không rắc rối, mapping thuận tuý.Cách này, bạn có thể nhận thấy rằng nó có thể chứa dữ liệu hướng đối tượng trong một relational database.Cho axa nàychúng ta có thể thử nhận rằng chúng ta phải điều khiển xem dữ liệu chứa thế nào.Trong thực tế, chúng ta sẽ không may mắn: dưới sự dkhiển của Database Administrators(DBAs) hay database đã tồn tại như là một hệ thống kế thừa và chúng ta ko thể modify- trong trường hợp này chúng ta phải “reverse engineer” mapping của chúng ta. Chúng ta không thể tìm ra làm thế nào để viết code để gắn dữ liệu giữa một hệ thống phần mềm với database vì đó có lẽ là khó khăn duy nhất trong toàn bộ hệ thống.Nếu bạn sử dụng một framework như EJB công cụ sẽ xuất ra các dòng code bạn cần cho việc mô tả axa cơ sở ở đây. Để làm bất cứ điều gì đòi hỏi nhiều nỗ lực và nghiên cứu về database’s programming interface (giao diện lập trình của ban). Đối với yêu cầu của chúng ta,nó đủ để chỉ ra các kĩ thuật để sử dụng câu lệnh SQL đọc dữ liệu từ CSDLとりu đối tượng runtime và sau đó viết thêm các dữ liệu mới. Cho tới khi lớp DB được đóng gói bởi lớp nghiệp vụ chúng ta chỉ việc chúng ta có thể hoàn toàn nhàn rỗi cho việc đưa lớp business cho lợi ích của chương trình khách và lớp DB cho lợi ích của DB-mã mapping có thể nghỉ ngơi.

Bảng

Mô hình quan hệ dựa trên các bảng dữ liệu chứa các dòng và các cột, mỗi cột chứa một kiểu riêng biệt. Chuẩn SQL định nghĩa các kiểu mà bạn có thể lựa chọn trong một dropdownlist.Mà bạn có thể tham khảo:

- VARCHAR(X):một chuỗi mà có thể nhiều nhất là x ký tự
- INTERGER:một kiểu số nguyên mà thường nhung ko phải luôn luôn là 32bits
- DATE: kiểu ngày tháng trong lịch
- TIMESTAMP: một sự kết hợp của ngày và thời gian trong ngày
- BOOLEAN: true hay false

Khóa

Một khóa là một giá trị hay kết hợp của nhiều giá trị,và dành riêng cho dòng.Mỗi bảng có tập các thuộc tính gọi là “Key” định danh duy nhất cho mỗi thực thể. Tìm kiếm một bản ghi sử dụng khoá kết hợp thì chậm hơn tìm kiếm bản ghi sử dụng khoá đơn giản .

Vd trên bảng Address chúng ta có thể sử dụng House và Pothcode nhu một khoá bởi vì việc kết hợp đó đảm bảo sự duy nhất cho bất kì một căn nhà nào.Tuy nhiên việc kết hợp của các chuỗi không tiện lợi và chàm cho sử dụng.Do đó trong ví dụ Address một khoá giả ID được đặt ra để đồng nhất tính duy nhất cho mỗi vị trí.Nó cho phép chúng ta tìm

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

đến bản ghi mà trong thực tế có 2 khách hàng có địa chỉ giống nhau mà ta chỉ việc chèn vào ID thứ 2 nhưng địa chỉ vẫn chỉ là một.

Ánh xạ mô hình đổi tượng thành mô hình quan hệ

Để truy xuất vào cơ sở dữ liệu theo cách hướng đổi tượng, cần phải có một giao diện chuyển đổi logic đổi tượng sang logic quan hệ. Giao diện này gọi là sự ánh xạ đổi tượng có tính quan hệ. Một ORM được tạo từ những đối tượng cho phép truy xuất vào dữ liệu và giữ các qui tắc nghiệp vụ bên trong nó. Một lợi ích của lớp trừu tượng đổi tượng có tính quan hệ là nó ngăn bạn sử dụng cú pháp đặc trưng cho 1 cơ sở dữ liệu. Nó tự động chuyển đổi những lời gọi model object (đối tượng mô hình) thành những câu SQL tối ưu cho cơ sở dữ liệu hiện tại. Điều này đồng nghĩa với việc chuyển đổi sang hệ cơ sở dữ liệu khác khi đã đi 1 nửa dự án trở nên dễ dàng. Hãy tưởng tượng rằng bạn đang viết 1 phiên bản prototype cho 1 ứng dụng nhưng khách hàng chưa quyết định hệ cơ sở dữ liệu nào phù hợp với yêu cầu của họ. Bạn có thể bắt đầu ứng dụng của bạn với SQLite chẳng hạn, và chuyển đổi sang MySQL, PostgreSQL hay Oracle khi nào khách hàng có quyết định. Chỉ cần thay đổi 1 dòng trong file cấu hình và mọi thứ vẫn hoạt động.

Một lớp trừu tượng thu gọn logic dữ liệu. Phần còn lại của ứng dụng không cần phải biết về các câu truy vấn SQL, và phần SQL truy xuất dữ liệu là dễ viết. Những lập trình viên chuyên lập trình cơ sở dữ liệu cũng biết 1 cách rõ ràng cần phải làm gì.

Việc sử dụng đổi tượng thay cho bản ghi (record) và class thay cho table còn có 1 lợi ích khác: bạn có thể thêm các bộ truy xuất khác cho các table của bạn. Ví dụ nếu bạn có 1 table Client với 2 field là FirstName và LastName, bạn có thể cần Name. Trong thế giới hướng đổi tượng, điều này được thực hiện dễ dàng bằng cách thêm phương thức truy xuất cho class Client như sau:

Mã:

```
Code: Select all
public function getName()
{
    return $this->getFirstName().' '.$this->getLastName();
}
```

Tất cả những hàm truy xuất dữ liệu lặp lại và logic nghiệp vụ của dữ liệu có thể được bảo trì trong các object này. Ví dụ có một class ShoppingCart (giỏ hàng) có chứa các item (cũng là đối tượng). Để lấy tổng số tiền của giỏ hàng, bạn có thể thêm phương thức `getTotal()` như sau:

Mã:

```
Code: Select all
public function getTotal()
{
```

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

```
$total = 0;  
foreach ($this->getItems() as $item)  
{  
    $total += $item->getPrice() * $item->getQuantity();  
}  
return $total;  
}
```

8.3.3 Ánh xạ các lớp thực thể

Xu thế phát triển phần mềm hiện nay là phát triển phần mềm hướng đối tượng. Trong phát triển phần mềm hướng đối tượng quan tâm đến đối tượng cả về dữ liệu và hành động của nó, dữ liệu của đối tượng phải được lưu trữ. Do sự khác nhau giữa object model và relational model:

Object Model dựa trên nguyên lý phát triển phần mềm như coupling, cohesion, encapsulation. Relation Model dựa trên các nguyên lý toán học

Object quan tâm đến cả dữ liệu và hành động trong khi đó relational model chỉ quan tâm đến dữ liệu

Một hướng giải quyết cho sự khác nhau này là gộp 2 model nhưng đó không phải là ý kiến hay. Một hướng giải quyết khá tốt là việc ánh xạ từ object model sang relational model.

Để ánh xạ một entity (business object) từ object – oriented model sang relational schema, cần tạo ra một bảng với tên giống như tên class. Mỗi dòng biểu diễn một đối tượng duy nhất trong business domain. Thuộc tính của class có thể không được ánh xạ vào bảng vì không phải thuộc tính nào cũng persistence. Ví dụ như thuộc tính diemTB trong class Diem có thể tính toán không nhất thiết phải lưu vào database. Thuộc tính cũng có thể được vào một cột với kiểu dữ liệu tương ứng trong SQL hoặc một số cột. Ví dụ thuộc tính địa chỉ trong class Sinhvien là một đối tượng và được ánh xạ đến một số cột trong database

Vì lợi ích của lập trình hướng đối tượng việc tạo ra một thuộc tính kiểu integer là ID được gọi là primary key là một vấn đề quan trọng. Lợi ích của việc thêm ID là nó làm đơn giản việc ánh xạ cột (quan trọng nếu chúng ta tự viết code) và nó làm cho việc đọc các đối tượng hoặc chuyển vị trí từ đối tượng này sang đối tượng khác dễ dàng và hiệu quả hơn. Sử dụng ID dễ dàng maintenance các quan hệ giữa các đối tượng, cho phép chúng ta đơn giản hóa các key trong relational database

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Object - oriented model

Diachi	
-sonha : String	
-duong : String	
-phuong : String	
-quan : String	
-thanhhpho : String	

Relational schema

Diachi		
+ID	integer(10)	Nullable = false
SONHA	varchar(50)	Nullable = false
DUONG	varchar(50)	Nullable = false
PHUONG	varchar(50)	Nullable = false
QUAN	varchar(50)	Nullable = false
THANHPHO	varchar(50)	Nullable = false

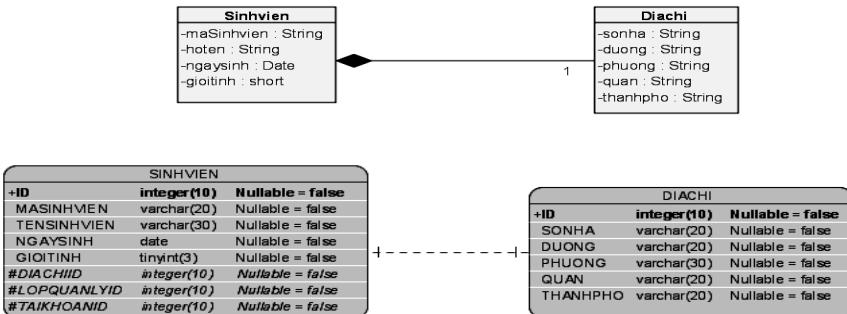
8.3.4 Ánh xạ hợp thành

Hợp thành 1-1

Với quan hệ 1 -1 chúng ta có thể thêm một foreign key vào một trong các bảng thực thể, một foreign key là một trường hoặc một nhóm trường tham chiếu đến khoá chính của một bảng khác. Relational database có thuộc tính 2 chiều nên chúng ta không cần đặt foreign key trong cả 2 bảng.

Một hướng giải quyết khác thay thế foreign key, chúng ta có thể gộp 2 bảng thành một. Nhưng 2 bảng chỉ nên gộp khi có một bảng không được biểu diễn như một thực thể

Trong trường hợp có một optional association (multiplicities 1 and 0..1), chúng ta có đặt thêm một khoá ngoại vào optional end. Trong relational database có hỗ trợ cột có thuộc tính nullable, nhưng để đơn giản tốt hơn nên tránh dùng.



One to many association

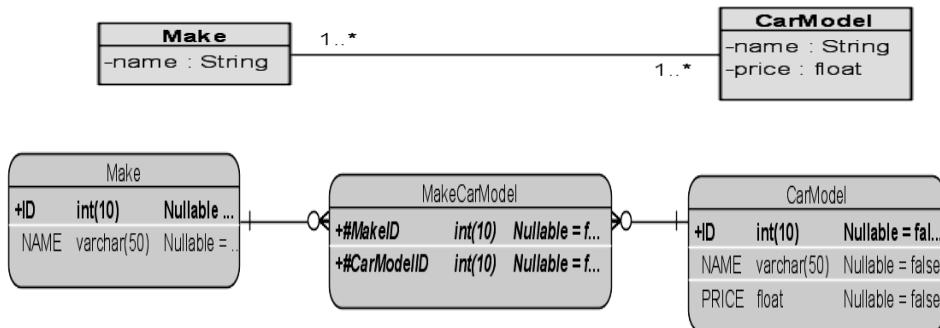
Với quan hệ nhiều nhiều, chúng ta có thể thêm một foreign key vào bảng “many”



CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

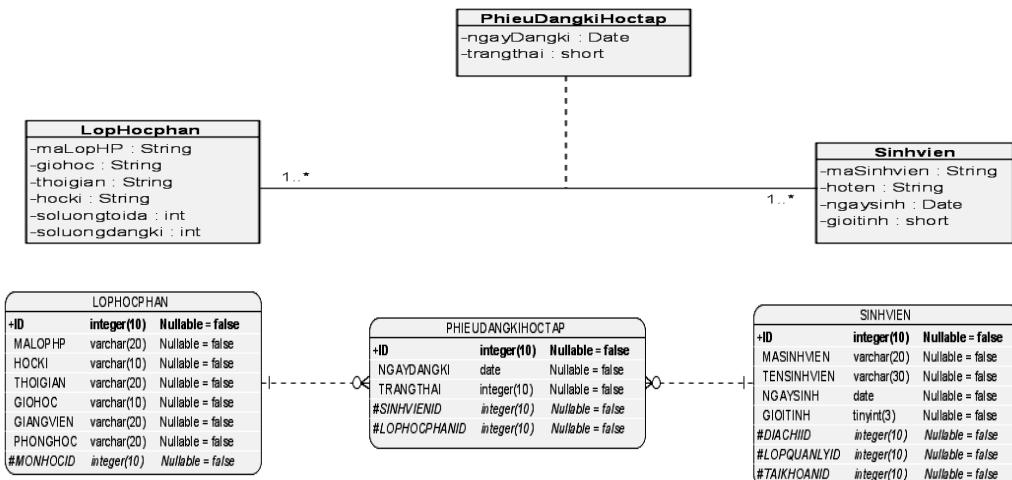
Many to Many Association

Trong relational database mỗi giá trị trong một cột phải nguyên tử vì vậy quan hệ nhiều - nhiều, một foreign key là chưa đủ để xác định nhiều entity ở mỗi đầu liên kết. Cách giải quyết là sử dụng một link table, mỗi hàng trong link table biểu diễn một liên kết giữa một entity trong một bảng đến một entity trong bảng khác. Link table có khoá chính là hợp của 2 khoá ngoại ánh xạ đến 2 bảng. Chúng ta có thể sử dụng một link table để lưu trữ quan hệ một - một, một - nhiều. Tuy nhiên nó sử dụng tốt nhất cho quan hệ nhiều - nhiều và association class



Association class

Association class được tạo ra theo một liên kết, nó biểu diễn các attribute và operation chỉ tồn tại khi liên kết tồn tại. Association class phải được ánh xạ sang một link table, không quan tâm đến multiplicity ở mỗi đầu liên kết. Không giống như link table đơn giản, table biểu diễn association class có thêm các cột chứa thuộc tính của nó. Chúng có thể có một cột ID làm khoá chính (nếu association class biểu diễn một entity)



8.3.5 Ánh xạ trạng thái đối tượng

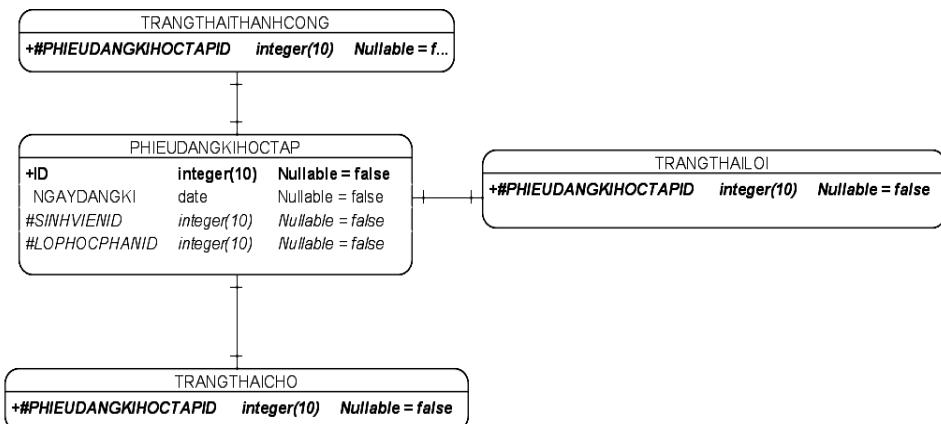
Một đối tượng có một associated state machine có thể được biểu diễn bằng state machine diagram, chúng ta cần phải thu lại những trạng thái của đối tượng. Trong business layer,

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

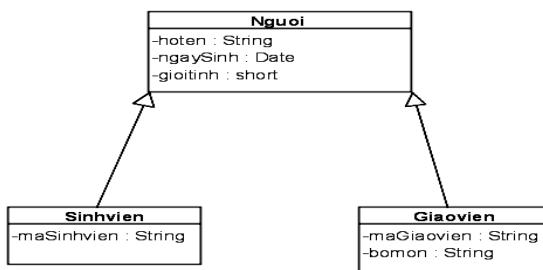
object có thể có một trường đơn để biểu diễn trạng thái của nó kiểu String hoặc int, khi ánh xạ vào database bảng phải thêm một cột có kiểu varchar hoặc integer để mô tả trạng thái

PHIEUDANGKIHOCRAP		
+ID	integer(10)	Nullable = false
NGAYDANGKI	date	Nullable = false
TRANGTHAI	integer(10)	Nullable = false
#SINHVIENID	integer(10)	Nullable = false
#LOPHOCPHANID	integer(10)	Nullable = false

Một cách khác là tạo ra một state object, khi ánh xạ sang database tạo ra một bảng cho mỗi state, sử dụng khoá ngoại để chỉ ra đối tượng có trạng thái đó, khi trạng thái có thuộc tính thì lưu chúng vào các cột mở rộng



Mapping inheritance



Có 3 cách để ánh xạ quan hệ kế thừa sang relational database

Cách 1: Sử dụng một bảng cho entity class hierarchy, tất cả các thuộc tính hierarchy được lưu trữ trong một bảng.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

NGUOI		
+ID	integer(10)	Nullable = false
HOTEN	varchar(30)	Nullable = false
NGAYSINH	date	Nullable = false
GIOITINH	tinyint(3)	Nullable = false
MASINHVIEN	varchar(20)	Nullable = false
MAGIAOVIEN	varchar(20)	Nullable = false
BOMON	varchar(20)	Nullable = false
#DIACHIID	integer(10)	Nullable = false
#TAIKHOANID	integer(10)	Nullable = false
#KHOAID	integer(10)	Nullable = false

Ưu điểm:

- Đơn giản

- Tính đa hình được hỗ trợ khi một người có thể thay đổi vai trò hoặc có nhiều vai trò (ví dụ 1 người có thể vừa là customer vừa là employee)

Nhược điểm:

- Mỗi lần thêm một thuộc tính mới vào bất kỳ bảng nào trong quan hệ kế thừa đều phải thêm một cột vào bảng
- Tăng sự trùng lặp trong class hierarchy
- Các cột trong bảng có thể null

Cách 2: Sử dụng một bảng cho một class cụ thể, mỗi bảng chứa cả thuộc tính của super class và thuộc tính của chúng

SINHVIEN		
+ID	integer(10)	Nullable = false
MASINHVIEN	varchar(20)	Nullable = false
TENSINHVIEN	varchar(30)	Nullable = false
NGAYSINH	date	Nullable = false
GIOITINH	tinyint(3)	Nullable = false
#DIACHIID	integer(10)	Nullable = false
#LOPQUANLYID	integer(10)	Nullable = false
#TAIKHOANID	integer(10)	Nullable = false

GIAOVIEN		
+ID	integer(10)	Nullable = false
MAGIAOVIEN	varchar(20)	Nullable = false
TENGIAOVIEN	varchar(30)	Nullable = false
NGAYSINH	date	Nullable = false
GIOITINH	tinyint(3)	Nullable = false
#DIACHIID	integer(10)	Nullable = false
#TAIKHOANID	integer(10)	Nullable = false
#KHOAID	integer(10)	Nullable = false

Lợi điểm:

Dữ liệu của đối tượng chỉ lưu trực trong một bảng, không phải di chuyển qua các bảng để lấy dữ liệu

Nhược điểm

Khi thêm một thuộc tính vào superclass thì phải sửa ở tất cả các bảng là subclass của nó

Không hỗ trợ nhiều vai trò

Cách 3: Sử dụng một bảng cho một class

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

NGUOI		
+ID	integer(10)	Nullable = false
HOTEN	varchar(30)	Nullable = false
NGAYSINH	date	Nullable = false
GIOITINH	tinyint(3)	Nullable = false
#DIACHIID	integer(10)	Nullable = false

GIAOVIEN		
+ID	integer(10)	Nullable = false
MAGIAOVIEN	varchar(20)	Nullable = false
BOMON	varchar(20)	Nullable = false
#TAIKHOANID	integer(10)	Nullable = false
#KHOAID	integer(10)	Nullable = false

SINHVIEN		
+ID	integer(10)	Nullable = false
MASINHVIEN	varchar(20)	Nullable = false
#LOPQUANLYID	integer(10)	Nullable = false
#TAIKHOANID	integer(10)	Nullable = false

Lợi điểm:

Thoả mãn khái niệm object – oriented nhất. Hỗ trợ đa hình

Dễ dàng sửa superclass và thêm các subclass mới

Nhược điểm:

Có quá nhiều bảng

Mất nhiều thời gian để đọc và ghi dữ liệu, vì phải xử lý trên nhiều bảng

8.4 HOÀN THIỆN GIAO DIỆN NGƯỜI DÙNG

Tiếp theo, chúng ta sẽ xem xét việc thiết kế giao diện người dùng. Chúng ta có một số ít các phác thảo không rõ ràng đã sử dụng trong quá trình bổ sung các yêu cầu hệ thống. Phần này bao gồm các gợi ý và lời khuyên để tạo ra các giao diện tốt và đơn giản cho khách hàng. Từ những giai đoạn đầu của quá trình phát triển, khi tiến hành tìm hiểu yêu cầu và phân tích bài toán, việc xét đến các chức năng trong giao diện người dùng là rất hữu ích. Bởi với 1 phương pháp luận “hướng usecases”, cách thức các tác nhân tương tác với hệ thống là vấn đề quan trọng nhất, trong khi hầu hết các tác nhân đều là con người. Tóm lại, chúng ta có 2 vấn đề sau :

- Những phác thảo giao diện người dùng. Chúng được sử dụng để giúp chúng ta tạo ra các “usecase hệ thống” trong suốt quá trình tìm hiểu yêu cầu, cùng với sự cộng tác của các khách hàng.
- Các đối tượng biên trong biểu đồ giao tiếp. Trong quá trình phân tích động, chúng ta sẽ sử dụng các biểu đồ giao tiếp để thể hiện cách thức thực hiện các usecases. Trong các biểu đồ, mọi tác nhân được hiện thị tương tác với hệ thống thông qua các đối tượng biên.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Tuy nhiên, chúng ta vẫn cần thiết kế giao diện người dùng. Chúng ta phải bám theo các đối tượng biên đơn giản, các phác thảo giao diện người dùng không rõ ràng và các usecases chính xác, biến chúng thành các mô tả giao diện người dùng có thể được thực thi (code) trực tiếp.

Bạn có thể ngạc nhiên khi biết thực tế, hầu hết các hệ thống phần mềm được thiết kế thành công mà không xem xét đến giao diện người dùng chi tiết mà gần như chỉ tập trung hoàn toàn vào các đối tượng biên trong hệ thống. Có 2 lý do chính :

- Các hành vi đúng của hệ thống phụ thuộc vào cấu trúc nội tại của nó chứ không phải cách mọi người tương tác với nó. Tương tự như việc một chiếc xe bao gồm một động cơ, 4 bánh xe và một khung xe. Tùy thuộc vào giao diện và cách chúng ta sử dụng, chúng ta có thể có một chiếc xe chở người, hoặc một chiếc xe kéo, hay một chiếc xe ủi. Dù chọn cách sử dụng nào, nó vẫn là một chiếc xe.
- Chúng ta muốn viết những đoạn mã có thể tái sử dụng. Nếu chúng ta tập trung vào 1 tập các giao diện cụ thể, chúng ta có nguy cơ sẽ tạo ra một hệ thống chỉ làm việc với những giao diện này – đây là một trong những vấn đề trong phát triển phần mềm truyền thống : “ chỉ giải quyết vấn đề của hôm nay trong ngày hôm nay mà quên ngày mai ”. Hướng usecases có vẻ như chống lại nguyên tắc tái sử dụng, tuy nhiên, các usecases chỉ là 1 cách hay để đảm bảo những người phát triển không động đến các vấn đề không liên quan.

Thay vì nghiên cứu sâu lý thuyết tương tác người – máy, chúng ta sẽ xem xét một số nguyên lý cơ bản để thiết kế giao diện người dùng tốt (đặc biệt là cho các khách hàng nhỏ truy cập hệ thống đa tầng)

Sử dụng usecase

Từ cái nhìn hệ thống, usecases đơn giản chỉ cần giúp những người phát triển đi đúng đường. Mặt khác, đối với người dùng, usecases là tất cả mọi thứ. Như vậy usecases nên được sử dụng để cấu trúc nền giao diện người dùng. Nói chung, chúng ta nên cố gắng xây dựng các usecases đơn giản: chúng nên bao gồm quá nhiều chức năng hoặc quá khó để quản lý. Vì vậy, mặc dù chúng ta mong chờ mỗi giao diện người dùng sẽ đại diện cho một số usecases thì các giao diện chính vẫn phải đơn giản.

Chúng ta chờ đợi một nhóm các usecases liên quan được thể hiện trong cấu trúc giao diện người dùng. Ví dụ : khách hàng iCoot được thể hiện với một giao diện người dùng đơn, dựa trên nền web, bao gồm hàng tá các usecases. Với giao diện đơn này, chúng ta chờ đợi sẽ có các nhóm như “ đặt ”, “ cho thuê ”, “ xem ”... Chúng ta cũng nên

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

tránh việc chia một usecase duy nhất hoặc các usecases liên quan thành hơn một giao diện.

Hãy làm nó đơn giản

Giữ mọi thứ đơn giản là một nguyên lý tốt. Một số người dùng là người mới, đặc biệt là những người dùng truy cập hệ thống của chúng ta trên Internet, do đó, đơn giản rất quan trọng. Có một lý do lớn cho việc xây dựng một giao diện đơn giản và ngăn nắp đối với các người dùng chuyên gia : chúng ta không muốn các chức năng rắc rối trong giao diện người dùng cản trở việc sản xuất của họ.

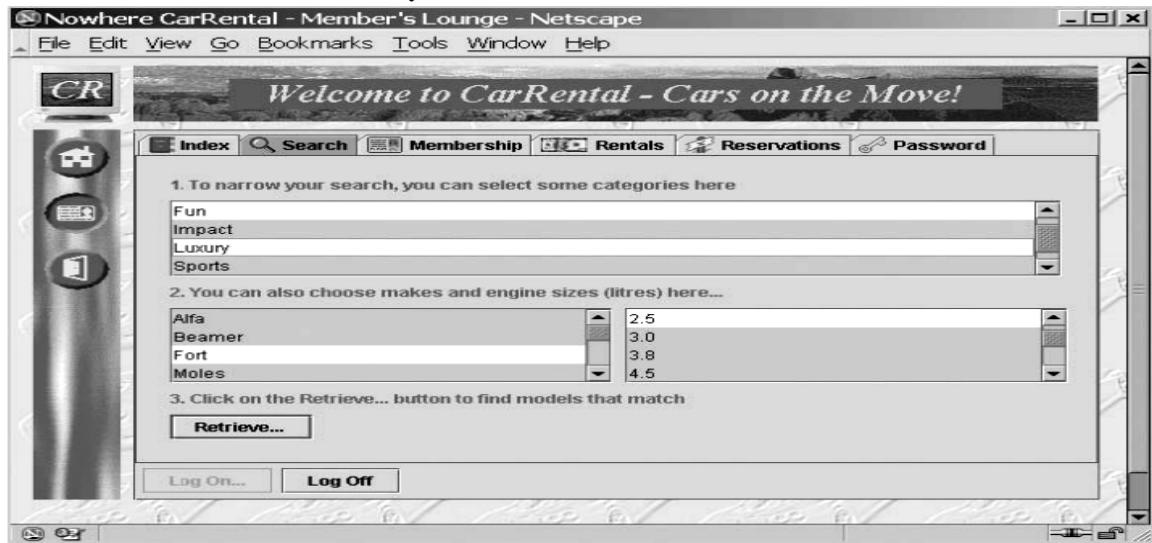
Trong thương mại điện tử, chúng ta không muốn có một “rào cản” trước hàng hóa mà chúng ta bán. Khi một khách hàng vào trang web của chúng ta, nếu chúng được trình bày với các đoạn hướng dẫn sử dụng dài dòng, họ sẽ bỏ đi ngay. Vì vậy, để khách hàng có thể sử dụng trang web của chúng ta ngay lập tức, chúng ta phải hạn chế bớt, xây dựng hướng dẫn từng bước một, như : “click here to buy” hay “enter your credit card details below and click Next”.

Một giao diện càng đơn giản thì càng dễ dàng thêm vào các nền nguyên thủy (ví dụ như điện thoại di động, các hộp set – top, trợ lý số cá nhân, đồ gia dụng...) => ???? Sự cần thiết tạo ra các giao diện người dùng động là một lí do cho việc sử dụng lớp điều khiển trong thiết kế của bạn.

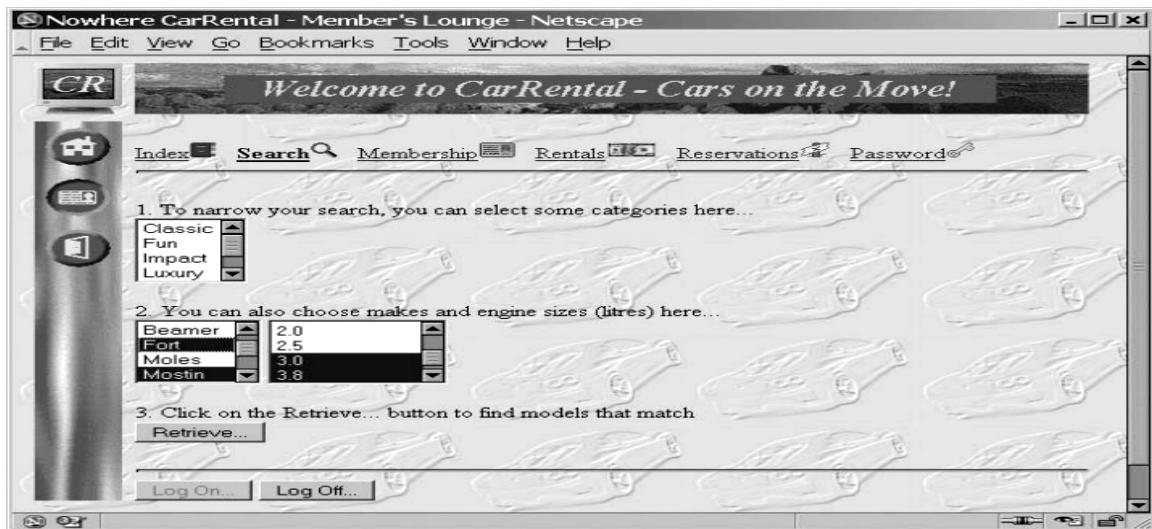
Sử dụng Notebooks :

Bạn nên sử dụng Notebooks để nhóm các usecases có liên quan. Notebook là một tập các trang mà chỉ có một trang tại một thời điểm, các trang khác sẵn sàng thông qua các tabs khác. Lợi ích của notebooks là kích thước và vị trí của nó như nhau, người dùng có thể tập trung vào một vị trí trên màn hình, thậm chí với cả các tương tác lâu có liên quan nhiều hơn đến một usecase. Điều này cải thiện kinh nghiệm và năng suất người dùng. Notebooks được hỗ trợ bởi hầu hết các thư viện giao diện, như Swing của Java (h10.21 : iCoot chạy trên nền applet swing).

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON



Đối với giao diện HTML/CGI, chúng ta có thể chỉ mô phỏng một notebooks – kết quả cuối cùng là như nhau, nhưng người dùng phải chịu đựng sự trì hoãn lâu hơn khi chuyển đến trang mới (h10.22 cho thấy iCoot chạy như một mô phỏng notebook HTML/CGI)



H10.22: iCoot chạy như một mô phỏng notebook HTML/CGI

Thông thường, một usecase sẽ ánh xạ chính xác tới một trang duy nhất. Thậm chí một usecase phức tạp cũng có thể được thể hiện trong một panel duy nhất.

Sử dụng wizards

Wizard là chuỗi các trang hướng dẫn người dùng, từng bước một để thực hiện một hành động phức tạp. Khi người dùng chọn một hành động, chúng được trình bày trong một trang đại diện cho bước 1, người dùng hoàn thành bước 1 và click chuột vào nút Next để tiếp tục bước 2... Trước khi hoàn tất một hoạt động, người dùng có thể vào lại các bước trước đó bằng cách sử dụng nút Back. Mỗi trang của wizard bao gồm các trường, list mà người dùng sử dụng để nhập dữ liệu cần thiết cho bước hiện tại. Bình thường, mỗi trang

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

sẽ có hướng dẫn đơn giản, ví dụ như “nhập mật khẩu cũ của bạn vào ô dưới đây rồi chọn Next ...”

Việc thực hiện từng bước với những hướng dẫn đơn giản của wizard cho phép người dùng thực thi một hành động phức tạp mà không cần nhớ làm thế nào. Ngoài ra, một wizard, như một notebook, giúp người dùng tập trung vào một vị trí trên màn hình. Cũng như với notebooks, nếu chúng ta sử dụng giao diện HTML/CGI thuần, chúng ta có thể chỉ mô phỏng wizard, tuy nhiên, kết quả cuối cùng có thể tốt hơn nếu chúng ta không sử dụng chúng cho tất cả các usecases. (Trong trường hợp này, sự thay thế là một trang HTML rất dài, bao gồm tất cả các bước của một hành động : bất tiện cho người dùng khi lần lượt chuyển qua các bước)

Tránh nhiều cửa sổ

Nhiều cửa sổ, cửa sổ chồng được tạo ra vì lợi ích của các máy tính hơn là vì những người dùng mới. Thậm chí nếu người dùng đủ kiến thức máy tính để mở một trình duyệt web, chúng ta cũng không nên mong đợi họ có thể đối phó với nhiều cửa sổ trình duyệt hoặc các hộp thoại pop – up. Trong iCoot, người dùng hướng tới một chiếc xe đặc biệt, và click vào nút “make a reservation” để đặt trước. Khi có một liên kết khả thi với việc đặt một mô hình xe hơi, hệ thống cần yêu cầu người dùng xác nhận. Điều tự nhiên đối với một người phát triển hệ thống đa cửa sổ là mở ra một hộp thoại. Tuy nhiên, iCoot có thể được chạy trên một TV (với sự trợ giúp của một hộp set – top) Do giới hạn của độ phân giải và kích thước màn hình, hộp set – top không hỗ trợ các hộp thoại, vì vậy không cho phép các message được hiển thị (h10.23).



Wrong

Hình 10.23: Các vấn đề với nhiều cửa sổ

Có thể tránh sử dụng nhiều cửa sổ (như trong hình 10.24) bằng cách xếp chồng lên trang trước nội dung của hộp thoại. Nếu người dùng click vào “No”, hệ thống sẽ trả

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

về trang trước đó (tương đương với việc đóng hộp thoại). Nếu người dùng click vào “Yes”, việc đặt hàng được thực hiện và hệ thống trở về trang trước.



Hình 10.24: Tránh nhiều cửa sổ

Giao diện hiển thị trong hình 10.24 là một wizard chạy trong một notebook : nút xuống bên trái màn hình tương ứng với các thẻ tab của notebook cho phép người dùng chuyển đổi giữa các usecases chính; trong trang notebook, chúng ta thấy một hướng dẫn wizard cho người dùng thông qua U7 : usecase “Make Reservation” (là phần mở rộng của U1 : “Browse the Index”)

Vì vậy, notebooks và wizard có thể được dùng cùng nhau để nhóm các tương tác phức tạp vào cùng một vị trí duy nhất trên màn hình. Do đó, người dùng không thấy một mồi hay nhầm lẫn. Chúng ta chọn quy tắc “một thời điểm chỉ làm một việc”, thậm chí khi chúng ta thiết kế các giao diện người dùng theo quy tắc này, chúng ta vẫn có thể cho phép người dùng chuyển đổi giữa các hành động dở chừng. Ví dụ : trong hình 10.24, người dùng có thể bỏ qua câu hỏi hiện tại và chuyển sang trang “đổi mã pin”. Khi người dùng trở lại trang Index, câu hỏi tương tự sẽ chờ được trả lời.

8.5 THIẾT KẾ CÁC DỊCH VỤ NGHIỆP VỤ

Khi chúng ta quyết định chọn giao diện người dùng và giao diện của tầng nghiệp vụ (*business layer*), chúng ta có thể thiết kế tầng server: bao gồm những đối tượng server, kết hợp với những đối tượng trong tầng nghiệp vụ, cung cấp một giao diện đơn giản có lợi cho client.

Business service là những truy vấn và lệnh mà tầng giữa (*middle tier*) cung cấp cho client của nó. Ví dụ, với iCoot, chúng ta có thể đưa ra tập các business service sau:

- Đọc tiêu đề chỉ mục từ *Catalog*.
- Đọc *CarModel* từ một tiêu đề chỉ mục được đưa ra.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

- Đọc mọi *CarModel Category*.
- Đọc tất cả kích thước động cơ *CarModel*.
- Đọc tất cả *Make* của *CarModel*.
- Đọc *CarModel* cho một tập *Category* được đưa ra, kích thước động cơ và *Make*.
- Đọc chi tiết của một *Member* được đưa ra.
- Thay đổi password của một *Member*
- Đọc những *Car* được thuê từ một *Member* được đưa ra.
- Đọc những danh sách đăng ký *Reservation* của một *Member* được đưa ra.
- Hủy một *Reservation*

Business service đưa ra một bản tóm tắt ngắn gọn về luồng thông tin giữa giao diện người dùng (user interface) bên phía client và logic nghiệp vụ tại tầng giữa (*middle tier*). Business service có thể được suy ra từ các usecase, đồng thời có xem xét đến cả thiết kế giao diện người dùng và kiến trúc hệ thống. (ý tưởng chia interface của server thành các dịch vụ mà nó cung cấp có thể được áp dụng cho bất kì loại giao tiếp nào, nghĩa là chúng ta có thể mở rộng những ý kiến đã được thảo luận trong phần này thành vấn đề làm thế nào để các lớp trong một hệ thống con có thể thông tin được với nhau).

Để những business service phù hợp với interface của HTML/CGI, chúng ta phải thừa nhận rằng giao diện người dùng được trải ra giữa tầng client và tầng giữa: sự kết hợp giữa những mẫu HTML, servlet, và trang kết quả JSP bao gồm một giao diện người dùng đơn, mặc dù thực tế là một vài phần chạy trên tầng giữa. Chúng ta thậm chí có thể thực thi những servlet trên một máy thuộc tầng giữa khác không thực thi các tiến trình nghiệp vụ, để làm giảm việc tải các đối tượng nghiệp vụ - trong trường hợp này, máy servlet là một client của tầng giữa hay đúng hơn là một phần của tầng giữa. Vì thế, trong phần này, client có thể là:

- Một đoạn code, ví dụ là một applet, chạy trên một máy riêng biệt với tầng nghiệp vụ.
- Những servlet chạy trong một tiến trình riêng biệt hay trên một máy riêng biệt với tầng nghiệp vụ.

Cài đặt giao tiếp client-server theo khuôn mẫu business service cho phép chúng ta:

- Đơn giản hóa code bên phía client. Bởi vì hầu hết giao diện bên phía client chỉ cần một tập con các khả năng của hệ thống, mỗi cái có thể được đưa ra một khung nhìn rút gọn.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

- Phức tạp hóa tầng nghiệp vụ. Chúng ta có thể làm cho tầng nghiệp vụ phức tạp, mạnh mẽ hay hỗ trợ việc sử dụng lại tùy theo ý muốn của chúng ta, hay thậm chí là thay đổi hoàn toàn, mà không cần lo lắng về việc ảnh hưởng đến client.
- Xây dựng một tầng server có thể cắm được. Chúng ta thích cung cấp nhiều loại giao diện để phù hợp với cài đặt của từng người dùng, với khả năng của từng client (applet/RMI hay HTML/CGI hay servlet), nghĩa là không cần lúc nào cũng phải cài đặt lại những dịch vụ tầng giữa.

8.5.1 Sử dụng proxy và copy

Khi một client yêu cầu một business service, kết quả có thể bao gồm một hay nhiều đối tượng nghiệp vụ. Ví dụ, nếu yêu cầu của một client là “Đọc tất cả đối tượng *Make* của *CarModel*”, kết quả sẽ là một danh sách các nhãn hiệu (*make*). Cũng như khi gọi những đối tượng nghiệp vụ, một client có thể gửi những đối tượng nghiệp vụ đến server. Ví dụ, nếu yêu cầu của client là “Hủy bỏ một Reservation”, đơn đặt chỗ cần phải được chỉ rõ.

Vì thế, theo logic, các đối tượng nghiệp vụ phải truyền qua lại giữa client và server. Nếu chúng ta làm việc với một hệ thống một tầng, đó không phải là vấn đề: chúng ta chỉ cần truyền con trỏ đối tượng bên trong hệ thống chạy của chúng ta và truyền tin báo đến đối tượng ứng với những điều chúng ta cần. Tuy nhiên, khi chúng ta làm việc với một mạng, những điều ấy không hề đơn giản.

Với sự giúp đỡ của các công nghệ như RMI, chúng ta có thể sắp xếp các tin báo truyền giữa các đối tượng qua mạng: mỗi đối tượng được chứa trong một hệ thống chạy đơn, nhưng tin báo có thể đến từ bên ngoài. Hoặc chúng ta có thể truyền bản sao của đối tượng: mỗi hệ thống có một bản sao những đối tượng mà nó cần và gửi tin báo đến các đối tượng đó như bình thường. Trong trường hợp giao tiếp giữa các tầng trong một hệ thống con, chúng ta cũng lựa chọn tương tự: chúng ta có thể truyền tham chiếu của đối tượng giữa các tầng hoặc là bản sao của đối tượng? (chúng ta sẽ truyền tham chiếu qua biên của một tầng mở và truyền bản sao qua một tầng đóng).

Chính xác hơn, chúng ta có hai lựa chọn: **proxies** là những đối tượng client mà biết làm thế nào để truyền tin báo chúng nhận được đến một đối tượng nghiệp vụ thực ở một nơi nào đó trong hệ thống. **Copies** là những đối tượng client chứa một bản sao dữ liệu của một đối tượng nghiệp vụ thực. Những thuận lợi khi sử dụng **proxies** là:

- Tất cả client nhìn các đối tượng là giống nhau, vì thế chúng luôn làm việc với cùng một thông tin.
- Tất cả hệ thống kết hợp vào một không gian đơn nhất: những đối tượng phân tán được giải quyết theo cách thức giống như những đối tượng cục bộ.

Những bất lợi của **proxies** là:

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

- Những đối tượng tầng nghiệp vụ phải được đảm bảo. Ví dụ, chúng ta phải đảm bảo rằng khi chúng ta nhận được một tin báo từ client thì client phải được phép gửi tin báo. Điều này làm phức tạp tầng nghiệp vụ. (mô hình bảo mật khác, trong đó những đối tượng nghiệp vụ chỉ được truy cập bởi tầng server và tầng server chỉ phải đảm bảo cho tầng nghiệp vụ của nó, dễ dàng cài đặt hơn nhiều).
- Trao đổi qua mạng tầng lên. Trong một hệ thống hướng đối tượng, chúng ta thường tìm nơi có chứa đối tượng và gửi cho nó tin báo. Với kĩ thuật **proxy**, việc tìm kiếm đối tượng nhanh nhưng quá trình gửi tin báo đến đối tượng thường chậm (bởi vì mỗi tin báo phải đi qua mạng).
- Gánh nặng đặt vào tầng giữa tầng lên: tầng nghiệp vụ, chạy ở tầng giữa, thực thi tất cả những phương thức trong suốt thời gian chạy.

Những thuận lợi của việc sử dụng **copy**:

- Giúp làm giảm lưu lượng trao đổi qua mạng: chúng ta nhận được nhiều thông tin hơn khi bắt đầu (tất cả dữ liệu của đối tượng) nhưng phương thức được thực thi nhanh hơn (bởi vì nó chạy ngay tại client).
- Không phải tất cả mọi tiến trình đều thực hiện ở tầng giữa, bởi vì một vài phương thức nghiệp vụ được thực thi trên client.
- Truy cập trực tiếp vào những phương thức nghiệp vụ không cần phải đảm bảo an toàn bởi vì chỉ có tầng server là được truy cập trực tiếp.
- Việc truy cập vào đối tượng không cần phải chú ý đến việc truy cập đồng thời, bởi vì khi một bản sao được tạo ra bên phía client thì chỉ có mỗi mình client là được truy cập vào đối tượng ấy.

Những bất lợi khi sử dụng **copy** là:

- Có quá nhiều dữ liệu được sao chép. Ví dụ, nếu chúng ta sao chép một *CarModel*, chúng ta có thể phải sao chép toàn bộ đối tượng *CarModelDetail*, *Make*, *Reservation*, *Member* của nó. Đó là những đối tượng được dùng để tạo ra một đơn đặt hàng.
- Bản sao có thể bị sai lệch khi bản chính ở server bị thay đổi bởi vì các bản sao ở các client là độc lập với bản chính ở server và chỉ được sao chép khi bắt đầu.

Với một vài ứng dụng, các bản sao chép ở client có thể bị lỗi thời. Ví dụ, nếu một người dùng đưa ra một yêu cầu cho công cụ tìm kiếm từ một trình duyệt web, trang hiển thị kết quả sẽ bị lỗi thời khi trang web được thêm vào, cập nhật hay bị xóa bỏ. Người dùng phải chấp nhận rằng kết quả trả về chỉ chính xác tại thời điểm truy vấn được thực thi.

Bằng lập trình một cách cẩn thận, chúng ta có thể làm giảm một vài bất lợi của **proxy** và **copy**. Ví dụ, khi sử dụng proxy, chúng ta có thể lưu tạm một vài dữ liệu của đối

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

tượng để làm giảm lưu lượng truyền tải qua mạng, tuy nhiên cần phải đảm bảo rằng dữ liệu được lưu không bị sai lệch. Với phương pháp **copy**, chúng ta sẽ chỉ sao lưu những đối tượng được yêu cầu, bằng cách đó có thể làm giảm lượng dữ liệu cần truyền. Ví dụ, khi chúng ta đọc một *CarModel*, client có thể được nhận chỉ những thuộc tính quan trọng nhất – bất kì những đối tượng được tham chiếu sẽ có giá trị *null*, chúng chỉ được tải khi client thao tác thực sự trên chúng (bằng cách sử dụng đóng gói, tiến trình này có thể được vô hình với mã bên client).

Nếu chúng ta may mắn, chúng ta sẽ có quyền truy cập vào framework hay thư viện hỗ trợ việc cấu hình proxy, cấu hình copy và hỗn hợp của cả hai. Tuy nhiên, dù với công nghệ nào, người phát triển cũng cần phải lựa chọn kỹ thuật muôn sử dụng.

Khi chúng ta chấp nhận việc làm bằng tay, chúng ta có thể sử dụng phương pháp: **lightweight copy**. Với phương pháp này, khi client yêu cầu một đối tượng nghiệp vụ, nó sẽ chỉ được nhận những thông tin quan trọng mà nó cần về đối tượng đó, và thông tin sẽ không được truyền cho client nếu đã có từ trước. Nói cách khác, khi một client cần xác định một đối tượng nghiệp vụ với server, client chỉ cần truyền thông tin tổng quát đến máy chủ.

Để lightweight copy có thể làm việc được, chúng ta cần dịch vụ nghiệp vụ (để có thể đưa ra những thông tin mà client cần) và thông tin tổng quát (để client có thể lấy đối tượng về từ server). Thông tin truyền đến client sẽ bao gồm bản sao đơn giản của các đối tượng (số, chuỗi...) và thông tin tổng quát cho tất cả các thuộc tính còn lại (khi client cần đến chúng).

8.5.2 Phân loại các dịch vụ nghiệp vụ

Chúng ta mong muốn các dịch vụ nghiệp tạo thành các nhóm có hành vi liên quan đến nhau, nói cách khác đó là thông điệp giữa các đối tượng. Chúng ta mong muốn các đối tượng dịch vụ này có vài thuộc tính của riêng và chúng ta sẽ không mong muốn chúng ghi các trạng thái thay mặt cho các khách hàng (mỗi khách hàng phải nhớ trạng thái của mình). Các đối tượng dịch vụ nằm trong tầng dịch vụ của một hệ thống đa tầng.

Các đối tượng dịch vụ được thiết kế cho iCoot được thể hiện như hình 1. Các dịch vụ nghiệp vụ được phân loại như đặt phòng, xác thực, thông tin thành viên, thông tin sản phẩm và cho thuê (tương ứng là ReservationsServer, AuthenticationServer, MembershipServer, CatalogServer and RentalsServer, respectively).

CatalogServer
+readCategoryNames():String[]
+readMakeNames():String[]
+readEngineSizes():int[]
+readIndexHeadings():String[]
+readCarModels(q:PCatalogQuery):PCarModel[]

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

```
+readCarModels (heading:String) : PCarModel[]  
+readCarModelDetails (carModelId:int) : PCarModelDetails
```

AuthenticationServer

```
+logon (memberShipNumber:String, password:String, steal:bo  
olean) : long  
+logoff (sessionId:long)
```

MembershipServer

```
+readMember (sessionId:long) : PMember  
+changePassword (sessionId:long, old:String, ne  
w:String)
```

RentalsServer

```
+readRentals (sessionId:long) : PCar[]
```

ReservationsServer

```
+readReservations (sessionId:long) : Preservations[]  
+createReservation (sessionId:long, movieId:int)  
+deleteReservation (sessionId:long, reservationId:int)
```

Hình 1: Server Objects for iCoot

Các thông điệp dịch vụ được thể hiện trong Hình 1 đã được điều chỉnh để bảo đảm rằng không phải khách hàng cũng không máy chủ sẽ truyền thông tin cho phía bên kia không cần thiết. Ví dụ, hãy xem xét CatalogServer. Để tìm kiếm các mẫu xe, việc đầu tiên là khách hàng phải lấy tất cả những các tên category, kích thước động cơ và tạo các tên. Những phản hồi từ máy chủ như các đối tượng *String* và đối tượng *int*.

Những khách hàng sau đó cho phép người sử dụng tạo một câu truy vấn ('Tất cả các mẫu xe' thể thao được làm bởi Alpha Rodeo hoặc Beamer', ví dụ như vậy). Máy khách gửi truy vấn đến máy chủ như là một *PCatalogQuery*. Bên trong các *PCatalogQuery* có ba thuộc tính mảng để xác định các tên category, kích thước động cơ và tạo tên mà người sử dụng thích. Máy chủ trả về một mảng các đối tượng *PCarModel* đã được matching. (P là viết tắt của giao thức này - nó sẽ giúp các nhà phát triển các đối tượng máy chủ để tránh bất kỳ tên nào xung đột với các đối tượng dịch vụ quan trọng. Mảng đã được sử dụng bởi vì chúng nhỏ gọn hơn so với collections.)

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Để giảm bớt lượng thông tin trả lại cho khách hàng, các đối tượng *PCarModel* không chứa bất kỳ chi tiết, chỉ cần giá, số mẫu và định danh tổng quát. Khi người dùng yêu cầu chi tiết của một mô hình xe cụ thể, chúng ta không phải thông qua toàn bộ *PCarModel*, cái mà các thuộc tính của máy chủ đã phải có, khách hàng thông qua chỉ là định danh tổng quát, như là một tham số để thông báo *readCarModelDetails*.

8.5.3 Xác nhận các phiên

Thông thường, để vô hiệu hóa các hacker, chúng ta cần hạn chế khách hàng truy cập vào các dịch vụ đặc quyền. Ví dụ, cho iCoot, chúng ta có các dịch vụ đặc quyền, chẳng hạn như '*Reserve a CarModel*', yêu cầu khách hàng phải đăng nhập vào và dịch vụ không đặc quyền, chẳng hạn như '*Read the index headlings from the Catalog*', mà có sẵn cho tất cả mọi người, ngay cả các hackers.

Một số các giao thức client-server, như HTTP, có một cơ chế tiêu chuẩn cho các thách thức đăng nhập hiển thị trên một màn hình cho người dùng: tên người dùng và mật khẩu được gõ vào và được xác nhận trên máy chủ; nếu thành công, khách hàng được cấp một định danh duy nhất cho phiên đó. Tuy nhiên, nếu chúng ta muốn cung cấp các dịch vụ nghiệp vụ Pluggable, có nghĩa là các dịch vụ sẽ làm việc với bất kỳ loại nào từ đầu đến cuối, chúng ta đã cài đặt một cơ chế cho chính mình. Hình 1 cho thấy một cách để làm điều này. Tất cả những dịch vụ đặc quyền đưa ra một số miễn là tham số đầu tiên của chúng: điều này phải là định danh cho một phiên được tạo bởi lớp dịch vụ, hoặc sử dụng các dịch vụ đặc quyền sẽ thất bại. Để giữ một định danh phiên, khách hàng phải sử dụng các thông điệp đăng nhập trên *AuthenticationServer*. (Việc lấy cấp các thông số được sử dụng bởi khách hàng để xác định xem bất kỳ phiên hiện tại cho thành viên phải được chấm dứt - đây là một phần của cơ chế single-log-on).

Khi phương thức đăng nhập được gọi, *AuthenticationServer* kiểm tra số lượng và mật khẩu thành viên bằng cách sử dụng tầng nghiệp vụ. Nếu các thông tin của khách hàng là chính xác, *AuthenticationServer* tạo ra một số ngẫu nhiên kết hợp nó với Member tương ứng ở tầng nghiệp vụ. Sau đó, bất cứ khi nào một dịch vụ đặc quyền được gọi, các đối tượng dịch vụ liên quan có thể sử dụng để định danh phiên tìm kiếm trên Member trước khi tiếp tục quá trình. đương nhiên, nếu việc chứng thực bản gốc hoặc định danh phiên là không hợp lệ, khách hàng nhận được một thông báo lỗi. Các định danh của phiên phải khó khăn để làm giả - một cách ngẫu nhiên tạo ra số có độ dài 64-bit là đủ. Các hacker thường không bận tâm cố gắng để đoán ra số ngẫu nhiên có độ dài 64bit bởi vì cơ hội của họ thành công là rất nhỏ.

Một cách khác của việc cung cấp một dịch vụ cơ động đặc quyền/-không đặc quyền có cơ chế cho việc tạo ra định danh cho một phiên của đối tượng. Thông qua đóng gói, điều này sẽ giúp cho chúng ta thêm linh hoạt trong các loại khó khăn để giả mạo thông tin mà chúng ta đã chọn để sử dụng.

8.5.4 Hiện thực hóa dịch vụ nghiệp vụ

Và bây giờ chúng ta thiết kế tầng nghiệp vụ (business service) (những thông điệp từ những đối tượng dịch vụ). Chúng ta cần phải giải quyết vấn đề cung ta sẽ cài đặt tầng nghiệp vụ thế nào. Để làm điều này, chúng ta cần phải xem lại biểu đồ use case, biểu đồ tuần tự cái mà chỉ cho chúng ta thấy các thông điệp cần được gửi đi. Để có một tên gọi hợp lý hơn, chúng tôi gọi công việc này là: **hiện thực quá tầng dịch vụ nghiệp vụ**. Nó giống như là việc hiện thực hóa usecase chúng ta làm trong suốt quá trình phân tích, thông qua biểu đồ usecase và biểu đồ hợp tác để tập trung vào các đối tượng nghiệp vụ sẽ hỗ trợ một việc cài đặt nào đó?. Chúng ta nên dùng biểu đồ tuần tự hơn là biểu đồ hợp tác trong tài liệu này bởi vì biểu đồ tuần tự thì mô tả chi tiết hơn (chúng ta đang đặc tả về việc cài đặt, vì vậy chúng ta cần nhiều thông tin để hiển thị hơn)

Hình 10.24 chỉ ra một biểu đồ tuần tự cho phương thức đăng xuất của AuthenticationSever. Bên trên đó là các đối tượng được gọi trong quá trình tương tác. Không giống như các đối tượng trong biểu đồ giao tiếp, các đối tượng chỉ ra trong biểu đồ tuần tự không phải gạch chân. Luồng thời gian được biểu diễn từ trên xuống dưới, vì vậy, bắt đầu từ đỉnh, tác nhân Member chỉ ra việc bắt đầu log-off với AuthenticationServlet, cái mà gửi thông điệp logoff tới AuthenticationServer, và gửi thông điệp findById tới MemberHome và tiếp tục.

Cái bảng đường hình dọc trong biểu đồ tuần tự được gọi đường thời gian thực. Những hình chữ nhật dọc xám được gọi là những thanh hoạt động, nó chỉ ra khi nào một đối tượng đang thực thi một phương thức nào đó. (Những thanh này có thể trắng hoặc bỗ qua trong biểu đồ vẽ tay). Mỗi biểu đồ có thể được đóng trong một khung - một hộp với một toán hạng ở góc trên cùng bên trái, giống như một thứ mà chúng ta đã nhìn thấy sớm

Cho biểu đồ thiết kế các lớp của chúng ta. (Tất cả các biểu đồ ML có thể được đóng trong một khung, toán hạng chỉ ra nội dung của biểu đồ đó). Trong trường hợp sử dụng một biểu đồ tuần tự này, toán tử là một ds theo tên biểu đồ tuần tự. Cho một biểu đồ lớp, chúng ta sử dụng toán tử pkg để chỉ ra các gói chứa những lớp này. Trong quyển sách này, những khung này chỉ là nơi bao quanh biểu đồ tuần tự và các bậc thiết kế lớp từ iCoot. Các khung này có thể mô tả một vòng lặp hoặc tham chiếu đến một biểu đồ tuần tự khác.

Cho việc hiện thực hóa dịch vụ nghiệp vụ, chúng ta cần phải chỉ ra các phương thức cái mà sử dụng giữa tầng đối tượng dịch vụ và tầng đối tượng nghiệp vụ, nhưng chúng ta không cần phải chỉ ra bên trong công việc của tầng đối tượng nghiệp vụ. Phụ thuộc vào kích thước, độ lớn của chương trình, nó có thể hoặc không khả thi khi chỉ ra toàn bộ mỗi đối tượng nghiệp vụ trong biểu đồ tuần tự. Thông thường, chúng ta mong đợi chỉ ra những kịch bản quan trọng nhất. Usecase sẽ xác định cho chúng ta quyết định cái nào quan trọng nhất. Bên trong mỗi usecase, cho thể có những kịch bản bình thường

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

và một vài kịch bản tương tự nhau: vì mục đích của chúng ta, nó đủ để chỉ ra hết những phần cơ bản cho hầu hết các usecase thông dụng.

Khi chúng ta vẽ biểu đồ tuần tự, chúng ta sẽ phát hiện càng ngày càng có nhiều phương thức trong giao diện của tầng đối tượng nghiệp vụ. Kết quả phương thức của tầng đối tượng nghiệp vụ được phản hồi qua pha đặc tả, để hỗ trợ hơn cho việc này, chúng ta sử dụng với trí giác, kinh nghiệm, thư viện, mẫu hay một framework. Trong pha đặc tả, chúng ta cần hoàn thiện giao diện đối tượng và miêu tả những hành vi yêu cầu của đối tượng.

8.6 SỬ DỤNG FRAMEWORK, KHUÔN MẪU, VÀ THƯ VIỆN

Một khuôn mẫu là một giải pháp giải quyết gọn nhẹ cho những vấn đề lập trình nhỏ trong mô hình hướng đối tượng, là một nhóm cộng tác giữa các đối tượng. Khuôn mẫu cho phép người phát triển thực hiện công việc nhanh hơn và chất lượng mã nguồn tốt hơn, tránh bị trùng lặp. Một vài khuôn mẫu cơ sở được đề cập tại một phần khác trong quyển sách này: singletons(nơi mà một lớp chỉ có một thể hiện); factories (cho việc tạo những đối tượng); homes(cho việc tạo các đối tượng và tìm một đối tượng khác) và states(đại diện cho vòng đời của một đối tượng). Mỗi một khuôn mẫu đều có tên của nó, một vài dòng miêu tả và ví dụ để sử dụng nó. Mỗi một người phát triển cần phải làm quen với những khuôn mẫu thông dụng.

Trong một vài trường hợp, một framework như là một khuôn mẫu; nó là cái cách để đặt các thành phần của một hệ thống cùng nhau. Tuy nhiên, có 2 vấn đề khác biệt đó là; đầu tiên nó có thể phát triển lớn hơn, thứ 2 là một vài mã nguồn đã được viết sẵn cho người sử dụng(dùng công cụ để sinh mã hoặc viết theo một khuôn dạng được định nghĩa sẵn).

Giống như pattern, người sử dụng cần phải tìm kiếm một framework nào có thể được thiết kế để giải quyết vấn đề của bạn, vì vậy người sử dụng cần tránh viết mã nguồn không thật sự cần thiết. Một trong những framework toàn diện và phổ biến đó là Enterprise Beans. EJB framework cho phép người phát triển đối tượng nghiệp vụ logic tầng giữa mà không phải viết một dòng mã ngòn nào để xử lý persistence, giao dịch, bảo mật, chống giao tranh, phân tán và đa luồng. Mặc dù nó là một framework lớn nhưng nó rất hay để học tập.

Một thư viện là một tập các lớp được viết trước mà được cửa dụng như cái làm trước. Bạn có thể giải quyết vấn đề lĩnh vực của bạn bằng việc sử dụng thư viện và tránh xa việc viết lại các hàm, các chức năng phù hợp đã được xây dựng. Thư viện Java2 Platform là một ví dụ, Nó có 3 phiên bản Cho doanh nghiệp, Standard và Micro dành cho những nghiệp vụ khác nhau : hệ thống lớn (thương mại điện tử và hệ thống phân tán..) hay những hệ thống vừa phải (desktop) hay là những hệ thống nhỏ như điện thoại di động hay PDA

8.7 GIAO DỊCH

8.7.1 Định nghĩa giao dịch

Mỗi hệ thống hiện nay, nếu nó ngày càng lớn thì phải xây dựng các transaction. Một transaction được biết đến như một đơn vị công việc, nó sử dụng để ngăn một số truy cập vào database. Một ứng dụng doanh nghiệp truy cập và lưu trữ thông tin trong một hoặc nhiều database. Vì những thông tin này quyết định đến hoạt động kinh doanh của doanh nghiệp nên nó phải tin cậy và đúng đắn. Tính toàn vẹn dữ liệu sẽ mất nếu nhiều chương trình cho phép update những thông tin giống nhau đồng thời. Nó cũng sẽ mất đi nếu hệ thống xảy ra lỗi trong khi đang xử lý một số transaction, dữ liệu không được cập nhập hoàn chỉnh. Để ngăn cản vấn đề này software transaction đảm bảo tính toàn vẹn dữ liệu.

Transaction được sử dụng để đảm bảo:

- Thông tin trong database không bị sai lạc do các vấn đề hệ thống. Chúng ta muốn chắc chắn rằng khi database di chuyển từ trạng thái này sang trạng thái khác dữ liệu không bao giờ được update một phần, nó chỉ có thể update toàn bộ hoặc không update gì cả.
- Client không lấy phải các thông tin cũ. Chúng ta muốn tránh xác tình huống như: Client A đọc địa chỉ của một khách hàng, client B sửa địa chỉ của khách hàng đó, Client A làm một số action dựa trên địa chỉ cũ.

Một số khái niệm transaction

- Begin transaction: Bắt đầu transaction.
- Commit transaction: hoàn tất transaction.
- Rollback transaction: quay lui transaction(những pending updates bị loại bỏ do đó dữ liệu cơ sở trong thông tin cũ sẽ không bị thay đổi).
- Short transaction: transaction có tác dụng đối với một hàng.
- Long transaction: transaction có tác dụng đối với một vài hàng có quan hệ.

Mô tả Transaction

- 1 database client bắt đầu một transaction, truy cập dữ liệu và sau đó hoàn tất transaction.
- Nếu sự xác nhận thành công, tất cả updates được tạo ra từ lúc bắt đầu transaction được đưa vào database và client có thể đảm bảo rằng nó làm việc trên thông tin mới.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

- Nếu commit bị fail do một vấn đề hệ thống hoặc do xung đột với các transaction khác, thì client có thể roll back transaction, loại bỏ các pending update
- Hệ quản trị CSDL đảm bảo rằng các sự truy cập trong 1 transaction tất cả là thành công hoặc tất cả thất bại (fail), không có update phần nào; 1 transaction có thể kết thúc theo 2 cách: với một commit hoặc với một rollback. CSDL sẽ chuyển trạng thái toàn vẹn này sang trạng thái toàn vẹn khác.

Mô phỏng một giao dịch nghiệp vụ

Một chương trình tài chính, ví dụ : chuyển tiền từ một checking account sang một saving account sử dụng các bước liệt kê ở đoạn mã giả dưới đây:

```
Begin transaction  
Debit cheking account.  
Credit saving account.  
Update history log  
Commit transaction
```

Tất cả 3 bước trên đều phải hoàn thành. Nếu một disk drive bị hỏng trong bước debit, transaction sẽ rollback và loại bỏ những dữ liệu được sửa trong debit statement. Mặc dù transaction bị fail, toàn vẹn dữ liệu sẽ không bị thay đổi.

8.7.2 Đồng thời

Điều khiển đồng thời (việc sử dụng các transaction để điều khiển truy cập đồng thời vào dữ liệu từ nhiều client) có thể là pessimistic hoặc optimistic. Với pessimistic concurrency, DBMS đảm bảo rằng không có transactions khác có thể thực hiện truy cập xung đột trong khi một transaction active. Với optimistic concurrency, transactions truy cập được tắt cả nhưng khi một transaction committed, DBMS kiểm tra rằng không có transactions nào thực hiện các truy cập xung đột.

Mặc định, các relational database sử dụng pessimistic concurrency. Optimistic concurrency thường được dùng nhiều trong object-oriented databases, mặc dù đôi khi nó được cung cấp như là một lựa chọn trong top 10 của object-oriented frameworks. (nói một cách khác để cài đặt optimistic concurrency cho một relation database là đọc lại 1 hàng trước khi update nó: nếu dữ liệu trong hàng có bị thay đổi, chúng ta biết rằng một transaction khác đã sửa nó; tuy nhiên điều này chỉ là giải pháp một phần).

Để hiện thực pessimistic concurrency, 1 relation DBMS lock (khóa) dữ liệu được truy cập trong 1 transaction cho đến khi transaction đó kết thúc. Ví dụ: nếu 1 transaction sửa 1 số dư tài khoản số 123, DBMS khóa hàng chứa số dư tài khoản mới lại; transactions khác bị chặn không được đọc hàng cho đến khi transaction đầu tiên kia được hoàn thành (và khóa được bỏ). Locks còn ứng dụng để đọc: ví dụ nếu 1 transaction đọc 1

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

số dư tài khoản của tài khoản số 456, transactions khác sẽ không được phép sửa hàng chửa dữ liệu cũ cho đến khi transaction đầu tiên kết thúc. (các trường hợp còn lại, transaction đầu tiên phải là một nửa công việc của nó liên quan đến dữ liệu cũ và một nửa công việc của nó liên quan đến dữ liệu mới,...) .

8.7.3 Sử dụng giao dịch với đối tượng

Khi ánh xạ từ 1 mô hình đối tượng đến một cơ sở dữ liệu quan hệ, chúng ta gặp phải một mâu thuẫn đáng quan tâm. Trong một xử lý, chúng ta có 2 biểu đồ đối tượng phức tạp, toàn bộ các chunks của nó chúng ta muốn xử lý trong một transaction đơn dài. Ở một cách xử lý khác, chúng ta có một relational database trải rộng trên nhiều bảng với một pessimistic locking schema chúng ta nên sử dụng short transactions, để tránh khoá một vùng lớn database. (Điều này giải thích tại sao optimistic concurrency là phổ biến với object- oriented database và object- oriented frameworks.). Nếu may mắn, bạn sẽ sử dụng 1 framework như là EJB (giải quyết hầu hết các vấn đề cho bạn). Nếu bạn ánh xạ bằng tay thì chú ý các lời khuyên sau:

- Tổ chức các đối tượng và các đường dẫn truy cập giảm sự chồng chéo.
- Sử dụng primary keys để truy cập database đúng trọng tâm. Vấn đề này với hầu hết DBMS đó là chúng khóa một bảng thực thể nếu chúng không thể đảm bảo rằng các hàng đó sẽ được truy cập. Ví dụ, nếu ID là primary key trong bảng Customer , 1 câu lệnh thực thi như: “Lấy cho tôi tên của customer 789” sẽ khóa một hàng có ID = 789; 1 câu lệnh không chứa khoá chính như là: “Lấy cho tôi các Customer với họ là “Bloggs” ” phải khóa toàn bộ bảng. Do đó, tạo ra sự đảm bảo rằng bạn chọn 1 primary key cho mỗi thực thể và nói cho database biết về nó khi đó database cơ sở của bạn truy cập vào các primary keys bất cứ khi nào có thể.
- Giữ các transactions ngắn. Bạn có thể muốn điều khiển sự bắt đầu và kết thúc của một transaction, cho nên bạn có thể tạo một vài đối tượng truy cập 1 lần, không làm quá sức.

8.7.4 Giao dịch ở tầng trên

Các giao dịch có một ảnh hưởng nối tiếp: Thực tế chúng ở trong tầng CSDL và thường thấy rõ trong tầng persistence layer; một khi chúng rõ ràng trong tầng persistence layer, chúng thường trở nên rõ ràng trong tầng nghiệp vụ; và một khi chúng rõ ràng trong tầng nghiệp vụ, những người phát triển phải hiểu chúng và sử dụng chúng như thế nào cho đúng đắn.

8.8 XỬ LÝ ĐA NHIỆM

Thông thường, khi sử dụng máy tính, người dùng luôn muốn có thể thực hiện đồng thời nhiều việc khác nhau, ví dụ như vừa viết mail, vừa đọc mail, hay lướt web. Điều đó cũng

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

đòi hỏi 1 server có thể thực hiện đồng thời nhiều việc (đáp ứng nhiều request từ các client khác nhau). Do vậy, hầu hết các hệ điều hành đều cho phép chạy đa nhiệm - **multitasking**: mỗi chương trình chạy được coi như 1 tiến trình – **process** độc lập với phần code và dữ liệu được bảo vệ của riêng nó.

Một số ngôn ngữ lập trình cho phép chúng ta thực hiện nhiều tác vụ nhỏ trong một tiến trình đơn. Các tác vụ này thường được gọi là các tiểu trình (thread). Mỗi luồng thực thi 1 hoạt động trong chương trình, và nó chạy song song với các hoạt động khác.

8.8.1 Quản lý đa nhiệm

Mỗi tiến trình được quản lý bởi hệ điều hành có thể ở trong trạng thái *idle* – tạm ngưng (ví dụ để đợi người dùng nhập dữ liệu) hoặc *active* – hoạt động (thực hiện các thao tác tính toán). Vì số tiến trình thường nhiều hơn số CPU, hệ điều hành sẽ phải chia sẻ thời gian CPU giữa các tiến trình đang hoạt động: hệ điều hành cho phép mỗi tiến trình chạy trong vòng 1 thời gian ngắn sau đó chuyển qua cho tiến trình kế tiếp. Việc phân lát thời gian (**time-slicing**) này được điều khiển bởi một đoạn phần mềm gọi là **scheduler** – chương trình lập lịch. Chúng ta không cần phải biết chi tiết về thuật toán và scheduler sử dụng để phân bổ thời gian giữa các tiến trình, chỉ cần xem như mỗi tiến trình đều được chia sẻ công bằng. Để thuận tiện hơn, hầu hết các hệ điều hành đều cho phép chúng ta gán **priority** – mức ưu tiên cho mỗi tiến trình, qua đó 1 số tiến trình sẽ nhận được nhiều thời gian hơn các tiến trình khác. Ví dụ, chúng ta có thể gán mức ưu tiên cao cho việc nhận biết hoặc động click chuột và mức ưu tiên thấp hơn cho các ứng dụng người dùng.

Mặc dù các hệ điều hành chuẩn đều có thể tránh việc các tiến trình không truy nhập và mã nguồn và dữ liệu của nhau, nhưng việc của người lập trình vẫn là đảm bảo rằng việc truy nhập các tài nguyên bên ngoài (như các tệp và cơ sở dữ liệu) phải được quản lý đúng đắn. Ví dụ, khi một trình duyệt word mở 1 tệp, nó có thể **lock** - khóa tệp đó lại để tránh việc chỉnh sửa tệp ở cùng thời điểm; việc truy nhập đồng thời vào 1 tài nguyên có tính mở cao như cơ sở dữ liệu thường được điều khiển thông qua việc kết hợp quản lý giao dịch và các quy luật nghiệp vụ.

Đối với cá nhân người sử dụng, multitasking - đa nhiệm cho phép mở nhiều ứng dụng cùng 1 lúc trên máy tính. Chúng ta có thể chuyển đổi giữa các ứng dụng ngay lập tức, chạy mỗi lúc 1 ứng dụng, hay chạy nhiều ứng dụng cùng 1 lúc, mỗi ứng dụng lại kết thúc độc lập với nhau. Đa nhiệm còn có 1 ưu điểm nữa, đó là khi chạy các tính toán đòi hỏi thời gian dài, chúng ta không cần phải chờ nó kết thúc mới có thể làm việc khác: ví dụ, khi đang đợi việc tìm kiếm trên Internet kết thúc, chúng ta có thể kiểm tra giờ hoặc nhận tin nhắn.

Đối với server, ngoài việc có thể phục vụ đồng thời nhiều client, đa nhiệm còn cho phép server phục vụ các client tốt hơn. Ví dụ, thử tưởng tượng 1 đoạn mã search.pl dùng để chạy chương trình tìm kiếm Internet dựa trên HTML/CGI. Các thao tác tìm kiếm được gọi từ một số client sẽ thực hiện rất nhanh, có thể chỉ vài mili giây, trong khi các

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

client khác lại mất đến vài giây. Nếu 1 client bắt đầu gọi thao tác tìm kiếm lâu dài và 1 client khác gọi thao tác tìm kiếm đơn giản hơn, thì thao tác tìm kiếm thứ hai sẽ được thực hiện ngay lập tức mà không cần phải đợi thao tác đầu tiên kết thúc.

8.8.2 Quản lý đa tiến trình (Controlling Multiple Threads)

Tiêu tiến trình (thread) khác với tiến trình (process) ở chỗ, chúng đều sử dụng chung 1 vùng dữ liệu bên trong tiến trình. Bởi vậy, ngoài việc phải bảo vệ tài nguyên bên ngoài, lập trình viên còn phải bảo vệ cả dữ liệu bên trong. (Mã nguồn bên trong mỗi tiến trình thường được giấu để tránh các luồng nhờ hệ thời gian chạy (run-time system), bởi vậy chúng ta không cần phải thực hiện các thao tác đặc biệt để bảo vệ nó.) Trong mọi phương diện khác, các luồng chỉ như những tiến trình nhỏ (mini-process): chúng được điều khiển bởi scheduler và chúng ta có thể gán mức ưu tiên lên chúng.

Đối với client, đa luồng có những ưu điểm sau đây:

- Người dùng có thể chạy nhiều ứng dụng cùng lúc và làm được nhiều việc với từng ứng dụng: đơn cử như ứng dụng e-mail, chúng ta có thể sửa thư, nhận thông báo khi có thư mới, xem giờ thời gian thực, và nhiều việc khác.
- Người dùng có thể tương tác với giao diện người sử dụng ngay cả khi ứng dụng đang bận. Thủ tướng tượng một công cụ truy vấn cơ sở dữ liệu khi người dùng gõ lệnh truy vấn và nhấn nút Tìm; và, trong lúc câu truy vấn đang thực hiện, người dùng phát hiện ra mình đã gõ nhầm lỗi chính tả trong câu truy vấn. Nếu công cụ truy vấn chỉ có 1 luồng chạy, người dùng không thể sửa lại câu truy vấn cho đến khi kết quả sai được trả lại và hiển thị. Mặt khác, nếu chúng ta cài đặt cho giao diện người dùng và cơ sở dữ liệu truy vấn chạy với nhiều luồng riêng biệt, người dùng có thể gọi 1 lệnh tìm kiếm khác trước khi lệnh đầu tiên kết thúc: ứng dụng có thể ngừng luồng chạy sai ngay lập tức và kết quả sai sẽ không bao giờ được hiển thị.
- Giao diện người dùng có thể được cập nhật ngay cả khi ứng dụng đang bận. Ta coi một công cụ truy vấn chạy như một luồng đơn. Nếu người dùng bắt đầu việc tìm kiếm, và trước khi hiển thị kết quả, làm thay đổi kích thước cửa sổ ứng dụng, chuyện gì sẽ xảy ra? Việc kéo góc cửa sổ và di chuyển nó dọc trên màn hình được thực hiện bởi hệ điều hành (desktop), và khung cửa sổ sẽ di chuyển theo như mong muốn. Tuy nhiên, phần bên trong cửa sổ được vẽ bởi ứng dụng tìm kiếm: vì ứng dụng này đang bận, nên phần bên trong cửa sổ sẽ không được hiển thị cho đến khi kết quả truy vấn thực hiện xong. Người sử dụng sẽ thấy một giao diện nghèo nàn không hiển thị được theo thời gian mong muốn. Nếu chúng ta sử dụng 1 luồng riêng cho thao tác truy vấn, người dùng sẽ có thể thay đổi kích thước cửa sổ trong khi đợi kết quả mà việc hiển thị vẫn diễn ra ngay lập tức.

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Đối với server, ưu điểm của đa luồng là:

- Cho phép phục vụ đồng thời nhiều client mà không cần chạy nhiều tiến trình. Chi phí bỏ ra để cài đặt, thực thi và gỡ bỏ các tiến trình đắt hơn nhiều so với các luồng. Cụ thể, một chiếc máy sẽ hỏng nếu bạn bắt nó phải chạy 1000 tiến trình đồng thời nhưng lại có thể vui vẻ thực thi 4 tiến trình với 250 luồng mỗi cái. Với một số ứng dụng mạng cụ thể, điều này là đặc biệt quan trọng.
- Làm giảm **latency** (độ trễ) của server. Ví dụ, nếu một máy tính tầng trung truy nhập vào cơ sở dữ liệu sử dụng 1 server đơn luồng, máy tính tầng trung đó sẽ phải chờ câu lệnh truy vấn được thực thi tại máy tính tầng dữ liệu. Nếu sử dụng đa luồng, máy tính tầng trung vẫn có thể làm những việc khác trong khi đợi thực hiện lệnh truy vấn.
- Làm giảm time-out. Với 1 số giao thức, lời gọi từ client sẽ bị loại tự động nếu như server không đáp lại trong 1 khoảng thời gian nhất định. Nếu tất cả các client phải đợi để được phục vụ bởi 1 server đơn luồng, sẽ nảy sinh nhiều time-out. Nếu sử dụng đa luồng, các yêu cầu ngắn sẽ được giải quyết ngay: time-out chỉ có thể xảy ra khi mạng gặp trục trặc, server quá tải hay gặp phải những yêu cầu quá phức tạp.

Về mặt lý tưởng, ngôn ngữ lập trình và hệ thống thời gian chạy của nó có thể kiểm soát được các chi tiết vụn vặt của đa luồng như lập lịch, mức độ ưu tiên, phân lát thời gian... Nhờ vậy, người lập trình chỉ cần phải viết đoạn mã để chạy cho các luồng và thực thi chúng.

8.8.3 An toàn luồng (Thread Safety)

Đa luồng cũng gây ra 1 số vấn đề, vì các luồng có thể bị gián đoạn trước khi chúng kết thúc (để cho luồng khác được chạy). Giả sử với scenario sau:

Hai luồng A và B cùng truy nhập vào đối tượng O.

- Luồng A bắt đầu đọc 1 phần của O, là F, sử dụng phương thức get.
- Khi luồng A đọc được 1 nửa giá trị, trình lập lịch ngắt nó để cho luồng B chạy.
- Luồng B bắt đầu sửa đổi F, thông qua phương thức set.
- Trình lập lịch cho phép B kết thúc quá trình sửa đổi trước khi đánh thức A.
- Khi A được đánh thức, nó tiếp tục đọc phần còn lại của F.

Luồng A lúc này đã đọc 1 nửa giá trị cũ và 1 nửa giá trị mới, điều này hiển nhiên là vô nghĩa. Dữ liệu bị hỏng cũng ảnh hưởng đến các tài nguyên bên ngoài.

Khi truy nhập dữ liệu trong 1 cơ sở dữ liệu, hệ quản trị cơ sở dữ liệu DBMS cung cấp 1 cơ chế chuyển giao phức tạp để đảm bảo rằng dữ liệu không bị ảnh hưởng. Tuy

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

nhiên, bên trong đoạn mã đa luồng, chúng ta cần phải tự bảo vệ dữ liệu của mình. Điểm mấu chốt để bảo vệ dữ liệu là 1 chương trình hướng đối tượng để đảm bảo rằng mỗi phần chỉ có thể được truy nhập qua 1 đối tượng quản lý dữ liệu. Nhờ đó, khi chắc chắn rằng chỉ có 1 luồng truy nhập vào đối tượng đó tại 1 thời điểm (loại trừ lẫn nhau) chúng ta biết dữ liệu sẽ được an toàn. Ngôn ngữ lập trình cho phép chúng ta điều khiển việc loại trừ lẫn nhau này.

Đoạn mã nguồn nhằm đảm bảo an toàn khi sử dụng đa luồng được gọi là **thread-safe** - an toàn luồng hay **MT-safe** (ngược lại sẽ là **not thread-safe** - mất an toàn luồng hoặc **MT-hot**). Nói tóm lại, chúng ta cần phải đảm bảo an toàn luồng cho mọi đối tượng và đảm bảo đa luồng cho mọi ứng dụng.

Tính bất biến

Một đối tượng bất biến – **immutability** là đối tượng mà dữ liệu của nó không thể bị thay đổi. Dữ liệu ở đây tức là:

- Các giá trị của miền đối tượng.
- Các giá trị chứa trong tài nguyên bên ngoài mà đối tượng đó quản lý (văn bản trong file).
- Các giá trị nằm trong các đối tượng được đối tượng đó trả đến.
- ...

Nói cách khác, để 1 đối tượng thực sự bất biến, phải đảm bảo không thể thay đổi bản thân đối tượng và các dữ liệu liên quan đến đối tượng, trực tiếp hoặc gián tiếp, bên trong hoặc bên ngoài. Ưu điểm của đối tượng bất biến là luôn đảm bảo được an toàn luồng – vì không có dữ liệu nào bị thay đổi đồng nghĩa với không có dữ liệu nào bị mất mát. Chúng còn rất hiệu quả (có thể truyền trong suốt và lưu trữ trong các vùng nhớ chỉ đọc). Một số ngôn ngữ cũng cung cấp các phương tiện đảm bảo bất biến – ví dụ như từ khóa **const** trong C++ và **final** trong Java.

Mặc dù đối tượng bất biến là 1 ý tưởng hay, nhưng hầu hết các đối tượng đều cần phải thay đổi: ví dụ, trong 1 đối tượng Người thì địa chỉ của người đó phải thay đổi được. Vì thế, chúng ta cần phải đảm bảo an toàn luồng cho cả những đối tượng hay biến đổi.

Fixed Values

Đảm bảo an toàn luồng cho các đối tượng luôn là 1 thách thức. Đặc biệt là khi các đối tượng được sử dụng đồng thời. Một vấn đề ở đây là **deadlock**. Deadlock là tình trạng mà 1 luồng A phải đợi 1 luồng B thực hiện xong, trong khi chính luồng B lại đang đợi luồng A thực hiện xong: kết quả là 2 luồng này phải chờ đợi nhau mãi mãi. Để tránh xảy ra deadlock, chúng ta cần phải xem xét cách thức các đối tượng liên kết với nhau, và các luồng chạy qua chúng như thế nào.

Thread A	Thread B
----------	----------

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

Need a resource held by B	Need a resource held by A
Wait for B	Wait for A
Deadlock	

Một thủ thuật đơn giản để thực hiện an toàn luồng là sử dụng **fixed values**. Fixed value chính là 1 miền bất biến trong 1 đối tượng: ví dụ, 1 đối tượng Math có thể chứa giá trị Pi không đổi bên trong nó. Fixed value, vì là bất biến nên nó tự động đảm bảo an toàn luồng.

Từ việc quyết định miền nào của đối tượng là fixed value, chúng ta có thể chia 1 đối tượng ra làm 2 phần: fixed value, là phần không cần bảo vệ, và changeable value, và phần cần phải bảo vệ. Fixed value chỉ có thể trở nên bất biến sau khi đối tượng được khởi tạo. Nói cách khác, chúng ta có thể thao tác với các fixed value bên trong phương thức khởi tạo (constructor) của đối tượng: chỉ cần chúng ta không thay đổi giá trị sau khi phương thức khởi tạo kết thúc, thì sẽ không có vấn đề gì. Lý do là chỉ duy nhất 1 luồng có thể truy nhập vào 1 phương thức khởi tạo, chính là luồng gọi hệ thời gian chạy tạo ra đối tượng; ngoài ra không luồng nào có thể truy nhập vào bên trong đối tượng vì lúc ấy đối tượng chưa được tạo ra.

Đồng bộ hóa (Synchronization) trong Java

Chúng ta có thể giải quyết hầu hết các vấn đề trong đa luồng bằng cách đóng gói các tài nguyên chung vào 1 đối tượng. Khi đó nhiệm vụ của 1 đối tượng là đảm bảo chỉ cho phép 1 luồng được truy nhập trong 1 thời gian.

Trong Java, 1 phương thức có thể được gán nhãn đồng bộ **synchronized**: hệ thời gian chạy sẽ đảm bảo trong 1 thời gian, chỉ duy nhất 1 luồng được gọi 1 phương thức synchronized của đối tượng. Việc này được thực hiện nhờ liên kết 1 **lock** với mỗi đối tượng, dưới sự điều khiển của 1 **monitor**. Luồng đầu tiên gọi đến 1 phương thức synchronized của đối tượng sẽ được chấp nhận bởi monitor, mà các luồng khác sẽ bị khóa không thể truy nhập phương thức synchronized cho đến khi luồng thứ nhất thực hiện xong. Điều này trong Java không được áp dụng với các phương thức không đồng bộ: các luồng có thể tự do truy nhập đối tượng ở thời điểm bất kỳ.

8.9 KẾT LUẬN

Trong chương này, chúng ta đã xem xét vấn đề thiết kế hệ thống con, một quá trình quyết định chính xác các đối tượng nào chúng ta dự định cài đặt và các giao diện nào chúng ta cần có:

- Chúng ta đã xem xét thiết kế tàng nghiệp vụ và cách rút ra từ mô hình phân tích lớp

CHƯƠNG 8. THIẾT KẾ CÁC HỆ THỐNG CON

- Chúng ta xem xét cách ánh xạ mô hình đối tượng thành lược đồ cơ sở dữ liệu quan hệ.
- Sau khi trình bày một vài hướng dẫn cho thiết kế giao diện người sử dụng, chúng ta đã bàn cách nhóm các phương tiện từ tầng trung gian thành các lớp dịch vụ nghiệp vụ che dấu sự phức tạp của tầng nghiệp vụ

BÀI TẬP

1. Trình bày cách thiết kế tầng nghiệp vụ dựa trên biểu đồ lớp phân tích
2. Dead lock là gì? Nó được cài đặt trong java thế nào? Nhóm đã sử dụng trong thiết kế hệ thống của mình không?
3. Trình bày thiết kế giao diện của hệ quản lý học theo tín chỉ
4. Hoàn thiện bài tập nhóm với các bước thiết kế chi tiết trình bày trong chương này

CHƯƠNG 9 MẪU THIẾT KẾ SỬ DỤNG LẠI

9.1 GIỚI THIỆU

Vấn đề trong thiết kế phần mềm hướng đối tượng

Thiết kế một hệ phần mềm hướng đối tượng là một công việc khó và việc thiết kế một phần mềm hướng đối tượng phục vụ cho mục đích dùng lại còn khó hơn. Chúng ta phải tìm ra những đối tượng phù hợp, đại diện cho một lớp các đối tượng. Sau đó thiết kế giao diện và cây kế thừa cho chúng, thiết lập mối quan hệ giữa chúng. Thiết kế của chúng ta phải đảm bảo là giải quyết được các vấn đề hiện tại, có thể tiến hành mở rộng trong tương lai mà tránh phải thiết kế lại phần mềm. Và một tiêu chí quan trọng là phải nhỏ gọn. Việc thiết kế một phần mềm hướng đối tượng phục vụ cho mục đích dùng lại là một công việc khó, phức tạp vì vậy chúng ta không thể mong chờ thiết kế của mình sẽ là đúng, và đảm bảo các tiêu chí trên ngay được. Thực tế là nó cần phải được thử nghiệm sau vài lần và sau đó nó sẽ được sửa chữa lại. Đứng trước một vấn đề, một người phân tích thiết kế tốt có thể đưa ra nhiều phương án giải quyết, anh ta phải duyệt qua tất cả các phương án và rồi chọn ra cho mình một phương án tốt nhất. Phương án tốt nhất này sẽ được anh ta dùng đi dùng lại nhiều lần mỗi khi gặp vấn đề tương tự.

Lịch sử mẫu thiết kế

Ý tưởng dùng mẫu thiết kế xuất phát từ ngành kiến trúc, Alexander, Ishikawa, Silverstein, Jacobson, Fiksdahl-King và Angel (1977) lần đầu tiên đưa ra ý tưởng dùng các mẫu chuẩn trong thiết kế xây dựng và truyền thông. Họ đã xác định và lập kho dữ liệu các mẫu có liên quan để có thể dùng giải quyết các vấn đề thường xảy ra trong thiết kế các cao ốc. Mỗi mẫu này là một cách thiết kế đã được phát triển hàng trăm năm như là các giải pháp cho các vấn đề thường gặp trong lĩnh vực xây dựng. Mặc dù ngành công nghệ phần mềm không có lịch sử phát triển lâu dài như ngành kiến trúc nhưng nhiều nghiên cứu đã chỉ ra rằng mẫu thiết kế được xem là giải pháp tốt để giải quyết vấn đề xây dựng hệ thống phần mềm.

Suốt những năm đầu 1990, nhiều Hội thảo đã nỗ lực để đưa ra danh sách các mẫu và lập kho tư liệu về chúng. Những người tham gia đã hướng đến cung cấp một số kiểu cấu trúc ở mức khái niệm cao hơn đối tượng và lớp để tổ chức các lớp. Người ta cho rằng việc dùng các kỹ thuật hướng đối tượng độc lập sẽ không mang lại những cải tiến đáng kể đối với chất lượng cũng như hiệu quả của công việc phát triển phần mềm. Mẫu được

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

xem là cách tổ chức việc phát triển hướng đối tượng nhằm đến tính hiệu quả trong thực tiễn.

Cuốn sách “Design patterns : Elements of Reusable Object Oriented Software” đã được xuất bản đúng vào thời điểm diễn ra hội nghị OOPSLA’94. Mặc dù, đây là một tài liệu sơ cấp trong việc làm nổi bật ảnh hưởng của mẫu đối với việc phát triển phần mềm, sự đóng góp của nó là xây dựng các mẫu thành các danh mục (catalogue) với định dạng chuẩn được dùng làm tài liệu cho mỗi mẫu và nổi tiếng với tên Gang of Four (bộ tứ). Tiếp theo sau đó, rất nhiều các cuốn sách khác liên tiếp xuất hiện và các định dạng chuẩn khác được đưa ra.

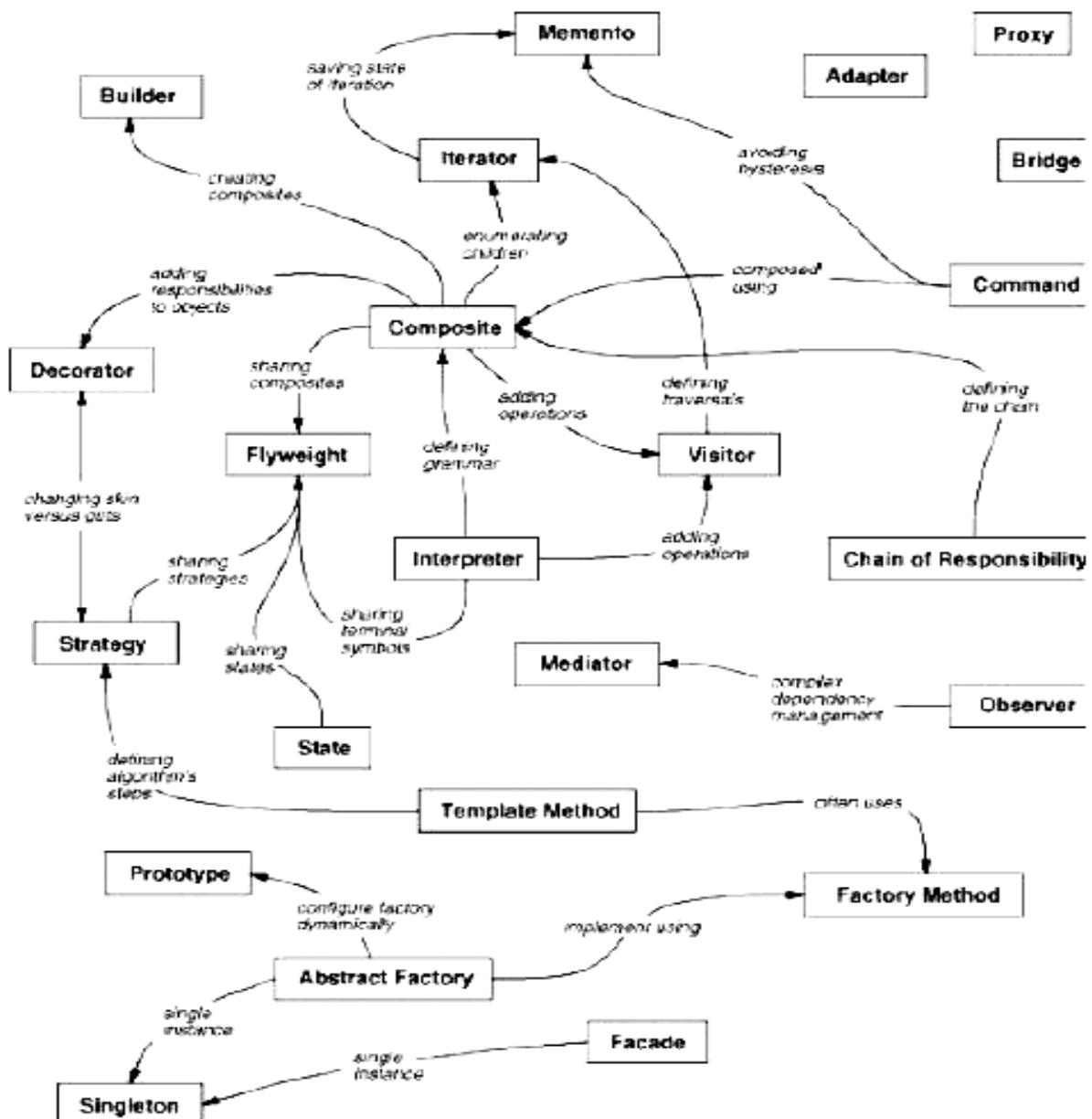
Năm 2000 Evitts đã tổng kết cách các mẫu xâm nhập vào thế giới phần mềm (cuốn sách của ông lúc bấy giờ chỉ nói về những mẫu có thể được sử dụng trong UML chứ chưa đưa ra khái niệm những mẫu thiết kế một cách tổng quát). Ông công nhận Kent Beck và Ward Cunningham là những người phát triển những mẫu đầu tiên với SmallTalk và kết quả của họ được báo cáo tại hội nghị OOPSLA’87. Có 5 mẫu mà Kent Beck và Ward Cunningham đã tìm ra trong việc kết hợp các người dùng của một hệ thống mà họ đang thiết kế. Năm mẫu này đều được áp dụng để thiết kế giao diện người dùng trong môi trường Windows.

9.2 MẪU THIẾT KẾ LÀ GÌ?

Mẫu thiết kế là tập các giải pháp cho vấn đề phổ biến trong thiết kế các hệ thống máy tính. Đây là tập các giải pháp đã được công nhận là tài liệu có giá trị, những người phát triển có thể áp dụng giải pháp này để giải quyết các vấn đề tương tự. Giống như với các yêu cầu của thiết kế và phân tích hướng đối tượng (nhằm đạt được khả năng sử dụng các thành phần và thư viện lớp), việc sử dụng các mẫu cũng cần phải đạt được khả năng tái sử dụng các giải pháp chuẩn đối với vấn đề thường xuyên xảy ra.

Christopher Alexander nói rằng :” Mỗi một mẫu mô tả một vấn đề xảy ra lặp đi lặp lại trong môi trường và mô tả cái cốt lõi của giải pháp để cho vấn đề đó.Bằng cách nào đó bạn đã dùng nó cả triệu lần mà không làm giống nhau 2 lần”.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DUNG LAI



Hình 9.1: Mối quan hệ giữa các mẫu thiết kế

Mẫu thiết kế không phải là một phần của UML cốt lõi, nhưng nó lại được sử dụng rộng rãi trong thiết kế hệ thống hướng đối tượng và UML cung cấp các cơ chế biểu diễn mẫu dưới dạng đồ họa.

9.3 HỆ THỐNG CÁC MẪU THIẾT KẾ

Hệ thống các mẫu thiết kế có 23 mẫu được định nghĩa trong cuốn sách “Design patterns Elements of Reusable Object Oriented Software”. Hệ thống các mẫu này có thể nói là đủ và tối ưu cho việc giải quyết hết các vấn đề của bài toán phân tích thiết kế và xây dựng

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

phần mềm hiện nay. Hệ thống các mẫu thiết kế được chia thành 3 nhóm: Creational, nhóm Structural và nhóm Behavioral.

Nhóm Creational

Gồm có 5 pattern: AbstractFactory, Abstract Method, Builder, Prototype, và Singleton. Nhóm này liên quan tới việc tạo ra các thể nghiệm (instance) của đối tượng, tách biệt với cách được thực hiện từ ứng dụng. Muốn xem xét thông tin của các mẫu trong nhóm này thì phải dựa vào biểu đồ nào phụ thuộc vào chính mẫu đó, mẫu thiên về hành vi hay cấu trúc.

Nhóm Structural

Gồm có 7 mẫu : Adapter, Bridge, Composite, Decorator, Facade, Proxy và Flyweight. Nhóm này liên quan tới các quan hệ cấu trúc giữa các thể nghiệm, dùng kế thừa, kết tập, tương tác. Để xem thông tin về mẫu này phải dựa vào biểu đồ lớp của mẫu.

Nhóm Behavioral

Gồm có 11 mẫu : Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy và Visitor. Nhóm này liên quan đến các quan hệ gán trách nhiệm để cung cấp các chức năng giữa các đối tượng trong hệ thống. Đối với các mẫu thuộc nhóm này ta có thể dựa vào biểu đồ cộng tác và biểu đồ diễn tiến. Biểu đồ cộng tác và biểu đồ diễn tiến sẽ giải thích cho ta cách chuyển giao của các chức năng.

9.4 NỘI DUNG CÁC MẪU THIẾT KẾ

9.4.1 Nhóm Creational

Abstract factory

Đặt vấn đề

Chúng ta có thể để ý thấy trong các hệ điều hành giao diện đồ họa, một bộ công cụ muốn cung cấp một giao diện người dùng dựa trên chuẩn “nhìn và cảm nhận” (look and feel), chẳng hạn như chương trình trình diễn tài liệu power point. Có rất nhiều kiểu giao diện như vậy và cả những hành vi giao diện người dùng khác nhau được thể hiện ở đây như thanh cuộn tài liệu (scroll bar), cửa sổ (window), nút bấm (button), hộp soạn thảo (editbox)...Nếu xem chúng là các đối tượng thì chúng ta thấy chúng có một số đặc điểm và hành vi khá giống nhau về mặt hình thức nhưng lại khác nhau về cách thực hiện. Chẳng hạn đối tượng button và window, editbox có cùng các thuộc tính là chiều rộng, chiều cao, toạ độ...và có các phương thức là Resize(), SetPosition()...Tuy nhiên, các đối tượng này không thể gộp chung vào một lớp được vì theo nguyên lý xây dựng lớp các đối

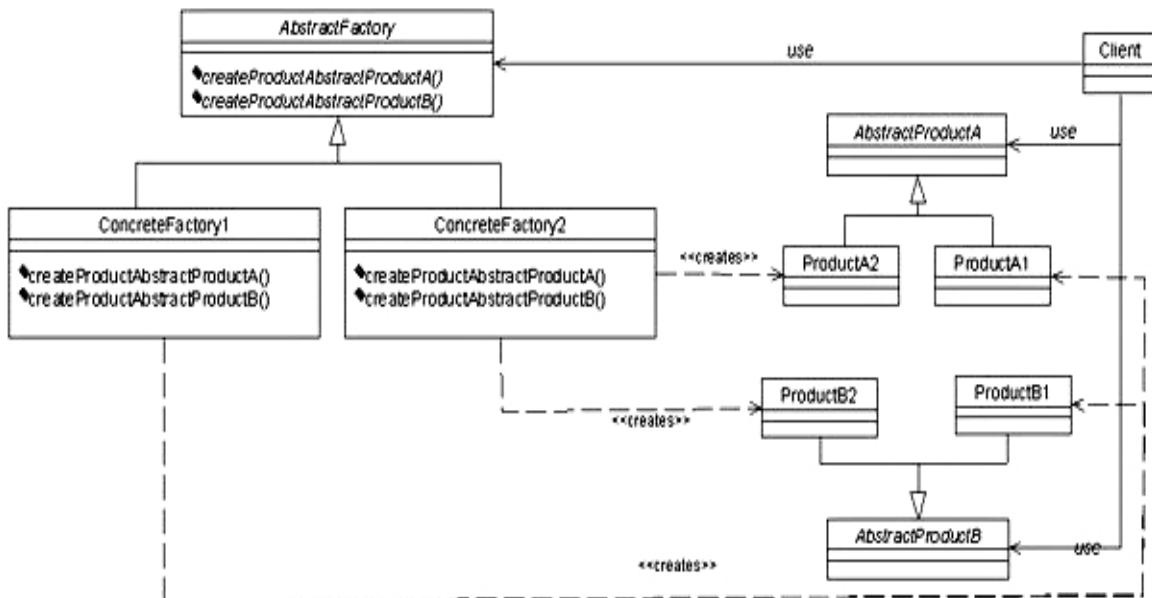
CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

tượng thuộc lớp phải có các phương thức hoạt động giống nhau. Trong khi ở đây tuy rằng các đối tượng có cùng giao diện nhưng cách thực hiện các hành vi tương ứng lại hoàn toàn khác nhau.

Vấn đề đặt ra là phải xây dựng một lớp tổng quát, có thể chứa hết được những điểm chung của các đối tượng này để từ đó có thể dễ dàng sử dụng lại, ta gọi lớp này là WidgetFactory. Các lớp của các đối tượng window, button, editbox kế thừa từ lớp này. Trong thiết kế hướng đối tượng, xây dựng một mô hình các lớp như thế được tối ưu hóa như sau:

Định nghĩa Mẫu AbstractFactory là một mẫu thiết kế có thể cung cấp cho trình khách một giao diện cho một họ hoặc một tập các đối tượng thuộc các lớp khác nhau nhưng có cùng chung giao diện với nhau mà không phải trực tiếp làm việc với từng lớp con cụ thể.

Cấu trúc mẫu



Trong đó:

- **AbstractFactory**: là lớp trừu tượng, tạo ra các đối tượng thuộc 2 lớp trừu tượng là: **AbstractProductA** và **AbstractProductB**.
- **ConcreteFactoryX**: là lớp kế thừa từ **AbstractFactory**, lớp này sẽ tạo ra một đối tượng cụ thể.
- **AbstractProduct**: là các lớp trừu tượng, các đối tượng cụ thể sẽ là các thể hiện của các lớp dẫn xuất từ lớp này.

Tình huống áp dụng

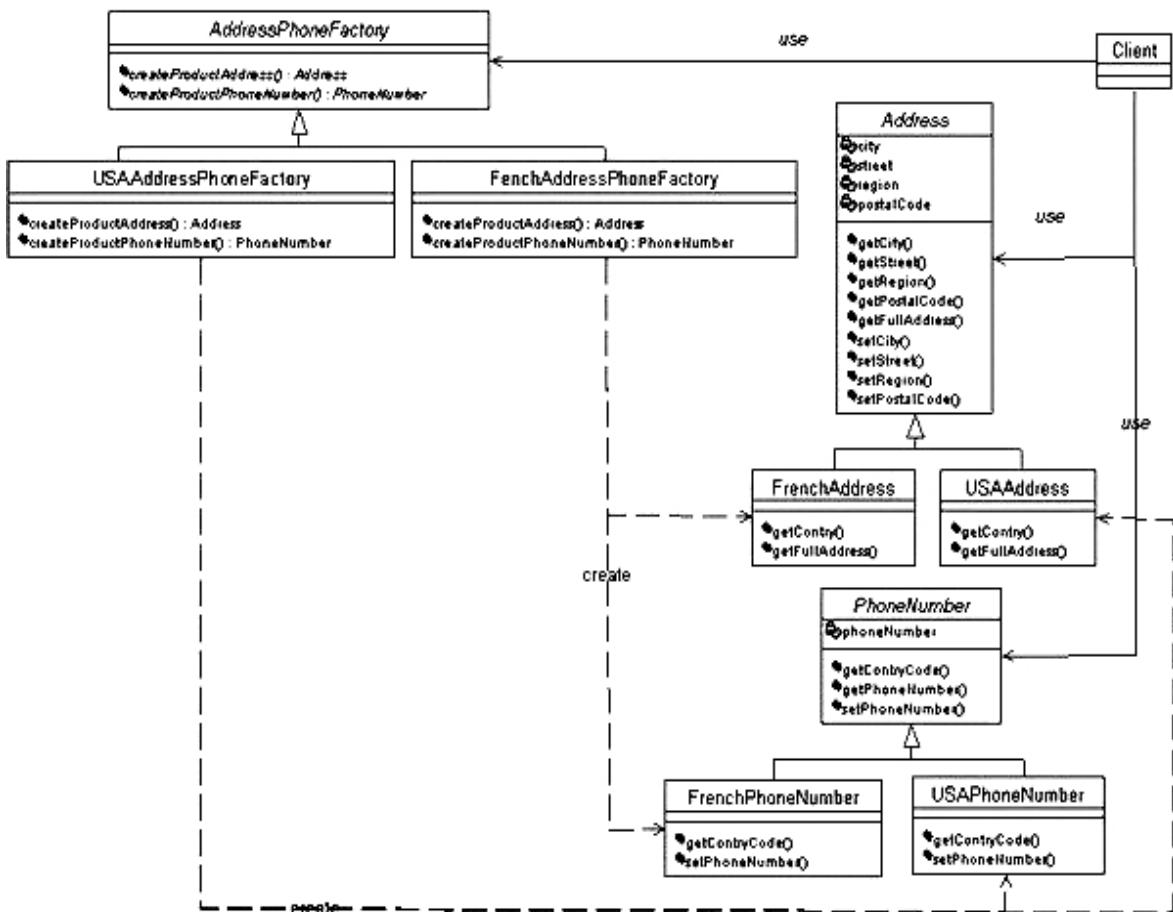
CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

- Phía trình khách sẽ không phụ thuộc vào việc những sản phẩm được tạo ra như thế nào.
- Ứng dụng sẽ được cấu hình với một hoặc nhiều họ sản phẩm.
- Các đối tượng cần phải được tạo ra như một tập hợp để có thể tương thích với nhau.
- Chúng ta muốn cung cấp một tập các lớp và muốn thể hiện các ràng buộc, các mối quan hệ giữa chúng mà không phải là các thực thi của chúng (interface).

Lớp WidgetFactory có 2 phương thức là CreateScrollBar() và CreateWindow() đây là lớp giao diện trừu tượng tổng quát cho tất cả các MotifWidgetFactory và PMWidgetFactory. Các lớp MotifWidgetFactory và PMWidgetFactory kế thừa trực tiếp từ lớp WidgetFactory. Trong sơ đồ trên còn có 2 nhóm lớp Window và ScrollBar, chúng đều là các lớp trừu tượng. Từ lớp Window sinh ra các lớp con cụ thể là PMWindow và MotifWindow. Từ lớp ScrollBar sinh ra các lớp con cụ thể là PMSearchBar và MotifScrollbar. Các đối tượng thuộc lớp này được các đối tượng thuộc lớp Factory (MotifWidgetFactory và PMWidgetFactory) gọi trong các hàm tạo đối tượng. Đối tượng trong ứng dụng (đối tượng khách - client) chỉ thông qua lớp giao diện của các đối tượng MotifWidgetFactory và PMWidgetFactory và các đối tượng trừu tượng Window và ScrollBar để làm việc với các đối tượng PMWindow, MotifWindow, PMSearchBar, MotifScrollbar. Điều này có được nhờ cơ chế binding trong các ngôn ngữ hỗ trợ lập trình hướng đối tượng như C++, C#, Java, Small Talk... Các đối tượng PMWindow, MotifWindow, PMSearchBar, MotifScrollbar được sinh ra vào thời gian chạy chương trình, nên trình ứng dụng (đối tượng thuộc lớp client) chỉ cần giữ một con trỏ trỏ đến đối tượng thuộc lớp WidgetFactory, và thay đổi địa chỉ trỏ đến nó thì có thể làm việc với tất cả các đối tượng ở trên. Những tình huống thiết kế thường có cùng một cách giải quyết như thế này đã được chứng tỏ là tối ưu. Nó được tổng quát hóa thành một mẫu thiết kế gọi là **AbstractFactory**.

Ví dụ: Giả sử ta cần viết một ứng dụng quản lý địa chỉ và số điện thoại cho các quốc gia trên thế giới. Địa chỉ và số điện thoại của mỗi quốc gia sẽ có 1 số điểm giống nhau và 1 số điểm khác nhau. Ta xây dựng sơ đồ lớp như sau:

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Ta sẽ xây dựng các phương thức tạo Address, và PhoneNumber cụ thể trong các lớp USAAddressPhoneFactory, FrenchAddressPhoneFactory.

Với phương thức **createProductAddress()** của lớp **USAAddressPhoneFactory** sẽ trả về đối tượng của lớp **USAAddress**.

Với phương thức **createProductAddress()** của lớp **FrenchAddressPhoneFactory** sẽ trả về đối tượng của lớp **FrenchAddress**.

Tương tự với **PhoneNumber**.

AddressFactory.java

```

public interface AddressFactory {
    public Address createAddress();
    public PhoneNumber createPhoneNumber();
}
  
```

Address.java

```

public abstract class Address {
    private String street;
    private String city;
    private String region;
    private String postalCode;
  
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
public static final String EOL_STRING =
        System.getProperty("line.separator");
public static final String SPACE = " ";
public String getStreet() {
    return street;
}
public String getCity() {
    return city;
}
public String getPostalCode() {
    return postalCode;
}
public String getRegion() {
    return region;
}
public abstract String getCountry();
public String getFullAddress() {
    return street + EOL_STRING + city + SPACE +
postalCode + EOL_STRING;
}

public void setStreet(String newStreet) {
    street = newStreet;
}
public void setCity(String newCity) {
    city = newCity;
}
public void setRegion(String newRegion) {
    region = newRegion;
}
public void setPostalCode(String newPostalCode) {
    postalCode = newPostalCode;
}
}
```

USAddressFactory.java

```
public class USAddressFactory implements AddressFactory{
    public Address createAddress() {
        return new USAddress();
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
    }
public PhoneNumber createPhoneNumber() {
    return new USPhoneNumber();
}
}

USAddress.java
public class USAddress extends Address{
private static final String COUNTRY = "UNITED STATES";
private static final String COMMA = ",";
public String getCountry(){
    return COUNTRY;
}
public String getFullAddress() {
    return getStreet() + EOL_STRING + getCity() + COMMA +
SPACE + getRegion() + SPACE + getPostalCode() + EOL_STRING
+ COUNTRY + EOL_STRING;
}
}
```

Tương tự cho lớp PhoneNumber và USAPhoneNumber

Mẫu liên quan

AbstractFactory thường được cài đặt cùng với singleton, FactoryMethod đôi khi còn dùng cả Prototype. Các lớp con cụ thể (concrete class) thường được cài đặt bằng singleton. Bởi singleton có thể tạo ra những đối tượng đồng nhất cho dù chúng ta gọi nó ở đâu trong chương trình. Các mẫu này sẽ được nói kỹ hơn ở các phần sau.

Builder

Đặt vấn đề

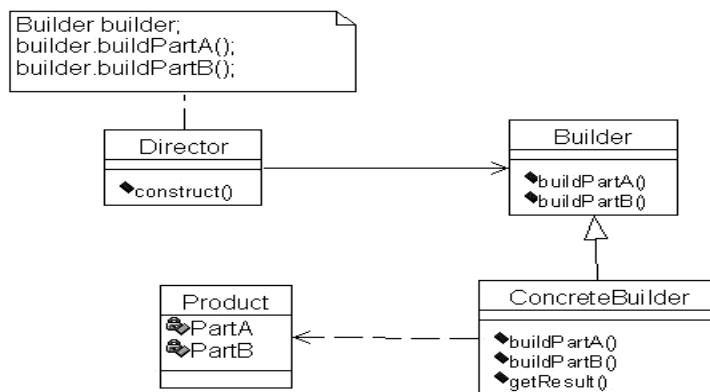
Trong những ứng dụng lớn, với nhiều các chức năng phức tạp và mô hình giao diện đồ sộ. Việc khởi tạo ứng dụng thường gặp nhiều khó khăn. Chúng ta không thể dồn tất cả công việc khởi tạo này cho một hàm khởi tạo . Vì như thế sẽ rất khó kiểm soát hết và hồn nữa không phải lúc nào các thành phần của ứng dụng cũng được khởi tạo một cách đồng bộ. Có thành phần được tạo ra vào lúc dịch chương trình nhưng cũng có thành phần tùy theo từng yêu cầu của người dùng, tùy vào hoàn cảnh của ứng dụng, mà nó sẽ được tạo ra. Do vậy người ta giao công việc này cho một đối tượng chịu trách nhiệm khởi tạo, và chia việc khởi tạo ứng dụng riêng rẽ, để có thể tiến hành khởi tạo riêng biệt ở các hoàn cảnh khác nhau. Hãy tưởng tượng việc tạo ra đối tượng giống như việc chúng ta tạo ra chiếc xe đạp. Đầu tiên ta tạo ra khung xe, sau đó tạo ra bánh xe, xích, líp... Việc tạo ra các

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

bộ phận này không nhất thiết phải được thực hiện một cách đồng thời hay theo một trật tự nào cả và nó có thể được tạo ra một cách độc lập bởi nhiều người. Nhưng trong một mô hình sản xuất như vậy, bao giờ việc tạo ra chiếc xe cũng được khép kín để tạo ra chiếc xe hoàn chỉnh, đó là nhà máy sản xuất xe đẹp. Ta gọi đối tượng nhà máy sản xuất xe đẹp này là builder (người xây dựng).

Định nghĩa: Builder là mẫu thiết kế hướng đối tượng được tạo ra để chia một công việc khởi tạo một đối tượng phức tạp thành các đối tượng riêng rẽ từ đó có thể tiến hành khởi tạo đối tượng ở các hoàn cảnh khác nhau.

Cấu trúc mẫu



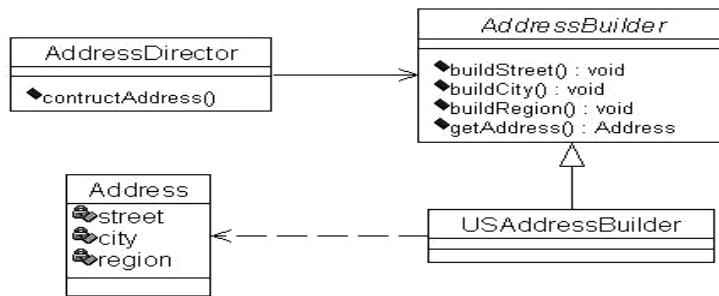
Trong đó:

- AddressDirector là lớp tạo ra đối tượng Address
- AddressBuilder là lớp trùu tượng cho phép tạo ra 1 đối tượng Address bằng các phương thức khởi tạo từng thành phần của Address
- USAddressBuilder là lớp tạo ra các Address. USAddressBuilder sẽ tạo ra địa chỉ theo chuẩn của USA.

Ví dụ

Ta lại xét đối tượng Address, có các thành phần sau: Street, City và Region. Ta phân rã việc khởi tạo một đối tượng Address thành các phần: buildStreet, buildCity và buildRegion.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Address.java

Address.java

```

class Address() {
    private String street;
    private String city;
    private String region;
    /**
     * @return the city
     */
    public String getCity() {
        return city;
    }
    /**
     * @param city the city to set
     */
    public void setCity(String city) {
        this.city = city;
    }
    /**
     * @return the region
     */
    public String getRegion() {
        return region;
    }
    /**
     * @param region the region to set
     */
    public void setRegion(String region) {
        this.region = region;
    }
    /**
     * @return the street
     */
  
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
public String getStreet() {  
    return street;  
}  
/**  
 * @param street the street to set  
 */  
public void setStreet(String street) {  
    this.street = street;  
}  
}  
}  
AddressBuilder.java
```

```
abstract class AddressBuilder{  
    abstract public void buildStreet(String street){}  
    abstract public void buildCity(String city){}  
    abstract public void buildRegion(String region){}  
}
```

```
USAddressBuilder.java
```

```
class USAddressBuilder extends AddressBuilder {  
    private Address add;  
    public void buildStreet(String street){  
        add.setStreet(street);  
    }  
    public void buildCity(String city){  
        add.setCity(city);  
    }  
    public void buildRegion(String region){  
        add.setRegion(region);  
    }  
    public Address getAddress(){  
        return add;  
    }  
}
```

```
AddressDirector.java
```

```
class AddressDirector{
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
public void Construct(AddressBuilder builder, String street,
String city, String region){
builder.buildStreet(street);
builder.buildCity(city);
builder.buildRegion(region);
}
```

Client.java

```
class Client{
AddressDirector director = new AddressDirector();
USAAddressBuilder b = new USAAddressBuilder();
director.Construct(b, "Street 01", "City 01", "Region 01");
Address add = b.getAddress();
}
```

Các mẫu liên quan

Builder thường được cài đặt cùng với các mẫu như Abstract Factory. Tầm quan trọng của Abstract Factory là tạo ra một dòng các đối tượng dẫn xuất (cả đơn giản và phức tạp). Ngoài ra Builder còn thường được cài đặt kèm với Composite pattern. Composite pattern thường là những gì mà Builder tạo ra.

Factory Method

Đặt vấn đề

Các Framework thường sử dụng các lớp trừu tượng để định nghĩa và duy trì mối quan hệ giữa các đối tượng. Một framework thường đảm nhiệm việc tạo ra các đối tượng hoàn chỉnh. Việc xây dựng một framework cho ứng dụng có thể đại diện cho nhiều đối tượng tài liệu cho người dùng. Có 2 loại lớp trừu tượng chủ chốt trong framework này là lớp ứng dụng và tài liệu. Cả 2 lớp đều là lớp trừu tượng và trình khách phải xây dựng các dẫn xuất, các lớp con để hiện thực hóa, tạo ra đối tượng phù hợp với yêu cầu của ứng dụng. Chẳng hạn để tạo ra một ứng dụng drawing, chúng ta định nghĩa một lớp DrawingApplication và một lớp DrawingDocument. Lớp ứng dụng chịu trách nhiệm quản lý tài liệu và chúng ta sẽ tạo ra chúng khi có nhu cầu (chẳng hạn khi người dùng chọn Open hoặc New từ menu).

Bởi vì một lớp Document cá biệt như thế này được tạo ra từ các dẫn xuất của lớp Abstract Document (trong framework) để đưa ra các thê nghiệm cho một ứng dụng Drawing, lớp ứng dụng không thể biết trước được lớp dẫn xuất của Abstract Document nào sẽ được tạo ra để trình bày, mà nó chỉ biết khi nào một đối tượng tài liệu nào nên được tạo chứ không biết loại đối tượng tài liệu nào để tạo. Điều này tạo ra một sự tiến thoái lưỡng nan: framework phải thê nghiệm một lớp, nhưng nó chỉ biết về lớp trừu

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

tượng của nó, mà lớp trùu tượng này lại không thể thử nghiệm trực tiếp được.Nếu ai từng làm việc với giao diện ứng dụng đơn tài liệu và đa tài liệu trong ngôn ngữ Visual C++, chúng ta cũng sẽ bắt gặp một vấn đề tương tự. Đối tượng MainFrame có thể tạo ra một đối tượng view mỗi khi người dùng nhấn chuột vào menu View hay Open, nhưng MainFrame hoàn toàn không hề biết một chút thông tin nào trước đó về View vì nó không chứa bất cứ một thử nghiệm nào của View.

Mẫu Abstract Method đưa ra một giải pháp cho vấn đề này.Nó đóng gói thông tin về lớp dẫn xuất Document nào được tạo và đưa ra ngoài framework. Lớp dẫn xuất ứng dụng định nghĩa lại một phương thức trùu tượng CreateDocument() trên lớp Application để trả về một đối tượng thuộc lớp dẫn xuất của lớp document.

```
class Application
{
//Các khai báo khác
Public:
Document* CreateDocument();
//Các khai báo khác
};
Document* Application::CreateDocument()
{
Document* newDocument = new MyDocument();
Return newDocument;
}
```

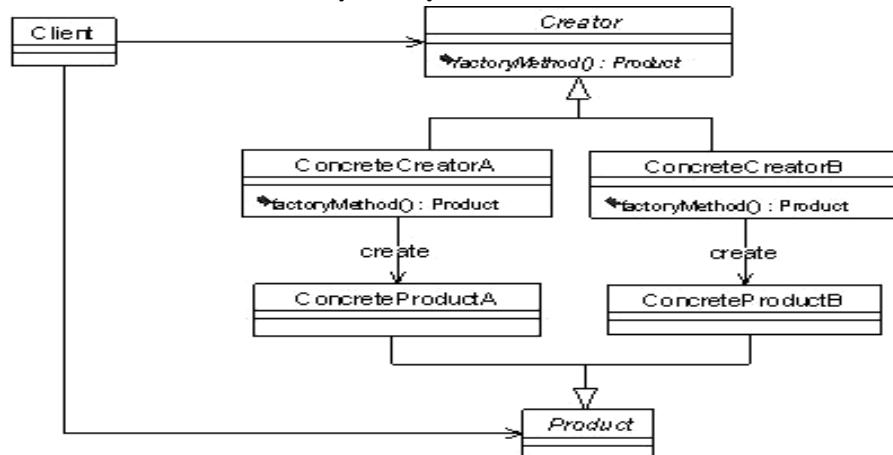
Khi một đối tượng thuộc lớp dẫn xuất của Application được tạo ra, nó có thể tạo ra luôn các đối tượng tài liệu đặc biệt cho ứng dụng mà không cần biết về các lớp đó.Chúng ta gọi CreateDocument là một factory method bởi vì nhiệm vụ của nó là sản xuất ra các đối tượng.

Định nghĩa Factory Method

Factory Method là một giao diện cho việc tạo ra một đối tượng, nhưng để cho lớp dẫn xuất quyết định lớp nào sẽ được tạo. Factory method để cho một lớp trì hoãn sự thử nghiệm một lớp con.

Cấu trúc mẫu

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Trong đó:

- Creator là lớp trừu tượng, khai báo phương thức `factoryMethod()` nhưng không cài đặt
- Product cũng là lớp trừu tượng
- `ConcreteCreatorA` và `ConcreteCreatorB` là 2 lớp kế thừa từ lớp Creator để tạo ra các đối tượng riêng biệt
- `ConcreteProductA` và `ConcreteProductB` là các lớp kế thừa của lớp Product, các đối tượng của 2 lớp này sẽ do 2 lớp `ConcreteCreatorA` và `ConcreteCreatorB` tạo ra

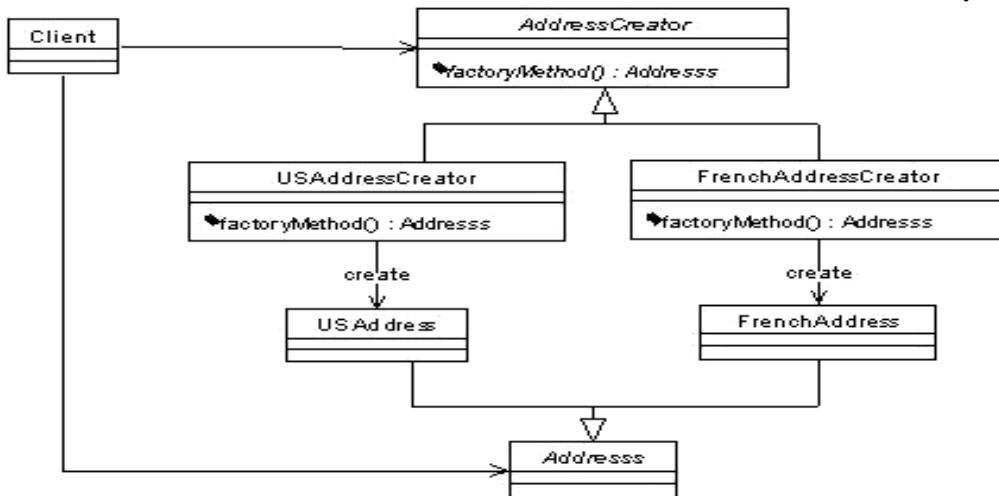
Tình huống áp dụng

- Khi bạn muốn tạo ra một framework có thể mở rộng, có nghĩa là nó cho phép tính mềm dẻo trong một số quyết định như chỉ ra loại đối tượng nào được tạo ra
- Khi bạn muốn 1 lớp con, mở rộng từ 1 lớp cha, quyết định lại đối tượng được khởi tạo
- Khi bạn biết khi nào thì khởi tạo một đối tượng nhưng không biết loại đối tượng nào được khởi tạo
- Bạn cần một vài khai báo chồng phương thức tạo với danh sách các tham số như nhau, điều mà Java không cho phép. Thay vì điều đó ta sử dụng các Factory Method với các tên khác nhau

Ví dụ

Ta xét lại ví dụ về các địa chỉ ở phần Abstract Pattern

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Mẫu liên quan

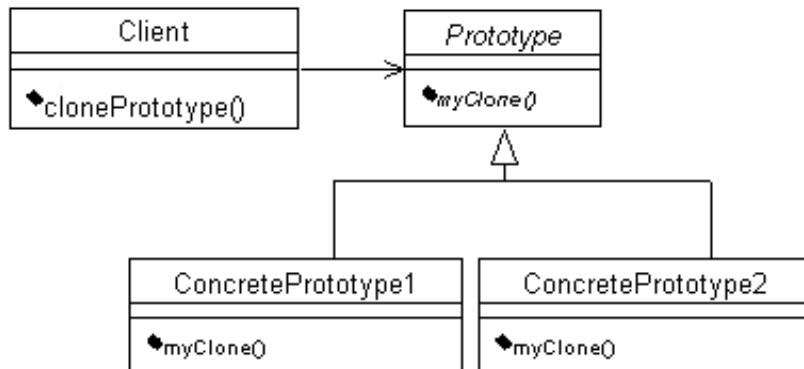
Abstract Factory thường được cài đặt cùng với Factory Method. Lớp Factory Method thường được gọi là Template Method. Trong ví dụ về ứng dụng Drawing trên, NewDocument là một template method. Prototype không cần đến một lớp con Creator, tuy nhiên thường đòi hỏi một phương thức để tạo thể nghiệm trên lớp dẫn xuất.

Prototype

Định nghĩa

Prototype là mẫu thiết kế chỉ định ra một đối tượng đặc biệt để khởi tạo, nó sử dụng một thể nghiệm sơ khai rồi sau đó sao chép ra các đối tượng khác từ mẫu đối tượng này.

Cấu trúc mẫu



Trong đó:

- Prototype là lớp trừu tượng cài đặt phương thức `myClone()` – một phương thức copy bản thân đối tượng đã tồn tại.
- `ConcretePrototype1` và `ConcretePrototype2` là các lớp kế thừa lớp `Prototype`.

Tình huống áp dụng

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

- Khi bạn muốn khởi tạo một đối tượng bằng cách sao chép từ một đối tượng đã tồn tại

Ví dụ



Mẫu liên quan

Prototype và Abstract Factory liên quan đến nhau chặt chẽ, có thể đổi chơi nhau theo nhiều kiểu. Tuy nhiên chúng cũng có thể kết hợp cùng nhau. Một Abstract Factory có thể chứa một tập các Prototype vô tính và trả về các đối tượng sản xuất.

Singleton

Đặt vấn đề

Ta hãy xem xét một đối tượng quản lý tài nguyên trong các ứng dụng. Mỗi ứng dụng có một bộ quản lý tài nguyên, nó cung cấp các điểm truy cập cho các đối tượng khác trong ứng dụng. Các đối tượng (ta gọi là đối tượng khách) có thể thực hiện lấy ra từ bộ quản lý tài nguyên những gì chúng cần và thay đổi giá trị nằm bên trong bộ quản lý tài nguyên đó. Để truy cập vào bộ quản lý tài nguyên đối tượng khách cần phải có một thể nghiệm của bộ quản lý tài nguyên, như vậy trong một ứng dụng sẽ có rất nhiều thể nghiệm của bộ quản lý tài nguyên được tạo ra.

```
class ResourceManager {  
private int x;  
public ResourceManager () {x = 0; ....}  
void SetX(int _x) { x = _x; }  
void GetX() { return x; }  
}  
class ClientObject1 {  
public void DoSomething () {  
ResourceManager rm;  
printf("x = %d", rm.GetX());  
x = 100;  
}  
}  
class ClientObject2 {  
public void DoSomething ()
```

```

{
ResourceManager rm;
printf("x = %d", rm.GetX());
x = 500;
}
}

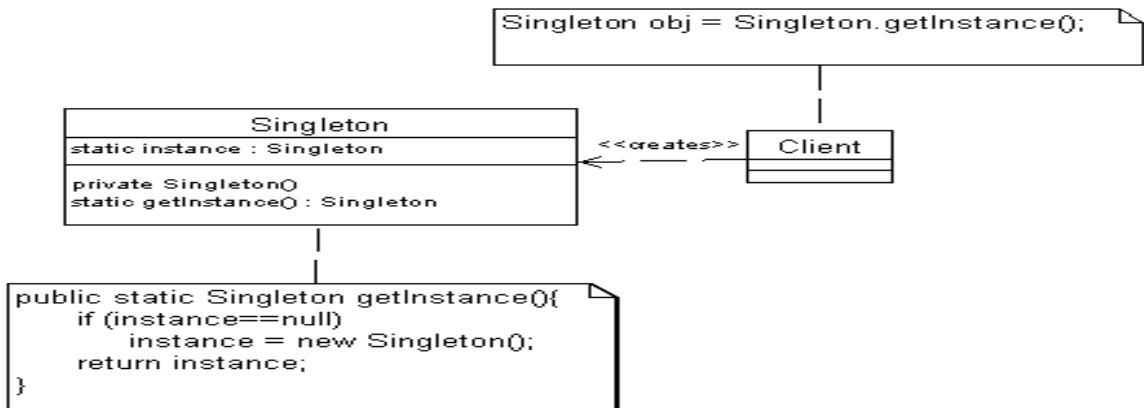
```

Trong ví dụ trên hàm DoSomething() của ClientObject1 khi truy cập vào đối tượng thuộc lớp ResourceManager sẽ in ra màn hình X = 0; và đặt vào đó x = 100; Sau đó hàm Dosomething() của ClientObject2 khi truy cập vào đối tượng thuộc lớp ResourceManager sẽ in ra màn hình X = 0 và đặt vào đó x = 500; Rõ ràng là tài nguyên mà các đối tượng khác truy cập vào không thể hiện sự thống nhất lẫn nhau. Điều mà lẽ ra khi giá trị x trong ResourceManager bị ClientObject1 thay đổi thì ClientObject2 phải nhận biết được điều đó và in ra màn hình X=100 . Nhưng theo logic cài đặt ở trên thì khi đối tượng thuộc lớp ClientObject1 truy cập đến ResourceManager nó tạo ra một Instance và đặt thuộc tính x = 100; Đối tượng ClientObject2 truy cập đến ResourceManager nó tạo ra một Instance và đặt thuộc tính x = 500. Hai Instance này hoàn toàn độc lập nhau về vùng nhớ do đó mà tài nguyên x mà nó quản lý cũng là 2 tài nguyên độc lập với nhau.Vấn đề đặt ra là phải tạo ra một bộ quản lý tài nguyên hoàn hảo tạo ra mọi thể nghiệm giống nhau tại nhiều nơi nhiều thời điểm khác nhau trong ứng dụng. Singleton cung cấp cho ta một cách giải quyết vấn đề này.

Định nghĩa

Singleton là mẫu thiết kế nhằm đảm bảo chỉ có duy nhất một thể nghiệm và cung cấp điểm truy cập của nó một cách thống nhất toàn cục.

Cấu trúc mẫu



Ví dụ

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
public class Singleton {  
    private String _strName;  
    private static Singleton instance;  
    private Singleton(String name) {  
        _strName = name;  
    }  
  
    public static Singleton getInstance(String name) {  
        if (instance == null)  
            instance = new Singleton(name);  
        return instance;  
    }  
  
    public void printName() {  
        System.out.println(this._strName);  
    }  
}
```

Mẫu liên quan

Có rất nhiều mẫu có thể cài đặt bổ sung từ việc sử dụng Singleton, chẳng hạn như Abstract Factory, Builder, và Prototype.

9.4.2 Nhóm Structural

Adapter

Đặt vấn đề

Đôi khi một lớp công cụ được thiết kế cho việc sử dụng lại nhưng không thể sử dụng lại chỉ bởi giao diện không thích hợp với miền giao diện đặc biệt mà một ứng dụng yêu cầu. Adapter đưa ra một giải pháp cho vấn đề này.

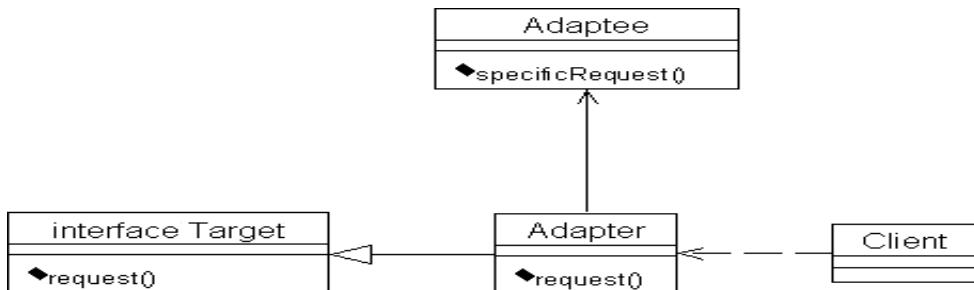
Trong một trường hợp khác ta muốn sử dụng một lớp đã tồn tại và giao diện của nó không phù hợp với giao diện của một lớp mà ta yêu cầu. Ta muốn tạo ra một lớp có khả năng được dùng lại, lớp đó cho phép kết hợp với các lớp không liên quan hoặc không được dự đoán trước, các lớp đó không nhất thiết phải có giao diện tương thích với nhau. Với đối tượng adapter, ta cần sử dụng một vài lớp con đã tồn tại, nhưng để làm cho các giao diện của chúng tương thích với nhau bằng việc phân lớp các giao diện đó là việc làm không thực tế, để làm được điều này ta dùng một đối tượng adapter để biến đổi giao diện lớp cha của nó.

Định nghĩa

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Adapter là mẫu thiết kế dùng để biến đổi giao diện của một lớp thành một giao diện khác mà clients yêu cầu. Adapter ngăn cản các lớp làm việc cùng nhau đó không thể làm bằng cách nào khác bởi giao diện không tương thích.

Cấu trúc mẫu



Trong đó:

- Tagret là một interface định nghĩa chức năng, yêu cầu mà Client cần sử dụng
- Adaptee là lớp có chức năng mà Target cần sử dụng để tạo ra được chức năng mà Target cần cung cấp cho Client
- Adapter thực thi từ Target và sử dụng đối tượng lớp Adaptee, Adapter có nhiệm vụ gắn kết Adaptee vào Target để có được chức năng mà Client mong muốn

Trường hợp ứng dụng

- Muốn sử dụng 1 lớp có sẵn nhưng giao tiếp của nó không tương thích với yêu cầu hiện tại
- Muốn tạo 1 lớp có thể sử dụng lại mà lớp này có thể làm việc được với những lớp khác không liên hệ gì với nó, và là những lớp không cần thiết tương thích trong giao diện.

Ví dụ

Một hệ thống PhoneTarget cần thực hiện một chức năng gì đó, trong đó có một phương thức trả về số điện thoại trong một chuỗi đầu vào. Trước đó ta đã có một lớp có một chức năng là lấy các kí tự số trong một chuỗi. Giờ ta muốn sử dụng chức năng lấy kí tự số vào hệ thống lấy số điện thoại.

```
public interface PhoneTarget{
    public String getPhoneNumber(String message);
    //lấy số điện thoại trong 1 chuỗi
}
```

```
public GetNumberAdaptee{
    public String getNumber(String str){
        //lấy ra dạng số trong 1 chuỗi
    }
}
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

<...get number>

}

...

}

```
public Adapter implements PhoneTarget{  
    public String getPhoneNumber(String message) {  
        GetNumberAdaptee obj = new  
GetNumberAdaptee;  
        String str = obj.getNumber(message);  
        return "84"+str;  
    }  
}
```

Mẫu liên quan

Bridge có một cấu trúc tương tự như một đối tượng của Adapter, nhưng Bridge có một mục đích khác. Nó chia giao diện từ các phần cài đặt của nó ra riêng rẽ để từ đó có thể linh hoạt hơn và độc lập nhau. Sử dụng một Adapter đồng nghĩa với việc thay đổi giao diện của một đối tượng đã tồn tại. Decorator nâng cấp một đối tượng khác mà không làm thay đổi giao diện của nó. Một Decorator do đó mà trong suốt với ứng dụng hơn là một Adapter. Như một hệ quả Decorator hỗ trợ cơ chế kết tập để quy mà điều này không thể thực hiện được đối với các Adapter thuận tuý.

Proxy định nghĩa một đại diện cho một đối tượng khác và không làm thay đổi giao diện của nó.

Bridge

Đặt vấn đề

Khi một lớp trừu tượng (abstraction) có thể có một vài thành phần bổ sung thêm thì cách thông thường phù hợp với chúng là sử dụng kế thừa. Một lớp trừu tượng định nghĩa một giao diện cho trừu tượng đó, và các lớp con cụ thể thực hiện nó theo các cách khác nhau. Nhưng cách tiếp cận này không đủ mềm dẻo. Sự kế thừa ràng buộc một thành phần bổ sung thêm là cố định cho abstraction, điều này làm nó khó thay đổi, mở rộng, và sử dụng lại các abstraction, các thành phần bổ sung một cách độc lập. Trong trường hợp này dùng một mẫu Bridge là thích hợp nhất. Mẫu Bridge thường được ứng dụng khi :

- Ta muốn tránh một ràng buộc cố định giữa một abstraction và một thành phần bổ sung thêm của nó.
- Cả hai, các abstraction và các thành phần cài đặt của chúng nên có khả năng mở rộng bằng việc phân chia lớp. Trong trường hợp này, Bridge pattern cho phép ta

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

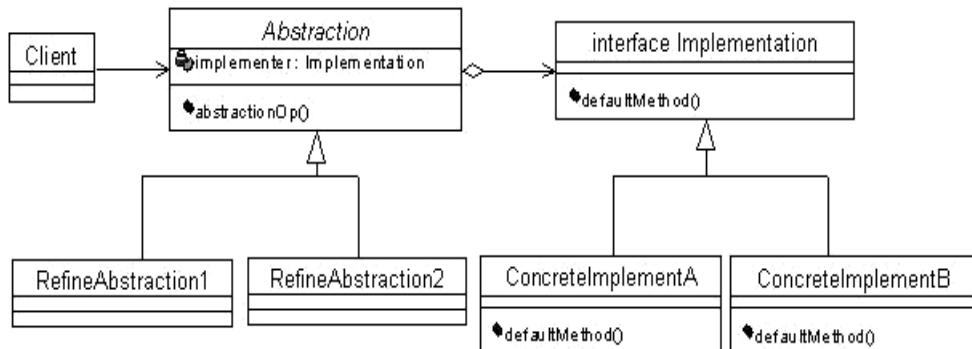
kết hợp các abstraction và các thành phần bổ sung thêm khác nhau và mở rộng chúng một cách độc lập.

- Thay đổi trong thành phần được bổ sung thêm của một abstraction mà không ảnh hưởng đến với các client, tức là mã của chúng không nên đem biên dịch lại.
- Ta muốn làm ẩn đi hoàn toàn các thành phần bổ sung thêm của một abstraction khỏi các client.
- Ta có một sự phát triển rất nhanh các lớp, hệ thống phân cấp lớp chỉ ra là cần phải tách một đối tượng thành hai phần.
- Ta muốn thành phần bổ sung thêm có mặt trong nhiều đối tượng, và việc này lại được che khỏi client (client không thấy được).

Định nghĩa

Bridge là mẫu thiết kế dùng để tách riêng một lớp trừu tượng khỏi thành phần cài đặt của nó để có được hai cái có thể biến đổi độc lập.

Cấu trúc mẫu



Trong đó:

- Abstraction: là lớp trừu tượng khai báo các chức năng và cấu trúc cơ bản, trong lớp này có 1 thuộc tính là 1 thể hiện của giao tiếp Implementation, thể hiện này bằng các phương thức của mình sẽ thực hiện các chức năng abstractionOp() của lớp Abstraction
- Implementation: là giao tiếp thực thi của lớp các chức năng nào đó của Abstraction
- RefineAbstraction: là định nghĩa các chức năng mới hoặc các chức năng đã có trong Absrtaction.
- ConcreteImplement: là các lớp định nghĩa tường minh các thực thi trong lớp giao tiếp Implementation

Trường hợp ứng dụng

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

- Khi bạn muốn tạo ra sự mềm dẻo giữa 2 thành phần ảo và thực thi của một thành phần, và tránh đâm môi quan hệ tĩnh giữa chúng
- Khi bạn muốn những thay đổi của phần thực thi sẽ không ảnh hưởng đến client
- Bạn định nghĩa nhiều thành phần ảo và thực thi.
- Phân lớp con một cách thích hợp, nhưng bạn muốn quản lý 2 thành phần của hệ thống một cách riêng biệt.

Ví dụ

```
class MyAbstraction{  
    private MyImplementation myImp;  
    public void method01(){  
        myImp.doMethod1();  
    }  
    public String method2(){  
        myImp.doMethod2();  
    }  
}  
  
interface class MyImplementation{  
    public void doMethod1();  
    public String doMethod2();  
}  
  
class RefineAbstraction1 extends MyAbstraction{  
    //các định nghĩa riêng, tường minh  
}  
  
class ConcreteImpleA extend MyImplement{  
    //các định nghĩa riêng, tường minh  
}  
  
class RefineAbstraction2 extends MyAbstraction{  
    //các định nghĩa riêng, tường minh  
}  
  
class ConcreteImpleB extend MyImplement{  
    //các định nghĩa riêng, tường minh  
}
```

Các mẫu liên quan

Abstract Factory cũng có thể tạo ra và cấu hình một Bridge. Adapter có thể được cơ cấu theo hướng để 2 lớp không có quan hệ gì với nhau có thể làm việc với nhau được. Nó thường ứng dụng cho các hệ thống sau khi đã được thiết kế. Bridge xét ở một khía cạnh khác nó kết thúc một thiết kế để lớp trừu tượng và lớp cài đặt có thể tùy biến một cách độc lập.

Composite

Đặt vấn đề

Các ứng dụng đồ họa như bộ soạn thảo hình vẽ và các hệ thống lưu giữ biểu đồ cho phép người sử dụng xây dựng lên các lược đồ phức tạp xa với các thành phần cơ bản, đơn giản. Người sử dụng có thể nhóm một số các thành phần để tạo thành các thành phần khác lớn hơn, và các thành phần lớn hơn này lại có thể được nhóm lại để tạo thành các thành phần lớn hơn nữa. Một cài đặt đơn giản có thể xác định các lớp cho các thành phần đồ họa cơ bản như Text và Line, cộng với các lớp khác cho phép hoạt động như các khuôn chứa các thành phần cơ bản đó.

Nhưng có một vấn đề với cách tiếp cận này, đó là, mã sử dụng các lớp đó phải tác động lên các đối tượng nguyên thủy (cơ bản) và các đối tượng bao hàm các thành phần nguyên thủy ấy là khác nhau ngay cả khi hầu hết thời gian người sử dụng tác động lên chúng là như nhau. Có sự phân biệt các đối tượng này làm cho ứng dụng trở nên phức tạp hơn. Composite pattern đề cập đến việc sử dụng các thành phần để quy để làm cho các client không tạo ra sự phân biệt trên.

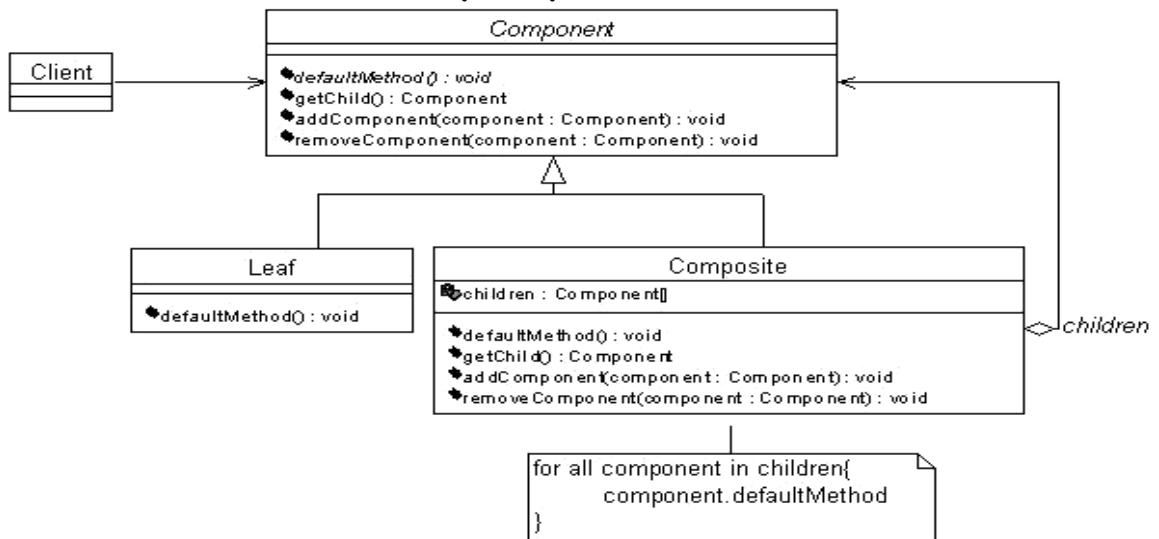
Giải pháp của Composite pattern là một lớp trừu tượng biểu diễn cả các thành phần cơ bản và các lớp chứa chúng. Lớp này cũng xác định các thao tác truy nhập và quản lý các con của nó.

Định nghĩa

Composite là mẫu thiết kế dùng để tạo ra các đối tượng trong các cấu trúc cây để biểu diễn hệ thống phân lớp: bộ phận – toàn bộ. Composite cho phép các client tác động đến từng đối tượng và các thành phần của đối tượng một cách thống nhất.

Cấu trúc mẫu.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Trong đó:

- **Component**: là một giao tiếp định nghĩa các phương thức cho tất cả các phần của cấu trúc cây. Component có thể được thực thi như một lớp trừu tượng khi bạn cần cung cấp các hành vi cho tất cả các kiểu con. Bình thường, các Component không có các thể hiện, các lớp con hoặc các lớp thực thi của nó, gọi là các nốt, có thể có thể hiện và được sử dụng để tạo nên cấu trúc cây
- **Composite**: là lớp được định nghĩa bởi các thành phần mà nó chứa. Composite chứa một nhóm động các Component, vì vậy nó có các phương thức để thêm vào hoặc loại bỏ các thể hiện của Component trong tập các Component của nó. Những phương thức được định nghĩa trong Component được thực thi để thực hiện các hành vi đặc tả cho lớp Composite và để gọi lại phương thức đó trong các nốt của nó. Lớp Composite được gọi là lớp nhánh hay lớp chứa
- **Leaf**: là lớp thực thi từ giao tiếp Component. Sự khác nhau giữa lớp Leaf và Composite là lớp Leaf không chứa các tham chiếu đến các Component khác, lớp Leaf đại diện cho mức thấp nhất của cấu trúc cây

Trường hợp ứng dụng

- Khi có một mô hình thành phần với cấu trúc nhánh - lá, toàn bộ - bộ phận, ...
- Khi cấu trúc có thể có vài mức phức tạp và động
- Bạn muốn thăm cấu trúc thành phần theo một cách qui chuẩn, sử dụng các thao tác chung thông qua mối quan hệ kế thừa.

Ví dụ

Trở lại ví dụ về dự án, một Project(Composite) có nhiều tác vụ Task(Leaf), ta cần tính tổng thời gian của dự án thông qua thời gian của tất cả các tác vụ

```
public interface TaskItem{
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
public double getTime();  
}  
  
public class Project implements TaskItem{  
    private String name;  
    private ArrayList subtask = new ArrayList();  
  
    public Project(){}  
    public Project(String newName){  
        name = newName;  
    }  
    public String getName(){ return name; }  
    public ArrayList getSubtasks(){ return subtask; }  
    public double getTime(){  
        double totalTime = 0;  
        Iterator items = subtask.iterator();  
        while(items.hasNext()){  
            TaskItem item = (TaskItem)items.next();  
            totalTime += item.getTime();  
        }  
        return totalTime;  
    }  
  
    public void setName(String newName){ name = newName; }  
  
    public void addTaskItem(TaskItem element){  
        if (!subtask.contains(element)){  
            subtask.add(element);  
        }  
    }  
    public void removeTaskItem(TaskItem element){  
        subtask.remove(element);  
    }  
}  
  
public class Task implements TaskItem{  
    private String name;  
    private double time;  
  
    public Task(){}  
    public Task(String newName, double newTimeRequired){  
        name = newName;
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
    time = newTimeRequired;  
}  
  
public String getName(){ return name; }  
public double getTime(){  
    return time;  
}  
public void setName(String newName){ name =  
newName; }  
public void setTime(double newTimeRequired){ time =  
newTimeRequired; }  
}
```

Các mẫu liên quan

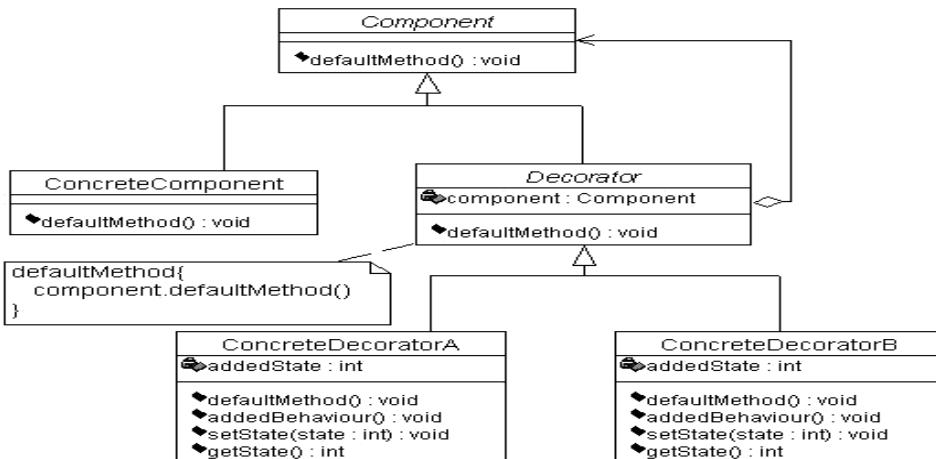
Một mẫu mà thường dùng làm thành phần liên kết đến đối tượng cha là Chain of Responsibility. Mẫu Decorator cũng thường được sử dụng với Composite. Khi Decorator và Composite cùng được sử dụng cùng nhau, chúng thường sẽ có một lớp cha chung. Vì vậy Decorator sẽ hỗ trợ thành phần giao diện với các phương thức như Add, Remove và GetChild. Mẫu Flyweight để cho chúng ta chia sẻ thành phần, nhưng chúng sẽ không tham chiếu đến cha của chúng. Mẫu Iterator có thể dùng để duyệt mẫu Composite. Mẫu Visitor định vị thao tác và hành vi nào sẽ được phân phối qua các lớp lá và Composite.

Decorator

Định nghĩa

Gắn một vài chức năng bổ sung cho các đối tượng (gán động). Decorator cung cấp một số thay đổi mềm dẻo cho các phân lớp để mở rộng thêm các chức năng.

Cấu trúc mẫu.



Trong đó:

- Component: là một interface chứa các phương thức ảo (ở đây là defaultMethod)
- ConcreteComponent: là một lớp kế thừa từ Component, cài đặt các phương thức cụ thể (defaultMethod được cài đặt tường minh)
- Decorator: là một lớp ảo kế thừa từ Component đồng thời cũng chứa 1 thê hiện của Component, phương thức defaultMethod trong Decorator sẽ được thực hiện thông qua thê hiện này.
- ConcreteDecoratorX: là các lớp kế thừa từ Decorator, khai báo tường minh các phương thức, đặc biệt trong các lớp này khai báo tường minh các “trách nhiệm” cần thêm vào khi run-time

Trường hợp ứng dụng

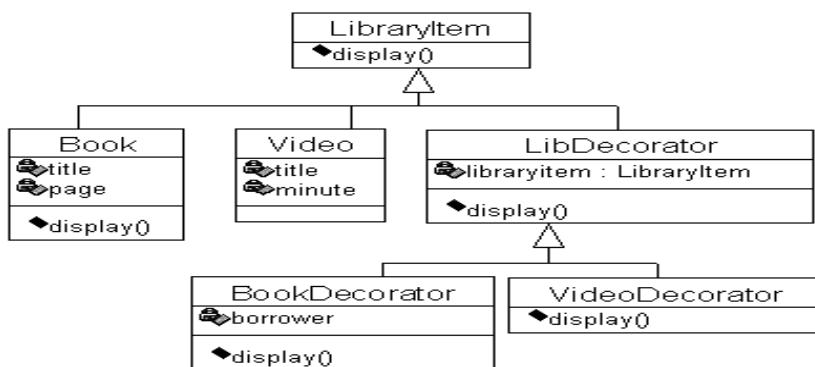
Khi bạn muốn thay đổi động mà không ảnh hưởng đến người dùng, không phụ thuộc vào giới hạn các lớp con

Khi bạn muốn thành phần có thể thêm vào hoặc rút bỏ đi khi hệ thống đang chạy

Có một số đặc tính phụ thuộc mà bạn muốn ứng dụng một cách động và bạn muốn kết hợp chúng vào trong một thành phần.

Ví dụ

Giả sử trong thư viện có các tài liệu: sách, video... Các loại tài liệu này có các thuộc tính khác nhau, phương thức hiển thị của giao tiếp LibraryItem sẽ hiển thị các thông tin này. Giả sử, ngoài các thông tin thuộc tính trên, đôi khi ta muốn hiển thị độc giả mượn một cuốn sách (chức năng hiển thị độc giả này không phải lúc nào cũng muốn hiển thị), hoặc muốn xem đoạn video(không thường xuyên).



Giao tiếp LibraryItem định nghĩa phương thức display() cho tất cả các tài liệu của thư viện:

```
public interface LibraryItem{
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
    public void display(); // đây là defaultMethod  
}
```

Các lớp tài liệu:

```
public class Book implements LibraryItem{  
    private String title;  
    private int page;  
    public Book(String s, int p){  
        title = s;  
        page = p;  
    }  
    public void display(){  
        System.out.println("Title: " + title);  
        System.out.println("Page number: " + page);  
    }  
}  
  
public class Video implements LibraryItem{  
    private String title;  
    private int minutes;  
    public Video(String s, int m){  
        title = s;  
        minutes = m;  
    }  
    public void display(){  
        System.out.println("Title: " + title);  
        System.out.println("Time: " + minutes);  
    }  
}
```

Lớp ảo Decorator thư viện

```
public abstract class LibDecorator implements LibraryItem{  
    private LibraryItem libraryitem;  
  
    public LibDecorator(LibraryItem li){  
        libraryitem = li;  
    }  
    public void display(){  
        libraryitem.display();  
    }  
}
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Các lớp Decorator cho mỗi tài liệu thư viện cần bổ sung trách nhiệm ở thời điểm run-time:

```
public class BookDecorator extends LibDecorator{
    private String borrower;
    public BookDecorator(LibraryItem li, String b) {
        super(li);
        borrower = b;
    }
    public void display(){
        super.display();
        System.out.println("Borrower: " + borrower);
    }
}
public class VideoDecorator extends LibDecorator{
    public VideoDecorator(LibraryItem li) {
        super(li);
    }
    public void display(){
        super.display();
        play(); //phương thức play video
    }
}
```

Các mẫu liên quan

Mẫu Decorator khác với Adapter, Decorator chỉ thay đổi nhiệm vụ của đối tượng, không phải là thay đổi giao diện của nó như Adapter. Adapter sẽ mang đến cho đối tượng một giao diện mới hoàn toàn. Decorator cũng có thể coi như một Composite bị thoái hoá với duy nhất một thành phần. Tuy nhiên, một Decorator thêm phần nhiệm phụ, nó là phần đối tượng được kết tập vào. Một Decorator cho phép chúng ta thay đổi bề ngoài của một đối tượng, một strategy cho phép chúng ta thay đổi ruột của đối tượng. Chúng là 2 cách luân phiên nhau để ta thay đổi một đối tượng.

Facade

Đặt vấn đề

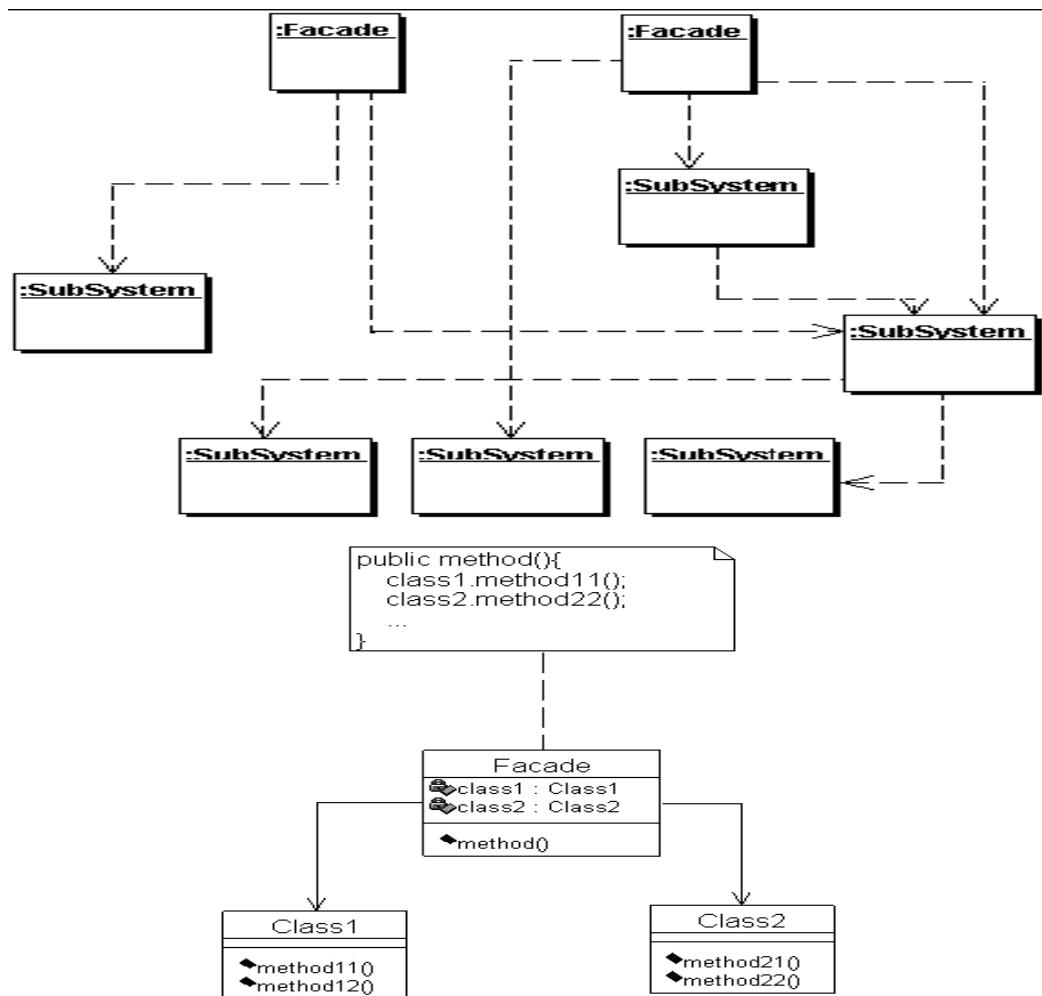
Khi cấu trúc hóa một hệ thống thành các hệ thống con sẽ giúp làm giảm độ phức tạp của hệ thống. Một mục tiêu thiết kế thông thường là tối thiểu hóa sự giao tiếp và phụ thuộc giữa các hệ thống con. Một cách để đạt được mục tiêu này là đưa ra đối tượng facade, đối tượng này cung cấp một giao diện đơn giản để dễ dàng tổng quát hóa cho một hệ thống con hơn.

Định nghĩa

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Mẫu Facade cung cấp một giao diện thống nhất cho một tập các giao diện trong một hệ thống con. Facade định nghĩa một giao diện ở mức cao hơn, giao diện này làm cho hệ thống con được sử dụng dễ dàng hơn.

Cấu trúc mẫu.



Trong đó

- Class1 và Class2 là các lớp đã có trong hệ thống
- Facade là lớp sử dụng các phương thức của Class1 và Class2 để tạo ra một giao diện mới, thường được sử dụng, đỡ phức tạp hơn khi sử dụng riêng Class1 và Class2

Trường hợp ứng dụng

- Làm cho một hệ thống phức tạp dễ sử dụng hơn bằng cách cung cấp một giao tiếp đơn giản mà không cần loại bỏ các lựa chọn phức tạp

Mẫu liên quan

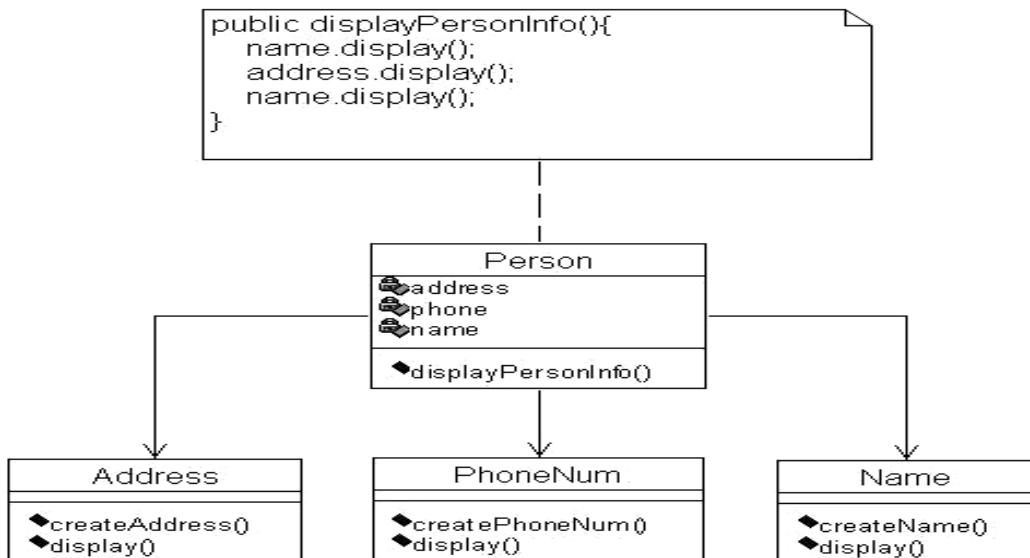
Abstract Factory có thể được sử dụng cùng với Facade để cung cấp một giao diện cho việc tạo hệ thống con một cách độc lập hệ thống con. Abstract Factory cũng có thể được sử dụng như một sự thay thế cho Facade để ẩn các lớp nền đặc biệt.

Mediator tương tự như Facade ở chỗ trùu tượng chức năng của một lớp đã tồn tại. Tuy nhiên mục đích của Mediator là trùu tượng một cách chuyên quyền sự giao tiếp giữa đối tượng cộng tác, thường chức năng trung tâm không thuộc về bất kỳ đối tượng cộng tác nào. Một đối tượng cộng tác với Mediator được nhận và giao tiếp với mediator thay vì giao tiếp với nhau một cách trực tiếp. Trong hoàn cảnh nào đó, Facade chỉ đơn thuần là trùu tượng giao diện cho một môt đối tượng hệ thống con để làm nó dễ sử dụng hơn, nó không định nghĩa một chức năng mới và lớp hệ thống con không hề biết gì về nó.

Thường thì một đối tượng Facade là đủ. Do đó đối tượng Facade thường là Singleton.

Ví dụ

Giả sử hệ thống cũ đã có các lớp: Địa chỉ, Số điện thoại, Tên tuổi về thông tin của một người, giờ ta muốn quản lý các thông tin trên của một người, ta tận dụng lại hệ thống cũ, xây một lớp Người sử dụng lại các lớp ở trên.



Flyweight

Vấn đề đặt ra

Một vài ứng dụng có thể được lợi từ việc sử dụng các đối tượng xuyên suốt thiết kế của chúng, nhưng một cài đặt không tốt sẽ cần trả điều đó. Trong tình huống này chúng ta sẽ dùng mẫu thiết kế Flyweight để giải quyết.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

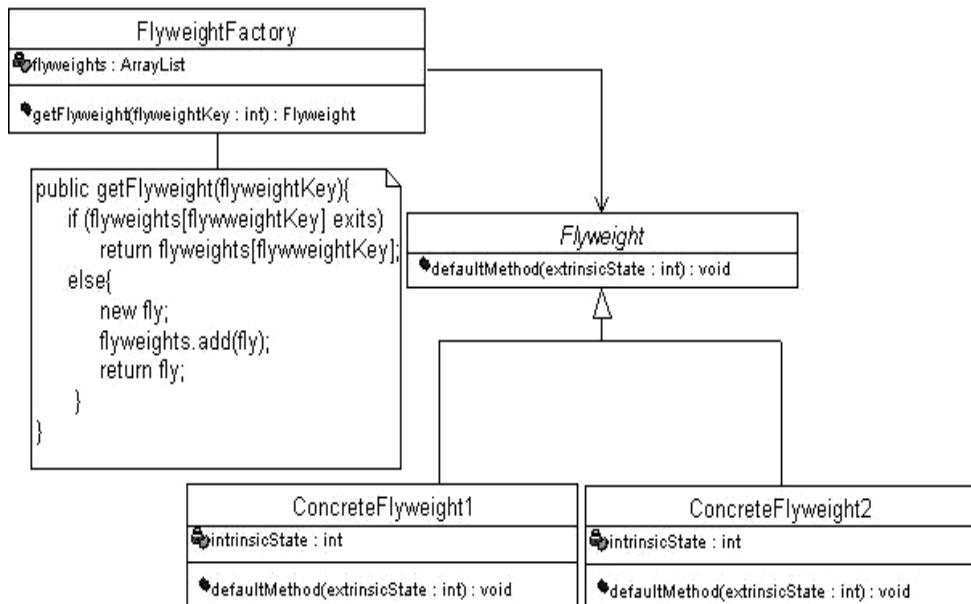
Định nghĩa

Mẫu thiết kế Flyweight là mẫu thiết kế được sử dụng việc chia sẻ để trợ giúp một lượng lớn các đối tượng một cách hiệu quả.

Việc sử dụng mẫu này cần chú ý rằng các hiệu ứng của Flyweight pattern đòi hỏi rất nhiều vào việc nó được sử dụng ở đâu và sử dụng nó như thế nào. Sử dụng Flyweight pattern khi tất cả cá điều sau đây là đúng:

- Một ứng dụng sử dụng một lượng lớn các đối tượng.
- Giá thành lưu trữ rất cao bởi số lượng các đối tượng là rất lớn.
- Hầu hết trạng thái của các đối tượng có thể chịu tác động từ bên ngoài.
- Ứng dụng không yêu cầu đối tượng đồng nhất. Khi các đối tượng flyweight có thể bị phân tách, việc kiểm tra tính đồng nhất sẽ trả về đúng cho các đối tượng được định nghĩa dựa trên các khái niệm khác nhau.

Cấu trúc mẫu



Trong đó:

FlyweightFactory: tạo ra và quản lý các đối tượng Flyweight

Flyweight là một giao tiếp định nghĩa các phương thức chuẩn

ConcreteFlyweightX: là các lớp thực thi của Flyweight, các thể hiện của các lớp này sẽ được sử dụng tùy thuộc vào điều kiện ngoài.

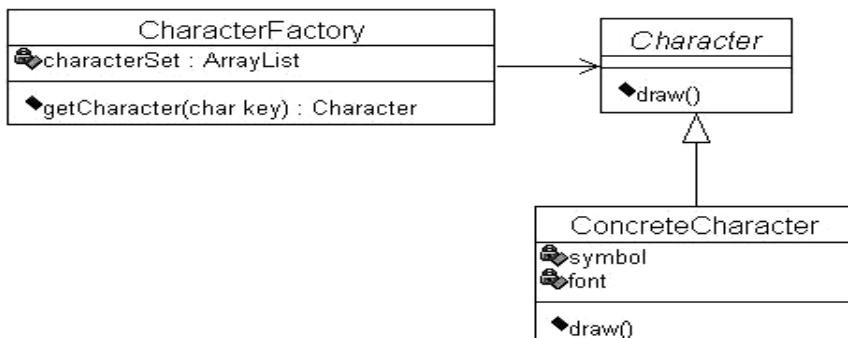
Trường hợp sử dụng

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

- Ứng dụng sử dụng nhiều đối tượng giống hoặc gần giống nhau
- Với các đối tượng gần giống nhau, những phần không giống nhau có thể tách rời với các phần giống nhau để cho phép các phần giống nhau có thể chia sẻ
- Nhóm các đối tượng gần giống nhau có thể được thay thế bởi một đối tượng chia sẻ mà các phần không giống nhau đã được loại bỏ
- Nếu ứng dụng cần phân biệt các đối tượng gần giống nhau trong trạng thái gốc của chúng

Ví dụ

Một ví dụ cổ điển của mẫu flyweight là các kí tự được lưu trong một bộ xử lý văn bản (word processor). Mỗi kí tự đại diện cho một đối tượng mà có dữ liệu là loại font (font face), kích thước font, và các dữ liệu định dạng khác. Bạn có thể tưởng tượng là, với một tài liệu (document) lớn với cấu trúc dữ liệu như thế này thì sẽ bộ xử lý văn bản sẽ khó mà có thể xử lý được. Hơn nữa, vì hầu hết dữ liệu dạng này là lặp lại, phải có một cách để giảm việc lưu giữ này - đó chính là mẫu Flyweight. Mỗi đối tượng kí tự sẽ chứa một tham chiếu đến một đối tượng định dạng riêng rẽ mà chính đối tượng này sẽ chứa các thuộc tính cần thiết. Điều này sẽ giảm một lượng lớn sự lưu giữ bằng cách kết hợp mọi kí tự có định dạng giống nhau trở thành các đối tượng đơn chỉ chứa tham chiếu đến cùng một đối tượng đơn chứa định dạng chung đó.



Giao tiếp Character định nghĩa phương thức vẽ một kí tự

```
public interface Character {  
    public void draw();  
}
```

Lớp kí tự thực thi từ giao tiếp Character. Sở dĩ ta định nghĩa Character và ConcreteCharacter riêng là vì trong cấu trúc sẽ có một phần nữa: phần không giống nhau giữa các đối tượng Character (mà ta không bàn tới)

```
public class ConcreteCharacter implements Character {
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
private String symbol;
private String font;
public ConcreteCharacter(String s, String f) {
    this.symbol = s;
    this.font = f;
}
public void draw() {
    System.out.println("Symbol " + this.symbol + "
with font " + this.font );
}
}
```

Lớp khởi tạo các kí tự theo symbol và font, mỗi lần khởi tạo nó sẽ lưu các đối tượng này vào vùng nhớ riêng của nó, nếu đối tượng ký tự symbol + font đã có thì nó sẽ lấy ra chứ không khởi tạo lại

```
import java.util.*;
public class CharacterFactory {
    private Hashtable pool = new Hashtable<String,
Character>();

    public int getNum() {
        return pool.size();
    }

    public Character get(String symbol, String fontFace)
{
    Character c;
    String key = symbol + fontFace;
    if ((c = (Character)pool.get(key)) != null) {
        return c;
    } else {
        c = new ConcreteCharacter(symbol,
fontFace);
        pool.put(key, c);
        return c;
    }
}
Lớp Test
```

```

import java.util.*;
public class Test {
    public static void main(String[] agrs) {
        CharacterFactory characterfactory = new
CharacterFactory();
        ArrayList<Character> text = new
ArrayList<Character>();
        text.add(0, characterfactory.get("a",
"arial"));
        text.add(1, characterfactory.get("b", "time"));
        text.add(2, characterfactory.get("a",
"arial"));
        text.add(0, characterfactory.get("c",
"arial"));
        for (int i = 0; i < text.size(); i++) {
            Character c = (Character)text.get(i);
            c.draw();
        }
    }
}

```

Như vậy 'a' + 'arial' gọi 2 lần như chỉ khởi tạo có 1 lần mà thôi.

Mẫu liên quan

Mẫu Flyweight thường kết hợp với mẫu Composite để cài đặt cấu trúc phân cấp lôgic trong giới hạn của một sơ đồ vòng xoắn trực tiếp với các lá chia sẻ của nó.

Thường tốt nhất là cài đặt các mẫu State và Strategy giống như là flyweight.

Proxy

Đặt vấn đề

Lý do để điều khiển truy nhập tới một đối tượng là làm theo toàn bộ quá trình tạo ra và khởi đầu của nó cho tới tận khi thực sự chúng ta cần sử dụng nó. Trong trường hợp này ta nên dùng mẫu thiết kế proxy. Proxy có thể được ứng dụng tại bất cứ nơi nào mà ở đó cần có một sự tham chiếu tới một đối tượng linh hoạt hơn, tinh xảo hơn so với việc sử dụng một con trỏ đơn giản.

Sau đây là một vài trường hợp thông thường trong đó proxy được vận dụng:

- Một remote proxy cung cấp một biểu diễn (một mẫu) cục bộ cho một đối tượng trong một không gian địa chỉ khác.
- Một virtual proxy tạo ra một đối tượng có chi phí cao theo yêu cầu.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

- Một protection proxy điều khiển việc truy nhập đối tượng gốc. Các protection proxy rất hữu ích khi các đối tượng có các quyền truy nhập khác nhau.
- Một smart reference là sự thay thế cho một con trỏ rỗng cho phép thực hiện các chức năng thêm vào khi một đối tượng được truy nhập.

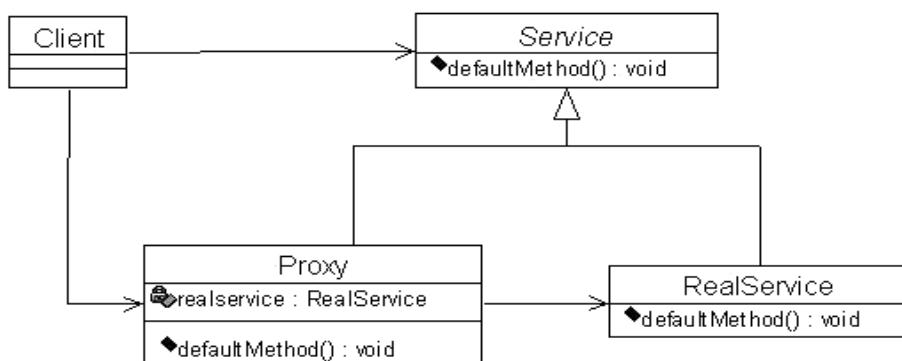
Các trường hợp sử dụng điển hình:

- Đếm số tham chiếu tới đối tượng thực, do đó nó có thể tự động giải phóng khi có không nhiều các tham chiếu.
- Tải một đối tượng liên tục vào bộ nhớ khi nó được tham chiếu lần đầu tiên.
- Kiểm tra đối tượng thực đó có được khóa hay không trước khi nó bị truy nhập để đảm bảo không đối tượng nào khác có thể thay đổi nó.

Định nghĩa

Cung cấp một đại diện cho một đối tượng khác để điều khiển việc truy nhập nó.

Cấu trúc mẫu



Trong đó:

Service: là giao tiếp định nghĩa các phương thức chuẩn cho một dịch vụ nào đó

RealService: là một thực thi của giao tiếp Service, lớp này sẽ báo tường minh các phương thức của Service, lớp này xem như thực hiện tất cả các yêu cầu từ Service

Proxy: kế thừa Service và sử dụng đối tượng của RealService

Trường hợp ứng dụng

- Sử dụng mẫu Proxy khi bạn cần một tham chiếu phức tạp đến một đối tượng thay vì chỉ một cách bình thường
- Remote proxy – sử dụng khi bạn cần một tham chiếu định vị cho một đối tượng trong không gian địa chỉ(JVM)
- Virtual proxy – lưu giữ các thông tin thêm vào về một dịch vụ thực vì vậy chúng có thể hoãn lại sự truy xuất vào dịch vụ này

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

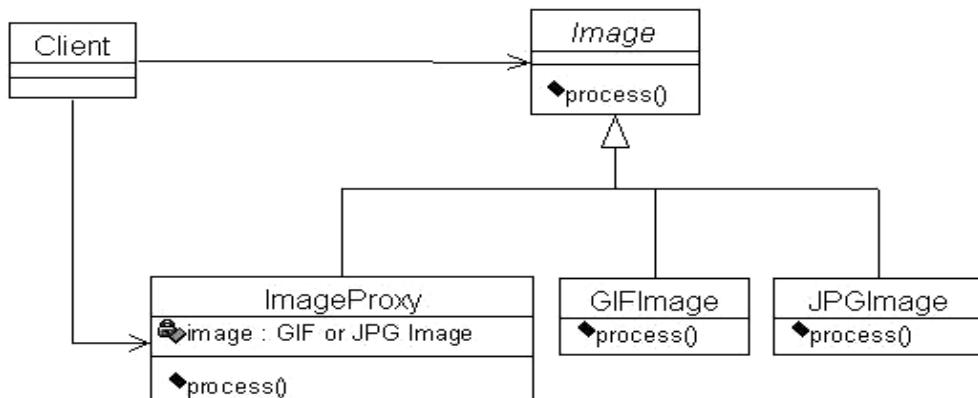
- Protection proxy – xác thực quyền truy xuất vào một đối tượng thực

Ví dụ

Ví dụ lớp Image là một interface định nghĩa các phương thức xử lý ảnh, nó có các lớp con là GIFImage và JPGImage.

Theo hướng đối tượng thì thiết kế như thế có vẻ hợp lý, Client chỉ cần sử dụng lớp Image là đủ, còn tùy thuộc vào loại ảnh sẽ có các phương thức khác nhau. Nhưng trong trường hợp, trên ứng dụng web chẳng hạn, một lúc ta đọc lên hàng trăm ảnh các loại và ta còn muốn xử lý tùy vào một điều kiện nào đó (ví dụ chỉ xử lý khi là ảnh JPG hoặc GIF). Nếu ta đặt điều kiện IF Image (sau đó sẽ tùy điều kiện này rồi xử lý riêng) thì không hợp lý, nếu đặt trong Client, nếu mỗi lần cần thay đổi IF ta lại sửa Client thì không hợp lý khi Client là một ứng dụng lớn.

Ta sử dụng Proxy, lớp ImageProxy chỉ là lớp đại diện cho Image, kế thừa từ Image và sử dụng các lớp GIFImage hay JPGImage. Khi cần thay đổi ta chỉ cần thay đổi trên Proxy mà không cần tác động đến Client và Image.



```
public interface Image {
    public void process();
}

public class JPGImage implements Image {
    public void process() {
        System.out.print("JPG Image");
    }
}

public class GIFImage implements Image {
    public void process() {
        System.out.print("GIF Image");
    }
}
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
public class ImageProxy implements Image {  
    private Image image;  
    public void process() {  
        if (image == null)  
            image = new JPGImage(); //tạo đối tượng ảnh JPG,  
        chỉ mang tính minh họa  
        image.process();  
    }  
}
```

Mẫu liên quan

Một Adapter cung cấp một giao diện khác với đối tượng mà nó thích nghi. Trong trường hợp này, một Proxy cung cấp cùng một giao diện như vậy giống như một chủ thẻ. Mặc dù decorator có thể có cài đặt tương tự như các proxy, decorator có một mục đích khác. Một decorator phụ thêm nhiều nhiệm vụ cho một đối tượng nhưng ngược lại một proxy điều khiển truy cập đến một đối tượng. Proxy tùy biến theo nhiều cấp khác nhau mà chúng có thể sẽ được cài đặt giống như một decorator. Một proxy bảo vệ có thể được cài đặt chính xác như một decorator. Mặt khác một proxy truy cập đối tượng từ xa sẽ không chứa một tham chiếu trực tiếp đến chủ thẻ thực sự nhưng chỉ duy nhất có một tham chiếu trực tiếp, giống như ID của host và địa chỉ trên host vậy. Một proxy ảo sẽ bắt đầu với một tham chiếu gián tiếp chẳng hạn như tên file nhưng rốt cuộc rồi cũng sẽ đạt được một tham chiếu trực tiếp.

9.4.3 Nhóm Behavior

Các mẫu hành vi là các mẫu tập trung vào vấn đề giải quyết các thuật toán và sự phân chia trách nhiệm giữa các đối tượng. Mẫu hành vi không chỉ miêu tả mô hình của các đối tượng mà còn miêu tả mô hình trao đổi thông tin giữa chúng; đặc trưng hoá các luồng điều khiển phức tạp, giúp chúng ta tập trung hơn vào việc xây dựng cách thức liên kết giữa các đối tượng thay vì các luồng điều khiển.

Mẫu hành vi kiểu lớp sử dụng tính kế thừa để phân phối hành vi giữa các lớp. Dưới đây chúng ta sẽ nghiên cứu hai mẫu thuộc loại đó : Temple Method và Interpreter. Trong hai mẫu đó, Temple Method là mẫu đơn giản và thông dụng hơn. Nó định nghĩa trừu tượng từng bước của một thuật toán; mỗi bước sử dụng một hàm trừu tượng hoặc một hàm nguyên thuỷ. Các lớp con của nó làm sáng tỏ thuật toán cụ thể bằng cách cụ thể hoá các hàm trừu tượng. Mẫu Interpreter diễn tả văn phạm như một hệ thống phân cấp của các lớp và trình phiên dịch như một thủ tục tác động lên các thể hiện của các lớp đó.

Mẫu hành vi kiểu đối tượng lại sử dụng đối tượng thành phần thay vì thừa kế. Một vài mẫu miêu tả cách thức một nhóm các đối tượng ngang hàng hợp tác với nhau để thi hành các tác vụ mà không một đối tượng riêng lẻ nào có thể tự thi hành. Một vấn đề

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

quan trọng được đặt ra ở đây là bằng cách nào các đối tượng ngang hàng đó biết được sự tồn tại của nhau. Cách đơn giản nhất và cũng “dư thừa” nhất là lưu trữ các tham chiếu trực tiếp đến nhau trong các đối tượng ngang hàng. Mẫu Mediator tránh sự thừa thãi này bằng cách xây dựng kết nối trung gian, liên kết gián tiếp các đối tượng ngang hàng.

Mẫu Chain of Responsibility xây dựng mô hình liên kết thậm chí còn “lỏng” hơn. Nó cho phép gửi yêu cầu đến một đối tượng thông qua chuỗi các đối tượng “ứng cử”. Mỗi ứng cử viên có khả năng thực hiện yêu cầu tùy thuộc vào các điều kiện trong thời gian chạy. Số lượng ứng cử viên là một con số mở và ta có thể lựa chọn những ứng cử viên nào sẽ tham gia vào chuỗi trong thời gian chạy.

Mẫu Observer xây dựng và vận hành một sự phụ thuộc giữa các đối tượng. Một ví dụ cụ thể của mẫu này là mô hình Model/View/Controller của Smalltalk trong đó tất cả các views của model đều được thông báo khi trạng thái của model có sự thay đổi.

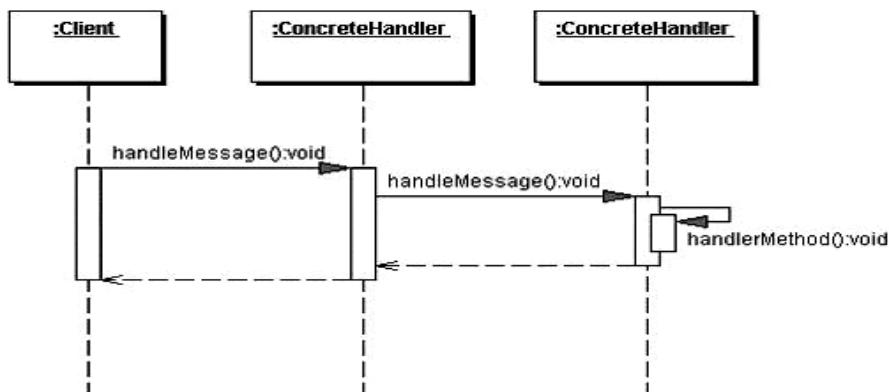
Một số mẫu hành vi khác lại quan tâm đến việc đóng gói các hành vi vào một đối tượng và “uỷ thác” các yêu cầu cho nó. Mẫu Strategy đóng gói một thuật toán trong một đối tượng, xây dựng cơ chế khiến cho việc xác định và thay đổi thuật toán mà đối tượng sử dụng trở nên đơn giản. Trong khi đó mẫu Command lại đóng gói một yêu cầu vào một đối tượng; làm cho nó có thể được truyền như một tham số, được lưu trữ trong một history list hoặc thao tác theo những cách thức khác. Mẫu State đóng gói các trạng thái của một đối tượng, làm cho đối tượng có khả năng thay đổi hành vi của mình khi trạng thái thay đổi. Mẫu Visitor đóng gói các hành vi vốn được phân phối trong nhiều lớp và mẫu Iterator trừu tượng hoá cách thức truy cập và duyệt các đối tượng trong một tập hợp.

Daaychuyền trách nhiệm (Chain of Responsibility)

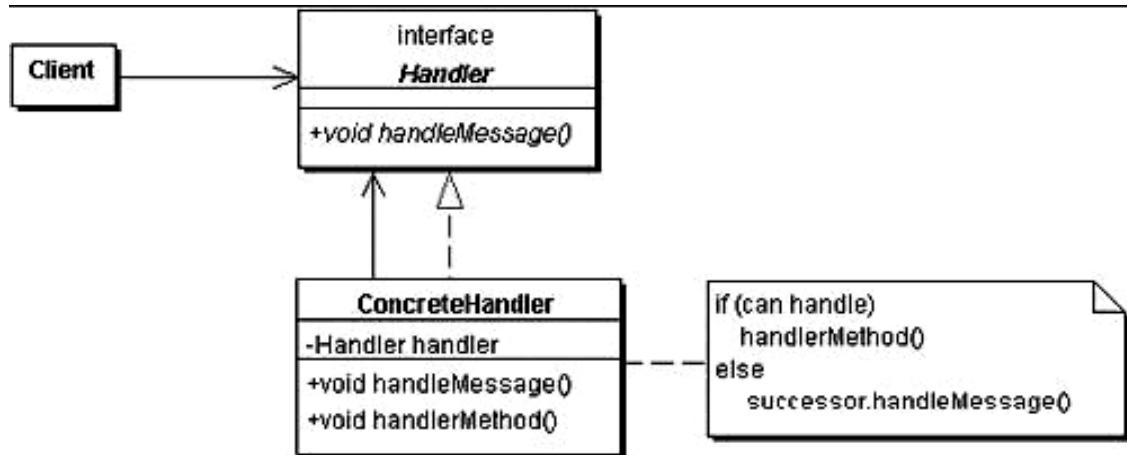
Đặt vấn đề

Mẫu này thiết lập một chuỗi bên trong một hệ thống, nơi mà các thông điệp hoặc có thể được thực hiện ở tại một mức nơi mà nó được nhận lần đầu hoặc là được chuyển đến một đối tượng mà có thể thực hiện điều đó.

Cấu trúc mẫu



CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Trong đó:

Handler: là một giao tiếp định nghĩa phương thức sử dụng để chuyển thông điệp qua các lần thực hiện tiếp theo.

ConcreteHandler: là một thực thi của giao tiếp **Handler**. Nó giữ một tham chiếu đến một **Handler** tiếp theo. Việc thực thi phương thức `handleMessage` có thể xác định làm thế nào để thực hiện phương thức và gọi một `handlerMethod`, chuyển tiếp thông điệp đến cho **Handler** tiếp theo hoặc kết hợp cả hai

Trường hợp ứng dụng

- Có một nhóm các đối tượng trong một hệ thống có thể đáp ứng tất cả các loại thông điệp giống nhau
- Các thông điệp phải được thực hiện bởi một vài các đối tượng trong hệ thống
- Các thông điệp đi theo mô hình “thực hiện – chuyển tiếp”, một vài sự kiện có thể được thực hiện tại mức mà chúng được nhận hoặc tại ra, trong khi số khác phải được chuyển tiếp đến một vài đối tượng khác

Ví dụ

Hệ thống quản lý thông tin các nhân có thể được sử dụng để quản lý các dự án như là các liên hệ.

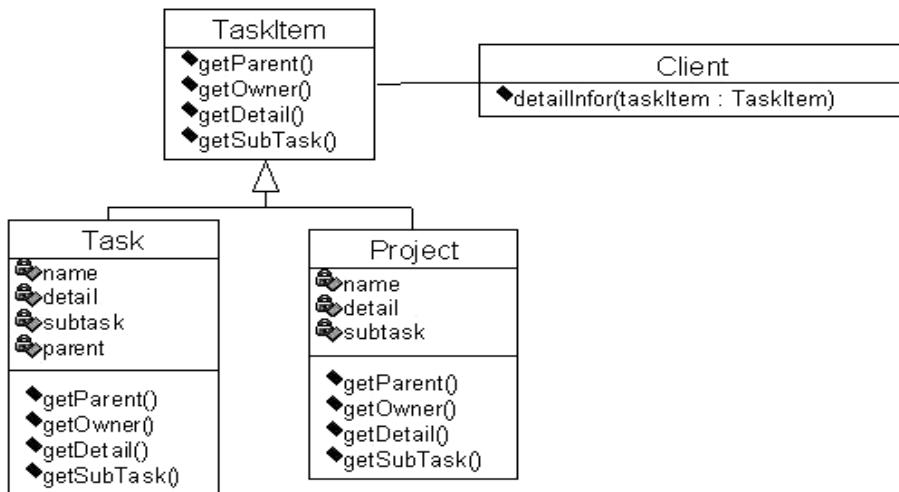
Ta hình dung một cấu trúc cây như sau: đỉnh là một dự án, các đỉnh con là các tác vụ của dự án đó, và cứ như vậy, mỗi đỉnh con tác vụ lại có một tập các đỉnh con tác vụ khác.

Để quản lý cấu trúc này ta thực hiện như sau:

- Ở mỗi đỉnh ta lưu các thông tin như sau: tên tác vụ, đỉnh cha, tập các đỉnh con
- Ta xét thông điệp sau: duyệt từ đỉnh gốc (project cốt sở) in ra các thông tin

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

- Như vậy với thông điệp này, việc in thông tin ở một đỉnh là chưa đủ, ta phải chuyển tiếp đến in thông tin các đỉnh con của đỉnh gốc và chuyển tiếp cho đến khi không còn đỉnh con thì mới dừng



Giao tiếp TaskItem định nghĩa các phương thức cho project cơ sở và các tác vụ

```

public interface TaskItem {
    public TaskItem getParent();
    public String getDetails();
    public ArrayList getProjectItems();
}
  
```

Lớp Project thực thi giao tiếp TaskItem, nó là lớp đại diện cho các đỉnh gốc trên cùng của cây

```

public class Project implements TaskItem {
    private String name;
    private String details;
    private ArrayList subtask = new ArrayList();

    public Project() { }
    public Project(String newName, String newDetails) {
        name = newName;
        details = newDetails;
    }
    public String getName() {
        return name;
    }
    public String getDetails() {
        return details;
    }
}
  
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
    }
    public ProjectItem getParent() {
        return null;
        //vì project là ở mức cơ sở, đỉnh gốc trên cùng nên
        không có cha
    }
    public ArrayList getSubTask() {
        return subtask;
    }

    public void setName(String newName) {
        name = newName;
    }
    public void setDetails(String newDetails) {
        details = newDetails;
    }

    public void addTask(TaskItem element) {
        if (!subtask.contains(element)){
            subtask.add(element);
        }
    }

    public void removeProjectItem(TaskItem element) {
        subtask.remove(element);
    }
}
```

Lớp Task thực thi giao tiếp TaskItem đại diện cho các tác vụ, các đỉnh không phải ở gốc của cây

```
public class Task implements TaskItem {
    private String name;
    private ArrayList subtask = new ArrayList();
    private String details;
    private TaskItem parent;

    public Task(TaskItem newPassword) {
        this(newPassword, "", "");
    }
    public Task(TaskItem newPassword, String newName, String
newDetails,) {
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
parent = newParent;
name = newName;
details = newDetails;
}

public String getDetails() {
    if (primaryTask) {
        return details;
    }
    else{
        return parent.getDetails() + EOL_STRING + "\t"
+ details;
    }
}

public String getName() {
    return name;
}
public ArrayList getSubTask(){ return subtask; }
public ProjectItem getParent() {
    return parent;
}

public void setName(String newName) {
    name = newName;
}
public void setParent(TaskItem newParent) {
    parent = newParent;
}
public void setDetails(String newDetails) {
    details = newDetails;
}

public void addSubTask(TaskItem element) {
    if (!subtask.contains(element)){
        subtask.add(element);
    }
}

public void removeSubTask(TaskItem element) {
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
        subtask.remove(element);  
    }  
}
```

Lớp thực thi test mẫu

```
public class RunPattern {  
    public static void main(String [] arguments){  
        Project project = new Project("Project 01", "Detail  
of Project 01");  
        //Khởi tạo, thiết lập các tác vụ con ...  
        detailInfor(project);  
    }  
    private static void detailInfor (TaskItem item){  
        System.out.println("TaskItem: " + item);  
        System.out.println(" Details: " +  
item.getDetails());  
        System.out.println();  
        if (item.getSubTask() != null){  
            Iterator subElements =  
item.getSubTask().iterator();  
            while (subElements.hasNext()){  
                detailInfor ((TaskItem)subElements.next());  
            }  
        }  
    }  
}
```

Các mẫu liên quan

Dây chuyền trách nhiệm thường được kết hợp trong mối quan hệ với composite. Có một thành phần cha có thể hành động như là successor của nó.

Command

Đặt vấn đề

Đôi khi chúng ta gặp tình huống trong đó cần phải gửi yêu cầu đến một đối tượng mà không biết cụ thể về đối tượng nhận yêu cầu cũng như phương thức xử lý yêu cầu. Lấy ví dụ về bộ công cụ giao diện người sử dụng cho phép xây dựng các menu. Rõ ràng, nó không thể xử lý trực tiếp yêu cầu trên các đối tượng menu vì cách thức xử lý phụ thuộc vào từng ứng dụng cụ thể.

Mẫu Command cho phép bộ công cụ như trên gửi yêu cầu đến các đối tượng chưa xác định bằng cách biến chính yêu cầu thành một đối tượng. Khái niệm quan trọng nhất

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

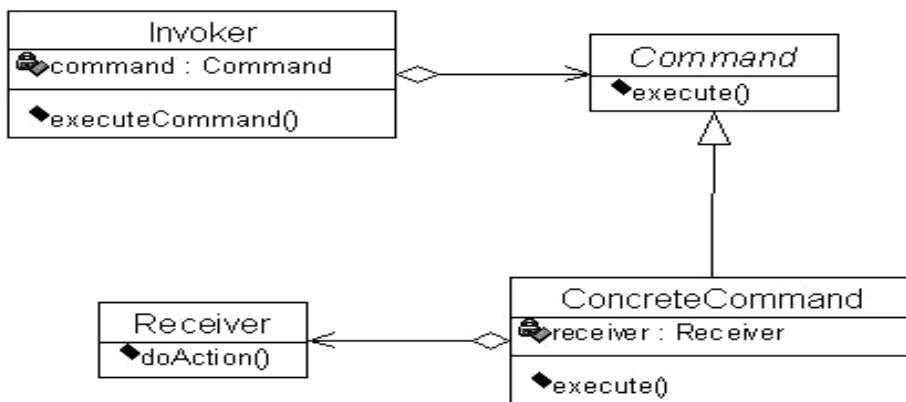
của mẫu này là lớp trừu tượng Command có chức năng định nghĩa giao diện cho việc thi hành các yêu cầu. Trong trường hợp đơn giản nhất, ta chỉ cần định nghĩa một thủ tục ảo Execute trên giao diện đó. Các lớp con cụ thể của Command sẽ xác định cặp đối tượng nhận yêu cầu-các thao tác bằng cách lưu trữ một tham chiếu đến đối tượng nhận yêu cầu và định nghĩa lại thủ tục Execute để gọi các thủ tục xử lý.

Theo cách này, chương trình sẽ có nhiệm vụ tạo ra các menu, menuitem và gán mỗi menuitem với một đối tượng thuộc lớp con cụ thể của Command. Khi người sử dụng chọn một menuitem, nó sẽ gọi hàm command->Execute() mà không cần con trỏ command trả về đến loại lớp con cụ thể nào của lớp Command. Hàm Execute() sẽ có nhiệm vụ xử lý yêu cầu.

Định nghĩa

Mẫu Command đóng gói yêu cầu như là một đối tượng, làm cho nó có thể được truyền như một tham số, được lưu trữ trong một history list hoặc thao tác theo những cách thức khác nhau.

Cấu trúc mẫu



Trong đó:

Command: là một giao tiếp định nghĩa các phương thức cho Invoker sử dụng

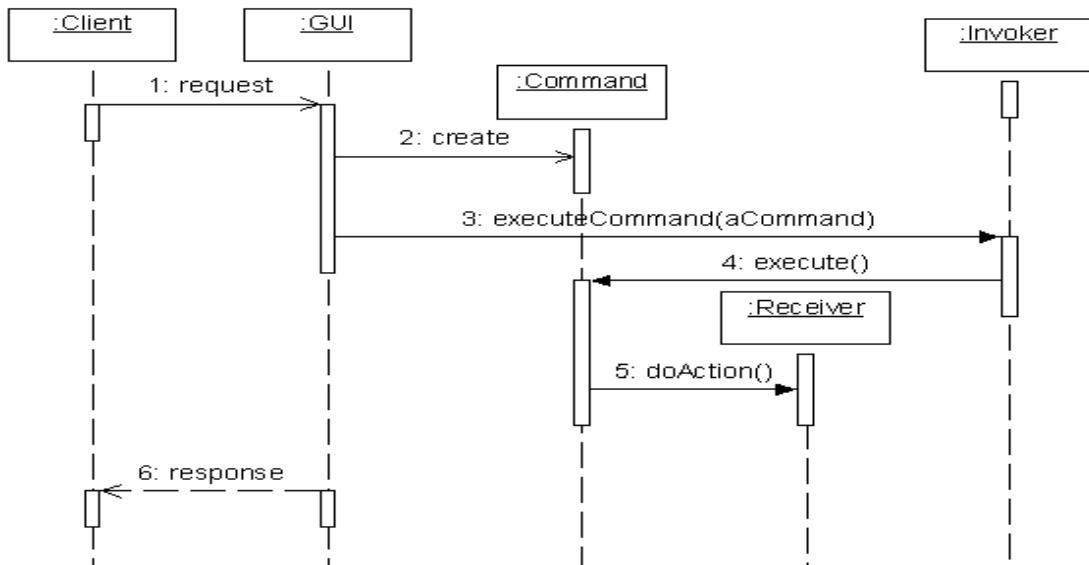
Invoker: lớp này thực hiện các phương thức của đối tượng Command

Receiver: là đích đến của Command và là đối tượng thực hiện hoàn tất yêu cầu, nó có tất cả các thông tin cần thiết để thực hiện điều này

ConcreteCommand: là một thực thi của giao tiếp Command. Nó lưu giữ một tham chiếu Receiver mong muốn.

Luồng thực thi của mẫu Command như sau:

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Client gửi yêu cầu đến GUI của ứng dụng

Ứng dụng khởi tạo một đối tượng Command thích hợp cho yêu cầu đó (đối tượng này sẽ là các ConcreteCommand)

Sau đó ứng dụng gọi phương thức executeCommand() với tham số là đối tượng Command vừa khởi tạo

Invoker khi được gọi thông qua phương thức executeCommand() sẽ thực hiện gọi phương thức execute() của đối tượng Command tham số

Đối tượng Command này sẽ gọi tiếp phương thức doAction() của thành phần Receiver của nó, được khởi tạo từ đầu, doAction() chính là phương thức chính để hoàn tất yêu cầu của Client.

Ứng dụng

Dùng mẫu Command khi :

- Tham số hoá các đối tượng theo thủ tục mà chúng thi hành, như đối tượng MenuItem ở trên.
- Xác định, xếp hàng và thi hành các yêu cầu tại những thời điểm khác nhau.
- Cho phép quay ngược. Thủ tục Execute của lớp Command có thể lưu lại trạng thái để cho phép quay ngược các biến đổi mà nó gây ra. Khi đó lớp Command trừu tượng cần định nghĩa thêm hàm Unexecute để đảo ngược các biến đổi. Các commands đã được thi hành sẽ được lưu trong một danh sách, từ đó cho phép undo và redo không giới hạn mức.
- Cần hỗ trợ ghi lại các commands đã được thi hành để thi hành lại trong trường hợp hệ thống gặp sự cố.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

- Thiết kế một hệ thống với các thủ tục bậc cao được xây dựng dựa trên các thủ tục nguyên thuỷ. Cấu trúc này thường gặp trong các hệ thống thông tin hỗ trợ “phiên giao dịch”. Một phiên giao dịch là một tập hợp các sự thay đổi lên dữ liệu. Mẫu Command cung cấp cách thức mô tả phiên giao dịch. Nó có giao diện chung cho phép khởi xướng các phiên làm việc với cùng một cách thức và cũng cho phép dễ dàng mở rộng hệ thống với các phiên giao dịch mới

Ví dụ

```
public interface Command{  
    public void execute();  
}  
  
public class ConcreteCommand implements Command{  
    private Receiver receiver;  
    public void setReceiver(Receiver receiver) {  
        this.receiver = receiver;  
    }  
    public Receiver getReceiver() {  
        return this.receiver;  
    }  
  
    public void execute () {  
        receiver.doAction();  
    }  
}  
  
public class Receiver{  
    private String name;  
    public Receiver(String name) {  
        this.name = name  
    }  
    public void doAction(){  
        System.out.print(this.name + " fulfill  
request!");  
    }  
}  
  
public class Invoker{  
    public void executeCommand(Command command) {  
        command.execute();  
    }  
}
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

}

```
public class Run{  
    public static void main(String[] agrs){  
        Command command = new ConcreteCommand();  
        command.setReceiver(new Receiver("NguyenD"));  
        Invoker invoker = new Invoker();  
        Invoker.executeCommand(command);  
    }  
}
```

Các mẫu liên quan

Một Composite có thể được sử dụng để cài đặt các MacroCommands. Một Memmento có thể lưu lại các trạng thái để Command yêu cầu phục hồi lại các hiệu ứng của nó. Một command phải được sao lưu trước khi nó được thay thế bằng các hành động trước đó như là một Prototype.

Iterator

Đặt vấn đề

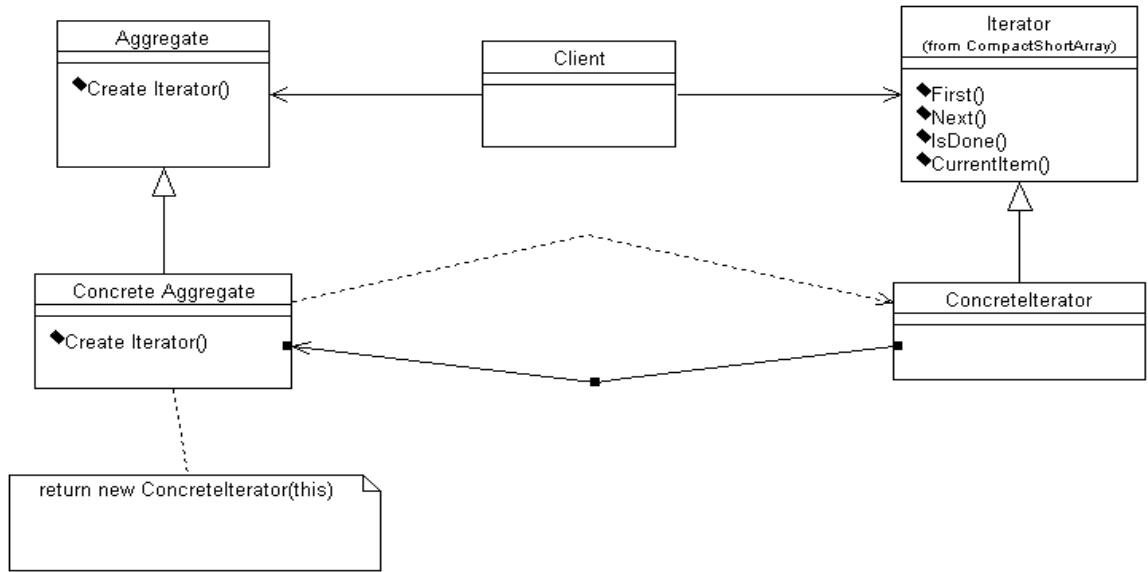
Một đối tượng tập hợp như là một danh sách cũng cung cấp cho ta các phương thức truy cập các thành phần của nó. Tuy nhiên đôi lúc chúng ta cần duyệt các thành phần của danh sách theo những cách thức và tiêu chí khác nhau. Chúng ta không nên làm phồng giao diện của danh sách List với các phương thức cho các cách thức duyệt.

Mẫu Iterator cho phép chúng ta duyệt danh sách dễ dàng bằng cách tách rời chức năng truy cập và duyệt ra khỏi danh sách và đặt vào đối tượng iterator. Lớp Iterator sẽ định nghĩa một giao diện để truy cập các thành phần của danh sách, đồng thời quản lý cách thức duyệt danh sách hiện thời.

Định nghĩa: Mẫu Iterator cung cấp khả năng truy cập và duyệt các thành phần của một tập hợp không cần quan tâm đến cách thức biểu diễn bên trong.

Cấu trúc mẫu

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Ích lợi

Hỗ trợ nhiều phương án duyệt một tập hợp.

Đơn giản hóa giao diện tập hợp.

Cho phép có lớn hơn một cách thức duyệt đối với một tập hợp tại một thời điểm.

Ứng dụng trong các trường hợp sau

Truy cập các thành phần của một tập hợp mà không cần quan tâm đến cách thức biểu diễn bên trong.

Hỗ trợ nhiều phương án duyệt của các đối tượng tập hợp.

Cung cấp giao diện chung cho việc duyệt các cấu trúc tập hợp.

Các mẫu liên quan

Iterator thường được sử dụng để duyệt một cấu trúc đệ quy như Composite.

Đa hình một Iterator dựa trên FactoryMethod để tạo thể nghiệm cho các lớp con tương ứng của Iterator.

Memento thường được sử dụng cùng với Iterator. Một Iterator có thể sử dụng một Memento để nắm bắt trạng thái của một Iterator. Iterator lưu trữ các memento ở bên trong.

Interpreter

Đặt vấn đề

Nếu một dạng bài toán có tần suất xuất hiện tương đối lớn, người ta thường biểu diễn các thể hiện cụ thể của nó bằng các câu trong một ngôn ngữ đơn giản. Sau đó ta có thể xây dựng một trình biên dịch để giải quyết bài toán bằng cách biên dịch các câu.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

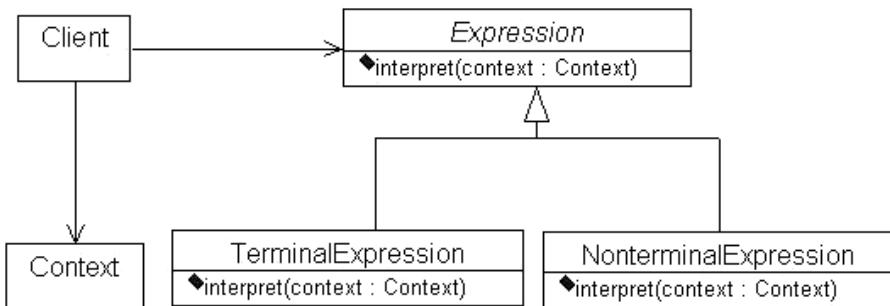
Lấy ví dụ, tìm kiếm các xâu thoả mãn một mẫu cho trước là một bài toán thường gặp và các “biểu diễn thông thường” tạo thành ngôn ngữ dùng để diễn tả các mẫu của xâu. Thay vì xây dựng từng thuật toán riêng biệt để tương ứng mỗi mẫu với một tập các xâu, người ta xây dựng thuật toán tổng quát có thể phiên dịch các “biểu diễn thông thường” thành tập các xâu tương ứng.

Mẫu Interpreter miêu tả cách thức xây dựng cấu trúc ngữ pháp cho những ngôn ngữ đơn giản, cách thức biểu diễn câu trong ngôn ngữ và cách thức phiên dịch các câu đó.

Định nghĩa

Interpreter đưa ra một ngôn ngữ, xây dựng cách diễn đạt ngôn ngữ đó cùng với một trình phiên dịch sử dụng cách diễn tả trên để phiên dịch các câu.

Cấu trúc mẫu



Trong đó:

Expression: là một giao tiếp mà thông qua nó, client tương tác với các biểu thức

TerminalExpression: là một thực thi của giao tiếp Expression, đại diện cho các nốt cuối trong cây cú pháp

NonterminalExpression: là một thực thi khác của giao tiếp Expression, đại diện cho các nút chưa kết thúc trong cấu trúc của cây cú pháp. Nó lưu trữ một tham chiếu đến Expression và triệu gọi phương thức diễn giải cho mỗi phần tử con

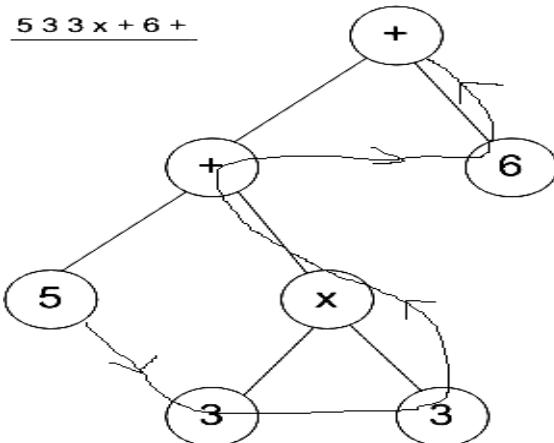
Context: chứa thông tin cần thiết cho một vài vị trí trong khi diễn giải. Nó có thể phục vụ như một kênh truyền thông cho các thể hiện của Expression

Client: hoặc là xây dựng hoặc là nhận một thể hiện của cây cú pháp ảo. Cây cú pháp này bao gồm các thể hiện của TerminalExpression và NonterminalExpression để tạo nên câu đặc tả. Client triệu gọi các phương thức diễn giải với ngữ cảnh thích hợp khi cần thiết.

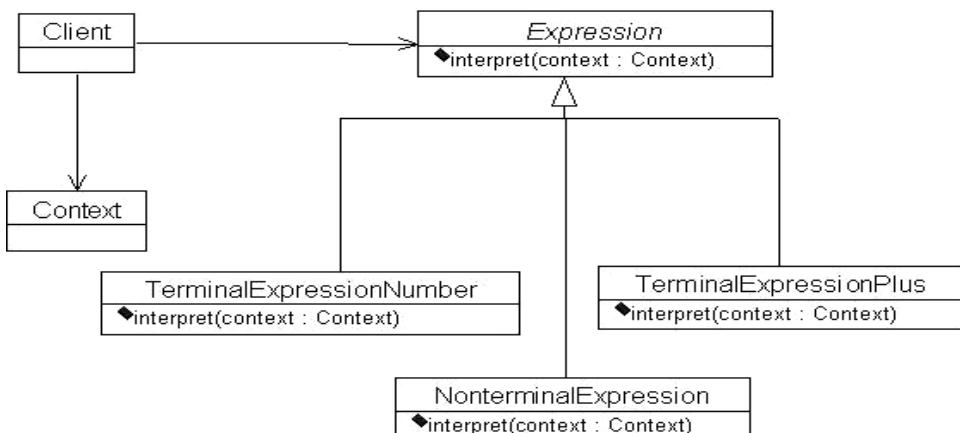
Ví dụ

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Tính kết quả của biểu thức $5 + 3 \times 3 + 6$. Bài toán này ta có thể chia thành các bài các bài toán nhỏ hơn: Tính $3 \times 3 = a$; Sau đó tính $5 + a = b$; Sau đó tính $b + 6$. Ta biểu diễn bài toán thành cấu trúc cây và duyệt cây.



Mã nguồn cho ví dụ này như sau:



```

import java.util.Stack;
public class Context extends Stack<Integer>{
}
public interface Expression {
    public void interpret(Context context);
}
public class TerminalExpressionNumber implements Expression {
    private int number;
    public TerminalExpressionNumber(int number) {
        this.number = number;
    }
    public void interpret(Context context) {
        context.push(this.number);
    }
}

```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
    }

}

public class TerminalExpressionPlus implements Expression {
    public void interpret(Context context) {
        //Cong 2 phan tu phia tren dinh Stack
        context.push(context.pop() +
context.pop());
    }
}

public class TerminalExpressionMutil implements Expression{
    public void interpret(Context context) {
        //Nhan 2 phan tu phia tren dinh Stack
        context.push(context.pop() *
context.pop());
    }
}

import java.util.ArrayList;
public class NonterminalExpression implements Expression {
    private ArrayList<Expression> expressions;//them
chieu den mang Expression con
    public ArrayList<Expression> getExpressions() {
        return expressions;
    }
    public void setExpressions(ArrayList<Expression>
expressions) {
        this.expressions = expressions;
    }

    public void interpret(Context context) {
        if (expressions != null){
            int size = expressions.size();
            for (Expression e : expressions){
                e.interpret(context);
            }
        }
    }
}
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
import java.util.*;
public class Client {
    public static void main(String[] agrs) {
        Context context = new Context();
        // 3 3 *
        ArrayList<Expression> treeLevel1 = new
ArrayList<Expression>();
        treeLevel1.add(new TerminalExpressionNumber(3));
        treeLevel1.add(new TerminalExpressionNumber(3));
        treeLevel1.add(new TerminalExpressionMutil());
        // 5 (3 3 *) +
        ArrayList<Expression> treeLevel2 = new
ArrayList<Expression>();
        treeLevel2.add(new TerminalExpressionNumber(5));
        Expression nonexpLevel1 = new
NonterminalExpression();
        ((NonterminalExpression)nonexpLevel1).setExpressions(t
reeLevel1);
        treeLevel2.add(nonexpLevel1);
        treeLevel2.add(new TerminalExpressionPlus());
        // (5 (3 3 *) +) 6 +
        ArrayList<Expression> treeLevel3 = new
ArrayList<Expression>();
        Expression nonexpLevel2 = new
NonterminalExpression();
        ((NonterminalExpression)nonexpLevel2).setExpression
s(treeLevel2);
        treeLevel3.add(nonexpLevel2);
        treeLevel3.add(new TerminalExpressionNumber(6));
        treeLevel3.add(new TerminalExpressionPlus());

        for(Expression e : treeLevel3){
            e.interpret(context);
        }

        if (context != null)
            System.out.print("Ket qua: " +
context.pop());
    }
}
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Ứng dụng

Sử dụng mẫu Interpreter khi cần phiên dịch một ngôn ngữ mà ta có thể miêu tả các câu bằng cấu trúc cây cú pháp. Mẫu này hoạt động hiệu quả nhất khi:

- Cấu trúc ngữ pháp đơn giản. Với các cấu trúc ngữ pháp phức tạp, cấu trúc lớp của ngữ pháp trở nên quá lớn và khó kiểm soát, việc tạo ra các cây cú pháp sẽ tốn thời gian và bộ nhớ.
- Hiệu quả không phải là yếu tố quan trọng nhất. Các cách thức biên dịch hiệu quả nhất thường không áp dụng trực tiếp mẫu Interpreter mà phải biến đổi các biểu diễn thành các dạng khác trước.

Các mẫu liên quan

Cây cú pháp trùu tượng là một thể nghiệm trong mẫu Composite.

Flyweight chỉ ra cách chia sẻ ký pháp đầu cuối trong phạm vi của cây cú pháp trùu tượng.

Interpreter thường sử dụng một Iterator để duyệt cấu trúc.

Visitor có thể được sử dụng để duyệt hành vi trên mỗi nút trong cây cú pháp trùu tượng của lớp.

Mediator

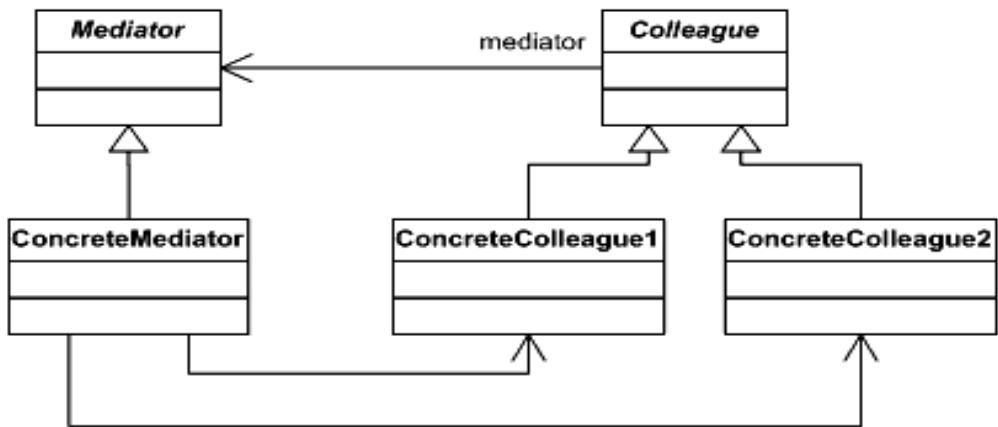
Đặt vấn đề

Cách thiết kế hướng đối tượng khuyến khích việc phân bố các ứng xử giữa các đối tượng. Việc phân chia đó có thể dẫn đến cấu trúc trong đó tồn tại rất nhiều liên kết giữa các đối tượng mà trong trường hợp xấu nhất là tất cả các đối tượng đều liên kết trực tiếp với nhau.

Định nghĩa: Mediator dùng để đóng gói cách thức tương tác của một tập hợp các đối tượng. Giảm bớt liên kết và cho phép thay đổi cách thức tương tác giữa các đối tượng.

Cấu trúc mẫu

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Mediator (IChatroom)

Định nghĩa một giao diện cho việc giao tiếp với đối tượng cộng tác.

ConcreteMediator (Chatroom)

Xây dựng các hành vi tương tác giữa các đối tượng colleague

Xác định các đối tượng colleague

Colleague classes (Participant)

Mỗi lớp Colleague đều xác định đối tượng Mediator tương ứng

Mỗi đối tượng colleague trao đổi với đối tượng mediator khi muốn trao đổi với colleague khác.

Ứng dụng

Mẫu Mediator có những ưu và nhược điểm sau :

Giảm việc chia lớp con

Giảm liên kết giữa các colleagues.

Đơn giản hóa giao thức giữa đối tượng bằng cách thay các tương tác nhiều - nhiều bằng tương tác 1- nhiều giữa nó và các colleagues.

Trừu tượng hoá cách thức tương tác hợp tác giữa các đối tượng. Tạo ra sự tách biệt rõ ràng hơn giữa các tương tác ngoài và các đặc tính trong của đối tượng.

Điều khiển trung tâm: thay sự phức tạp trong tương tác bằng sự phức tạp tại mediator.

Ứng dụng mediator vào trong các trường hợp sau:

Một nhóm các đối tượng trao đổi thông tin một cách rõ ràng nhưng khá phức tạp dẫn đến hệ thống các kết nối không có cấu trúc và khó hiểu.

Việc sử dụng lại một đối tượng gấp khó khăn vì nó liên kết với quá nhiều đối tượng khác.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Cho phép tùy biến một ứng xử được phân tán trong vài lớp mà không phải phân nhiều lớp con.

Các mẫu liên quan

Facade khác với Mediator ở chỗ nó trùu tượng một hệ thống con của các đối tượng để cung cấp một giao diện tiện ích hơn. Giao thức của nó theo một hướng duy nhất đó là, các đối tượng Facade tạo ra các yêu cầu của các lớp hệ thống con nhưng không có chiều ngược lại. Ngược lại Mediator cho phép các hành vi kết hợp mà các đối tượng cộng tác không thể cung cấp và giao thức này là không đa hướng.

Các cộng tác có thể đi giao tiếp với Mediator bằng cách sử dụng mẫu Observer.

Memento

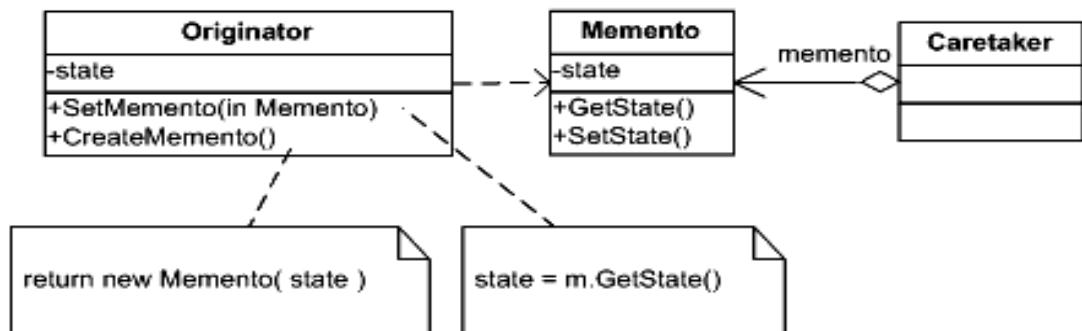
Đặt vấn đề

Đôi lúc, việc lưu lại trạng thái của một đối tượng là cần thiết. Ví dụ khi xây dựng cơ chế checkpoints và undo để phục hồi hệ thống khỏi trạng thái lỗi. Tuy nhiên các đối tượng thường phải che dấu một vài hoặc tất cả các thông tin trạng thái của mình, làm cho chúng không thể được truy cập từ ngoài. Vấn đề này có thể giải quyết với mẫu Memento. Memento là đối tượng có chức năng lưu lại trạng thái nội tại của một đối tượng khác. Cơ chế undo sẽ yêu cầu một memento ban đầu khi nó cần khôi phục lại trạng thái của ban đầu. Cũng chỉ đối tượng ban đầu mới có quyền truy xuất và lưu trữ thông tin vào memento vì nó trong suốt đối với các đối tượng còn lại.

Định nghĩa

Memento là mẫu thiết kế có thể lưu lại trạng thái của một đối tượng để khôi phục lại sau này mà không vi phạm nguyên tắc đóng gói.

Cấu trúc mẫu



Memento

Lưu trữ trạng thái của đối tượng Originator.

Bảo vệ, chống truy cập từ các đối tượng khác originator.

Originator

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Tạo memento chứa hình chụp trạng thái của mình

Sử dụng memento để khôi phục về trạng thái cũ.

Caretaker

Có trách nhiệm quản lý các memento.

Không thay đổi hoặc truy xuất nội dung của memento.

Đặc trưng

Memento có những đặc điểm

Đảm bảo ranh giới của sự đóng gói.

Đơn giản Originator. Trong các thiết kế hỗ trợ khôi phục trạng thái khác, Originator có chức năng lưu trữ các phiên bản trạng thái của mình và do đó phải thi hành các chức năng quản lý lưu trữ.

Sử dụng memento có thể gây ra chi phí lớn nếu Originator có nhiều thông tin trạng thái cần lưu trữ hoặc nếu việc ghi lại và khôi phục trạng thái diễn ra với tần suất lớn.

Việc đảm bảo chỉ originator mới có quyền truy cập memento là tương đối khó xây dựng ở một số ngôn ngữ lập trình.

Chi phí ẩn của việc lưu trữ memento : caretaker có nhiệm vụ quản lý cũng như xoá bỏ các memento nó yêu cầu tạo ra. Tuy nhiên nó không được biết kích thước của memento là bao nhiêu và do đó có thể tồn tại nhiều không gian bộ nhớ khi lưu trữ memento.

Ứng dụng

Cần lưu lại trạng thái nội bộ của một đối tượng để có thể khôi phục về trạng thái đó sau này.

Xây dựng giao diện trực tiếp để truy xuất thông tin trạng thái sẽ phai bầy cấu trúc và phá hỏng tính đóng gói của đối tượng.

Các mẫu liên quan

Các Command có thể sử dụng memento để duy trì trạng thái cho các thao tác có khả năng phục hồi được. Các Memento có thể được sử dụng cho vòng lặp sớm hơn.

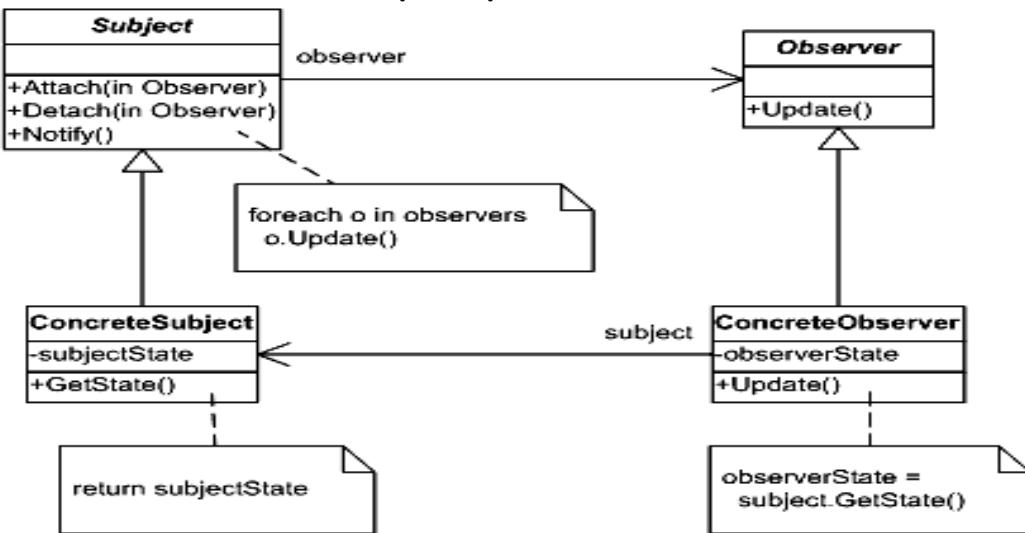
Observer

Định nghĩa

Observer định nghĩa một phụ thuộc 1- nhiều giữa các đối tượng để khi một đối tượng thay đổi trạng thái thì tất cả các phụ thuộc của nó được nhận biết và cập nhật tự động.

Cấu trúc mẫu

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Subject (Stock)

Hiểu về các Observer của nó. Một số lượng bất kỳ Observer có thể theo dấu một chủ đề nào đó.

Cung cấp một giao diện cho việc gắn và tách các đối tượng Observer

ConcreteSubject (IBM)

Lưu trữ trạng thái của ConcreteObserver cần quan tâm.

Gửi tín hiệu đến các observer của nó khi trạng thái của nó đã thay đổi.

Observer (IIInvestor)

Định nghĩa một giao diện cập nhật cho các đối tượng mà sẽ nhận tín hiệu của sự thay đổi tại chủ đề.

ConcreteObserver (Investor)

Duy trì một tham chiếu tới một đối tượng ConcreteSubject.

Lưu trữ các trạng thái cố định.

Cài đặt giao diện cập nhật của Observer để giữ các trạng thái cố định của nó.

Mẫu liên quan

Bằng việc đóng gói các ngữ nghĩa cập nhật phức tạp, ChangeManager hoạt động như một Mediator giữa các chủ đề và các Observer.

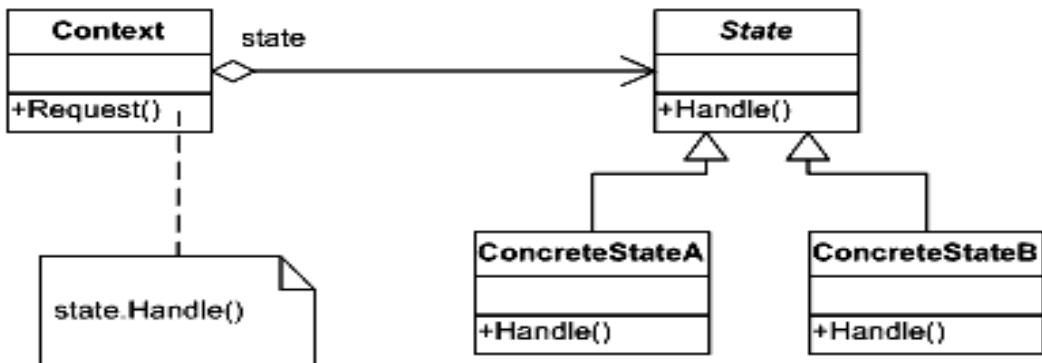
ChangeManager có thể sử dụng mẫu Singleton để cho việc truy nhập nó là đồng nhất và tổng thể.

State

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Định nghĩa: Observer là mẫu thiết kế cho phép một đối tượng thay đổi các hành vi của nó khi các trạng thái bên trong của nó thay đổi. Đối tượng sẽ xuất hiện để thay đổi các lớp của nó.

Cấu trúc mẫu



Context (Account)

Định nghĩa giao diện mà đối tượng khách quan tâm

Duy trì một thể nghiệm của một lớp ConcreteState mà định nghĩa trạng thái hiện tại

State (State)

Định nghĩa một giao diện cho việc đóng gói hành vi kết hợp với trạng thái đặc biệt của Context.

Concrete State (RedState, SilverState, GoldState)

Mỗi lớp con cài đặt một hành vi kết hợp với một trạng thái của Context.

Các mẫu liên quan

Mẫu Flyweight giải thích khi nào các đối tượng State có thể được phân tách và được phân tách như thế nào.

Các đối tượng State thường là các Singleton.

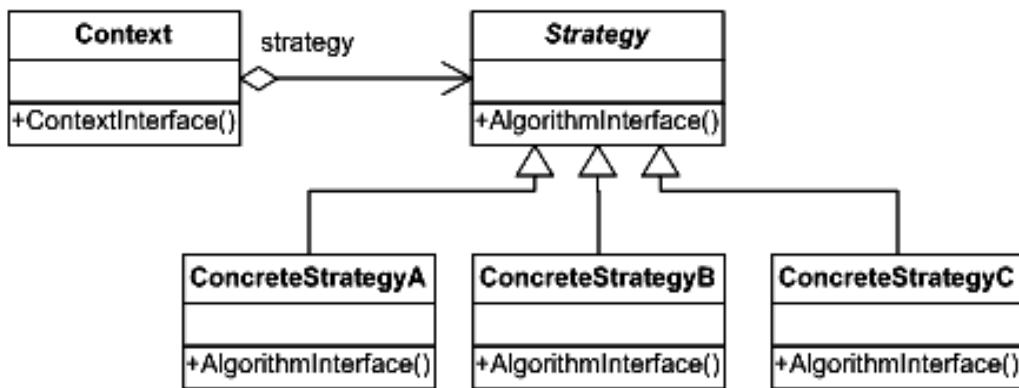
Strategy

Định nghĩa

Strategy là mẫu thiết kế dùng để định nghĩa một họ các thuật toán, đóng gói mỗi thuật toán đó và làm cho chúng có khả năng thay đổi dễ dàng. Strategy cho phép giả thuật tùy biến một cách độc lập tại các Client sử dụng nó.

Cấu trúc mẫu

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Strategy (SortStrategy)

Khai báo một giao diện thông thường tới tất cả các giải thuật được hỗ trợ. Context sử dụng giao diện này để gọi các giải thuật được định nghĩa bởi một ConcreteStrategy.

ConcreteStrategy (QuickSort, ShellSort, MergeSort)

Cài đặt giải thuật sử dụng giao diện Strategy

Context (SortedList)

Được cấu hình với một đối tượng ConcreteStrategy

Duy trì một tham chiếu tới một đối tượng Strategy

Có thể định nghĩa một giao diện để cho Strategy truy nhập dữ liệu của nó.

Các mẫu liên quan

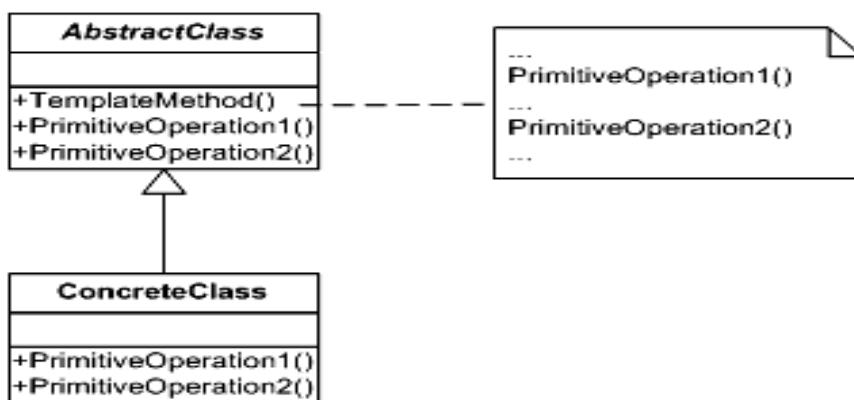
Các đối tượng Strategy thường tạo ra các Flyweight tốt hơn.

Template Method

Định nghĩa

Template Method là mẫu xác định khung của một giải thuật trong một thao tác, theo một số bước của các phân lớp. Template Method cho phép các lớp con xác định lại chắc chắn một số bước của một giải thuật bên ngoài cấu trúc của giải thuật đó.

Cấu trúc mẫu



AbstractClass

Định nghĩa các primitive operation (thao tác nguyên thủy) trừu tượng, các thao tác này định nghĩa các lớp con cụ thể để thực hiện các bước của một giải thuật.

Cài đặt một template method định nghĩa sườn của một giải thuật. Template method này gọi các thao tác nguyên thủy cũng như các thao tác được định nghĩa trong AbstractClass hoặc một số các đối tượng khác.

ConcreteClass

Thực hiện các thao tác nguyên thủy để thực hiện các bước đã chỉ ra trong các lớp con của giải thuật

Ứng dụng

Template Method nên sử dụng trong các trường hợp:

Thực hiện các phần cố định của một giải thuật khi đặt nó vào các lớp con để thực hiện hành vi có thể thay đổi.

Khi các lớp hành vi thông thường nên được phân tách và khoanh vùng trong một lớp thông thường để tránh sự giống nhau về mã.

Điều khiển mở rộng các lớp con. Ta có thể định nghĩa một template method, template method này gọi các thao tác “hook” tại các điểm đặc biệt, bằng cách đó cho phép các mở rộng chỉ tại các điểm đó.

Các mẫu liên quan

Các template Method sử dụng sự kế thừa để thay đổi các bộ phận của một giải thuật. Các Strategy sử dụng sự ủy nhiệm để thay đổi hoàn toàn một thuật toán.

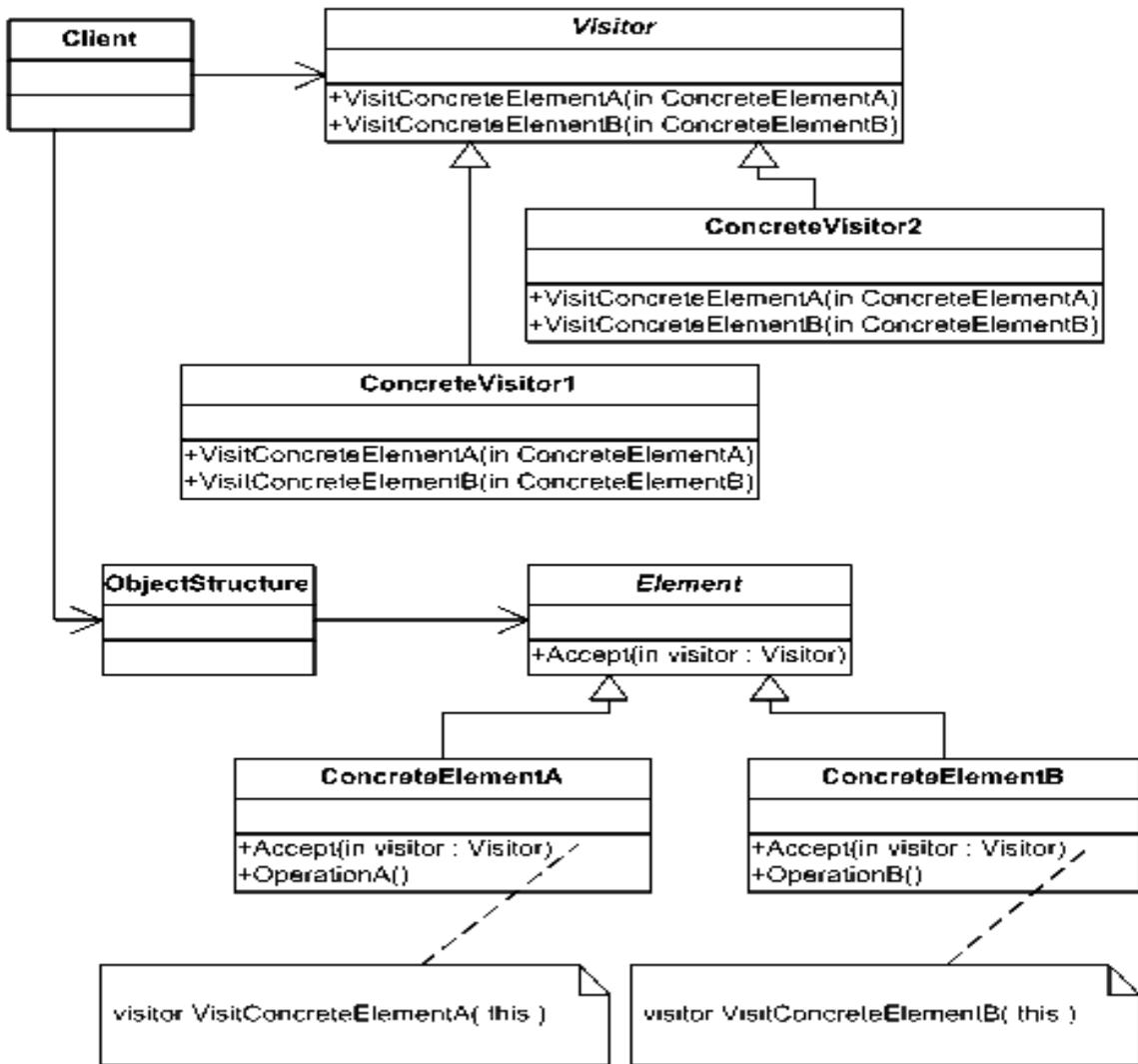
Visitor

Định nghĩa

Visitor là mẫu thiết kế xác định sườn của một giải thuật trong một thao tác, theo một số bước của các phân lớp. Template Method cho phép các lớp con xác định lại chắc chắn một số bước của một giải thuật bên ngoài cấu trúc của giải thuật đó.

Cấu trúc mẫu

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Visitor

Đưa ra một thao tác Visit cho mỗi lớp của **ConcreteElement** trong cấu trúc đối tượng. Tên và dấu hiệu của các thao tác này nhận dạng lớp gửi yêu cầu Visit tới visitor, nó cho phép visitor quyết định lớp cụ thể nào của thành phần được thăm. Sau đó visitor có thể truy nhập thành phần trực tiếp thông qua giao diện đặc biệt của nó.

ConcreteVisitor

Thực hiện mỗi thao tác được đưa ra bởi Visitor. Mỗi thao tác thực hiện một phần của giải thuật định nghĩa cho lớp phù hợp của đối tượng trong cấu trúc. **ConcreteVisitor** cung cấp ngữ cảnh cho giải thuật và lưu trữ trạng thái cục bộ của nó.

Element

Định nghĩa một thao tác Accept, thao tác này mang một visitor như là một đối số.

ConcreteElement

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Thực hiện một thao tác Accept, thao tác này mang một visitor như là một đối số.

ObjectStructure

Có thể đếm các thành phần của nó.

Có thể cung cấp một giao diện mức cao cho phép visitor thăm các thành phần của nó.

Có thể là một composite hoặc một sưu tập như danh sách hay tập hợp.

Ứng dụng: Sử dụng Visitor pattern khi:

Một cấu trúc đối tượng chứa đựng nhiều lớp của các đối tượng với các giao diện khác nhau, và ta muốn thực hiện các thao tác trên các đối tượng này thì đòi hỏi các lớp cụ thể của chúng.

Nhiều thao tác khác nhau và không có mối liên hệ nào cần được thực hiện trên các đối tượng trong một cấu trúc đối tượng, và ta muốn tránh “lạm hỏng” các lớp của chúng khi thực hiện các thao tác đó. Visitor cho phép ta giữ các thao tác có mối liên hệ với nhau bằng cách định nghĩa chúng trong một lớp. Khi một cấu trúc đối tượng được chia sẻ bởi nhiều ứng dụng, sử dụng Visitor để đặt các thao tác này vào trong các ứng dụng cần chúng.

Các lớp định nghĩa các cấu trúc đối tượng hiếm khi thay đổi, nhưng ta muốn định nghĩa các thao tác mới trên các cấu trúc. Thay đổi các lớp cấu trúc yêu cầu định nghĩa lại giao diện cho tất cả các visitor.

Mẫu liên quan

Các Visitor có thể được sử dụng để cung cấp một thao tác trên một cấu trúc đối tượng được định nghĩa bởi mẫu Composite. Visitor có thể được cung cấp để làm thông dịch.

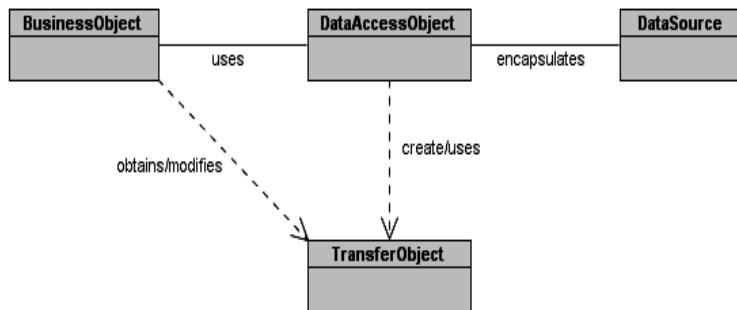
9.5 CASE STUDY: VÍ DỤ ĐỐI TƯỢNG TRUY CẬP DỮ LIỆU (J2EE Patterns)

Việc truy cập dữ liệu cần phải thay đổi tùy theo nguồn dữ liệu nghĩa là tùy thuộc vào kiểu của nó như CSDL quan hệ, CSDL hướng đối tượng, file... và cách cài đặt. Các ứng dụng có thể sử dụng JDBC API để truy cập dữ liệu trong hệ quản trị CSDL. JDBC API cho phép các ứng dụng JDBC sử dụng các lệnh SQL để truy cập dữ liệu. Tuy nhiên, trong một môi trường hệ quản trị CSDL quan hệ, cú pháp và định dạng của các lệnh SQL thực tế cũng biến đổi tùy thuộc vào sản phẩm CSDL cụ thể.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

Giải pháp là sử dụng Data Access Object (DAO) để trùu tượng và đóng gói các truy cập tới nguồn dữ liệu. DAO quản lý kết nối với nguồn dữ liệu để lấy và lưu trữ dữ liệu. DAO cài đặt cơ chế truy cập cần thiết để làm việc với nguồn dữ liệu.

Cấu trúc



Hình 9.12: Cấu trúc của DAO

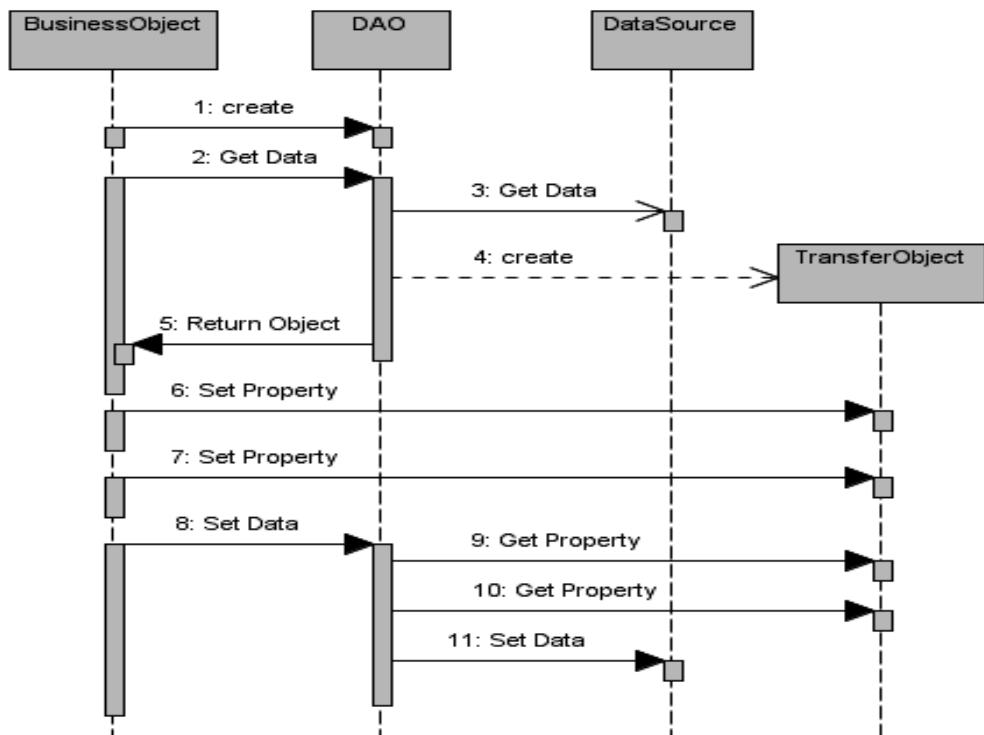
BusinessObject: Biểu diễn phía yêu cầu dữ liệu. Nó là đối tượng yêu cầu truy cập tới nguồn dữ liệu để lấy và lưu trữ dữ liệu. Nó có thể được cài đặt là session bean, entity bean hoặc đối tượng Java khác, servlet hoặc helper bean để truy cập tới nguồn dữ liệu.

DataAccessObject: Là đối tượng chính của bản mẫu này. Nó trùu tượng hóa các cài đặt việc truy cập dữ liệu bên dưới để BusinessObject có thể truy cập tới nguồn dữ liệu một cách trong suốt. BusinessObject cũng giao phó thao tác nạp và lưu trữ dữ liệu cho DataAccessObject.

DataSource: Biểu diễn cài đặt của nguồn dữ liệu. Nguồn dữ liệu có thể là CSDL như RDBMS, OODBMS, kho chứa XML, hệ thống file...

TransferObject: được dùng như một vật mang dữ liệu. DataAccessObject có thể sử dụng TransferObject để trả dữ liệu về phía yêu cầu. DataAccessObject cũng có thể nhận dữ liệu từ phía yêu cầu trong TransferObject để cập nhật dữ liệu trong nguồn dữ liệu.

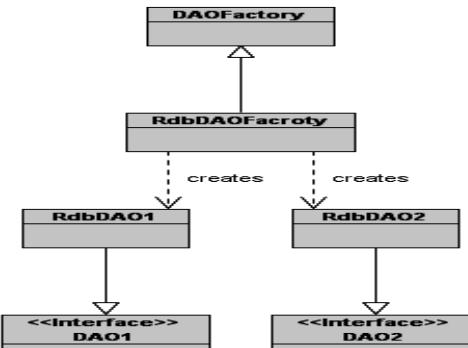
sd pattern



Hình 9.13: biểu đồ tuần tự của DAO

Các chiến lược

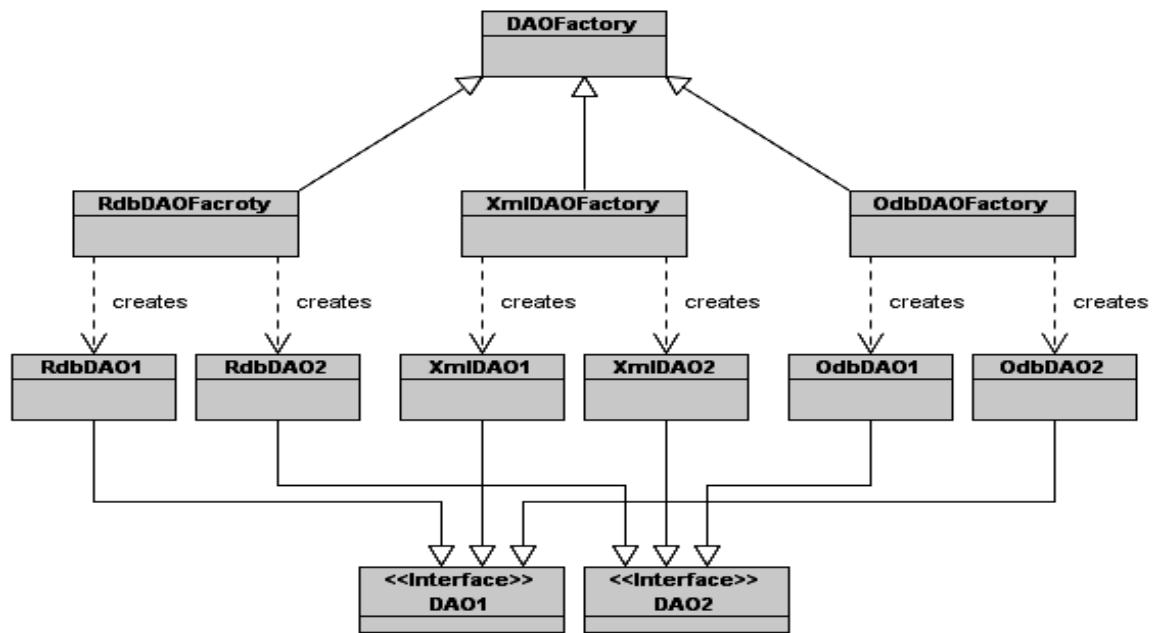
Factory Method



Hình 9.14: Biểu đồ lớp của DAO theo chiến lược Factory Method

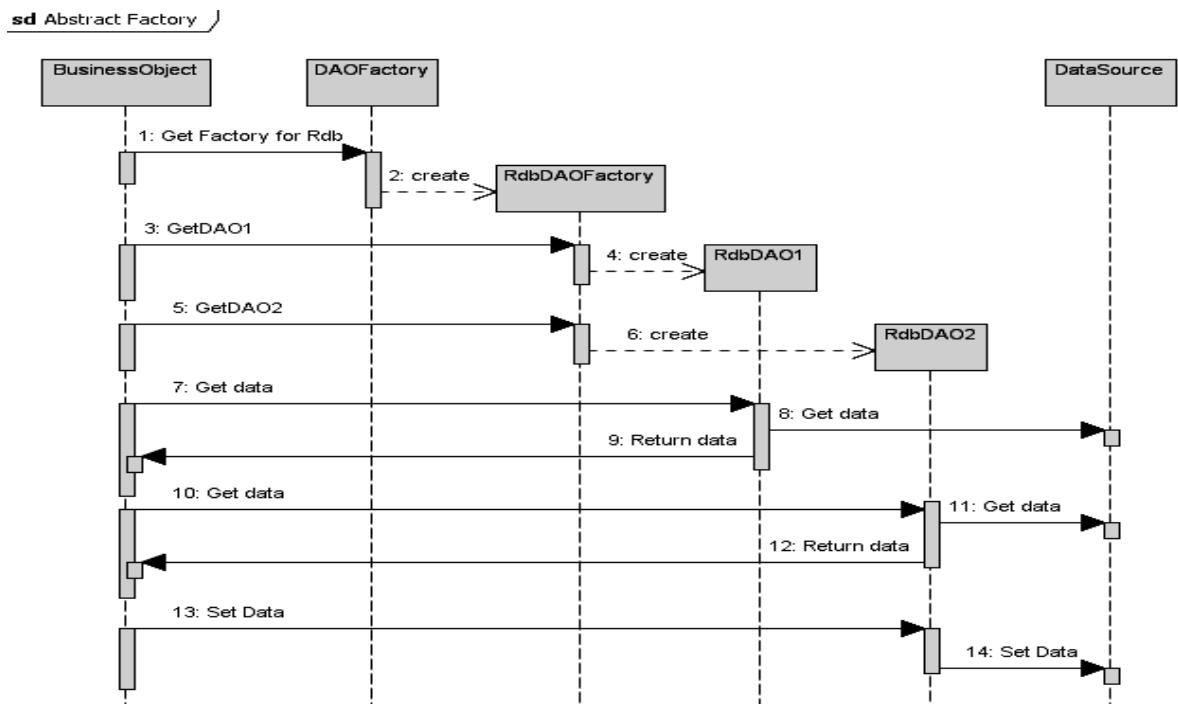
Abstract Factory

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Hình 9.14: Biểu đồ lớp của DAO theo chiến lược Factory Method

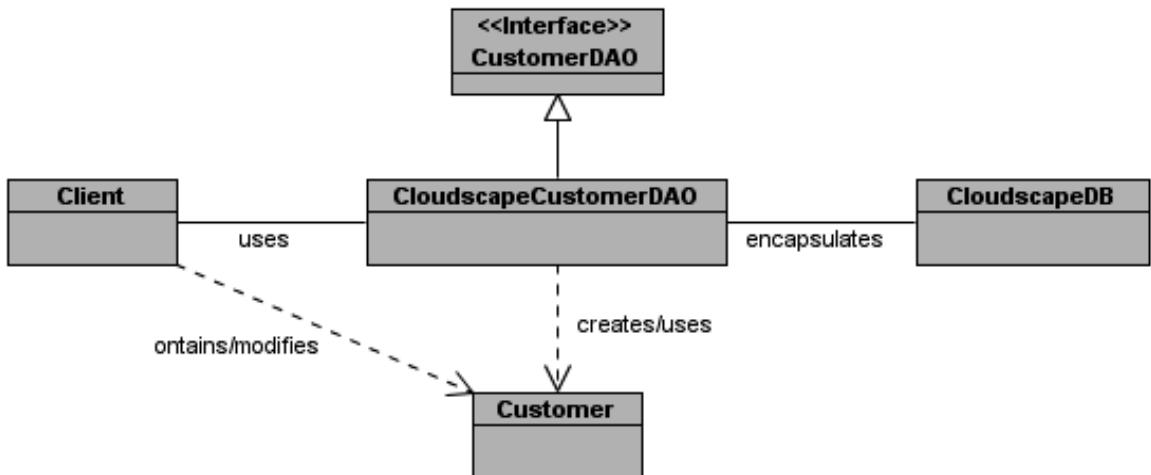
Biểu đồ tuần tự mô tả chiến lược Abstract Factory



Hình 1.15: biểu đồ tuần tự của DAO theo chiến lược Factory Method

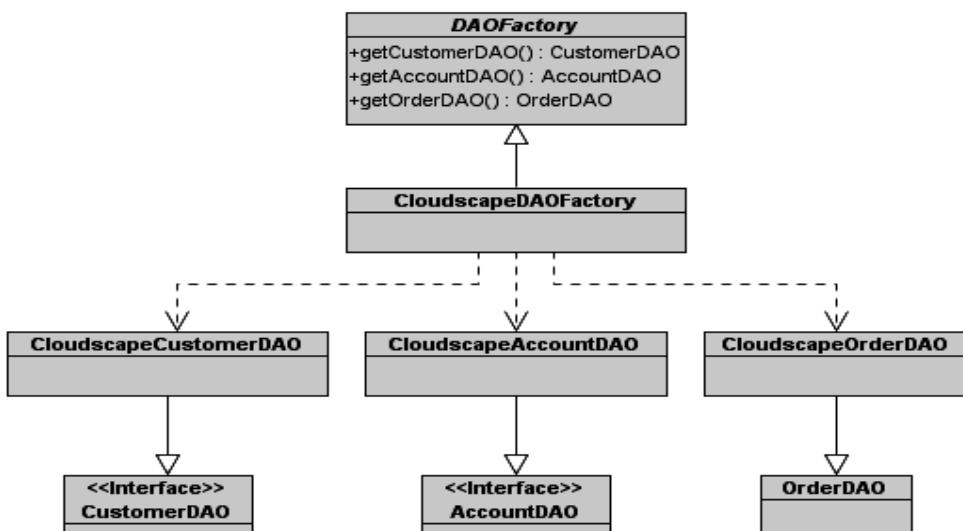
Sample Code: Customer

Class diagram



Hình 9.16: biểu đồ lớp của CustomerDAO

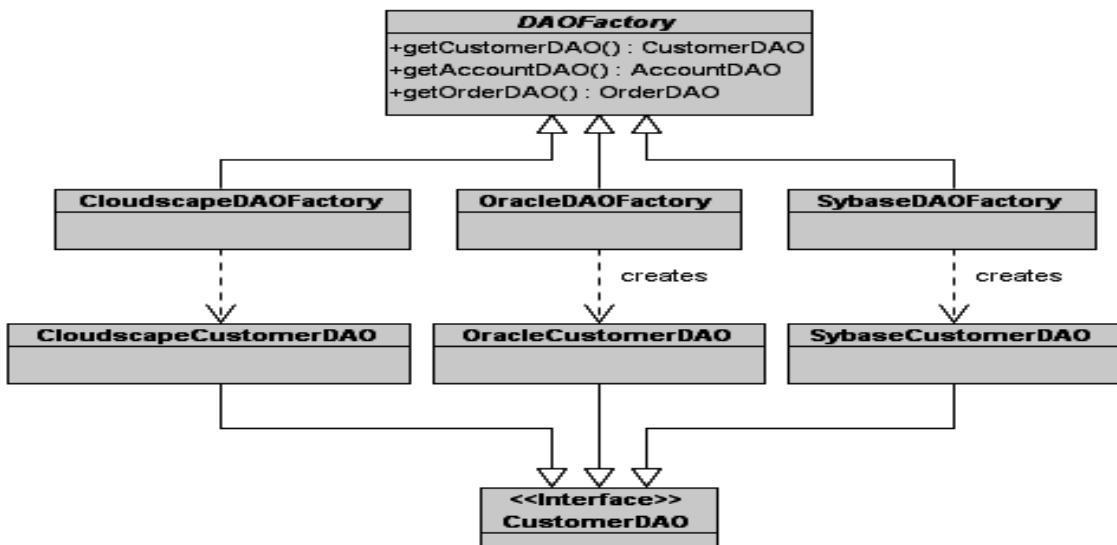
Cài đặt sử dụng chiến lược Factory Method



Hình 9.17: Biểu đồ lớp của CustomerDAO theo chiến lược Factory Method

Cài đặt sử dụng chiến lược Abstract Factory

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI



Hình 9.18: biểu đồ lớp của CustomerDAO theo chiến lược Factory Method

Cài đặt Factory cho sử dụng Abstract Factory

Lớp Abstract DAOFactory

```

// Abstract class DAO Factory
public abstract class DAOFactory {
    // List of DAO types supported by the factory
    public static final int CLOUDSCAPE = 1;
    public static final int ORACLE = 2;
    public static final int SYBASE = 3;
    ...
    // There will be a method for each DAO that can be
    // created. The concrete factories will have to
    // implement these methods.
    public abstract CustomerDAO getCustomerDAO();
    public abstract AccountDAO getAccountDAO();
    public abstract OrderDAO getOrderDAO();
    ...
    public static DAOFactory getDAOFactory(
        int whichFactory) {
        ...
        switch (whichFactory) {
            case CLOUDSCAPE:
                return new CloudscapeDAOFactory();
            case ORACLE :
                return new OracleDAOFactory();
            case SYBASE :
                ...
        }
    }
}
  
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
        return new SybaseDAOFactory();  
    ...  
    default :  
        return null;  
    }  
}  
}
```

Hiện thực hóa cài đặt DAOFactory cho Cloudscape

```
// Cloudscape concrete DAO Factory implementation  
import java.sql.*;  
public class CloudscapeDAOFactory extends DAOFactory {  
    public static final String DRIVER=  
        "COM.cloudscape.core.RmiJdbcDriver";  
    public static final String DBURL=  
        "jdbc:cloudscape:rmi://localhost:1099/CoreJ2EEDB";  
  
    // method to create Cloudscape connections  
    public static Connection createConnection() {  
        // Use DRIVER and DBURL to create a connection  
        // Recommend connection pool implementation/usage  
    }  
    public CustomerDAO getCustomerDAO() {  
        // CloudscapeCustomerDAO implements CustomerDAO  
        return new CloudscapeCustomerDAO();  
    }  
    public AccountDAO getAccountDAO() {  
        // CloudscapeAccountDAO implements AccountDAO  
        return new CloudscapeAccountDAO();  
    }  
    public OrderDAO getOrderDAO() {  
        // CloudscapeOrderDAO implements OrderDAO  
        return new CloudscapeOrderDAO();  
    }  
    ...  
}
```

Cài đặt DAO Interface cho Customer

```
// Interface that all CustomerDAOs must support  
public interface CustomerDAO {  
    public int insertCustomer(...);
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
public boolean deleteCustomer(...);  
public Customer findCustomer(...);  
public boolean updateCustomer(...);  
public RowSet selectCustomersRS(...);  
public Collection selectCustomersTO(...);  
...  
}
```

Cài đặt Cloudscape DAO cho Customer

```
// CloudscapeCustomerDAO implementation of the  
// CustomerDAO interface. This class can contain all  
// Cloudscape specific code and SQL statements.  
// The client is thus shielded from knowing  
// these implementation details.  
  
import java.sql.*;  
  
public class CloudscapeCustomerDAO implements  
    CustomerDAO {  
    public CloudscapeCustomerDAO() {  
        // initialization  
    }  
    // The following methods can use  
    // CloudscapeDAOFactory.createConnection()  
    // to get a connection as required  
    public int insertCustomer(...) {  
        // Implement insert customer here.  
        // Return newly created customer number  
        // or a -1 on error  
    }  
  
    public boolean deleteCustomer(...) {  
        // Implement delete customer here  
        // Return true on success, false on failure  
    }  
  
    public Customer findCustomer(...) {  
        // Implement find a customer here using supplied  
        // argument values as search criteria  
        // Return a Transfer Object if found,
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
// return null on error or if not found
}

public boolean updateCustomer(...) {
    // implement update record here using data
    // from the customerData Transfer Object
    // Return true on success, false on failure or
    // error
}

public RowSet selectCustomersRS(...) {
    // implement search customers here using the
    // supplied criteria.
    // Return a RowSet.
}

public Collection selectCustomersTO(...) {
    // implement search customers here using the
    // supplied criteria.
    // Alternatively, implement to return a Collection
    // of Transfer Objects.
}

...
}
```

Customer Transfer Object

```
public class Customer implements java.io.Serializable
{
    // member variables
    int CustomerNumber;
    String name;
    String streetAddress;
    String city;
    ...
    // getter and setter methods...
    ...
}
```

Sử dụng DAO và DAOFactory ở client code

```
...
// create the required DAO Factory
DAOFactory cloudscapeFactory =
    DAOFactory.getDAOFactory(DAOFactory.DAOCLLOUDSCAPE);
```

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

```
// Create a DAO  
CustomerDAO custDAO =  
    cloudscapeFactory.getCustomerDAO();  
  
// create a new customer  
int newCustNo = custDAO.insertCustomer(...);  
  
// Find a customer object. Get the Transfer Object.  
Customer cust = custDAO.findCustomer(...);  
  
// modify the values in the Transfer Object.  
cust.setAddress(...);  
cust.setEmail(...);  
// update the customer object using the DAO  
custDAO.updateCustomer(cust);  
  
// delete a customer object  
custDAO.deleteCustomer(...);  
// select all customers in the same city  
Customer criteria=new Customer();  
criteria.setCity("New York");  
Collection customersList =  
    custDAO.selectCustomersTO(criteria);  
// returns customersList - collection of Customer  
// Transfer Objects. iterate through this collection  
to  
    // get values.  
    ...
```

9.6 KẾT LUẬN

Trong chương này chúng ta đã xem xét:

- Cách sử dụng mẫu thiết kế để giải quyết bài toán tương tự đã được những người phát triển trước đó thực hiện và như vậy khỏi phải phí công sức một cách vô ích. Mẫu mô tả cách làm đặc biệt đã được chứng tỏ là hiệu quả trong các dự án thực tế có trước.
- Cách các mẫu được sử dụng để lưu lại tri thức, viết tài liệu lời giải và trợ giúp sử dụng lại.

CHƯƠNG 9. MẪU THIẾT KẾ SỬ DỤNG LẠI

BÀI TẬP

1. Sinh viên làm việc theo nhóm để quyết định sử dụng mẫu thiết kế nào cho dự án bài tập của mình.
2. Sử dụng mẫu thiết kế đã chọn để cài đặt hệ thống quản lý học theo tín chỉ

CHƯƠNG 10 ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

10.1 GIỚI THIỆU

Mặc dù theo một nghĩa nào đó chúng ta đã đặc tả hành vi của phần mềm tại tất cả các giai đoạn (từ các use case hệ thống cho tới lược đồ cơ sở dữ liệu). Vẫn đề ở đây liên quan tới interface của các thành phần hệ thống nói chung và các lớp nói riêng. Tại sao chúng ta phải viết đặc tả? Sau đây là 1 số nguyên nhân chính:

- Để loại bỏ sự nhập nhằng (tính đa nghĩa, sự tối nghĩa) còn lại sau các pha trước đó: Sự nhập nhằng có nghĩa là các sản phẩm có thể được hiểu theo nhiều cách. Các use case thông thường được viết theo ngôn ngữ tự nhiên, không rõ ràng; phân tích không có nghĩa là để ra lệnh cho người thiết kế rằng cách nào code thực sự nên được cài đặt, vì vậy các sản phẩm phân tích không phù hợp một cách chính xác với code cuối cùng.
- Còn đối với thiết kế, chúng ta có thể đặc tả chính xác các lớp được viết và các thông điệp, các trường nào nên tồn tại nhưng có thể không có tài liệu về cách các thông điệp đó nên làm việc cùng nhau như thế nào hoặc một tham số thông điệp hợp pháp cần thiết lập những gì, một giá trị trường dữ liệu hợp pháp hay chi tiết khác như thế.
- Để mở rộng sự hiểu biết của chúng ta về cài đặt. Chúng ta càng hiểu thiết kế của chúng ta hơn thì nó sẽ càng dễ hơn để cài đặt. Việc viết đặc tả yêu cầu chúng ta suy nghĩ sâu hơn về thiết kế, cho nên ta kết luận được với sự hiểu biết tốt hơn.
- Để làm tăng độ tin cậy rằng hệ thống sẽ hoạt động: thông thường, với sự cố gắng mô tả chính xác cách hoạt động của phần mềm, chúng ta nhận ra rằng chúng ta đã lỡ mất cơ hội gì đó hoặc đã giả sử điều gì đó là không thể, hoặc đã tạo một số sai sót khác. Như vậy, đặc tả cho phép chúng ta phát hiện ra các lỗi sớm nhất là nếu chúng ta đi thẳng vào cài đặt.
- Để giúp chúng ta gỡ rối phần mềm của chúng ta: Khi phần mềm của chúng ta hoạt động không đúng, chúng ta có thể kiểm tra mặc dù cài đặt không phù hợp với đặc tả. Vì vậy, chúng ta có 1 điểm bắt đầu cho việc xác định và loại bỏ các lỗi.
- Để giúp chúng ta kiểm tra phần mềm: Một đặc tả miêu tả cách phần mềm phải được hoạt động như thế nào và 1 test xác minh xem những hoạt động phần mềm của chúng ta như đã miêu tả không. Do đó các test có thể được dùng để kiểm tra xem phần mềm có phù hợp với đặc tả hay không. Các test dựa trên hệ thống use

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

case xác minh xem chúng ta có đưa ra 1 hệ thống mà đáp ứng các yêu cầu đặc tả hay không. Đặc tả dựa trên kiểm tra là cần thiết cho các library, pattern (khuôn mẫu), framwork và các object có khả năng sử dụng lại, bởi vì không cái gì trong số chúng involve(đòi hỏi, dính lứu) đến 1 hệ thống thực.

- Để giúp chúng ta sửa đổi phần mềm: Nếu chúng ta quyết định sửa đổi hệ thống theo 1 số cách bằng cách thêm chức năng. Ví dụ chúng ta có thể kiểm tra xem việc sửa đổi có làm phá vỡ đặc tả hay không. Nếu đặc tả bị phá vỡ, các sửa đổi đề ra không tương thích với hệ thống hiện tại (hoặc chúng ta có 1 lỗi trong đặc tả đó). Tốt hơn chúng ta nên xem xét thay đổi đã đề xuất cẩn thận hơn
- Để cho phép chúng ta chuyển tiếp tới cài đặt giao việc cho những người phát triển khác. Mặc dù những người phát triển khác vẫn sẽ tiếp tục liên hệ với chúng ta để làm rõ các chi tiết, they should need to do so less often.
- Để cung cấp tài liệu tốt hơn: tài liệu tốt hơn dẫn đến bảo trì dễ dàng hơn, sử dụng lại và ít lỗi hơn

10.2 ĐẶC TẢ LÀ GÌ?

Theo ngôn ngữ chung, một đặc tả đầy đủ, miêu tả rõ ràng các hành vi được yêu cầu của bộ phận phần mềm. Bộ phận phần mềm này có thể là toàn bộ hệ thống, hệ thống con, một tầng, một lớp hoặc một hàm. Mặc dù chúng ta muốn đặc tả của chúng ta trỏ nêu toàn diện và không mập mờ, điều đó không phải bao giờ cũng có thể trong thực tế. Tuy nhiên, ngay cả các đặc tả không hoàn chỉnh cũng có giá trị của chúng (chúng tốt hơn là không có gì cả).

Một đặc tả nhằm mô tả một hay nhiều các biên giới. Ví dụ, biên của của thư viện hàm bao gồm các ký hiệu (tên, kiểu trả về, kiểu tham số) của các hàm của nó và giá trị dữ liệu biến toàn cục của nó; biên của một đối tượng bao gồm các thông điệp của nó và các thuộc tính của nó; biên của một lớp bao gồm các thông điệp lớp của nó và các thuộc tính lớp. Mặc dù các thuộc tính phù hợp với biên đó một cách logic, chúng ta mong chúng được truy cập bởi các thông điệp lớp và đối tượng. Nếu chúng ta coi rằng mỗi biên có một khách hàng ở một bên và một nhà cung cấp ở bên kia thì đặc tả mô tả: Thông tin gì mà khách hàng có thể chuyển qua biên đó và khi nào họ được phép chuyển nó, ví dụ khách hàng chỉ có khả năng chuyển thông tin địa chỉ mới tới một đối tượng khách hàng đã giao dịch trước kia.

Thông tin gì khách hàng có thể lấy lại từ biên đó và khi nào họ được phép lấy lại nó. Xét ví dụ khách hàng có thể chỉ có khả năng lấy lại một danh sách các sách được bán từ một server sách trước kia nó đã đăng nhập.

Khi một sự kiện được quảng bá tới các khách hàng đã đăng ký và thông tin gì các khách hàng đó sẽ nhận. Ví dụ hệ thống chat phân phối, mỗi khách hàng sẽ được thông báo tên của mỗi khách hàng mới mà tham gia vào phòng chat. Tình trạng hợp pháp của

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

thông tin được quản lý bởi biên đó, xét ví dụ tập đối tượng có thể phát biểu rằng nó sẽ không bao giờ chứa số âm các đối tượng.

Hầu hết thời gian, một đặc tả được đề cập với biên giới chung của một mảng phần mềm. Tuy nhiên chúng ta có thể chỉ định bản chất của các biên bên trong tốt tương đương nhau. Ví dụ, xét một thư viện hàm đưa ra các điều kiện phân loại cho các cấu trúc sưu tập. Như một thư viện có thể có khả năng sử dụng lại *swap function* đổi với hai giá trị trao đổi trong một tập. Nó không chắc đúng là *swap function* đó có trở thành phần của interface chung của thư viện phân loại đó không bởi vì nó là một cài đặt chi tiết; Tuy nhiên chúng ta vẫn có thể xác định hoạt động của nó mà chúng ta đã phát biểu. Ví dụ, sau hàm đã được gọi, giá trị bên tay trái sẽ chiếm vị trí trước đó đã chiếm bởi giá trị bên tay phải và ngược lại. Có hai loại đặc tả: đặc tả hình thức là khoa học và nghiêm ngặt; đặc tả phi hình thức thì thực tế và cụ thể nhưng không kém hiệu quả.

10.3 ĐẶC TẢ HÌNH THỨC

Nhiều người cho rằng, phát triển phần mềm là một hoạt động khá “mơ hồ”. Về phương diện lịch sử, phần mềm đã được hy vọng phát triển nhiều hơn là nó sẽ làm được những gì nó cần làm. Điều này là vì không có kỹ thuật nào để khôi phục nó lại được, khác các ngành như nghề xây dựng hoặc phần cứng, có thể khôi phục lại bởi các máy móc và lý thuyết lượng tử. Hầu hết phần mềm tốt là kết quả của tính khéo léo của con người, kinh nghiệm, việc phỏng đoán hoạt động, sử dụng lại và kiểm tra.

Đây là điều tốt cho các ứng dụng thông thường như xử lý văn bản và các hệ thống quản lý khách hàng. Chúng ta chấp nhận hệ xử lý văn bản hiếm khi không hoạt động giữa chúng (đó là lý do chúng ta nên có đặc tính như tự động save). Chúng ta có thể mất thời gian, có thể trở nên nản chí và cáu giận, nhưng không có cuộc sống của ai bị gây nguy hiểm. Những thuận lợi của máy tính nói chung có nhiều tác dụng hơn những bất lợi dù các máy tính đó chưa hoàn chỉnh: tóm lại chúng ta được nhiều hữu ích hơn là không có máy tính. Tuy nhiên, chúng ta cần hướng tới hệ thống hoàn hảo hoặc gần hoàn hảo và đáng tin cậy hơn.

Các hệ thống an toàn tính mạng là một ví dụ. Các hệ thống này bao gồm phần mềm điều khiển các trạm năng lượng hạt nhân, hệ thống điều khiển vận chuyển hàng không, điều khiển các thiết bị ý tế. Chúng ta không chấp nhận để cho các hệ thống (hoặc các hệ điều hành) hỏng hóc. Tuy nhiên, trong thực tế chúng ta không thể đảm bảo tin cậy 100%, bởi vì có thể có quá nhiều khó khăn hoặc quá đắt hoặc yêu cầu kiến thức của từng điều kiện dị thường có thể xảy ra. Thay vì chúng ta bố trí cho hoặc là một thời gian dài giữa các lỗi (ví dụ, hệ thống điều khiển tại trạm năng lượng hạt nhân có thể được chấp nhận nếu nó gây ra trung bình một lỗi không quá một lần trong vòng một năm) hoặc độ tin cậy tốt hơn con người (ví dụ chương trình điều khiển máy bay có thể được chấp nhận nếu nó mắc 1/10 lỗi của một phi công).

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

Một cách để cải thiện độ tin cậy là tái tạo phần cứng hay phần mềm. Ví dụ, chúng ta có thể cài đặt ba máy tính để điều khiển một máy bay. Mỗi máy tính sẽ dùng phần cứng khác nhau, hệ thống điều hành khác nhau và phần mềm được phát triển bởi các đội khác nhau. Chúng ta có thể sắp đặt ba máy tính này trong một hệ thống, không có quyền điều khiển nào của máy bay bị bỏ qua trừ khi nó xảy ra đồng thời từ ít nhất hai trong số các máy tính. Nhiều lý luận cho rằng, sử dụng các kỹ thuật như thế không thể mắc các lỗi (nhưng chúng ta vẫn không thể tin hoàn toàn bởi vì hai trong số các máy tính có thể hỏng theo cách khác nhau trong cùng thời điểm, nguyên nhân là máy bay thực hiện một lệnh không đúng).

Một cách để cải thiện độ tin cậy bao gồm việc có gắng chứng minh theo chiều hướng toán học rằng phần mềm là chính xác (rằng nó làm những gì nó được cho là cần làm). Việc này bao gồm 3 bước:

- Đưa ra một đặc tả hình thức theo ngôn ngữ toán học để mô tả cách phần mềm nên hoạt động như thế nào.
- Chứng minh rằng đặc tả đó là khả thi – ví dụ như nó không có sự mâu thuẫn hoặc không thể thực hiện được hoặc không logic
- Chứng tỏ rằng phần mềm phù hợp với đặc tả.

Các ví dụ của ngôn ngữ đặc tả hình thức là Vienna Development Method (VDM), (khởi đầu từ IBM Vienna), Z (khởi đầu từ Oxford University) và Object Constraint Language (OCL). Ví dụ, nếu chúng ta muốn viết một đặc tả hình thức cho hành vi của hàm squareRoot(), ở đây có một số thứ mà chúng ta có thể muốn nói về nó:

- Giá trị nhập cần phải được xác thực, giả sử các số không là số phức
- Bình phương của kết quả sẽ tương đương với giá trị nhập vào(Điều này là một thủ thuật thông dụng khi đặc tả 1 hàm nghịch đảo: việc áp dụng hàm để kết quả phải sinh ra giá trị nhập vào)
- Kết quả sẽ là số dương. nhớ rằng $(2 \times 2 = 4)$ và $(-2 \times -2 = 4)$, vì vậy 4 có 2 căn bậc hai; chúng ta chọn để trả lại căn bậc 2 dương).

Đây là cách chúng ta có thể đặc tả các điều kiện biên của squareRoot() trong VDM:

```
squareRoot(x : R) y : R  
pre x ≥ 0  
post (y2 = x) ^ (y ≥ 0)
```

Trong đó, R thay cho các số thực, pre thay cho tiền điều kiện (những gì cần phải đúng trước khi hàm đó có thể gọi) và post thay cho hậu điều kiện (những gì sẽ phải đúng sau khi hàm đó được thực hiện). Mỗi biểu thức logic trong 1 đặc tả tham chiếu tới như sự xác nhận (nên $y \geq 0$ xác nhận rằng y sẽ lớn hơn hoặc bằng 0). Thậm chí nếu bạn không biết

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LÓP

VDM, dễ dàng để thấy rằng đặc tả bên trên đọc như sau: Hàm squareRoot() giữ 1 số thực x như tham số và trả lại 1 số thực y. Đôi với thao tác chính xác, hàm yêu cầu x lớn hơn hoặc bằng 0. Nếu tiền điều kiện phù hợp, hàm này đảm bảo rằng y được bình phương sẽ bằng x và y sẽ lớn hơn hoặc bằng 0. Vì vậy chúng ta có thể đặc tả chính xác hành vi của squareRoot() mà không phải nói bất cứ gì về việc cài đặt nó.

Đặc tả hình thức là quy định đòi hỏi phải có đào tạo và có kỹ năng tốt. Nó cũng là một quy trình dài dòng. Đôi lúc có thể để sử dụng một máy tính để chứng minh định lý tự động (automated theorem proving). Dưới các điều kiện rất hạn chế, chúng ta thậm chí có khả năng kiểm chứng sự thực thi có phù hợp với đặc tả không. Tuy nhiên không máy tính nào tự động viết được đặc tả. Thậm chí nếu bạn có thể dựa vào hệ điều hành, phần cứng hay trình biên dịch. Xét theo lý do này, phần mềm liên quan mạng sống đôi lúc được cài đặt theo ngôn ngữ assembly và sau đó được phát triển trên một hệ điều hành đã giàn thiểu, do đó chúng ta có thể chứng minh rằng kết quả cuối này là chính xác. Tuy nhiên, chúng ta cũng không thể chứng minh rằng đặc tả hình thức đáp ứng các yêu cầu hệ thống. Đặc tả hình thức thì khó và tốn nhiều thời gian để tạo ra và để sử dụng. Mặc dù đặc tả hình thức không phù hợp cho công nghiệp phần mềm nói chung, chúng ta vẫn có thể áp dụng các nguyên tắc cơ bản và có được nhiều lợi ích. Các kỹ thuật được mô tả trong phần còn lại của chương này liên quan tới đặc tả và với chỉ dẫn của lý luận đặc tả hình thức chúng ta có thể ít nhất được rõ hơn về đặc tả nên có gì.

10.4 ĐẶC TẢ KHÔNG HÌNH THỨC

Tất cả người lập trình sử dụng đặc tả thông tin tới một mức độ nhất định – ví dụ như thêm vào chú thích cho một hàm để giúp ích cho những người lập trình khác. Một chú thích sẽ mô tả một số hoặc tất cả những thông tin trong danh sách sau:

- Khi nào khách hàng có thể gọi hàm đó?
- Các tham số gì nên được chuyển?
- Loại kết quả nào được trả lại (kiểu và giá trị)?
- Hàm có hiệu ứng gì trên dữ liệu toàn cục?
- Hàm đưa ra hành động gì nếu có một vấn đề xảy ra ?

Thông tin trên cũng áp dụng tương tự cho thủ tục con, thủ tục và phương thức nếu chúng chỉ thay đổi trên ý tưởng cơ bản của một hàm. Thường thì một chú thích hàm không diễn đạt bất kỳ điều gì về việc cài đặt bên trong mà chỉ giống như bất kỳ dạng khác của đặc tả, chú thích chỉ miêu tả các điều kiện ở bên ngoài. Hiếm khi một chú thích diễn đạt một số thứ về việc cài đặt, nhưng chỉ khi cần thiết và chỉ dùng trong các ngôn ngữ trừu tượng. Ví dụ nó có thể hữu ích đối với một khách hàng, lập trình viên để nói về cân bằng thời gian và không gian của một hàm cụ thể, như hàm này chạy với tốc độ nlogn và yêu cầu $2n$ vị trí bên trong để xử lý. Đoạn mã C sau đây chứa một chú thích được sử dụng như là

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

một thông tin đặc tả của hàm squareRoot:

```
/*
Returns the positive square root of x.
Preconditions: x >= 0.

*/
float squareRoot(float x);
```

Bất kỳ ai đọc chú thích này đều hiểu rằng họ sẽ nhận lại căn bậc hai dương của x khi họ cấp một giá trị x dương. Tiền điều kiện nên quen thuộc cho hầu hết những người lập trình, hoặc dễ hiểu để họ thực hiện. Nếu bạn quan tâm về lượng tối đa có thể đọc được, bạn có thể sử dụng các yêu cầu thay thế cho các tiền điều kiện và đảm bảo rằng các hậu điều kiện cũng thay đổi hoặc bạn có thể viết các tiền điều kiện và hậu điều kiện theo ngôn ngữ tự nhiên. Thông tin đặc tả bên trên là không đầy đủ, khác với phiên bản VDM hình thức mà chúng ta đã xem trước. Ví dụ trên không nói gì về bình phương của kết quả tương đương với tham số hiện tại. Trong mục tiếp theo chúng ta sẽ xem tại sao hậu điều kiện cụ thể này lại không được chỉ ra trong lập trình.

Một ký hiệu hàm (tên, kiểu trả lại, các kiểu tham số) bao gồm nhiều thông tin trong nó vì vậy nó có thể tạo thành 1 phần của thông tin đặc tả. Ví dụ, float squareRoot(float x) ngũ ý rằng hàm này nhận giá trị float và trả lại giá trị float. Kết quả sẽ được lấy căn bậc hai tham số đó. Tuy nhiên thông tin đặc tả không nên dựa vào thông tin được bao hàm, do đó chủ thích phải rõ ràng. Nếu các chủ thích và các ký hiệu có thể được bao gồm trong một số module, mã nguồn không chỉ thay thế nơi thông tin đặc tả có thể xuất hiện.

Một loại khác của thông tin đặc tả mà chúng ta đã gặp trong cuốn sách này là use case. Mỗi use case miêu tả một số hoặc tất cả thông tin sau đây:

use case có thể được dùng khi nào (các precondition)?

use case phải làm gì (các bước và các hậu điều kiện)?

use case có tác động gì trên hệ thống?

điều gì diễn ra trong các trường hợp dị biệt?

Do đó nếu các use case miêu tả mọi thứ, 1 tác nhân bên trong cần để biết thứ tự sử dụng hệ thống 1 cách đúng đắn, chúng tương tự các comment hàm. Một số ngôn ngữ lập trình đặc biệt là Eiffel, có cú pháp đặc biệt để ghi lại các đặc tả không chính thức trong mã nguồn, phân biệt từ các chú thích

10.5 KIỂM TRA ĐỘNG

Người tạo ra đặc tả hình thức bằng cách sử dụng các ký hiệu toán học để phân biệt với

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

việc cài đặt phần mềm. Người đặc tả hình thức có thể chứng minh đặc tả họ đưa ra không có các lỗi logic. Trong các trường hợp hạn chế, họ thậm chí có thể chứng minh rằng một cài đặt phần mềm cụ thể tuân theo đặc tả của nó. Nhưng thông thường những người đặc tả hình thức dựa vào những người lập trình sử dụng tri thức và trực giác của họ để đưa ra code mà tuân theo đặc tả đó.

Phương pháp phi hình thức thì ngược lại, gồm tập kỹ thuật mà chúng ta có thể gọi là sự kiểm tra động. Một kiểm tra động là code được nhúng trong cài đặt của chúng ta để xác minh xem phần mềm đó đang tự chạy hay không (nó không phá vỡ đặc tả). Để xem xét cách kiểm tra động hoạt động, xét lại hàm square. Bên dưới là 1 cài đặt C của hàm này (việc tính toán được bỏ qua)

```
/*
Returns the positive square root of x.
Precondition: x >= 0.
*/
float squareRoot(float x) {
    if (x < 0) {
        fail("squareRoot", "Parameter x can't be negative");
    }
    ... /* Code to calculate y */
    return y;
}
```

Bên trong hàm này chúng ta có thể thấy 1 đoạn code kiểm tra precondition $x \geq 0$ đã thỏa mãn hay chưa. Nếu nó chưa thỏa mãn chương trình đưa ra, một hàm lỗi được đưa ra để in tên hàm và thông điệp lỗi và sau đó dừng chương trình đó (Sử dụng hàm exit trong C) (một số ngôn ngữ cung cấp nhiều hơn phương thức đánh dấu lỗi sử dụng việc quản lý ngoại lệ). Trình biên dịch máy khách hiện tại được bảo vệ bởi chính chúng: hầu hết thời gian máy khách gọi squareRoot với một số dương và hàm kế tiếp. Tuy nhiên hiếm khi máy khách gọi squareRoot với một số âm, khi đó nó báo rằng chương trình không thể tiếp tục cho tới khi vấn đề được giải quyết.

Bạn có thể nghĩ rằng hàm squareRoot của ta đang thực hiện việc kiểm tra lỗi một cách tuần tự. Nó chắc chắn thực hiện những gì mà trình biên dịch thường làm, dù muốn hay không chúng nên hiểu quy trình đặc tả. Điểm thú vị là đến khi chúng ta lo lắng là phần code kiểm tra lỗi này chỉ được đặt ở nơi kiểm tra để trình biên dịch máy khách không làm phá vỡ đặc tả, nếu phá vỡ đặc tả thì lỗi sẽ xuất hiện trong code của họ. Việc kiểm tra động chỉ thực hiện cho đặc tả phi hình thức mà không cho đặc tả hình thức. Việc này có nhiều nguyên nhân, một số được chỉ ra sau đây:

- Mục đích của đặc tả hình thức bảo đảm rằng việc thực thi không bị vi phạm đặc tả . Với việc kiểm tra động, ngược lại chúng ta chấp nhận rằng việc cài đặt theo

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

khuynh hướng chúa đựng các lỗi, cho nên chúng ta tìm thấy các vi phạm tại thời gian thực thi.

- Một số yêu cầu hình thức không khả thi được kiểm tra. Thực tế chúng ta không thể kiểm tra cái gì không có lý do để bỏ nó ra khỏi đặc tả hình thức.
- Một số yêu cầu hình thức không thể bị ép sử dụng code bắt buộc. (một ngôn ngữ lập trình bắt buộc, rất nhiều loại thông thường người trình phải cho biết chính xác máy tính phải thực hiện những gì; Với 1 ngôn ngữ lập trình, người lập trình thường là phát biểu kết quả sẽ xảy ra là gì; tương tự các yêu cầu vẫn nên chứa đựng trong đặc tả hình thức).
- Một số yêu cầu hình thức không bị ép buộc như code.

Xét ví dụ: Trong trường hợp squareRoot, chúng ta muốn đặc tả rằng kết quả được trả về của hàm đúng là căn bậc hai. Một cách khác để làm điều này là đặc tả căn bậc hai của kết quả là bằng với tham số đó, như đối với kết quả y , $(y * y) == x$. Nhưng lệnh số học không chính xác: căn bậc hai của 4.0 có thể trả về là 2.0, nhưng căn bậc hai 3.79512 có thể trả về 1.94811, 1 số không thực sự chính xác.

Để giải quyết sai sót này, bằng cách nào chúng ta có thể nói rằng một kết quả sẽ bị đóng trong vài giá trị? Có một lĩnh vực tính toán khác được gọi là phương thức số học dành cho loại vấn đề này. Để áp dụng phương pháp số học chúng ta sẽ cần tính 1 lỗi lớn nhất có thể và sử dụng vào trong đặc tả của chúng ta. Trong ví dụ squareRoot, nếu chúng ta tóm lược số lỗi lớn nhất có thể bên trong 1 hàm được gọi squareRootError và truy cập tới chức năng xâm phạm để trả lại giá trị dương cho tham số của nó. Chúng ta có thể định rõ kết quả y , $\text{pos}((y * y) - x) \leq \text{squareRootError}()$. Đặc tả này có thể được thêm vào chú thích hàm và mã hóa như 1 sự kiểm tra lúc cuối.

```
/*
Returns the positive square root of x.
Preconditions: x >= 0.
Postconditions: For result y,
(y >= 0) and pos((y * y) - x) <= squareRootError().
*/
float squareRoot(float x) {
    if (x < 0) {
        fail("squareRoot", "Parameter x can't be negative");
    }
    ... /* Code to calculate y */
    if (y < 0) {
        fail("squareRoot", "Calculation gave negative result");
    }
}
```

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

```
if (pos((y * y) - x) > squareRootError()) {  
    fail("squareRoot", "Calculation was inaccurate");  
}  
return y;  
}
```

Tốt hơn hết việc bảo vệ máy khách trình biên dịch khỏi các lỗi trong code của chúng và cũng đảm bảo ngăn chặn các lỗi trong code của mình. Nếu chúng ta không có khả năng để tính giá trị lỗi lớn nhất có thể (hoặc chúng ta mắc phải quá nhiều vấn đề) chúng ta vẫn nên thêm đặc tả tới chú thích như ngôn ngữ tự nhiên bình thường của kết quả là bằng x,

10.6 ĐẶC TẢ HƯỚNG ĐỐI TƯỢNG

Các ngôn ngữ đặc tả hình thức như là VDM thông thường được thiết kế để sử dụng cho việc phát triển phần mềm có cấu trúc. Từ đó ngôn ngữ này được mở rộng để làm việc với ngôn ngữ hướng đối tượng. Một số ngôn ngữ khác như OCL được thiết kế riêng cho việc phát triển phần mềm hướng đối tượng. Sự khác biệt chính giữa một ký hiệu cấu trúc và ký hiệu hướng đối tượng là hướng đối tượng sử dụng số lượng bất kỳ phạm vi lớp hơn là một phạm vi toàn cục đơn nhất. Thuật ngữ scope nghĩa là name space, partition, subarea, hoặc bất kỳ thứ gì mà bạn có thể đặt 1 hàng rào xung quanh.

Đặc tả hướng đối tượng đơn giản hơn đặc tả thủ tục bởi vì tất cả các quy tắc xuất hiện dựa theo các khái niệm có liên quan của chúng (một lớp hoặc đối tượng trong câu hỏi). Ngôn ngữ đặc tả hướng đối tượng phải cho phép chúng ta xác nhận:

- Khi nào một thông điệp có thể được gửi tới một đối tượng (các tiền điều kiện của message được biểu diễn dưới dạng các thuộc tính public).
- Các giá trị tham số của mỗi thông điệp (các tiền điều kiện của message – message preconditions)
- Tác động một thông điệp về việc nhận đối tượng (các tiền điều kiện message, biểu diễn dưới dạng các thuộc tính public)
- Các đáp ứng cho mỗi thông điệp (message postconditions)
- Các điều kiện mà luôn được đáp ứng bởi đối tượng đó (các biến của lớp, các điều kiện sẽ luôn đúng cho một thể hiện của một lớp, biểu diễn dưới dạng các thuộc tính public)

Cũng như các đối tượng, các thuộc tính và các message của đối tượng, Chúng ta cũng có các lớp, các thuộc tính lớp và các message của lớp. Như vậy 5 yêu cầu ở trên cũng có thể được ứng dụng cho các lớp mặc dù ký hiệu thường làm nó khó để nói lên sự khác nhau.

Xem ví dụ dưới đây xét lớp Container có message *add*. Cho rằng chúng ta muốn xác nhận như sau:

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

- Preconditon(Tiền điều kiện): Đối tượng tham chiếu được chuyển tới message *add* không phải là null.
- Precondion: Đối tượng được chuyển tới message *add* không tồn tại trong lớp Container.
- Postcondition: Sau message *add đó*, lớp Container sẽ có nhiều hơn một đối tượng mà trước đó lớp đã có.
- Postcondition: Sau message *add*, lớp Container sẽ chứa đối tượng mà đã được chuyển tới như là một tham số.
- Invariant: Lớp Container luôn luôn chứa số lượng các đối tượng cụ thể.

Trong 2 phần tới, chúng ta sẽ xem xét cách để biểu diễn các xác nhận của lớp Container trong OCL – ngôn ngữ đặc tả hình thức và trong Eiffel – ngôn ngữ lập trình hướng đối tượng bao gồm các cú pháp lệnh cho các đặc tả phi hình thức.

10.6.1 Đặc tả hình thức với OCL

OCL là ngôn ngữ đặc tả hình thức phù hợp với phương pháp thiết kế hướng đối tượng sử dụng các ký hiệu UML. Ví dụ lớp Container, chúng ta có thể sử dụng các đoạn sau đây của OCL. Trong đó phương thức *contains* trả lại giá trị *true* khi và chỉ khi bên nhận chứa ‘o’:

```
context Container::  
    add(o:Object)  
        pre: (o <> null) and not contains(o)  
  
        post: contains(o) and (size = size@pre + 1)  
context Container::  
    inv: size >= 0
```

OCL cho phép chúng ta kết nối các xác nhận (đòi hỏi) tới một lớp sử dụng từ khóa “context”, có nghĩa là chúng ta cũng có thể gọi tới các hàm toàn cục và dữ liệu toàn cục trong các đặc tả của chúng ta, đó chính là lợi ích của các kiểu ngôn ngữ lai như C++. Các Postcondition thường cần liên quan tới giá trị của một thuộc tính trước khi phương thức thực thi – điều này cho phép chúng ta nói tới hiệu quả mà một phương thức tồn tại các thuộc tính bên nhận. Trong trường hợp này, *size@pre* liên quan tới các giá trị của *size* trước khi *add* được thực thi. Fragment OCL ở trên có thể được hiểu như sau:

Khi gửi message *add* tới lớp Container, tham số o phải khác null và o không được tồn tại trong container. Khi mà phương thức đã hoàn thành thì container sẽ chứa o và kích cỡ của container sẽ lớn hơn so với trước đó. Kích cỡ của lớp Container luôn luôn lớn hơn hoặc bằng o.

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

10.6.2 Đặc tả phi hình thức trong Eiffel

Eiffel, ngôn ngữ được phát triển bởi Bertrand Meyer giữa năm 1980, được thiết kế cho các đặc tả phi hình thức với kiểm tra động. Lập trình viên Eiffel có thể đặt các assertion trong mã nguồn nhưng tách riêng các lời chú thích và các code cài đặt: các assertion được nhận ra bởi trình biên dịch Eiffel như là một phần của cú pháp. Eiffel sử dụng các yêu cầu thay cho *pre*, *ensure* thay thế cho *post*, và *old* thay cho *@pre*. Ví dụ lớp Container sử dụng Eiffel:

```
class CONTAINER
    feature {ANY} - Public stuff follows
        size: INTEGER
        add(o: OBJECT) is
            require
                (o != void) and not contains(o)
            do
                .....
            ensure
                contains(o) and (size = old size + 1)
            end
        invariant
            size >= 0
    end
```

Sự trộn lẫn giữa các đặc tả và cài đặt trong Eiffel là tiện lợi cho người lập trình. Hơn nữa khi trình biên dịch hiểu được cú pháp đó thì nó có thể được xác nhận (để chắc chắn rằng chúng không chứa các lỗi cú pháp hoặc các lỗi ngữ nghĩa nào).

Như chúng ta đã biết hàm *squareRoot* viết trong C, các lập trình viên có thể viết code để kiểm tra các khẳng định một cách linh động. Với Eiffel, trình biên dịch có thể dễ dàng xác định các khẳng định bởi vì chúng được tách ra khỏi các phần thân phương thức và nó có thể tự động sinh ra code kiểm tra vì trình biên dịch hiểu được cấu trúc của khẳng định.

Cho đến khi các tiền điều kiện và hậu điều kiện được kết nối với nhau thì hiệu quả của việc kiểm tra tự động như thế là nếu chúng ta chèn các lệnh *if* vào để kiểm tra thành phần *require* ở phần đầu phương thức và thành phần *ensure* ở phần kết thúc của phương thức. Nhưng bất biến về những gì? Các khẳng định trong mệnh đề bất biến luôn phải đúng cho các đối tượng của lớp. Tuy nhiên, để kiểm tra trên thực tế, chúng ta cần chạy một vài phần code và chỉ một nơi duy nhất trong ngôn ngữ hướng đối tượng là bên trong các phương thức. Bởi vậy các bất biến cần được kiểm tra bất cứ khi nào phương thức đó chạy.

CHƯƠNG 10. ĐẶC TÁ CÁC GIAO DIỆN CỦA LÓP

Nếu các bất biến được kiểm tra tại phần bắt đầu mỗi phương thức, chúng ta sẽ không thể chỉ ra nếu bản thân đối tượng đó phá vỡ bất biến đó; nếu chúng được kiểm tra ở cuối mỗi phương thức thì chúng ta không thể chỉ ra mặc dù bất biến đó đã bị mất khi phương thức được gọi. Bởi vậy các bất biến phải được kiểm tra khi bắt đầu mỗi phương thức (trước các tiền điều kiện) và ở cuối mỗi phương thức (sau các hậu điều kiện). Lý do để kiểm tra trước các tiền điều kiện và sau hậu điều kiện là các tiền điều kiện và hậu điều kiện có thể thay đổi các thuộc tính (ngay cả khi chúng không cần thay đổi). Trong thực tế, một bất biến có thêm một tiền điều kiện và một hậu điều kiện cho mỗi phương thức.

Bạn có thể nghĩ “Khi chúng ta luôn kiểm tra để các phương thức không làm phá vỡ các bất biến, chúng ta có thể chắc chắn rằng các bất biến sẽ an toàn khi chúng ta nhập một phương thức không”. Tuy nhiên, các đối tượng khác trong hệ thống có thể thay đổi các thuộc tính (bằng việc gửi cho chúng các thông điệp), có khả năng vi phạm các bất biến. Nếu các đối tượng khác kia không có phần kiểm tra các bất biến, chúng ta phải thừa nhận là các bất biến bị mất khi chúng ta bắt đầu mỗi phương thức.

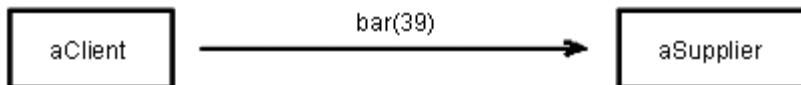
10.7 THIẾT KẾ BẰNG HỢP ĐỒNG

Bất cứ khi nào một phần của phần mềm đang chạy, chúng ta phải chấp nhận rằng cái gì đó có thể không đúng: có thể là lỗi của chương trình, người sử dụng nhập không đúng dữ liệu; chương trình dựa trên phần mềm nguồn mở có thể bị lỗi; có thể bị các lỗi trong hệ thống chạy, lỗi của hệ điều hành hoặc phần cứng. Nếu chúng ta xem mỗi phần của phần mềm là tiến trình riêng biệt đang chạy trên hạ tầng nào đó (hệ điều hành với phần cứng), chúng ta có thể chia toàn bộ phần mềm thành các hoạt động bên trong tiến trình và hành động bên ngoài tiến trình. Với quy mô lớn, chúng ta có thể điều khiển cái gì diễn ra bên trong tiến trình sử dụng thiết kế tốt, nguyên tắc lập trình tốt và các nguyên lý kiểm thử tốt. Tuy nhiên, chúng ta không thể điều khiển những gì xảy ra bên ngoài tiến trình. Thậm chí nếu chúng ta truy cập vào tiến trình khác mà chúng được cài đặt là tin cậy, việc giao tiếp giữa các tiến trình được cung cấp bởi cơ sở hạ tầng của chúng ta có thể bị lỗi; ngoài ra chúng ta phải thoát ra khỏi tiến trình của chúng ta để truy nhập phần cứng, như một mạng hoặc một hệ thống file mà lần lượt bị lỗi. Trước khi chúng ta có thể hy vọng viết phần mềm trong sáng, tin cậy và hiệu quả, chúng ta phải có sự hiểu biết rõ ràng về các loại lỗi khác nhau và ai có trách nhiệm sao chép, khắc phục mỗi loại.

Khi chúng ta bắt đầu học lập trình, hầu hết chúng ta đều cho rằng không có gì là không chính xác. Sau đó chúng ta chạy một chương trình, một lỗi xuất hiện và chương trình thất bại, vì vậy chúng ta học để ngăn chặn các lỗi tốt hơn là hủy bỏ chương trình của chúng ta – khắc phục lỗi – hoặc chắc chắn rằng các chương trình của chúng ta bị chết bởi các thông điệp giải thích từ hệ điều hành. Có nhiều lỗi có thể xảy ra cho nên có rất nhiều chỗ mà chúng ta phải kiểm tra các lỗi. Để tránh mọi thứ tồi tệ hơn, cách thực tế tốt nhất khuyên khích chúng ta cài đặt tách biệt các module phần mềm (hàm, hệ thống

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

con, các lớp) để xác định ai chịu trách nhiệm cho việc tìm lỗi hoặc xử lý lỗi. Điều này có thể dẫn đến số lượng code kiểm tra lỗi lớn, hầu hết chúng là dư thừa.



Hình 10.1 Gửi message cho đối tượng.

Hãy nhìn vào trường hợp xấu nhất trong chương trình hướng đối tượng: design by fear. Ví dụ một đối tượng gọi *aClient*, bên trong phương thức *foo*, muốn gửi một message *bar* đến *aSupplier* như Hình 10.1 (cả 2 đối tượng đang chạy trong tiến trình). Cho rằng code cho hai đối tượng này được viết độc lập (thậm chí 2 lớp được cài đặt bởi cùng một người, có thể có kẽ hở quan trọng giữa việc cài đặt cho mỗi lớp đủ để người cài đặt không còn tin tưởng vào lớp cũ hơn nữa).

Cho rằng người cài đặt lớp Client là Beryl và người cài đặt lớp Supplier là Fred. Với ý tưởng trên, Beryl sẽ viết đơn giản *result = aSupplier.bar(anObject)* và cho rằng *result* có giá trị. Với Fred sẽ cho rằng *bar* đã gọi đúng thực hiện việc tính toán và trả lại kết quả. Tuy nhiên, họ nhận ra rằng chưa hoàn chỉnh, cả hai cùng tiến hành cẩn trọng hơn.

Trong khi viết phương thức *foo*, Beryl nghĩ rằng “mình nên kiểm tra *aSupplier* xem ở bên trong trạng thái có thích hợp để nhận message của *bar* trước khi mình gửi nó”. Vì vậy, cô ấy đã thêm vào code phần kiểm tra trạng thái của *aSupplier* với ý nghĩ rằng “Mình không thể chắc chắn rằng tham số mà mình truyền qua có giá trị, vì vậy cần phải kiểm tra nó”. Cô ấy đã code thêm 2 phần kiểm tra và thêm vào code để gửi message. Với kết quả trả lại, Beryl nghĩ ”Mình không thể chắc chắn rằng kết quả trả lại có giá trị” nên cô ấy thêm code để test kết quả. Tuy nhiên, Beryl quá cẩn thận khi không thể tin vào giá trị trả về từ *aSupplier* sau khi nhận message phản hồi. Rốt cuộc, message có thể có hiệu quả không thích hợp mà không được báo trước bởi người cài đặt nó. Nên Beryl thêm vào 1/4 đoạn code kiểm tra để chắc chắn rằng *aSupplier* có giá trị trước khi tiến hành. Beryl thiết kế phương thức *foo* trong Java như sau:

CHƯƠNG 10. ĐẶC TÁ CÁC GIAO DIỆN CỦA LỚP

```
void foo() {
    ...
    if (! ... check aSupplier ...) {
        fail("Invalid state of a supplier");
    }
    if (! ... check anObject ...) {

        fail("Invalid parameter for a supplier message");
    }

    int result = aSupplier.bar(anObject);

    if (! ... check result ...) {
        fail("Invalid result from a supplier message");
    }
    if (! ... check aSupplier ...) {
        fail("Invalid state of a supplier");
    }
    ...
}

}
```

Hàm *fail* là một phương thức trên *Client* mà in ra tên lớp, tên của phương thức bị lỗi và các tham số được chuyển vào và sau khi ngắt tiến trình thì thoát ra với hàm *System.exit(-1)*. Khi Fred viết phương thức *bar*, anh ấy cũng làm theo cách tốt nhất để viết code dự phòng, mạch lạc. Trước khi anh ấy thực hiện bất cứ tính toán nào, anh đều kiểm tra xem đối tượng ở trạng thái hiện tại có hợp lệ để nhận *bar* không. Tiếp đó, anh ấy kiểm tra các tham số có hợp lệ hay không? Cuối cùng, thật là ngốc nghếch để tin vào client. Một lần nữa để chắc chắn rằng bên nhận đã sẵn sàng và dữ liệu hợp lệ được chuyển qua, Fred thêm vào code phần tính kết quả. Trước lệnh *return*, Fred quyết định rằng đó sẽ là một ý tưởng tốt để kiểm tra xem kết quả hợp lệ hay không. Vì có nhiều lỗi trong các phép tính mà anh ta đã viết hoặc mã lệnh khác mà anh ta đã gọi. Vẫn cảm thấy chưa hài lòng, Fred thêm vào một phần code kiểm tra để chắc chắn rằng việc tính toán không làm sai lệch đối tượng hiện tại. Fred đã thiết kế phương thức *bar* trong Java như sau:

```
public int bar(Object anObject) {
    if (! ... check this object ...) {
        fail("Invalid state of this object");
    }
    if (! ... check anObject ...) {
        fail("Invalid parameter from a client");
    }
    int result = ...
    if (! ... check result ...) {
        fail("Invalid result for a client");
    }
    if (! ... check this object ...) {
```

```

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP
    fail("Invalid state of this object");
}

return result;
}

```

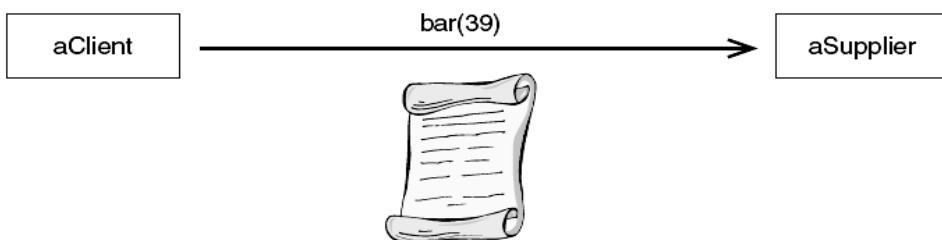
Kết quả cuối cùng là nhiều code và có lẽ là nhiều code kiểm tra lỗi hơn là code cho chức năng thông thường: Trạng thái của *aSupplier* được kiểm tra hai lần trước và sau khi tính toán, *anObject* được kiểm tra hai lần, *result* được kiểm tra hai lần. Mặc dù bạn có thể không hoàn toàn tỉ mỉ như Fred và Beyl, nhưng code của họ vẫn không dễ chịu và thân thiện. Tình huống này áp dụng tương đối tốt cho foo và bar được viết như các hàm, các thủ tục, hoặc các thủ tục con riêng riêng. Code kiểm tra lỗi có thể thậm chí dài dòng hơn nếu chúng ta tận dụng bất kỳ các kỹ thuật nào được liệt kê dưới đây:

- Mã kết quả : Một mã kết quả là một số được trả lại từ một routine (thường trình) để xác định thành công hay thất bại. Ví dụ như, zero xác định thành công, một giá trị âm xác định thất bại. Khi được sử dụng, các mã kết quả được áp dụng đặc trưng cho mỗi loại routine, không quan tâm đến kết quả trả về thông thường. Điều này dẫn đến các kiểm tra “khi mã kết quả là được”.
- Các biến lỗi toàn cục: Ở đây thay vì là trả về một giá trị thành công, lập trình viên sẽ thiết lập một giá trị toàn cục (Error trong thư viện hàm C là ví dụ tiêu biểu). Các hàm trả lại các giá trị của chúng như thông thường, cài đặt các biến lỗi nếu có vấn đề. Tuy nhiên, điều này dẫn đến nhiều kiểm tra “Nếu biến lỗi không là zero” (Biến lỗi toàn cục trong hướng đối tượng sẽ là một thuộc tính của lớp gọi là Error).
- Ngoại lệ: Một ngoại lệ là một lỗi được cảnh báo bởi supplier code và làm cho chương trình nhảy đến đoạn mã xử lý được giữ để tách từ các phần code thông thường. Để hiệu quả, các ngoại lệ có thể loại bỏ các khối kiểm tra lỗi từ code thông thường. Tuy nhiên, không có một quy tắc thiết kế nào chỉ rõ ai chịu trách nhiệm cho việc xử lý, vì thế các ngoại lệ làm cho vấn đề trở nên tội tệ hơn.

Trong năm 1980, Bert and Meyer mô tả cách để kết hợp các đặc trưng tốt nhất của các phương pháp hình thức và lập trình hướng đối tượng để dễ dàng để sinh code mạnh, trong sáng và hiệu quả. Meyer gọi kỹ thuật này là thiết kế theo hợp đồng (Design by Contract). Mặc dù Meyer mô tả và thể hiện kỹ thuật này trong Eiffel với cài đặt các yêu cầu sử dụng và kiểm tra động, chúng ta cũng có thể sử dụng các kỹ thuật này tương tự trong các ngôn ngữ khác – chúng ta đã thấy nó được sử dụng ở trong C.

Với thiết kế theo hợp đồng, chúng ta được yêu cầu để hình dung một hợp đồng kết nối giữa một đối tượng máy khách và đối tượng cung cấp với các điều bắt buộc giữa hai bên. Ví dụ như hình dưới:

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP



Bằng cách sử dụng kỹ thuật đặc tả hình thức, chúng ta có thể thấy rõ ràng về các điều cần đáp ứng hợp đồng giữa hai bên:

- Khách hàng phải tôn trọng các bất biến của bên cung cấp.
- Việc sử dụng phương thức bên cung cấp là tùy thuộc vào các tiền điều kiện của nó
- Việc cài đặt phương thức bên cung cấp phải đảm bảo các hậu điều kiện của nó.
- Việc cài đặt phương thức bên cung cấp phải đảm bảo các bất biến của nó.

Trong khi một phương thức đang thực hiện công việc của nó thì nó được phép vi phạm hợp đồng, miễn là hợp đồng được thỏa mãn khi công việc được hoàn thành. Ví dụ nếu chúng ta có invariant của aSupplier là $a + b = 4$ và hai thuộc tính đều có giá trị bằng 2 khi đó bar được gọi, nó sẽ hợp lệ cho bar để trả lại $a = 10$ và $b = -6$; Tuy nhiên nếu chúng ta không thể thiết lập cả 2 giá trị ở cùng một thời gian trong hầu hết các ngôn ngữ lập trình, $a + b$ sẽ có lúc trong bar có giá trị 12 hoặc -4. Một lớp cũng có thể có một hợp đồng bên trong nó, dưới dạng các bất biến cho các thuộc tính không public và các tiền điều kiện và hậu điều kiện cho các phương thức không public. Một hợp đồng bên trong giúp ích cho việc cài đặt và bảo trì hơn là hợp đồng bên ngoài.

10.7.1 Hợp đồng và kế thừa

Vì chương trình hướng đối tượng hỗ trợ kế thừa nên chúng ta cần phải xét đến hiệu quả của kế thừa trong hợp đồng. Bởi vì tính đa hình chúng ta phải đảm bảo cho mỗi lớp con trong hợp đồng tương đương hoặc tốt hơn các lớp cha, từ quan điểm nhìn của client (hoặc các lớp cha trong trường hợp đa kế thừa). Không tôn trọng quy tắc thì chúng ta có thể gây ra sự khó chịu cho một lập trình viên sử dụng đối tượng lớp con qua một biến của lớp cha. Ví dụ nếu hợp đồng của lớp Customer chỉ ra tất cả các khách hàng có ít nhất hai tài khoản ngân hàng và chúng ta tạo lớp con SimpleCustomer chỉ có một tài khoản, mọi sự truy nhập đối tượng SimpleCustomer qua một biến Customer phải cố gắng truy nhập vào tài khoản thứ hai không tồn tại.

Vì chúng ta có thể bỏ tính kế thừa lớp, nên các bất biến được kết hợp bằng sử dụng “and”: chúng ta chỉ có thể thêm các bất biến. Ví dụ lớp X có hằng là i1 và lớp con Y có hằng là i2, vậy thì lớp Y có hai biến hằng là i1 và i2. Điều này ngụ ý rằng chúng ta đang đáp ứng thêm sự đảm bảo về các thuộc tính.

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

Các tiền điều kiện và hậu điều kiện chỉ là một kết quả nếu chúng ta định nghĩa lại các phương thức liên kết trong lớp con. Với tiền điều kiện, chúng ta phải chắc chắn rằng phương thức định nghĩa lại chấp nhận ít nhất hoặc tất cả các thông điệp mà nó đã chấp nhận trong lớp con: chúng ta chỉ có thể làm yêu đi các tiền điều kiện. Bởi vậy bất kỳ tiền điều kiện nào thêm vào mà để định nghĩa lại phương thức thì được kết hợp với những các tiền điều kiện khác trong phương thức kế thừa sử dụng “or”. Ví dụ, nếu foo có một tiền điều kiện và lớp con thêm vào một cái nữa là pre2, hiệu quả chính là phương thức được định nghĩa lại có pre1 hoặc pre2.

Với hậu điều kiện chúng ta phải chắc chắn rằng mỗi lớp con đảm bảo ít nhất là bằng với lớp cha: chúng ta chỉ có thể làm tăng thêm hậu điều kiện. Bởi vậy, hậu điều kiện kết hợp với việc sử dụng “and”. Ví dụ khi foo có post1 và lớp con thêm post2 thì phương thức định nghĩa lại sẽ kết hợp cả 2 post1 và post2.

- Nếu các biến bị bỏ qua, điều đó có nghĩa là “không có cái gì đảm bảo cho lớp đó”; chúng ta thêm các biến vào lớp con khi đó sẽ không có ở lớp cha, tác động là các biến mà ta thêm vào ở lớp con.
- Nếu tiền điều kiện bị bỏ qua, nghĩa là “thông điệp áp dụng trong tất cả các trường hợp”, thêm vào tiền điều kiện phương thức định nghĩa lại trước đây không có, các pre mới thì sẽ không có hiệu quả (chúng ta không thể làm yêu phương thức đã chấp nhận mọi thứ).
- Nếu hậu điều kiện bị loại bỏ, nghĩa là “thông điệp không đảm bảo mọi thứ”, nếu chúng ta thêm vào hậu điều kiện cho phương thức định nghĩa lại trước đây không có, hiệu quả chính là ở hậu điều kiện trong phương thức định nghĩa lại (mọi cái chúng ta thêm vào làm mạnh hơn cái mà trước đó không có sự đảm bảo).

Với sự giúp đỡ của thiết kế bằng hợp đồng và kỹ thuật đặc tả hình thức, Beryl viết code rõ ràng hơn.

```
void foo() {  
    ...  
    if ( ! ... check aSupplier's invariants ... ) {  
        fail("Invariants broken for a supplier");  
    }  
    if ( ! ... check bar's preconditions ... ) {  
        fail("Preconditions broken for a supplier message");  
    }  
  
    int result = aSupplier.bar(anObject);  
    if ( ! ... check bar's postconditions ... ) {  
        fail("Postconditions broken by a supplier method");  
    }  
}
```

CHƯƠNG 10. ĐẶC TÁ CÁC GIAO DIỆN CỦA LỚP

```
if (! ... check aSupplier's invariants ...) {  
    fail("Invariants broken by a supplier method");  
}  
...  
}
```

Tương tự Fred cũng đã viết lại code của anh ấy:

```
public int bar(Object anObject) {  
    if (! ... check invariants ...) {  
        fail("Invariants broken");  
    }  
    if (! ... check preconditions ...) {  
  
        fail("Preconditions broken");  
    }  
  
    int result = ...  
  
    if (! ... check postconditions ...) {  
        fail("Postconditions broken");  
    }  
  
    if (! ... check invariants ...) {  
        fail("Invariants broken");  
    }  
  
    return result;  
}
```

10.7.2 Giảm lỗi – kiểm tra mã

Thuận lợi chính của thiết kế bằng hợp đồng là sự phân chia rõ ràng giữa các trách nhiệm của khách hàng và những trách nhiệm của người cung cấp. Trước khi thông điệp của bar được gửi, khách hàng đó kiểm tra. Như vậy khách hàng có trách nhiệm đảm bảo rằng hợp đồng đó không bị phá vỡ trước khi phương thức đó trả về. nó chỉ cho chúng ta cách để loại bỏ mã kiểm tra, một phần loại bỏ dư thừa. Kết quả này phát triển nhanh, hiệu năng tốt hơn, giảm lỗi và bảo trì dễ hơn.

Chúng ta có thể sử dụng thiết kế theo hợp đồng để giảm bớt mã kiểm tra hơn nữa được không? Được. Có nhiều dịp khi một phần code mà chúng ta viết không có hợp đồng.

Nếu người cung cấp không thể thoát khỏi phần quan trọng của hợp đồng, không có mã kiểm tra nào cần thiết. Theo nguyên tắc, các lỗi nền tảng có thể phá vỡ hợp đồng, nhưng nó sẽ không khả thi để chúng ta thêm mã cho các tình huống như vậy.

Xét cài đặt foo sau đây trên client:

```
void foo() {
```

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

```
Supplier aSupplier = new Supplier();  
int result = aSupplier.bar(new Plate("Wedgwood"));  
...  
}
```

Khi thông điệp bar được gửi, chúng ta biết rằng foo không thể vi phạm những bất biến của aSupplier vì nó hiển nhiên là một đối tượng có giá trị. Như vậy chúng ta không cần bất kỳ mã kiểm tra nào trong foo khác. Kết quả cuối cùng là chương trình không có bất cứ mã kiểm tra nào. Chúng ta xem rằng các nghĩa vụ được xác định tốt, tin cậy lẫn nhau và tính chất của code mà chúng ta đang viết có thể giảm một nửa so với số lượng mã kiểm tra, hoặc thậm chí loại bỏ tất cả. Những gì chúng ta vẫn chưa bao quát hết là cách chúng ta nên đề cập đến thực tế là phần mềm của chúng ta tồn tại trong một thế giới không hoàn hảo: code của khách hàng và nhà cung cấp có thể chứa các lỗi; nếu chúng ta vượt khỏi tiến trình của chúng ta tới tiến trình khác hoặc bộ phận phần cứng khác, điều khó chịu có thể xảy ra với việc điều khiển của chúng ta. Các lỗi bên trong tiến trình của chúng ta có thể được giải quyết bằng cách tuân theo hợp đồng. Các vấn đề bên ngoài tiến trình của chúng ta có thể được giải quyết bằng cách xây dựng các tường lửa ứng dụng.

10.7.3 Việc tuân theo các hợp đồng

Mục đích của hợp đồng là dựa trên những điều có thể chấp nhận được với điều kiện khách hàng phải đáp ứng những yêu cầu của nó, nhà cung cấp cũng phải đáp ứng những yêu cầu đó. Nhưng những người lập trình có thể đôi khi không đáp ứng được các yêu cầu, vì thế chúng ta cần kiểm tra một cách liên tục để tìm ra những lỗi sai. Nhưng ai sẽ chịu trách nhiệm kiểm tra những yêu cầu của hợp đồng có được thỏa mãn hay không? Và chúng ta nên làm gì nếu chúng ta tìm ra một lỗi sai?

Trong ví dụ client và supplier, phương thức Bar phải chịu trách nhiệm kiểm tra những điều kiện và tính bất biến trước khi nó trả lại giá trị, bởi vì chỉ có phương thức Bar hoặc một trong những supplier của nó mới có thể phá vỡ hợp đồng đó trong khoảng khởi đầu và kết thúc phương thức. Vì thế, việc kiểm tra liên tục đối với supplier (các hậu điều kiện và tính bất biến) nên xuất hiện ở cuối phương thức Bar.

Câu hỏi của người sẽ kiểm tra tính bất biến và các tiền điều kiện trước khi phương thức Bar bắt đầu thì phức tạp hơn. Client code có trách nhiệm để chắc chắn rằng client đã không phá vỡ một nửa hợp đồng. Nhưng nếu chúng ta kết luận rằng mỗi client của Bar phải có hợp đồng kiểm tra mã, thì mã sẽ gấp đôi lên.

Nguyên lý hướng đối tượng phát biểu rằng trách nhiệm nên được chia cho những đối tượng có liên quan. Đề nghị kiểm tra bên phía khách hàng của hợp đồng nên là việc liên quan đến supplier (bởi vì những điều kiện và tính bất biến có liên quan đến supplier). Như vậy để ngăn chặn một bản sao kiểm tra hợp đồng của client, chúng ta nên đặt nó ở

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

đầu của Bar. Bạn có thể thấy kết quả này ở một trong hai cách: một là Bar bảo vệ cho supplier khỏi những lỗi sai trong client; hai là Bar với điều kiện là một dịch vụ cho client, để kiểm tra là nó không có lỗi. Với dạng là một hợp đồng thì cách hiệu thứ hai thích hợp hơn.

Kết hợp những phương thức, bản thảo của hợp đồng, việc kiểm tra liên tục và tin cậy, ta cài đặt bằng java như sau:

```
public int bar(Object anObject) {  
    if (! ... check invariants ...) {  
        fail("Invariants broken");  
    }  
    if (! ... check preconditions ...) {  
        fail("Preconditions broken");  
    }  
    int result = ...  
    if (! ... check postconditions ...) {  
        fail("Postconditions broken");  
    }  
    if (! ... check invariants ...) {  
        fail("Invariants broken");  
    }  
    return result;  
}
```

Với việc cài đặt ở trên, không hợp đồng kiểm tả mã nào là cần thiết.

10.8 ĐẶT TẢ PHI HÌNH THỨC TRONG JAVA

Nếu chúng ta đang cài đặt thiết kế với một ngôn ngữ như là Eiffel, thực tế tốt nhất (sử dụng các đặc tả phi hình thức và kiểm tra động) là gắn vào chính ngôn ngữ của nó. Khi sử dụng những ngôn ngữ khác, chúng ta nên áp dụng những nguyên tắc tương tự, nhưng chúng ta phải viết mã kiểm tra động bằng tay. Trong phần này, chúng ta sẽ xem cách áp dụng nguyên lý đặc tả cho java, bởi java phổ biến, thực tế hơn Eiffel.

10.8.1 Viết tài liệu một hợp đồng với chú thích

Chúng ta nên luôn luôn hay chí ít nên viết tài liệu hợp đồng của các lớp sử dụng các chú thích trong mã nguồn. Vì mã lệnh của chúng ta sẽ hoàn thành để được dùng, được sử dụng lại hoặc được bảo trì bởi những người lập trình khi xem xét mã nguồn đó. Bất kỳ những chú giải nào của hợp đồng liên quan với mã nguồn cũng nên xuất hiện trong các module thiết kế và đặc tả.

Mỗi lớp nên có một chú thích ở đầu để miêu tả bất kỳ những bất biến nào cho những thuộc tính chung của nó. Có thể những người lập trình khác xem mã nguồn không

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

thể quen thuộc với kiểu hình thức, vì thế bạn có thể sử dụng những pha như đổi với tất cả các đối tượng của lớp này bằng cách áp dụng các điều kiện sau đây:

- Chú thích lớp có thể cũng chứa những ví dụ về cách các lớp nên được sử dụng như thế nào.
- Mỗi phương thức chung cho lớp nên có một lời chú thích ở đầu để mô tả phương thức nhằm đáp ứng những gì và liệt kê bất cứ tiền điều kiện và hậu điều kiện nào. Do những người đọc không am hiểu, bạn nên sử dụng các thuật ngữ như là yêu cầu và đảm bảo hơn là tiền điều kiện và hậu điều kiện.
- Hãy luôn giữ lập trường, khi cài đặt một phương thức mà trách nhiệm của những người gọi nó để chắc chắn tính bất biến và tiền điều kiện phải được đáp ứng trước khi phương thức được gọi. Trong mọi tình huống, việc mã hóa của bạn nên lạc quan. Tất nhiên, bạn nên chắc chắn rằng cài đặt của bạn vi phạm hậu điều kiện hoặc tính bất biến

10.8.2 Kiểm tra các điều kiện một cách linh động

Theo nguyên lý, nó là một ý kiến hay để thêm vào việc kiểm tra tính bất biến và kiểm tra tiêu kiện ở đầu một phương thức và kiểm tra hậu điều kiện và tính bất biến ở phần cuối. Tuy nhiên, không có cơ cấu ngôn ngữ cụ thể như những điều được cung cấp bởi Eiffel, bạn có thể chọn lựa nhiều hơn tùy thích.

Thật là một ý tưởng hay để thêm các kiểm tra ở đầu bất kỳ một phương thức public nào mà bạn muốn được sử dụng lại có hiệu quả (nếu code đã có sẵn cho 1 thư viện). Còn về việc kiểm tra ở cuối các phương thức public, nếu phương thức của bạn không thể dừng supplier một nửa hợp đồng, thì việc kiểm tra sẽ là thừa. Nếu như bạn nghĩ là mã lệnh của bạn có thể phá bỏ hợp đồng, thì có lẽ bạn không tin vào chính bản thân mình. Do kinh nghiệm của bạn với việc cài đặt code hướng đối tượng sẽ tăng lên, sự tin tưởng vào chính bạn sẽ lớn hơn lên, vì thế việc kiểm tra cũng trở nên không cần thiết.

Lý do thuyết phục nhất cần phải bao gồm việc kiểm tra ở cuối phương thức là nếu bạn tin rằng bên code thứ ba được gọi bởi phương thức của bạn có thể phá vỡ tính bất biến, bởi những thuộc tính đã được sửa lại: hầu hết code được gọi trong kiểu client-server và điều này không nên xảy ra thường xuyên.

10.8.3 Đánh dấu các vi phạm hợp đồng sử dụng ngoại lệ

Bất cứ lựa chọn nào về tính ép buộc hợp đồng cũng cần một phương pháp đánh dấu các vi phạm hợp đồng. Vì vậy, vấn đề được xếp theo bậc qua việc sử dụng thông điệp báo fail. Nhưng mã lệnh gì chúng ta cần làm trong việc đáp ứng phương thức fail do lỗi? Khả năng thứ nhất là thoát khỏi chương trình: Trong Java chúng ta có thể làm điều này với `System.exit(-1)`. Tuy nhiên kỹ thuật này loại bỏ hầu như toàn bộ thông tin về cách chúng

CHƯƠNG 10. ĐẶC TẢ CÁC GIAO DIỆN CỦA LỚP

ta đạt được vì những lỗi. Cuối mỗi phương thức sẽ cho chúng ta biết điều này, nhưng không phải phương thức nào cũng được gọi như thế.

Thay cho những hạn chế này, một phương thức trong Java có thể xử lý các biến lẻ, lệnh cho phương thức gọi cái mà phương thức hiện tại không thể tiếp tục. Java có một kiểu ngoại lệ riêng biệt được gọi là RuntimeException có thể sử dụng trong từng trường hợp được. RuntimeException nên được sử dụng để cho biết phương thức không thể tiếp tục được vì những lỗi trong việc thực hiện nó hoặc trong việc gọi phương thức. Việc gọi phương thức không đòi hỏi bắt buộc có RuntimeException. Thay vào đó, nó nên cho phép phương thức run-time bỏ đi ngoại lệ ở chính phương thức mà nó gọi, nơi mà nó sẽ được bỏ đi phương thức đã được gọi trước đó; cuối cùng, ngoại lệ sẽ nằm bắt bên ngoài tiến trình của main.

Khi code Java đang chạy trong một chương trình lớn, như servlet đang chạy trong một Web server hoặc một thành phần nghiệp vụ đang chạy trong GUI, chúng ta không nên cho phép những đối tượng RuntimeException hoặc bất kỳ kiểu ngoại lệ nào khác tham gia vào hàm main, nếu không toàn bộ chương trình sẽ ngừng hoạt động. Thay vào đó, code chung ở ngoài chương trình có thể xử lý những ngoại lệ đơn thuần và báo chúng với người quản trị hoặc người dùng, hoặc cả hai. Ví dụ, một Web server có thể bị gắn vào lỗi thì báo cho hệ thống ghi lại file và sau đó hiển thị thông điệp lỗi tới người dùng.

10.9 KẾT LUẬN

Trong chương này chúng ta đã xem xét:

- Các đặc tả là các mô tả rõ ràng, trọn vẹn về các hành vi yêu cầu của phần mềm. Đặc tả hình thức có tính khoa học, nghiêm ngặt và sử dụng ngôn ngữ chuyên dụng; đặc tả phi hình thức có tính thực tế, cục bộ và có thể nói bằng ngôn ngữ tự nhiên hoặc trong cấu trúc của ngôn ngữ như là Eiffel, UML.
- Các đặc tả có thể được viết theo lối hướng đối tượng. Đặc tả hướng đối tượng đòi hỏi các tiền điều kiện, hậu điều kiện và các bất biến cho một lớp.
- Thiết kế bằng hợp đồng kết hợp các phương pháp hình thức và chương trình hướng đối tượng làm cho nó dễ dàng tạo ra mã mạnh mẽ, trong sáng và hiệu quả. Một hợp đồng kết nối giữa đối tượng client và đối tượng supplier bắt buộc theo quy tắc cả hai bên đã thỏa thuận.
- Cách sử dụng các đặc tả và thiết kế bằng hợp đồng trong Java đặc biệt bằng việc sử dụng các chú thích và các đối tượng Exception. Chúng ta cũng xem xét các khảng định trong Java và coi nó như là vấn đề tiềm năng.