Tạp chí Tin học và Điều khiển học, T.22, S.1 (2006), 63–74

VỀ MỘT CÁCH TINH CHẾ MÔ HÌNH LỚP UML

NGUYỄN MẠNH ĐỨC[1], ĐẶNG VĂN ĐỨC[2]

[1]*Khoa Toán, Trường Đại học Sư phạm - Đại học Thái Nguyên*
[2]*Viện Công nghệ thông tin, Viện KH&CN Việt Nam*

Abstract. This paper presents a refinement method in the development process of software systems based on object-oriented relations. In the process of refinement the laws of refinement in conjunction with UML (Unified Modeling Language) diagrams have been used. This allows us to build a system, step by step, from a rudimentary model to a design model which can be implemented using an object-oriented programming language. The process of the refinement has been also explained by a software development example.

Tóm tắt. Bài báo trình bày một cách tinh chế hệ thống phần mềm trong quá trình phát triển dựa trên tiệm cận quan hệ hướng đối tượng. Trong quá trình tinh chế, các luật tinh chế kết hợp với các biểu đồ của UML đã được sử dụng để từng bước xây dựng mô hình từ thô sơ của hệ thống dần tới mô hình thiết kế gần với ngôn ngữ lập trình. Kỹ thuật này đã tạo điều kiện thuận lợi cho việc cài đặt hệ thống phần mềm bằng một ngôn ngữ lập trình hướng đối tượng bất kỳ.

1. GIỚI THIỆU

Thiết kế và phát triển hệ thống phần mềm theo phương pháp hướng đối tượng là rất phức tạp ([1, 4]). Đã có nhiều nghiên cứu nhằm giải quyết tính phức tạp này, trong đó có những nghiên cứu chỉ ra sự cần thiết phát triển công cụ hình thức hóa làm nền tảng cho việc phát triển phần mềm hướng đối tượng. Bài báo sẽ trình bày một cách tinh chế mô hình UML dựa trên lý thuyết lập trình thống nhất của Hoare và He ([2]), và đề xuất áp dụng chúng vào việc xây dựng hệ thống phần mềm với đảm bảo tính đúng đắn cao.

Trong tiến trình phát triển phần mềm RUP (Rational Unified Process) trên cơ sở ngôn ngữ mô hình hóa thống nhất UML ([1, 4, 7]), một số mô hình của UML được hình thành để biểu diễn và phân tích cấu trúc hệ thống trong pha nào đó của quá trình phát triển, thí dụ các biểu đồ lớp biểu diễn phân tích tĩnh (khung nhìn tĩnh), cơ cấu trạng thái đặc tả các hành vi động và tính phù hợp (khung nhìn hành vi). Việc ngôn ngữ UML cho khả năng sử dụng đồng thời nhiều khung nhìn trong việc mô hình hóa hệ thống là có nhiều thuận lợi. Người xây dựng mô hình có thể phân tách mô hình hệ thống thành một số khung nhìn khác nhau để quản lý theo cách thức riêng. Mỗi khung nhìn sẽ tập trung vào khía cạnh riêng biệt để phân tích và hiểu rõ các đặc trưng khác nhau của mô hình hệ thống. Tuy nhiên, việc sử dụng mô hình nhiều khung nhìn sẽ phải đối mặt với các khó khăn về thời gian khác nhau, do vậy một số mô hình tinh chế đã được đề xuất ([13]): 1) Nhất quán ngang mô hình, bao gồm nhiều khung nhìn khác nhau đòi hỏi tương thích về cú pháp và ngữ nghĩa với việc quan tâm về các khía cạnh của hệ thống được mô tả trong các mô hình con; 2) Biến đổi và phát triển mô

hình (nhất quán dọc mô hình) đòi hỏi mô hình được tinh chế phải có ngữ nghĩa phù hợp dọc theo suốt quá trình; 3) Theo dõi vết mô hình, sự thay đổi trong mô hình của một khung nhìn liên quan cần được chỉ dẫn thay đổi phù hợp trong các mô hình của các khung nhìn khác; 4) Tích hợp mô hình, mô hình của các khung nhìn khác nhau cần phải tích hợp trước khi có sản phẩm phần mềm...

Việc nghiên cứu tính chất và phân tích hình thức của các mô hình UML ([11, 13]) mới được tiến hành trong những năm gần đây. Tuy nhiên, phần lớn các nghiên cứu này chỉ liên quan tới hình thức của các biểu đồ riêng lẻ và tính nhất quán của các mô hình loại 1 hoặc loại 2. Ví dụ, các nghiên cứu tập trung nhiều vào tính nhất quán của biểu đồ lớp hoặc các máy trạng thái của chúng. Từ những kinh nghiệm phát triển hệ thống phần mềm theo hướng đối tượng, chúng tôi thấy rằng nhiều công đoạn khó khăn trong việc mô hình hóa hệ thống có thể giải quyết được bằng các quy tắc tinh chế mô hình UML mới được đề xuất gần đây. Phần tiếp theo của bài báo là trình bày các kết quả ứng dụng các quy tắc tinh chế mô hình trong bài toán thực tế.

2. CƠ SỞ TÍNH TOÁN

Một chương trình hoặc tập các lệnh được coi như là thiết kế (design) được xác định bằng cặp (α, P) ([2]), ở đây α biểu thị tập các biến đã biết trong chương trình, được gọi là bảng ký kiệu của thiết kế; P là tân từ xác định quan hệ giữa các giá trị khởi tạo của các biến trong chương trình và các giá trị kết thúc của nó và có dạng $p(x) \vdash R(x, x')$, cụ thể

$$p(x) \vdash R(x, x') \underline{\underline{\text{def}}} \text{ ok} \wedge p(x) \Rightarrow \text{ok}' \wedge R(x, x'),$$

trong đó, $p(x)$ được gọi là tiền điều kiện và phải có giá trị *true* trước khi chương trình bắt đầu, $R(x, x')$ là hậu điều kiện nhận được sau khi chương trình kết thúc, x và x' biểu diễn giá trị khởi đầu và kết thúc của biến x trong chương trình, ok và ok$'$ là các biến logic mô tả trạng thái hành vi ban đầu và cuối của chương trình: nếu chương trình được kích hoạt hợp thức ok là *true*, việc thực hiện chương trình cuối cùng thành công ok$'$ là *true*, ngược lại chúng là *false*. Ký hiệu def được hiểu là "được định nghĩa" hoặc "được xác định".

2.1. Biểu diễn hệ thống hướng đối tượng [10]

Một hệ thống hoặc chương trình hướng đối tượng S có dạng cdecls $\bullet$ P ([2]), ở đây, cdecls là phần khai báo một số hữu hạn các lớp, P được gọi là phương thức chính và có dạng (glb, c), trong đó glb là tập hữu hạn các biến chung và kiểu dữ liệu của chúng còn c là các lệnh, P có thể được hiểu như là phương thức chính nếu S được tạo bởi ngôn ngữ giống như Java. Phần khai báo lớp cdecls là thứ tự của các khai báo lớp $\text{cdecl}_1, ..., \text{cdecl}_k$, ở đây mỗi khai báo lớp cdecl_i có dạng như sau:

[private] class N [extends M] {

private	$T_1\ t_1 = a_1, ..., T_m\ t_m = a_m;$
protected	$U_1\ u_1 = b_1, ..., U_n\ u_n = b_n;$
public	$V_1\ v_1 = d_1, ..., V_k\ v_k = d_k;$
method	$m_1(\underline{T}_{11}\ \underline{x}_{11}, \underline{T}_{12}\ \underline{y}_{12}, \underline{T}_{13}\ \underline{z}_{13})\ \{c_1\};$
	...;
	$m_l(\underline{T}_{l1}\ \underline{x}_{l1}, \underline{T}_{l2}\ \underline{y}_{l2}, \underline{T}_{l3}\ \underline{z}_{l3})\ \{c_l\}$

}

trong đó,

+ Mỗi lớp có thể được khai báo private hoặc là public, ngầm định là public. Chỉ các lớp có kiểu public hoặc kiểu cơ sở mới được sử dụng trong các khai báo biến chung *glb*.

+ N và M là các tên lớp khác nhau và M được gọi là lớp cha của N.

+ Phần private khai báo các thuộc tính private của lớp, kiểu và các giá trị khởi tạo của chúng. Tương tự, các phần protected và public khai báo các thuộc tính protected và public của lớp.

+ Phần method khai báo các phương thức của lớp N, trong đó $m_1, m_2, ..., m_l$ là các phương thức, ở đây $(\underline{T}_{i1}\ \underline{x}_{i1})$, $(\underline{T}_{i2}\ \underline{y}_{i2})$, $(\underline{T}_{i3}\ \underline{z}_{i3})$ và c_i biểu diễn các tham biến giá trị, các tham biến kết quả, các tham biến giá trị kết quả và phần thân của phương thức m_i. Trong tài liệu này, phương thức còn được chỉ ra bởi $m(\text{paras})\{c\}$, trong đó paras là các tham biến và c là phần thân lệnh của m.

Chúng tôi quy ước sử dụng ngôn ngữ Java để viết đặc tả lớp. Khi viết các luật tinh chế, ký pháp sau được sử dụng để chỉ sự khai báo lớp N

$$N[M, \text{pri}, \text{pro}, \text{pub}, \text{op}],$$

trong đó, M là tên lớp cha của N; pri, pro và publần lượt là các tập thuộc tính private, protected và public của N; op là tập các phương thức của N. Ta có thể chỉ đưa ra các tham số liên quan cần thiết chẳng hạn như: nếu sử dụng $N[\text{op}]$ để chỉ lớp N với tập các phương thức op, $N[\text{pro}, \text{op}]$ chỉ lớp N với các thuộc tính protected là pro và các phương thức là op.

2.2. Biểu thức

Biểu thức trong ngôn ngữ hướng đối tượng có thể xuất hiện vế bên phải của các lệnh gán, được tạo lập theo các quy tắc như sau ([2, 5]):

$$e ::= x \mid \text{null} \mid \text{self} \mid e.a \mid e \text{ is } C \mid C(e) \mid f(e),$$

trong đó, x là biến đơn, null là kiểu đối tượng đặc biệt của lớp đặc biệt NULL và là lớp con của mọi lớp, null là duy nhất, self được sử dụng để chỉ đối tượng hoạt động trong phạm vi hiện tại (một số ngôn ngữ hướng đối tượng sử dụng là this), $e.a$ là thuộc tính a của e, e is C là kiểu kiểm thử (test), $C(e)$ là biểu thức có kiểu theo khuôn mẫu, $f(e)$ là phép toán gắn liền với các kiểu nguyên thủy.

2.3. Các lệnh

Phần này xem xét các lệnh hỗ trợ việc xây dựng chương trình hướng đối tượng tiêu biểu ([2, 5]).

$c ::= \text{skip}\mid\text{chaos} \mid \text{var}Tx = e \mid \text{end}x$	bỏ qua|không xác định|khai báo|kết thúc kh.báo
$\mid c; c \mid c \lhd b \rhd c \mid c \sqcap c$	trình tự | chọn theo điều kiện | không tiền định
$\mid b * c \mid \text{le}.m(e, v, u) \mid \text{le} := e \mid C.\text{new}(x)$	lặp | gọi phương thức | gán | tạo đối tượng mới

Ở đây b là biểu thức logic, c là lệnh, e là một biểu thức, le có thể xuất hiện ở vế trái của phép gán và có dạng $\text{le} ::= x \mid \text{le}.a$ với x là biến đơn còn a là thuộc tính của đối tượng. Đa số các lệnh có ý nghĩa truyền thống trong các ngôn ngữ lệnh, có thể xem chi tiết trong [2, 5]. Sau đây là các giải thích một số lệnh đặc trưng cho chương trình hướng đối tượng.

Lệnh gán $\text{le} := e$ được xác định đúng khi le và e đã được xác định đúng và kiểu của e là kiểu con của kiểu đã được khai báo le. Trong trường hợp lệnh gán đơn $x := e$, khi lệnh gán

được xác định đúng nó chỉ thay đổi x bởi gán x bằng giá trị của e. Trong trường hợp sự thay đổi thuộc tính của đối tượng, le.$a := e$ thay đổi thuộc tính a của đối tượng le tới giá trị e.

Phương thức gọi le.$m(e, v, vr)$ xác định đúng nếu le là đối tượng không rỗng và $m(x, y, z)$ là phương thức đã được khai báo trong kiểu le. Khi nó đã được xác định đúng, việc thực hiện các lệnh gán các giá trị của các tham số thực v và vr cho các tham số hình thức x và z của m của các đối tượng hoạt động le tham chiếu tới, rồi thực hiện thân của phương thức trong môi trường của lớp đối tượng hoạt động. Sau khi thực hiện các thân cuối cùng, giá trị của tham biến kết quả và tham biến giá trị kết quả y và z được trả lại hợp quy cách với các tham số thực r và vr.

Lệnh tạo đối tượng C.new(x) được xác định đúng nếu C là lớp đã được khai báo. Sự thực hiện của nó sẽ tạo ra đối tượng của lớp C với tham chiếu mới, gắn nó với biến x và gắn giá trị khởi đầu của các thuộc tính trong lớp C với các thuộc tính của x cũng vậy.

3. TINH CHẾ HỆ THỐNG ĐỐI TƯỢNG

3.1. Các khái niệm

Sau đây ta xem xét một số khái niệm liên quan tới quá trình tinh chế mô hình hệ thống ([9, 12]):

Định nghĩa 1.(tinh chế thiết kế) Thiết kế $D_2 = (\alpha, P_2)$ là tinh chế của thiết kế $D_1 = (\alpha, P_1)$ được chỉ ra bởi $D_1 \sqsubseteq D_2$, nếu P_2 kế thừa P_1, nghĩa là: $\forall x, x', ..., z, z', \text{ok}, \text{ok}', (P_2 \Rightarrow P_1)$.

Ở đây $x, ..., z$ là các biến chứa trong α. $D_1 \equiv D_2$ nếu và chỉ nếu $D_1 \sqsubseteq D_2$ và $D_2 \sqsubseteq D_1$.

Định nghĩa 2.(tinh chế dữ liệu) Cho ρ là ánh xạ (cũng có thể được coi như là thiết kế) từ α_2 tới α_1. Thiết kế $D_2 = (\alpha_2, P_2)$ là tinh chế của thiết kế $D_1 = (\alpha_1, P_1)$ dưới ρ, được chỉ ra bởi $D_1 \sqsubseteq_\rho D_2$, nếu $(\rho; P_1) \sqsubseteq (P_2; \rho)$. Trong trường hợp này ρ được gọi là ánh xạ tinh chế.

Định nghĩa 3.(tinh chế hệ thống) Cho S_1 và S_2 là các đối tượng chương trình có cùng một tập các biến chung glb.S_1 là tinh chế của S_2, được chỉ ra bởi $S_2 \sqsubseteq_{sys} S_1$, nếu hành vi của nó có thể kiểm soát và dự đoán nhiều hơn trong S_2, tức là: $\forall \underline{x}, \underline{x}', \text{ok}, \text{ok}', \ (S_1 \Rightarrow S_2)$.

Ở đây $\underline{x}$ là các biến trong glb. Ý nghĩa này là hành vi mở rộng của S_1, đó là các cặp tiền điều kiện và hậu điều kiện của các biến chung, và là tập con của S_2. Để phạm vi chương trình S_1 tinh chế S_2 khác, cầu chúng phải có cùng tập các biến chung và tồn tại ánh xạ tinh chế từ các biến của S_1 tới S_2 là giống hệt nhau trên các biến chung.

Định nghĩa 4. (tinh chế lớp) Cho cdecls_1 và cdecls_2 là hai phần khai báo. cdecls_1 tinh chế cdecls_2, được chỉ ra bởi $\text{cdecls}_1 \sqsupseteq_{class} \text{cdecls}_2$ nếu như phần trước có thể thay thế phần sau trong bất kỳ hệ thống đối tượng: $\text{cdecls}_1 \sqsupseteq_{class} \text{cdecls}_2 \ \underline{\underline{\text{def}}} \ \forall P. \ (\text{cdecls}_1 \bullet P \sqsupseteq_{sys} \text{cdecls}_2 \bullet P)$.

Ở đây P đóng vai trò cho phương thức chính (glb, c).

Phương thức chính tương ứng với chương trình ứng dụng đang sử dụng các dịch vụ (tức là các các phương thức của các lớp trong mô hình UML). Do đó, ở đây chúng tôi chỉ quan tâm tinh chế giữa các phần khai báo.

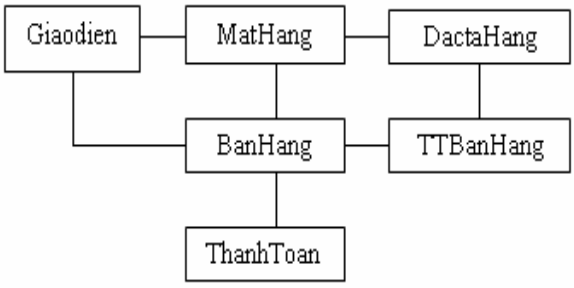
3.2. Tinh chế hệ thống

Phần này, chúng tôi trình bày quá trình xây dựng và phát triển một hệ thống hướng đối tượng, dựa trên việc áp dụng các quy tắc (luật) tinh chế ([3, 9, 12]). Tiến trình phát triển tăng dần của hệ thống thông qua trình tự của các bước tinh chế. Trong khi phát triển hệ thống chúng ta sẽ chỉ ra đó như là trình tự của các khai báo lớp. Ban đầu, phiên bản thô sơ của hệ thống là APP_0. Với sự hỗ trợ của các luật tinh chế, APP_0 sẽ được tinh chế thành APP_1. Tương tự, phiên bản APP_1 lại được tinh chế thành APP_2... tiếp tục ta đi tới mô hình thiết kế cuối cùng của hệ thống, khi nó đã gần với ngôn ngữ lập trình và thuận tiện cho việc cài đặt. Một cách trực giác, mỗi phiên bản của trình tự khai báo lớp được miêu tả bằng biểu đồ lớp UML tương ứng. Ở đây mô hình khái niệm được coi như là biểu đồ lớp mà trong đó các lớp chỉ có các thuộc tính và không có các phương thức, còn mô hình thiết kế như biểu đồ lớp mà trong đó các lớp có các thuộc tính và các đặc tả phương thức chính xác.

Các bước tinh chế hệ thống đối tượng sẽ được mô tả tiếp theo sau đây, thông qua ví dụ phân tích thiết kế hệ thống bán hàng trong một siêu thị.

3.2.1. Khởi tạo hệ thống

Tại thời điểm bắt đầu, hệ thống phải có các thành phần sau: **MatHang** (Mặt Hàng) như là cơ sở dữ liệu để lưu thông tin của tất cả các khả năng xảy ra trong việc bán các hàng hóa của siêu thị. Mỗi mục trong cơ sở dữ liệu là đặc tả hàng **DactaHang** (Đặc tả Hàng). Khi việc bán hàng bắt đầu, chúng ta cần phải xây dựng một đối tượng **BanHang** (Bán Hàng) bao gồm nhiều mục dòng bán hàng **TTBanHang** (Thông tin Bán Hàng) ghi lại tất cả các thông tin về sản phẩm đã được bán. Cuối cùng, khách hàng sẽ phải trả tiền. Do đó cần phải có đối tượng **ThanhToan** (Thanh Toán). Trong khi thực hiện chúng ta sẽ tạo nhiều thể hiện của BanHang, TTBanHang và ThanhToan. Cuối cùng chúng ta cần phải có lớp như là giao diện của hệ thống để người sử dụng truy nhập hệ thống, đó là lớp **Giaodien** (Giao diện).



Hình 1. Khởi tạo hệ thống

Sau khi phân tích sơ bộ quan hệ của 6 lớp đã đề cập ở trên, ta có "biểu đồ lớp" như ở Hình 1 coi như là khởi tạo hệ thống, mô tả nó có thể viết như sau:

APP_0 = Giaodien[]; MatHang[]; DactaHang[]; BanHang[]; TTBanHang[]; ThanhToan[];

3.2.2. Mô hình khái niệm

Các lớp trong APP_0 chưa có bất kỳ một thuộc tính nào. Bây giờ ta phải thêm các thuộc tính cho các lớp. Trong khi phân tích tinh chế, ta nhận thấy rằng các lớp cần phải có các thuộc tính như sau:

+ Giaodien với hoạt động như là giao diện của hệ thống phải duy trì ít nhất 3 thuộc tính: **bh** để tham chiếu tới đối tượng BanHang hiện tại, **BH** như là danh sách (List) các đối tượng BanHang lưu tất cả BanHang đang giữ, và **mh** tham chiếu tới cơ sở dữ liệu MatHang.

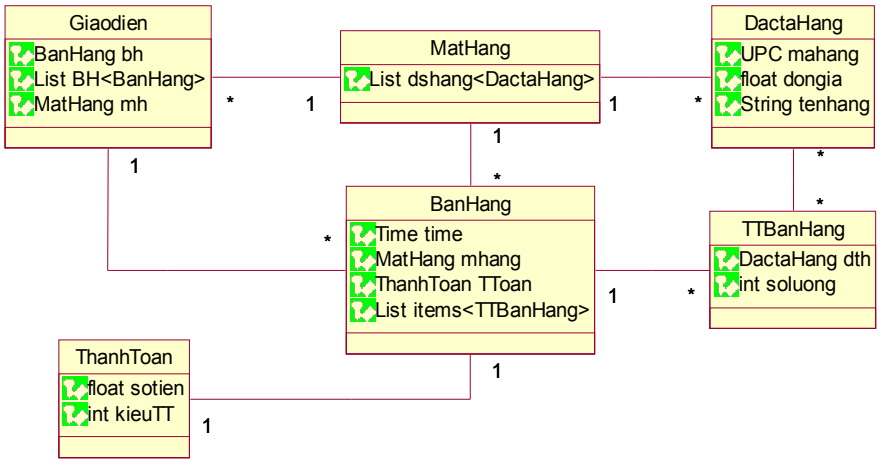
+ MatHang có **dshang** tham chiếu tới DactaHang của nó.

+ DactaHang phải có **tenhang**, thuộc tính **upc** có vai trò là mã của các sản phẩm hàng như là khóa trong cơ sở dữ liệu và thuộc tính đơn giá sản phẩm **dongia**.

+ BanHang phải có ít nhất 4 thuộc tính: thời gian bán **time**, **mhang** tham chiếu tới MatHang, **TToan** tham chiếu tới đối tượng ThanhToan và **items** tham chiếu tới danh sách TTBanHang.

+ TTBanHang phải có thuộc tính **dth** tham chiếu đúng tới DactaHang và số nguyên **soluong** biểu diễn số lượng, ghi chép bao nhiêu sản phẩm loại này đã được bán.

+ ThanhToan có thuộc tính sotien để lưu số tiền khách hàng đã thanh toán; và thuộc tính **kieuTT** biểu thị cách thanh toán, nếu kieuTT = 0 thì thanh toán bằng tiền mặt, nếu kieuTT = 1 thì thanh toán bằng thẻ tín dụng.



Hình 2. Mô hình khái niệm của hệ thống đã tinh chế

Trong mô hình quan hệ hướng đối tượng như trên, chúng ta cần phải bổ sung các thuộc tính và thay đổi thuộc tính private thành các thuộc tính protected theo các luật trong [3, 9]. Chẳng hạn:

Giaodien[ ]; cdecls $\sqsubseteq$ Giaodien [pri BanHang bh]; cdecls

và Giaodien [pri BanHang bh]; cdecls $\sqsubseteq$ Giaodien [pro BanHang bh]; cdecls

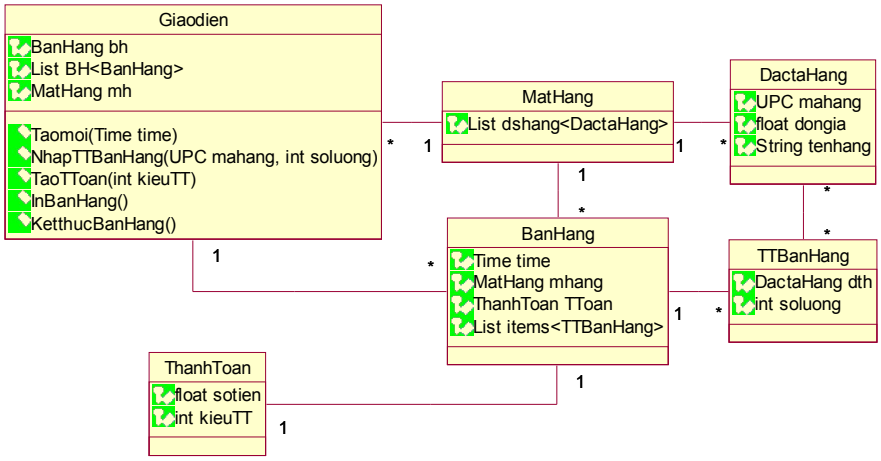
Áp dụng lặp lại 2 luật này để bổ sung cho tất cả các thuộc tính đã đề cập ở trên cho các lớp. Do đó, ta sẽ đạt được biểu đồ lớp mà trong đó các thuộc tính đã được bổ sung vào các lớp tương ứng, đó là mô hình khái niệm được biểu diễn trên Hình 2, trong đó tất cả các thuộc tính của các lớp đều là protected.

Chúng ta đã chỉ ra các lớp được miêu tả trên Hình 2 như hệ thống APP_1 là trình tự các khai báo lớp, điều đó chứng tỏ là $\text{APP}_0 \sqsubseteq \text{APP}_1$.

3.2.3. Bổ sung các phương thức giao diện

Bây giờ chúng ta sẽ tinh chế hệ thống bởi luật bổ sung phương thức [9, 12] cho các lớp để

có mô hình thiết kế, đó là mô hình khái niệm cộng thêm với tất cả các đặc tả phương thức. Chúng ta bắt đầu từ lớp điều khiển Giaodien là giao diện với mọi người sử dụng. Chúng ta nhận thấy Giaodien có ít nhất 5 phương thức: **Taomoi()** khởi đầu cho việc bán hàng là tạo mới đối tượng mh có kiểu là BanHang; **NhapTTBanHang()** để thêm thông tin bán hàng vào đối tượng BanHang; **TaoTToan()** để tính tổng giá tiền và tạo đối tượng ThanhToan; **InBanHang()** và **KetthucBanHang()** để in thông tin và kết thúc một phiên bán hàng. Hơn nữa, KetthucBanHang còn có thêm nhiệm vụ nữa là bổ sung tham chiếu của đối tượng mh tới danh sách BanHang. Sau đây là chi tiết đặc tả của các phương thức đã nêu trên:



Hình 3. Điều khiển Use Case

• Taomoi(Time time)

self.mh' $\neq$ null $\vdash$ self.bh'.time = time $\wedge$ self.bh'.mh = self.mh

• NhapTTBanHang (UPC mahang, int soluong)

self.mh $\neq$ null $\wedge$ self.bh $\neq$ null $\wedge$ soluong $\neq 0 \vdash$

$\exists$ item: TTBanHang $\bullet$ self.bh.items' = self.bh.item s $\cup$ {item} $\wedge$

item.mahang = mahang $\wedge$ item.soluong = soluong$\wedge$

($\exists m$: DactaMatHang $\bullet$ $m \in$ self.mh $\wedge$ item.m = $m \wedge m$.mahang=mahang)

• TaoTToan(int kieuTT)

self.bh $\neq$ null $\wedge$ kieuTT $\in \{0, 1\} \vdash \exists$TT: ThanhToan $\bullet$ self.bh.TT' = TT $\wedge$

$$\text{TT.sotien} = \sum_{\text{item} \in \text{items}} \text{item}.m.\text{dongia} \times \text{item.soluong} \wedge$$

((kieuTT = 0 $\wedge$ {Trả bằng tiền mặt}) $\vee$ (kieuTT=1 $\wedge$ {Trả bằng thẻ tín dụng}))

• InBanHang()

self.bh $\neq$ null $\wedge$ Thuchien(TaoTToan) $\vdash$ {In các thông tin, báo cáo bán hàng}

Ở đây hàm giả định Thuchien(TaoTToan) có biểu thị cho khách hàng thực hiện việc trả tiền.

• KetthucBanHang()

self.bh $\neq$ null Thuchien(TaoTToan) $\vdash$ self.bh' = null $\wedge$ self.BH' = self.BH $\cup$ {self.bh}

Áp dụng luật này với các lớp còn lại. Biểu đồ lớp này được mô tả trên Hình 3. Nó tương ứng với các khai báo lớp APP_2 và ta thấy rằng $\text{APP}_1 \sqsubseteq \text{APP}_2$.

3.2.4. Mô hình thiết kế

Ta sẽ phát triển tất cả các phương thức cho các lớp để hoàn thiện mô hình thiết kế, bao gồm các việc sau:

* Thứ nhất, ta sẽ đưa một số nhiệm vụ của Giaodien tới BanHang. Để hoàn thành việc này, trước tiên phải phát triển hai phương thức như mô tả sau đây cho lớp BanHang. Cũng như lý do ở phần trên, hệ thống mới được bổ sung các phương thức để tinh chế phiên bản trước đó.

• BanHang.TaoBanHang(UPC mahang, int soluong)

self.mh $\neq$ null $\wedge$ soluong $\neq 0 \vdash \exists$ item: TTBanHang • self.bh.item' = self.item $\cup\{item\}\wedge$ item.mahang = mahang $\wedge$ item.soluong = soluong $\wedge$ ($\exists m$: DactaMatHang•$m \in$ self.mh $\wedge$ item.$m = m \wedge m$.mahang = mahang)

• BanHang.TaoTToan(int kieuTT)

self.kieuTT $\in \{0, 1\} \vdash \exists$ TT: ThanhToan • self.bh.TT' = TT $\wedge$ TT.sotien = $\sum_{\text{item}\in\text{items}}$ item.m.dongia $*$ item.soluong $\wedge$ ((kieuTT = 0 $\wedge$ {Trả bằng tiền mặt}) $\vee$ (kieuTT=1 $\wedge$ {Trả bằng thẻ tín dụng}))

* Thứ hai, có thể thực hiện hoặc tinh chế trong mô hình này, các phương thức Giaodien.NhapTTBanHang() và Giaodien.TaoTToan() bằng các phương thức được phát triển như sau:

• Giaodien.NhapTTBanHang'(UPC mahang, int soluong) = self.bh.TaoBanHang(mahang, dongia)

• Giaodien.TaoTToan'(int kieuTT) = self.bh.TaoTToan(kieuTT)

Sau khi đã thêm các phương thức BanHang.TaoBanHang() và BanHang.TaoTToan() vào hệ thống, ta sẽ thay thế các phương thức đó lần lượt bằng Giaodien.NhapTTBanHang'() và Giaodien.TaoTToan'(). Ta chỉ ra hệ thống mới như là APP_3.

Theo ngữ nghĩa của mô hình ta có thể xác định rằng:

Giaodien.NhapTTBanHang() $\sqsubseteq$ Giaodien.NhapTTBanHang'()

Giaodien.TaoTToan() $\sqsubseteq$ Giaodien.TaoTToan'()

Do đó, áp dụng luật tinh chế phương thức [9] ta có $APP_2 \sqsubseteq APP_3$.

Tiếp theo, ta sẽ tiếp tục đưa các nhiệm vụ của BanHang tới MatHang và ThanhToan. Tương tự như quá trình trên, ta phát triển phương thức mới MatHang.Timkiem() và đưa nó vào phương thức BanHang.TaoBanHang(). Đặc tả của phương thức mới như sau:

• MatHang.Timkiem(UPC upc, DactaHang dt)

self.dshang $\neq$ null $\vdash$ (dt' = null) $\triangleleft(\exists dt \in$ self.dshang $\wedge$ dt.upc = upc)$\triangleright$(dt'=dt)

Phương thức này sẽ tìm kiếm mặt hàng hợp lệ từ MatHang và trả lại mặt hàng đó nếu việc tìm kiếm thành công. Nhờ sự hỗ trợ phương thức này, ta có thể cài đặt phương thức BanHang.TaoBanHang() như sau:

BanHang.TaoBanHang'(UPC upc, int soluong) = {
var MatHang mh;
Timkiem(upc, mh);
skip $\triangleleft(mh =$ null)$\triangleright$ {var TTBanHang ttbh; DactaHang(new(ttbh, [mh, soluong])}
self.items.Add(ttbh); end mh}

Tương tự, để tiến hành các nhiệm vụ của lớp BanHang tới lớp ThanhToan, ta phát triển phương thức mới ThanhToan.Tratien() như sau:

ThanhToan.Tratien() = {{Trả bằng tiền mặt} ◁(self.type= 0)▷ {Trả bằng thẻ tín dụng }}

Nhờ sự hỗ trợ phương thức trên, chúng ta cài đặt BanHang.TaoTToan(int kieuTT) như sau:

BanHang.TaoBanHang'(int kieuTT) = {

skip ◁ (kieuTT=0) ▷ {

var float $st = 0$;

for each (TTBanHang item ∈ self.items) st:=st+item.dongia∗item.soluong;

ThanhToan.new(self.ThanhToan, [st, kieuTT]);

self.ThanhToan.Tratien();}}

Trong mô hình quan hệ này, ta có thể thấy rằng:

BanHang.TaoBanHang() ⊑ BanHang.TaoBanHang'()

BanHang.TaoTToan() ⊑ BanHang.TaoTToan'()

Tương tự như quá trình trên, ta được hệ thống mới là APP_4 và có $APP_3 \sqsubseteq APP_4$.

3.2.5. Tinh chế phương thức

Sau quá trình phân tích xử lý trên, ta đã có mô hình thiết kế. Trong thực tế ta có thể cài đặt hệ thống này bằng bất kỳ ngôn ngữ lập trình hướng đối tượng nào. Nhưng ta vẫn có thể khắc phục được một số khiếm khuyết trong hệ thống như [6, 9] đã chỉ ra. Trong phần còn lại ta sẽ phân tích mô hình để nâng cao tính linh động và ổn định hệ thống.

Sau khi xem xét lại thiết kế, ta có thể tìm phần mã trình tiêu biểu để tinh chế lại. Đó là phương thức BanHang.TaoTToan() sử dụng các thuộc tính của TTBanHang nhiều lần. Điều đó có thể tốt hơn nếu sự tính toán xảy ra trong TTBanHang tự bản thân nó biến đổi phù hợp lại, hoặc tương tác giữa các lớp, cho nên ta sẽ trích ra phương thức trong lớp BanHang rồi chuyển tới lớp TTBanHang.

Chúng tôi đưa ra sự phân tích lại như sau:

+ Thứ nhất, nhờ sự hỗ trợ luật Extract Method ([9, 12]) ta có:

BanHang[TaoTToan()] ⊑ BanHang[TaoTToan()[Tong()\(item.dongia∗item.soluong)]]

Ở đây, Tong() = {return item.dongia ∗ item.soluong} và $[a\backslash b]$ có nghĩa là thay thế b bằng a.

+ Thứ hai, ta sẽ phân tích lại vế bên phải. Áp dụng luật Move Method ([9, 11]) ta có:

BanHang[TaoTToan()]; TTBanHang[] ⊑

BanHang[TaoTToan()[item.Tong()\Tong()]]; TTBanHang[Tong()]

Ở đây, trong lớp TTBanHang, Tong() = {return dongia∗soluong}.

Do đó nhận được các khai báo lớp mới như là APP_5 và có $APP_4 \sqsubseteq APP_5$.

3.2.6. Tinh chế lớp

Tiếp theo, xem lại lớp Giaodien. Nó có thuộc tính BanHang là danh sách bản ghi thông tin của tất cả các mặt hàng đã bán. Với mỗi cái đó, nó không phù hợp để ở lớp giao diện chính khi danh sách lớn lên. Mặt khác trường hợp đó có thể cho phép lớp Giaodien thực hiện xử lý song song. Chúng phải chia sẻ cùng một danh sách (ở đây ta có thể xem danh sách như là cơ sở dữ liệu với tất cả các bản ghi). Vì thế cần thiết phải có một lớp khác liên kết với danh sách. Chúng tôi trích thêm lớp mới LuuDS (lưu danh sách) để thay thế cho công việc trước đây. Theo luật Extract Class ([9, 12]):

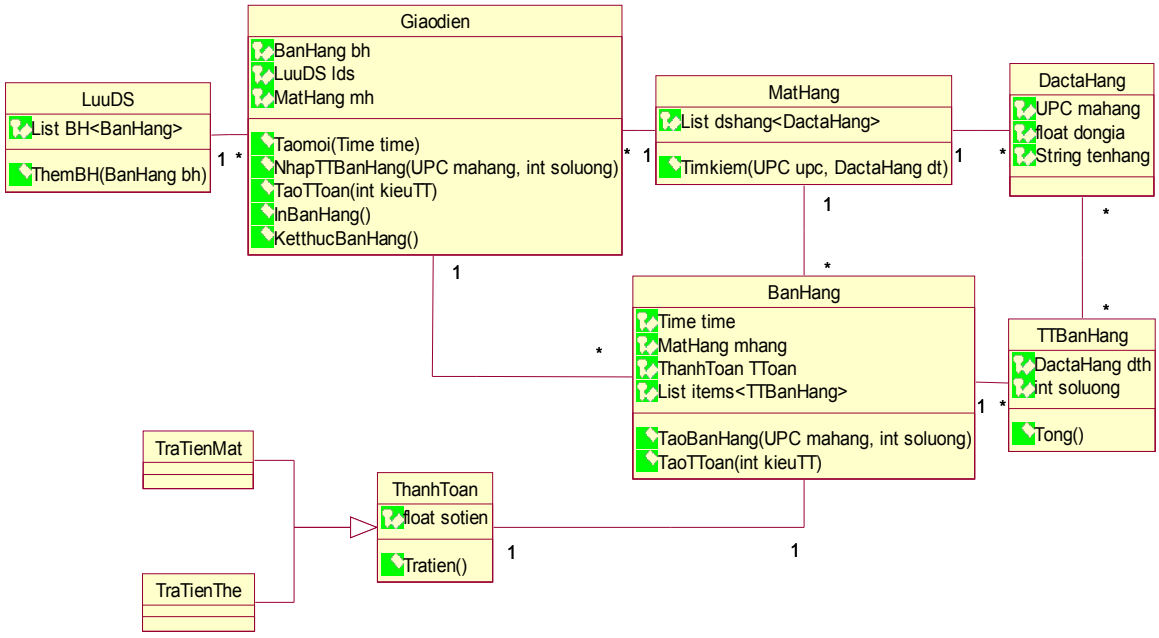
Giaodien[List BH⟨ BanHang⟩] ⊑ Giaodien[LuuDS lds]; LuuDS[List BH⟨ BanHang⟩]

Tương tự như trên, có thể trích ra phương thức ThemBH(BanHang bh) trong lớp Giaodien. Phương thức này được thêm vào đối tượng bh hiện tại tới danh sách bán hàng lds.BH. Sau đó chuyển nó vào lớp mới được phát triển LuuDS. Khai báo lớp tương ứng là APP_6 và có $\text{APP}_5 \sqsubseteq \text{APP}_6$.

3.2.7. Tinh chế theo mẫu định hướng

Bây giờ ta sẽ vào pha cuối cùng của việc tinh chế. Đó là tinh chế theo mẫu thiết kế định hướng Strategy [8, 12] cho hệ thống đang tồn tại.

Trong phương thức ThanhToan.Tratien() của hệ thống có đoạn mã $c_1 \triangleleft$ self.kieuTT = 0 $\triangleright c_2$, trong đó giá trị của self.kieuTT sẽ có tác dụng tới hành vi của phương thức. Bây giờ với sự định hướng bởi mẫu Strategy, ta sẽ phân tích lại nó bằng cách thay thế kiểu mã tính đa hình (cơ cấu nhiều trạng thái).



Hình 4. Mô hình thiết kế bởi các luật tinh chế

Nhờ sự hỗ trợ của luật Strategy ta có:

BanHang[TaoTToan(int kieuTT)]; ThanhToan[int kieuTT, Tratien()]

$\sqsubseteq$ BanHang[TaoTToan(int kieuTT)]; ThanhToan[Tratien()];
TraTienMat[ThanhToan, Tratien()]; TratienThe[ThanhToan, Tratien()]

Ở đây:

• Phương thức TaoTToan(int kieuTT) ở bên phải của $\sqsubseteq$ là khác với nó ở bên trái. Ta huỷ lệnh:

ThanhToan.new(self.ThanhToan, [sotien, kieuTT]);

từ phương thức cũ và thay thế nó bởi lệnh khác:

TraTienMat.new(self.ThanhToan, [sotien]) $\triangleleft$(kieuTT=0)$\triangleright$
TraTienThe.new(self.ThanhToan, [sotien]);

• Thân của phương thức ThanhToan.Tratien() là trống. Nó được cài đặt ở các lớp con:

TraTienMat.Tratien() {Trả bằng tiền mặt}
TraTienThe.Tratien() {Trả bằng thẻ tín dụng}

Như vậy, kiểu mã trình đã được thay thế bởi tính đa hình được đưa ra ở hai lớp con. Ta chỉ ra các khai báo lớp mới là APP_7 và dĩ nhiên là $\text{APP}_6 \sqsubseteq \text{APP}_7$. Toàn bộ quá trình tinh chế trên được biểu diễn trên biểu đồ lớp UML được mô tả trên Hình 4.

4. KẾT LUẬN

Bài báo đã trình bày quy trình tinh chế mô hình đối tượng thô ban đầu của bài toán, nhờ áp dụng các luật tinh chế bổ sung các thuộc tính và chuyển các thuộc tính sang kiểu protected, thu được mô hình khái niệm và biểu đồ use case của hệ thống. Tiếp tục áp dụng các luật bổ sung phương thức để thu được mô hình thiết kế. Sau đó áp dụng các luật tinh chế phương thức và luật tinh chế lớp để cải tiến mô hình thiết kế của hệ thống được phù hợp hơn. Với những lớp có khả năng đa hình thì áp dụng luật tinh chế theo mẫu định hướng, cụ thể ở đây là luật Strategy để thực hiện phương thức đa hình cho lớp. Với sự tinh chế từng bước trên sẽ thu được mô hình hệ thống sát thực hơn.

Qua quá trình tinh chế sẽ thu được mô hình thiết kế cuối cùng tương đối gần với mã có thể thực hiện được. Điều đó làm dễ dàng cho việc cài đặt hệ thống bằng bất kỳ ngôn ngữ lập trình hướng đối tượng, chẳng hạn như C#.Net hay Java. Việc phân tích, thiết kế và tinh chế như vậy sẽ đảm bảo tính ổn định, tính sử dụng lại, thuận lợi cho quá trình bảo trì phát triển hệ thống, góp phần nâng cao chất lượng hệ thống phần mềm.

Quá trình tinh chế theo cách thức chỉ ra ở trên đã được chúng tôi thực hiện trên công cụ Rational Rose, đảm bảo tính lôgic, tính đúng đắn và hệ thống ngày càng tối ưu hơn. Với khả năng tự động sinh mã trình, phần mềm hỗ trợ mô hình hóa Rational Rose, hoàn toàn có thể chuyển mô hình thiết kế sang khung chương trình hướng đối tượng trên ngôn ngữ lập trình mà nó hỗ trợ. Phương pháp này đã được chúng tôi áp dụng thử nghiệm hiệu quả trong việc phát triển "hệ thống bán hàng trong một siêu thị".

TÀI LIỆU THAM KHẢO

[1] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[2] C. A. R. Hoare and He Jifeng, *Unifying Theories of Programming*, Prentice Hall, 1998.

[3] He Jifeng, Li Xiaohan, and Zhiming Liu, *rCOS: Refinement Calculus for Object Systems*, Technical Report UNU/IIST No.322, UNU/IIST: International Institute for Software Technology, the United Nations University, Macau, May 2005.

[4] Ivar Jacopson, Gray Booch, and James Rumbaugh, *The Unified Software Development Process*, Addision-Wesley, 2000.

[5] J. He, Z. Liu, X. Li, and S. Qin, A relational model for object-oriented designs, *Pro APLA'2004 LNCS 3302*, Taiwan, 2004, Springer.

[6] Joshua Kerievsky, *Refactoring to Patterns*, Addison-Wesley, 2004.

[7] P. Kuchten, *The Rational Unified Process - An Introduction*, Addison-Wesley, 2000.

[8] Larman, *Applying UML and Patterns*, Prentice-Hall International, 2001.

[9] Martin Fowler, *Refactoring, Improving the Design of Existing Code*, Addison-Wesley, 2000.

[10] Nguyễn Mạnh Đức, Nguyễn Văn Vy, Đặng Văn Đức, Mô hình đại số quan hệ của hệ thống hướng đối tượng, *Tạp chí Tin học và Điều khiển học* **21** (3) (2005).

[11] R. J. R. Back, L. Petre, and I. P. Paltor, Formalizing UML use cases in the refinement calculus, *Proc. UML'99* Springer-Verlag, 1999.

[12] Quan Long, He Jifeng, Zhiming Liu, *Refactoring and pattern-directed refactoring: Aformal perspective*, Technical report UNU/IIST No.318, UNU/IIST: International Institute for Software Technology, the United Nations University, Macau, January 2005.

[13] P. Andre, A. Romanczuk, J. C. Royer, and A. Vasconcelos, Checking the consistency of UML class diagrams using Larch Prover, *Proc. ROOM'2000,* York, UK, 2000.

Nhận bài ngày 29 - 9 - 2005