ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Đào Thị Hường

MỘT SỐ PHƯƠNG PHÁP KIỂM CHỨNG CÁC HỆ THỐNG HƯỚNG ĐỐI TƯỢNG

TÓM TẮT LUẬN ÁN TIẾN SỸ CÔNG NGHỆ THÔNG TIN

Hà Nội - 2017

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Đào Thị Hường

MỘT SỐ PHƯƠNG PHÁP KIỂM CHỨNG CÁC HỆ THỐNG HƯỚNG ĐỐI TƯỢNG

Chuyên ngành: Kỹ thuật Phần mềm
Mã số: 62.48.01.03

TÓM TẮT LUẬN ÁN TIẾN SỸ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:
PGS.TS Trương Ninh Thuận

Hà Nội - 2017

Mục lục

Chương 1

Giới thiệu

1.1 Đặt vấn đề

Tái cấu trúc (*Refactoring*) "*là tiến trình thay đổi cấu trúc bên trong (internal structure) của hệ thống phần mềm mà không làm ảnh hưởng đến hành vi bên ngoài (external behavior) của nó.* Thực hiện tái cấu trúc trên các hệ thống phần mềm cho phép người phát triển có một kiến trúc chắc chắn, từ đó dễ dàng bảo trì hệ thống. Tuy nhiên, tiến trình này có thể tiêu tốn nhiều thời gian và dễ phát sinh lỗi nếu người phát triển không tuân thủ một cách nghiêm ngặt các quy trình và tiêu chuẩn phát triển phần mềm. Bài toán *kiểm chứng tính nhất quán giữa hệ thống phần mềm ban đầu và phiên bản tái cấu trúc của nó* là điều kiện cần, giúp đánh giá độ tin cậy của tiến trình tái cấu trúc và là cơ sở cho các hoạt động cải thiện chất lượng phần mềm sau này.

Các công trình nghiên cứu về kiểm chứng tính nhất quán của hệ thống phần mềm trong tiến trình tái cấu trúc rất đa dạng với nhiều cách tiếp cận khác nhau. Trong phạm vi nghiên cứu, luận án chỉ tập trung đến hai vấn đề chính đang thu hút được nhiều sự quan tâm là: (1) đặc tả hệ thống phần mềm cùng với các ràng của nó bằng phương pháp hình thức; (2) kiểm chứng sự bảo toàn về bất biến, hành vi và công cụ hỗ trợ kiểm chứng trong cấu trúc hệ thống phần mềm.

Một số nghiên cứu chuyên sâu khảo sát về lĩnh vực tái cấu trúc hệ thống phần mềm, đặc biệt là các nghiên cứu thực hiện kiểm chứng tính nhất quán của các hệ thống phần mềm trong tiến trình này. Có hai hướng tiếp cận nhận được sự quan tâm của nhiều nhà nghiên cứu: (1) hướng tiếp cận sử dụng các khẳng định (assertion) để biểu diễn hành vi của hệ thống (bao gồm bất biến, tiền/hậu điều kiện); (2) hướng thứ hai là sử dụng biến đổi mô hình (model transformation). Cụ thể, các nghiên cứu về bài toán kiểm chứng tính nhất quán trong tái cấu trúc được chia thành ba vấn đề chính sau:

- Kiểm tra sự bảo toàn về bất biến của hệ thống trong tiến trình tái cấu trúc (Invariants Preservation). Phương pháp đề xuất của họ dựa trên kỹ thuật biến đổi đồ thị với việc biểu diễn mô hình phần mềm bằng UML và ngôn ngữ OCL để biểu diễn các bất biến. Chưa có nghiên cứu nào thực sự đi sâu vào kiểm tra sự bảo toàn về bất biến trên biểu đồ lớp.
- Kiểm tra sự bảo toàn về hành vi của hệ thống trong tiến trình tái cấu trúc hệ thống phần mềm (Behaviors Preservation). Các nghiên cứu chuyên sâu về kiểm tra sự bảo toàn hành vi của hệ thống phần mềm (trong các giai đoạn thiết kế và cài đặt) thường biểu diễn hành vi của hệ thống bằng các khẳng định (assertions). Các nghiên cứu này kiểm chứng tính nhất quán chủ yếu ở giai đoạn mã nguồn, hoặc kiểm tra sự nhất quán giữa các giai khác nhau của tiến trình phát triển phần mềm (giai đoạn thiết kế với giai đoạn cài đặt mã nguồn).
- Công cụ hỗ trợ kiểm tra tính nhất quán của hệ thống trong tiến trình tái cấu trúc (Tool supports for Checking Consistency). Các công cụ hỗ trợ cho tiến trình tái cấu trúc hệ thống phần mềm được chia thành hai loại chính: (i) tìm kiếm vị trí tái cấu trúc và (ii) thực hiện tái cấu trúc. Tuy nhiên, hầu hết các công cụ này đều rơi vào loại thứ nhất và chưa có sự xem xét đến vấn đề bảo toàn hành vi của hệ thống sau tiến trình tái cấu trúc.

Các công trình nghiên cứu về kiểm chứng tính nhất quán của hệ thống phần mềm trong tiến trình tái cấu trúc có tính chất đa đạng cả về phương pháp và kỹ thuật được sử dụng. Giải quyết bài toán này là tìm lời giải đáp cho câu hỏi *"Hệ thống trước và sau khi tái cấu trúc có nhất quán với nhau hay không?"*. Bài toán này có dữ liệu đầu vào bao gồm: (i) hệ thống ban đầu (initial system) cùng với phiên bản tái cấu trúc của nó (evolution system) và (ii) *các ràng buộc về bất biến của lớp và hành vi của hệ thống*. Dữ liệu đầu ra là các kết quả thông báo mức độ nhất quán giữa các hệ thống trước và sau khi tái cấu trúc.

1.2 Các kết quả chính của luận án

Các kết quả nghiên cứu của luận án góp phần bổ sung và hoàn thiện các phương pháp kiểm chứng tính nhất quán trong tái cấu trúc hệ thống phần mềm. Cụ thể, luận án có bốn đóng góp chính sau:

- *Đề xuất phương pháp kiểm chứng tính bất biến* trong tái cấu trúc mô hình phần mềm.
- *Đề xuất phương pháp kiểm chứng sự bảo toàn hành vi* trong tái cấu trúc hệ thống phần mềm.
- *Xây dựng công cụ* CVT hỗ trợ kiểm chứng sự bảo toàn hành vi trong tái cấu trúc mô hình phần mềm.
- *Minh họa các phương pháp* đề xuất trên ví dụ về Hệ thống điều khiển giao thông đường bộ ARTC.

Về góc nhìn lý thuyết, luận án đề xuất các phương pháp kiểm chứng tính nhất quán bao phủ trên các ràng buộc về *bất biến của lớp* và *hành vi của hệ thống*, xuyên suốt từ giai đoạn thiết kế đến giai đoạn cài đặt trong chu trình phát triển phần mềm. *Về góc nhìn thực hành*, luận án cũng xây dựng được công cụ CVT hỗ trợ kiểm chứng tính nhất quán trong tái cấu trúc mô hình phần mềm. Như vậy, các kết quả nghiên cứu của luận án thể hiện được tính thống nhất về mặt lý thuyết, đồng thời khả thi về mặt thực hành. Mục tiêu cuối cùng của luận án là xây dựng giải pháp tương đối hoàn chỉnh cho bài toán kiểm chứng tính nhất quán trong tái cấu trúc hệ thống phần mềm.

1.3 Bố cục của luận án

Luận án *"Một số phương pháp kiểm chứng các hệ thống hướng đối tượng"* bao gồm sáu chương. Trong đó, Chương 1 trình bày bài toán mà luận án sẽ nghiên cứu, Chương 2 trình bày một cách tóm tắt các hướng nghiên cứu chính của bài toán kiểm chứng tính nhất quán trong tái cấu trúc. Chương 3 đề xuất phương pháp kiểm chứng các ràng buộc về bất biến trong tái cấu trúc biểu đồ lớp. Chương 4 đề xuất phương pháp kiểm chứng sự bảo toàn về hành vi trong tái cấu trúc hệ thống phần mềm. Công cụ hỗ trợ kiểm chứng được trình bày ở Chương 5. Cuối cùng, Chương 6 là kết luận và các hướng nghiên cứu tiếp theo của luận án.

Chương 2

KIẾN THỨC CƠ SỞ

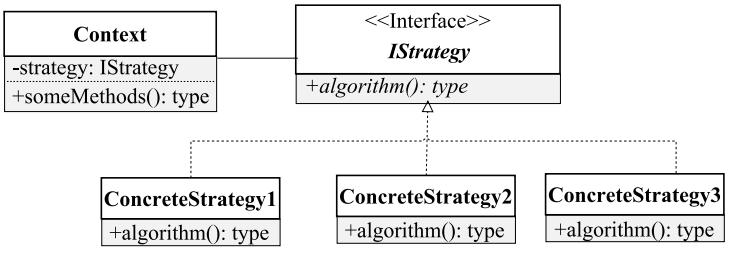
2.1 Tái cấu trúc

Tái cấu trúc (refactoring) "*là tiến trình thay đổi hệ thống phần mềm theo cách không làm thay đổi hành vi bên ngoài (external behavior) của hệ thống nhưng lại cải thiện được cấu trúc bên trong (internal structure) của nó*". Ý tưởng chính của tái cấu trúc là phân bố lại các lớp (classes), các thuộc tính (attributes) và các phương thức (methods) thông qua mối quan hệ phân cấp giữa các thành phần của hệ thống.

2.2 Mẫu thiết kế

"Mỗi mẫu thiết kế mô tả một vấn đề xảy ra nhiều hơn và nhiều hơn nữa trong môi trường của chúng ta và sau đó nó mô tả các giải pháp chính để xử lý vấn đề đó, theo cách mà bạn có thể sử dụng lại giải pháp này cả triệu lần". Các mẫu thiết kế cung cấp một khuôn mẫu chung để giải quyết nó các vấn đề có tính chất thường xuyên (lặp đi lặp lại) trong quá trình sử dụng phần mềm. Nói cách khác, mẫu thiết kế cung cấp giải pháp cho các vấn đề thiết kế phổ biến thường gặp trong lập trình, cụ thể là các vấn đề về kế thừa và tương tác nói chung. Mẫu thiết kế không sử dụng để xử lý các vấn đề về kiến trúc tổng thể trong quy mô lớn của toàn bộ mô hình phần mềm.

Mẫu Strategy *"Định nghĩa một tập hợp các thuật toán, đóng gói chúng thành từng loại và có thể hoán đổi cho nhau"*. Mẫu Strategy thực thi các thuật toán một cách linh hoạt, tùy thuộc vào ngữ cảnh cụ thể. Ý nghĩa thực sự của Strategy là *tách rời phần xử lý* một chức năng ra khỏi đối tượng. Sau đó tạo ra một *tập hợp các thuật toán* để xử lý chức năng đó và cho phép lựa chọn thuật toán đúng đắn nhất khi thực thi chương trình. Mẫu Strategy thường được sử dụng để thay thế cho sự kế thừa, khi muốn chấm dứt việc theo dõi và chỉnh sửa một chức năng qua nhiều lớp con.



Hình 2.1: Mẫu thiết kế Strategy.

Mẫu Strategy bao gồm các thành phần sau:

- *IStrategy*: Giao diện (*Interface*) được chia sẻ giữa các lớp con cụ thể trong họ các thuật toán. Lớp *Context* sử dụng giao diện để gọi đến các thuật toán đã được định nghĩa trong các lớp con;
- *ConcreteStrategy*: Nơi các thuật toán được cài đặt cụ thể;

– *Context*: Lớp dùng để tham chiếu đến kiểu *ISstrategy*. Trong một số trường hợp, *Context* có thể cài đặt các phương thức, bởi vậy các lớp *ConcreteStrategy* có thể truy cập đến các dữ liệu này.

Sự hợp tác giữa các thành phần trong mẫu Strategy:

– *IStrategy* và *Context* tương tác để thực hiện việc lựa chọn các thuật toán. Một *Context* có thể truyền tất cả các dữ liệu cần thiết bằng các thuật toán đến *IStrategy*. Ngoài ra, *Context* có thể truyền chính nó như là một đối số cho các hoạt động của *IStrategy*. Điều đó cho phép *IStrategy* gọi lại *Context* khi cần thiết;
– Một *Context* chuyển tiếp yêu cầu từ máy khách (client) đến các *IStrategy* của nó. Máy khách thường tạo và truyền đối tượng *ConcreteStrategy* đến *Context*; sau đó, nó tương tác riêng với *Context*. Thường có một gia đình các lớp *ConcreteStrategy* để máy khách lựa chọn.

Ưu điểm của việc sử dụng mẫu *Strategy* là đóng gói các thuật toán trong các lớp riêng biệt, điều này sẽ làm cho việc sử dụng lại mã được thuận tiện hơn nhiều và do đó, hành vi của lớp *Context* có thể thay đổi một cách linh hoạt tại thời điểm chạy chương trình.

Chương 3

Kiểm chứng tính bất biến trong tái cấu trúc mô hình

3.1 Giới thiệu

Tái cấu trúc thực hiện trên biểu đồ lớp của UML đương nhiên sẽ có ảnh hưởng trực tiếp lên cấu trúc và các ràng buộc bất biến của nó. Các nhà phát triển phần mềm cần phải kiểm soát và tác động để có được sự thay đổi như mong muốn . Chương này, luận án đề xuất phương pháp kiểm chứng tính nhất quán trong tái cấu trúc biểu đồ lớp theo cách bảo toàn các bất biến của nó.

Bất biến của lớp (*class invariants*) được định nghĩa như các khẳng định (assertions), biểu diễn cho các điều kiện ràng buộc trên các thuộc tính của lớp. Các điều kiện này phải được thỏa mãn bởi bất kỳ thể hiện nào của lớp.

Ý tưởng chính của phương pháp kiểm chứng tính bất biến trong tái cấu trúc biểu đồ lớp là (1) đặc tả một cách hình thức biểu đồ lớp cùng với bất biến của nó, (2) định nghĩa mẫu (template) để mô tả các phép toán tái cấu trúc và (3) xây dựng các luật áp dụng đối với từng phép toán tái sao cho phiên bản tái cấu trúc vẫn thỏa các ràng buộc bất biến trên mô hình ban đầu. Đóng góp chính của chương này là đề xuất mẫu mô tả phép toán và tích hợp các ràng buộc về bất biến trong mô hình phần mềm.

3.2 Phương pháp bảo toàn tính bất biến trong tái cấu trúc biểu đồ lớp của UML

3.2.1 Mô hình hóa biểu đồ lớp trong UML

Định nghĩa 3.1 (Biểu đồ lớp trong UML)**.** *Một biểu đồ lớp được biểu diễn bởi một bộ - 2 CD* $= \langle \Sigma_C, \Sigma_A \rangle$*, trong đó* Σ_C *là tập hợp các lớp và* Σ_A *là tập hợp các liên kết giữa các lớp.*

Định nghĩa 3.2 (Lớp)**.** *Một lớp* $C \in \Sigma_C$ *được biểu diễn bởi bộ - 3* $C = \langle N_C, M_C, A_C \rangle$*, trong đó* N_C *là tên của lớp,* M_C *là tập hợp các phương thức và* A_C *là tập hợp các thuộc tính.*

Định nghĩa 3.3 (Phương thức)**.** *Một phương thức* $m^i_C \in M_C$ *được biểu diễn bởi bộ - 3* $m^i_C = \left\langle N^i_{M_C}, P^i_{M_C}, R^i_{M_C} \right\rangle$*, trong đó* $N^i_{M_C}$ *là tên của phương thức,* $P^i_{M_C}$ *là danh sách tham số và* $R^i_{M_C}$ *là kiểu của phương thức.*

Định nghĩa 3.4 (Thuộc tính)**.** *Một thuộc tính* $a^i_C \in A_C$ *được biểu diễn bởi bộ - 3* $a^i_C = \left\langle N^i_{A_C}, T^i_{A_C}, P^i_{A_C} \right\rangle$*, trong đó* $N^i_{A_C}$ *là tên của phương thức,* $T^i_{A_i}$ *là kiểu của thuộc tính và* $P^i_{A_C}$ *là biểu thức logic vị từ biểu diễn cho các ràng buộc trên thuộc tính này.*

Định nghĩa 3.5 (Bất biến của lớp)**.** *Bất biến của lớp* C*, ký hiệu là* INV_C*, được xác định bởi sự kết hợp của các biểu thức logic vị từ biểu diễn cho ràng buộc trên các thuộc tính của lớp.*

Giả sử, $P^i_{A_C}$ biểu thức logic vị từ biểu diễn ràng buộc trên thuộc tính $a^i_C \in A_C$ (trong trường hợp thuộc tính a^i_C không có ràng buộc, giá trị của biểu thức $P^i_{A_C}$ sẽ được gán là đúng (true)) và ký hiệu $\mid A_C \mid = n$ (lực lượng của tập hợp A_C). Khi đó, bất biến của lớp C được mô tả bởi công thức sau đây:

$$INV_C = \bigwedge_{i=1}^{n} P^i_{A_C}$$

Định nghĩa 3.6 (Liên kết)**.** *Liên kết $as \in \Sigma_A$ được biểu diễn bởi bộ - 3 $as = \langle N_{as}, E_{as_1}, E_{as_2} \rangle$, trong đó N_{as} là tên của liên kết, E_{as_1}, E_{as_2} là mấu liên kết 1 và mấu liên kết 2.*

Định nghĩa 3.7 (Sự tương đương về ngữ nghĩa của hai phương thức)**.** *Cho cặp phương thức m^i_C và m^j_D, trong đó $m^i_C \in M_C$ và $m^j_D \in M_D$. Khi đó, m^i_C và m^j_D được gọi là tương đương về mặt ngữ nghĩa và biểu thị bởi biểu thức $M^i_C \cong M^j_D$ nếu và chỉ nếu:*

$$\begin{cases} N^i_{M_C} \equiv N^j_{M_D} & \text{// trùng tên} \\ P^i_{M_C} \equiv P^j_{M_D} & \text{//trùng danh sách tham số} \\ R^i_{M_C} \equiv R^j_{M_D} & \text{//trùng kiểu trả về} \end{cases}$$

Khi đó $M_C \cup M_D$ được phân chia thành các lớp tương đương và biểu diễn là $\mathcal{M}_{C,D} = \{m^{ij}_{C,D}\}$, ở đây:

$$m^i_C, m^j_D \in m^{ij}_{C,D} \Leftrightarrow \begin{cases} m^i_C \in M_C \\ m^j_D \in M_D \\ M^i_C \cong M^j_D \end{cases}$$

Chúng tôi ký hiệu phương thức m^i_C trong tập M_C và không thuộc bất kỳ tập $m^{ij}_{C,D}$ nào bằng biểu thức $M_C \setminus \mathcal{M}_{C,D}$.

Định nghĩa 3.8 (Cặp thuộc tính có thể kết hợp)**.** *Cho cặp thuộc tính a^i_C và a^j_D, trong đó $a^i_C \in A_C$ và $a^j_D \in A_D$. a^i_C và a^j_D được gọi là có thể kết hợp và biểu thị bằng biểu thức $A^i_C \cong A^j_D$ nếu và chỉ nếu:*

$$\begin{cases} N^i_{A_D} \equiv N^j_{A_D} & \text{//trùng tên} \\ T^i_{A_D} \equiv T^j_{A_D} & \text{//trùng kiểu thuộc tính} \end{cases}$$

Ở đây $P^i_{A_C}$ và $P^j_{A_D}$ có thể tương đương hoặc không tương đương.

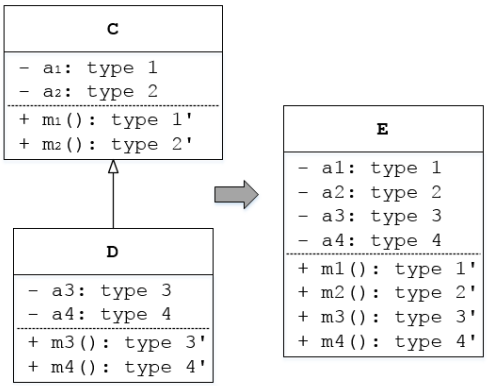
Khi đó $A_C \cup A_D$ được phân chia thành các lớp có thể kết hợp và biểu thị bởi $\mathcal{A}_{C,D} = \{a^{ij}_{C,D}\}$, trong đó:

$$a^i_C, a^j_D \in a^{ij}_{C,D} \Leftrightarrow \begin{cases} a^i_C \in A_C \\ a^j_D \in A_D \\ A^i_C \cong A^j_D \end{cases}$$

Chúng tôi ký hiệu a^i_C trong tập hợp A_C nhưng không thuộc về bất kỳ tập hợp $A^{ij}_{C,D}$ nào bởi biểu thức $A_C \setminus \mathcal{A}_{C,D}$.

Bảng 3.1: Mẫu của phép toán tái cấu trúc

Định danh	Diễn giải
Tên	Mô tả tên của phép toán tái cấu trúc
Tình huống áp dụng	Mô tả tình huống phổ biến nên áp dụng phép toán tái cấu trúc này
Mô hình khởi đầu	Mô tả thông tin về mô hình khởi đầu
Mô hình tiến hóa	Mô tả thông tin về mô hình sau khi áp dụng phép toán tái cấu trúc
Biểu diễn bằng UML	Biểu diễn phép toán bằng các ký pháp trong UML



Hình 3.1: Phép toán tái cấu trúc *Folding*.

3.2.2 Xây dựng tập luật áp dụng trong tái cấu trúc biểu đồ lớp của mô hình UML

Định nghĩa 3.9 (Tiến trình tái cấu trúc)**.** *Một tiến trình tái cấu trúc R được biểu diễn như sau:*

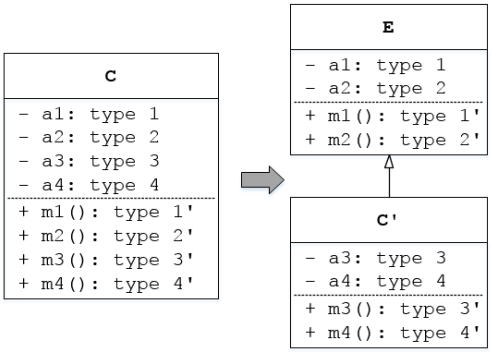
$$R : \mathcal{M} \xmapsto{OP_{Name}} \mathcal{M}'$$

trong đó $\mathcal{M}$ và $\mathcal{M}'$ tương ứng là các mô hình khởi đầu và mô hình tiến hóa, OP_{Name} là định danh của phép toán tái cấu trúc được áp dụng.

Để tạo điều kiên cho quá trình thực thi các phép toán tái cấu trúc, luận án định nghĩa mẫu của mỗi phép toán như trong Bảng 3.1. Sau đây, luận án trình bày lần lượt từng luật áp dụng đối với mỗi phép toán *Folding, Abstraction, Composition, Factoring* và *Unfolding* trong quá trình tái cấu trúc trên biểu đồ lớp.

3.2.2.1 Phép toán: Folding

- Tên gọi: *Folding*
- Tình huống áp dụng:
  - Thành phần: 2 lớp C và D có mối quan hệ kế thừa trực tiếp;
  - Mục đích: giảm mức độ phân cấp trong mô hình;
  - Thực thi: kết hợp 2 lớp C và D để thành lớp mới E, lớp này chứa đựng hành vi và thuộc tính của cả hai lớp ban đầu.
- Mô hình khởi đầu $\mathcal{M}$ bao gồm hai lớp C và D như sau:
  - lớp cơ sở $C = \langle N_C, M_C, A_C \rangle$;



Hình 3.2: Phép toán tái cấu trúc *Abstraction*.

  - $M_C = \{m_C^1, m_C^2, ..., m_C^n\}$;
  - $A_C = \{a_C^1, a_C^2, ..., a_C^m\}$;
  - $INV_C = \bigwedge_{i=1}^{m} P_{A_C}^i$, trong đó $P_{A_C}^i$ là biểu thức logic vị từ biểu diễn ràng buộc trên thuộc tính a_C^i.
- lớp dẫn xuất $D = \langle N_D, M_D, A_D \rangle$ được kế thừa trực tiếp từ lớp C;
  - $M_D = \{m_D^1, m_D^2, ..., m_D^p\}$;
  - $A_D = \{a_D^1, a_D^2, ..., a_D^q\}$;
  - $INV_D = \bigwedge_{i=1}^{q} P_{A_D}^i$, trong đó $P_{A_D}^i$ là biểu thức logic vị từ biểu diễn cho các ràng buộc trên thuộc tính a_D^i.
- Đặt $\mathcal{M}_{C,D} = \{m_{C,D}^{ij}\}$ và $| \mathcal{M}_{C,D} | = k \quad (0 \leq k \leq \min(n, p))$
- Đặt $\mathcal{A}_{C,D} = \{a_{C,D}^{ij}\}$ và $| \mathcal{A}_{C,D} | = h \quad (0 \leq h \leq \min(m, q))$
- Điều kiện ban đầu để thực hiện phép toán này là các tập hợp $\mathcal{M}_{C,D} = \varnothing$ và $\mathcal{A}_{C,D} = \varnothing$.

• Mô hình tiến hóa $\mathcal{M}'$ chỉ bao gồm một lớp E như sau:
- lớp $E = \langle N_E, M_E, A_E \rangle$.
  - $M_E = M_C \cup M_D$
  - $A_E = A_C \cup A_D$
  - $INV_E = \bigwedge_{i=1}^{m+q} P_{A_E}^i$, trong đó $P_{A_E}^i$ biểu thức logic vị từ biểu diễn ràng buộc trên thuộc tính a_E^i, và:

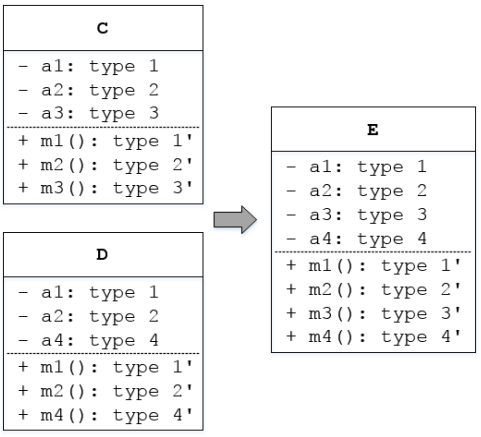
$$P_{A_E}^i = \begin{cases} P_{A_C}^i & \text{nếu } a_{A_E}^i \in A_C \\ P_{A_D}^i & \text{nếu } a_{A_E}^i \in A_D \end{cases}$$

• Biểu diễn bằng UML: Phép toán Folding được minh họa như trong Hình 3.1.

3.2.2.2 Phép toán: Abstraction

• Tên gọi: *Abstraction*
• Tình huống áp dụng:
- Thành phần: lớp C bao gồm danh sách dài các thuộc tính và phương thức, điều này gây khó khăn cho các hoạt động sử dụng lại;
- Mục đích: tạo lập một quan hệ phân cấp mới trong mô hình;
- Thực thi: tạo lập một lớp cơ sở mới E có hành vi trừu tượng hóa hơn so với lớp ban đầu, sau đó xác định lớp C' là các thành phần còn lại của lớp C sau khi đã trích lọc các thuộc tính và phương thức sang lớp E, kiến tạo mối quan hệ kế thừa trực tiếp giữa lớp cơ sở E và lớp dẫn xuất C'.

• Mô hình khởi đầu $\mathcal{M}$ bao gồm một lớp C:

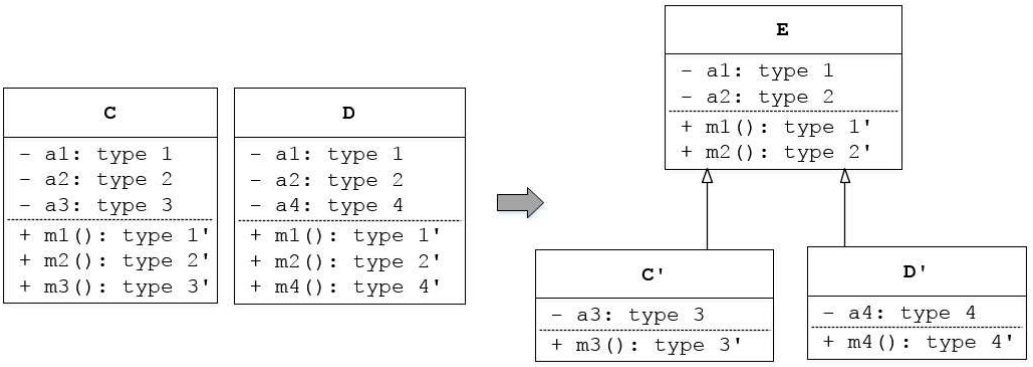


Hình 3.3: Phép toán tái cấu trúc *Composition*.

  - lớp $C = \langle N_C, M_C, A_C \rangle$;
    - $M_C = \{m_C^1, m_C^2, ..., m_C^p, ..., m_C^n\} \quad (1 \leq p \leq n)$. Giả sử rằng $M_C^1 = \{m_C^1, m_C^2, ..., m_C^p\}$ và $M_C^2 = \{m_C^{p+1}, m_C^{p+2}, ..., m_C^n\}$
    - $A_C = \{a_C^1, a_C^2, ..., a_C^q, ..., a_C^m\} \quad (1 \leq q \leq m)$.
      Giả sử rằng $A_C^1 = \{a_C^1, a_C^2, ..., a_C^q\}$ và $A_C^2 = \{a_C^{q+1}, a_C^{q+2}, ..., a_C^m\}$
    - $INV_C = \bigwedge_{i=1}^{m} P_{A_C}^i$, trong đó $P_{A_C}^i$ là ràng buộc trên thuộc tính a_C^i
- Mô hình tiến hóa $\mathcal{M}'$: không mất tính tổng quát, giả sử rằng tất cả các thuộc tính và phương thức được đưa lên lớp E, được sắp thứ tự tương ứng từ 1 đến p, và từ 1 đến k.
  Lớp cơ sở E là cha của lớp C' và được xây dựng như sau:
  - lớp $E = \langle N_E, M_E, A_E \rangle$, trong đó:
    - $M_E = M_C^1 = \{m_C^1, m_C^2, ..., m_C^p\}$
    - $A_E = A_C^1 = \{a_C^1, a_C^2, ..., a_C^q\}$
    - $INV_E = \bigwedge_{i=1}^{q} P_{A_E}^i$, ở đây $P_{A_E}^i \equiv P_{A_C^1}^i$ là ràng buộc trên thuộc tính a_E^i.
  - lớp $C' = \langle N_{C'}, M_{C'}, A_{C'} \rangle$, trong đó:
    - $M_{C'} = M_C^2 = \{m_C^{p+1}, ..., m_C^n\}$
    - $A_{C'} = A_C^2 = \{a_C^{q+1}, ..., a_C^m\}$
    - $INV_{C'} = \bigwedge_{i=1}^{m-q} P_{A_{C'}}^i$, sao cho $P_{A_{C'}}^i = P_{A_C^2}^{i+q}$ là ràng buộc trên thuộc tính $a_{C'}^i$
- Biểu diễn bằng UML: Phép toán Abstraction được minh họa như trong Hình 3.2.

3.2.2.3 Phép toán: Composition

- Tên gọi: *Composition*
- Tình huống áp dụng:
  - Thành phần: hai lớp C và D không có mối quan hệ kế thừa với nhau;
  - Mục đích: nhóm gộp các hành vi để loại bỏ quan hệ đa thừa kế trong mô hình;
  - Thực thi: xây dựng lớp E nhóm gộp các thuộc tính cũng như hành vi của cả 2 lớp C và D.
- Mô hình khởi đầu $\mathcal{M}$:
  - lớp $C = \langle N_C, M_C, A_C \rangle$ bao gồm các đặc tính như sau:
    - $M_C = \{m_C^1, m_C^2, ..., m_C^n\}$
    - $A_C = \{a_C^1, a_C^2, ..., a_C^m\}$
    - $INV_C = \bigwedge_{i=1}^{m} P_{A_C}^i$, trong đó $P_{A_C}^i$ biểu diễn ràng buộc trên thuộc tính a_C^i
  - lớp $D = \langle N_D, M_D, A_D \rangle$ được xác định như sau:
    - $M_D = \{m_D^1, m_D^2, ..., m_D^p\}$



Hình 3.4: Phép toán tái cấu trúc *Factoring*.

  - $A_D = \{a_D^1, a_D^2, ..., a_D^q\}$
  - $INV_D = \bigwedge_{i=1}^{q} P_{A_D}^i$, trong đó $P_{A_D}^i$ biểu diễn ràng buộc trên thuộc tính a_D^i
- $\mathcal{M}_{C,D} = \{m_{C,D}^{ij}\}$ và $\mid \mathcal{M}_{C,D} \mid = k (0 \leq k \leq \min(n, p))$
- $\mathcal{A}_{C,D} = \{a_{C,D}^{ij}\}$ và $\mid \mathcal{A}_{C,D} \mid = h \quad (0 \leq h \leq \min(m, q))$
- $\mathcal{M}_{C,D} \neq \varnothing$
- $\mathcal{A}_{C,D} \neq \varnothing$

- Mô hình tiến hóa $\mathcal{M}'$:
  - lớp $E = \langle N_E, M_E, A_E \rangle$ được xác định như sau:
    - $M_E = \{m_E^i : m_E^i \in \mathcal{M}_{C,D}\} \cup (M_C \setminus \mathcal{M}_{C,D}) \cup (M_D \setminus \mathcal{M}_{C,D})$
    - $A_E = \{a_E^i : a_E^i \in \mathcal{A}_{C,D}\} \cup (A_C \setminus \mathcal{A}_{C,D}) \cup (A_D \setminus \mathcal{A}_{C,D})$
    - $INV_E = \bigwedge_{i=1}^{m+q-h} P_{A_E}^i$, trong đó $P_{A_E}^i$ biểu diễn ràng buộc trên thuộc tính a_E^i, và:

$$P_{A_E}^i = \begin{cases} P_{A_C}^i \vee P_{A_D}^j & \text{nếu } a_{A_E}^i \in \mathcal{A}_{C,D} \\ P_{A_C}^i & \text{nếu } a_{A_E}^i \in (A_C \setminus \mathcal{A}_{C,D}) \\ P_{A_D}^i & \text{nếu } a_{A_E}^i \in (A_D \setminus \mathcal{A}_{C,D}) \end{cases}$$

- Biểu diễn bằng UML: Phép toán Composition được minh họa như trong Hình 3.3.

3.2.2.4 Phép toán: Factoring

- Tên gọi: *Factoring*
- Tình huống áp dụng:
  - Thành phần: 2 lớp C và D không có mối quan hệ kế thừa với nhau nhưng có một chung một số thuộc tính có thể kết hợp và phương thức tương đương về ngữ nghĩa;
  - Mục đích: loại bỏ các thuộc tính và phương thức giống nhau;
  - Thực thi: xây dựng một lớp mới E bằng cách trích lọc các thuộc tính có thể kết hợp và các phương thức tương đương về mặt ngữ nghĩa từ các lớp C và D, các lớp C' và D' tương ứng là phần còn lại của các lớp C và D sau khi trích chọn các thuộc tính và các phương thức. Các lớp C' và D' có mối quan hệ kế thừa trực tiếp từ lớp E.
- Mô hình khởi đầu $\mathcal{M}$:
  - lớp $C = \langle N_C, M_C, A_C \rangle$;
    - $M_C = \{m_C^1, m_C^2, ..., m_C^n\}$
    - $A_C = \{a_C^1, a_C^2, ..., a_C^m\}$
    - $INV_C = \bigwedge_{i=1}^{m} P_{A_C}^i$, trong đó $P_{A_C}^i$ biểu diễn ràng buộc trên thuộc tính a_C^i

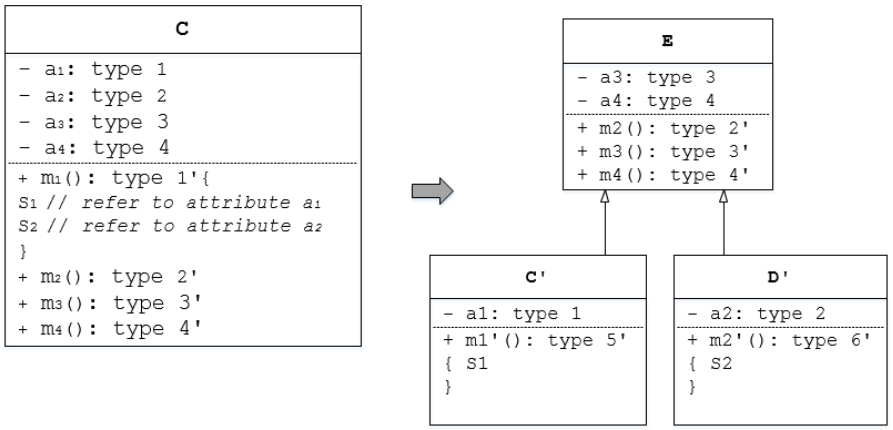
  - lớp $D = \langle N_D, M_D, A_D \rangle$;
    * $M_D = \{m_D^1, m_D^2, ..., m_D^p\}$
    * $A_D = \{a_D^1, a_D^2, ..., a_D^q\}$
    * $INV_D = \bigwedge_{i=1}^{q} P_{A_D}^i$, trong đó $P_{A_D}^i$ biểu diễn ràng buộc trên thuộc tính a_D^i
  - $\mathcal{M}_{C,D} = \{m_{C,D}^{ij}\}$ và $\mid \mathcal{M}_{C,D} \mid = k \quad (k \leq n \wedge k \leq p)$
  - $\mathcal{A}_{C,D} = \{a_{C,D}^{ij}\}$ và $\mid \mathcal{A}_{C,D} \mid = h \quad (h \leq m \wedge h \leq q)$
  - $\mathcal{M}_{C,D} \neq \varnothing$ và $\mathcal{A}_{C,D} \neq \varnothing$.
- Mô hình tiến hóa $\mathcal{M}'$:
  - lớp $C' = \langle N_{C'}, M_{C'}, A_{C'} \rangle$ được xác định như sau:
    * $M_{C'} = \{m_{C'}^i : m_{C'}^i \in (M_C \setminus \mathcal{M}_{C,D})\}$
    * $A_{C'} = \{a_{C'}^i : a_{C'}^i \in (A_C \setminus \mathcal{A}_{C,D})\}$
    * $INV_{C'} = \bigwedge_{i=1}^{(m-h)} P_{A_{C'}}^i$, $P_{A_{C'}}^i$ biểu diễn ràng buộc trên thuộc tính $a_{C'}^i$
  - lớp $D' = \langle N_{D'}, M_{D'}, A_{D'} \rangle$ được xác định như sau:
    * $M_{D'} = \{m_{D'}^i : m_{D'}^i \in (M_D \setminus \mathcal{M}_{C,D})\}$
    * $A_{D'} = \{a_{D'}^i : a_{D'}^i \in (A_D \setminus \mathcal{A}_{C,D})\}$
    * $INV_{D'} = \bigwedge_{i=1}^{(q-h)} P_{A_{D'}}^i$, $P_{A_{D'}}^i$ biểu diễn ràng buộc trên thuộc tính $a_{D'}^i$
  - lớp $E = \langle N_E, M_E, A_E \rangle$ được xác định như sau:
    * $M_E = \{m_E^i : m_E^i \in \mathcal{M}_{C,D}\}$
    * $A_E = \{a_E^i : a_E^i \in \mathcal{A}_{C,D}\}$
    * $INV_E = \bigwedge_{i=1}^{h} P_{A_E}^i$, trong đó $P_{A_E}^i$ biểu diễn ràng buộc trên thuộc tính a_E^i, và:

$$P_{A_E}^i = P_{A_C}^i \vee P_{A_D}^j \text{với } a_{A_E}^i \in \mathcal{A}_{C,D}$$

- Biểu diễn bằng UML: Phép toán Factoring được minh họa như trong Hình 3.4

3.2.2.5 Phép toán: Unfolding

- Tên gọi: *Unfolding*
- Tình huống áp dụng:
  - Thành phần: lớp C với một danh sách dài các phương thức, các phương thức này không tham chiếu một cách đồng bộ đến các thuộc tính của nó;
  - Mục đích: tối ưu hóa khả năng thực thi của mã nguồn;
  - Thực thi: nhóm các phương thức và các thuộc tính mà nó tham chiếu thành các lớp con bắt nguồn từ lớp C, các thuộc tính sẽ được phân chia thành các tập hợp rời rạc. Tạo mối quan hệ kế thừa từ phần còn lại của lớp C tới các lớp con vừa được tạo ra.
- Mô hình khởi đầu $\mathcal{M}$:
  - lớp $C = \langle N_C, M_C, A_C \rangle$;
    * $M_C = \{m_C^1, m_C^2, ..., m_C^n\}$
    * $A_C = \{a_C^1, a_C^2, ..., a_C^p, a_C^{p+1}, ..., a_C^{p+q}, a_C^{p+q+1}, ..., a^(p+q+m)_C\}$.
      Giả sử $A_C^1 = \{a_C^1, a_C^2, ..., a_C^p\}$, $A_C^2 = \{a_C^{p+1}, a_C^{p+2}, ..., a_C^{p+q}\}$
      and $A_C^3 = \{a_C^{p+q+1}, a_C^{p+q+2}, ..., a_C^{p+q+m}\}$
    * $INV_C = \bigwedge_{i=1}^{p+q+m} P_{A_C}^i$, trong đó $P_{A_C}^i$ biểu diễn ràng buộc trên thuộc tính a_C^i
  - Không mất tính tổng quát giả sử rằng:
    * đoạn mã nguồn S_1, S_2 thuộc về phương thức m_C^1 tham chiếu đến các thuộc tính $a_C^1, ..., a_C^p$ và $a_C^{p+1}, ..., a_C^{p+q}$, một cách tương ứng;
    * phương thức m_C^1 chỉ được tạo thành từ đoạn mã S_1 và S_2;
    * ở đây chúng ta chỉ thực hiện với phương thức m_C^1, các phương thức khác (nếu có xảy ra) sẽ được thực hiện một cách tương tự.
- Mô hình tiến hóa $\mathcal{M}'$:



Hình 3.5: Phép toán tái cấu trúc *Unfolding*.

- lớp $C' = \langle N_{C'}, M_{C'}, A_{C'} \rangle$ được định nghĩa như sau:
  - $M_{C'} = \{m^i_{C'} : m^i_{C'}$ được tạo thành từ đoạn mã $S_1\}$
  - $A_{C'} = A^1_C = \{a^i_C : 0 \leq i \leq p\}$
  - $INV_{C'} = \bigwedge_{i=1}^{p} P^i_{A_{C'}}$, trong đó $P^i_{A_{C'}} = P^i_{A_C}$, là ràng buộc trên thuộc tính $a^i_{C'}$
- lớp $D' = \langle N_{D'}, M_{D'}, A_{D'} \rangle$ được xác định như sau:
  - $M_{D'} = \{m^i_{D'} : m^i_{D'}$ được tạo thành từ đoạn mã $S_2\}$
  - $A_{D'} = A^2_C = \{a^i_C : (p+1) \leq i \leq q\}$
  - $INV_{D'} = \bigwedge_{i=p+1}^{p+q} P^i_{A_{D'}}$, trong đó $P^i_{A_{D'}} = P^i_{A_C}$, là ràng buộc trên thuộc tính $a^i_{D'}$
- lớp $E = \langle N_E, M_E, A_E \rangle$ được xác định như sau:
  - $M_E = \{m^i_C : m^i_C \in (M_C \setminus m^1_C)\}$
  - $A_E = A^3_C = \{a^i_C : a^i_C \in (A_C \setminus (A_{C'} \cup A_{D'}))\}$
  - $INV_E = \bigwedge_{i=p+q+1}^{p+q+m} P^i_{A_E}$, trong đó $P^i_{A_E} = P^i_{A_C}$, là ràng buộc trên thuộc tính a^i_E, và:
- các lớp C' và D' có mối quan hệ kế thừa trực tiếp từ lớp E và $A_{C'} \cap A_{D'} = \varnothing$.

- Biểu diễn bằng UML: Phép toán Unfolding được minh họa như trong Hình 3.5

3.3 Kết chương

Chương này luận án đề xuất phương pháp kiểm chứng tính bất biến trong tái cấu trúc biểu đồ lớp của UML. Để đạt được mục tiêu này, đầu tiên luận án tiến hành hình thức hóa biểu đồ lớp cùng với bất biến của nó. Kế tiếp, nhờ vào việc xác định các ràng buộc đối với từng phép toán, quá trình tái cấu trúc trên biểu đồ lớp được tiến hành theo cách bảo toàn các bất biến của nó. Đóng góp chính của luận án trong chương này là định nghĩa tập mẫu để mô tả phép toán và tích hợp bất biến của lớp vào trong tiến trình tái cấu trúc.

Các ràng buộc trên mô hình phần mềm được phân thành hai loại cơ bản: (1) Bất biến và (2) Tiền/hậu điều kiện. Trong chương này, chúng tôi đã đề xuất phương pháp tái cấu trúc trên biểu đồ lớp theo cách bảo toàn các bất biến của nó. Trong chương tiếp theo, luận án sẽ tiếp tục giải quyết bài toán tái cấu trúc trên hệ thống phần mềm mà vẫn đảm bảo sự duy trì các hành vi (tiền/hậu điều kiện) của hệ thống.

Chương 4

Kiểm chứng sự bảo toàn hành vi trong tái cấu trúc

Chương này luận án trình bày phương pháp kiểm chứng sự bảo toàn hành vi trong tái cấu trúc. Đối tượng chính mà luận án quan tâm đến sự bảo toàn hành vi là các kịch bản (scenarios) bị tác động trong tiến trình tái cấu trúc. Ý tưởng chính để giải quyết bài toán này là (1) đặc tả bằng phương pháp hình thức hệ thống phần mềm; (2) định nghĩa tập luật để quản lý sự biến đổi hành vi và (3) xây dựng các luật áp dụng trong kiểm chứng bảo toàn hành vi. Đóng góp chính của luận án trong chương này là đề xuất khái niệm kịch bản và kiểm chứng sự nhất quán về mặt hành vi trong tái cấu trúc.

4.1 Giới thiệu

Tái cấu trúc là một kỹ thuật mạnh mẽ được sử dụng để cải thiện cấu trúc bên trong của hệ thống phần mềm. Mẫu thiết kế cung cấp các giải pháp chung, tin cậy cho các vấn đề thường xảy ra trong lập trình. Sự kết hợp giữa tái cấu trúc và mẫu thiết là khá tự nhiên để tiến tới mục tiêu có được kiến trúc phần mềm chắc chắn. Trong chương này, luận án đề xuất phương pháp kiểm chứng sự bảo toàn hành vi trong tái cấu trúc hệ thống phần mềm, có sử dụng mẫu thiết kế. Phương pháp đề xuất áp dụng trong các giai đoạn thiết kế và cài đặt của quy trình phát triển phần mềm.

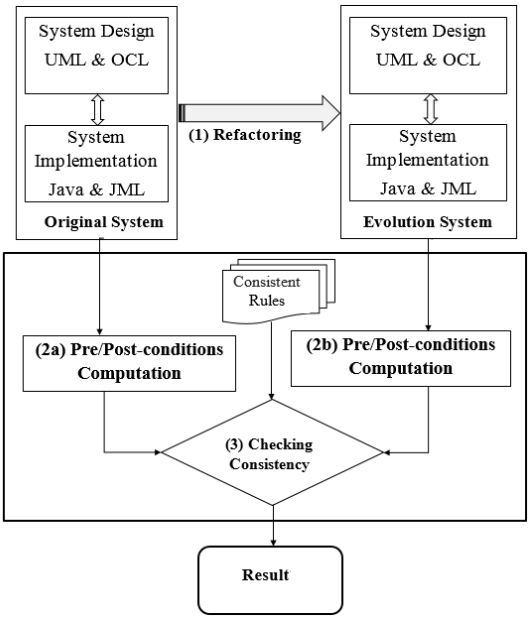
4.2 Kiểm chứng tính nhất quán về mặt hành vi trong tái cấu trúc hệ thống phần mềm

4.2.1 Tổng quan về quy trình kiểm chứng sự bảo toàn hành vi trong tái cấu trúc hệ thống phần mềm

Hình 4.1 mô tả quy trình với ba tiến trình chính: *(1) tái cấu trúc*, *(2) tính toán tiền và hậu điều kiện của các kịch bản* và *(3) thực hiện kiểm chứng.*

Tại giai đoạn thiết kế, luận án sử dụng các biểu đồ lớp và biểu đồ tuần tự (còn gọi là kịch bản) trong UML để mô hình hóa hệ thống phần mềm, ngôn ngữ ràng buộc đối tượng OCL để biểu diễn các hành vi (bao gồm bất biến, tiền và hậu điều kiện). Mô hình sau tái cấu trúc được tính toán lại các ràng buộc về hành vi. Cuối cùng, dựa trên tập luật đã xây dựng, tiến trình kiểm chứng được thực hiện. Quá trình kiểm chứng ở giai đoạn cài đặt được thực hiện một cách tương tự, chỉ khác biệt về chế tác phần mềm được tái cấu trúc là chương trình phần mềm. Trong luận án này, chúng tôi cài đặt hệ thống bằng ngôn ngữ lập trình Java và đặc tả hành vi bằng JML. Nhờ vào sự hỗ trợ của công cụ OpenJML tích hợp vào môi trường Eclipse, quá trình kiểm chứng được thực thi một cách tự động.

Kết quả cuối cùng về sự bảo toàn hành vi trong tái cấu trúc hệ thống là sự kết hợp các kết quả của hai quá trình kiểm chứng trong các giai đoạn thiết kế và cài đặt nói trên.



Hình 4.1: Tổng quan về quy trình kiểm chứng

4.2.2 Phương pháp kiểm chứng tính nhất quán trong tái cấu trúc mô hình phần mềm

4.2.2.1 Mô hình hóa hệ thống phần mềm bằng UML

Các thành phần của hệ thống được mô hình hóa một cách tuần tự dựa trên các định nghĩa sau đây:

Định nghĩa 4.1 (Mô hình)**.** *Một mô hình M là một bộ-2 $\langle C_M, S_M \rangle$, trong đó C_M là tập hợp các lớp và S_M là tập hợp hành vi của các kich bản có trong mô hình.*

Định nghĩa 4.2 (Lớp)**.** *Một lớp $Ci_M \in C_M$ được biểu diễn bởi bộ-3 $Ci_M = \langle OP_{CiM}, A_{CiM}, I_{CiM} \rangle$, ở đây OP_{CiM} là tập hợp các phương thức công khai, A_{CiM} là tập hợp các thuộc tính công khai, và I_{CiM} là tập trạng thái các bất biến của lớp.*

Định nghĩa 4.3 (Tiền điều kiện của phương thức trừu tượng)**.** *Tiền điều kiện $PRE_{op_{iM}}$ của phương thức op_{ei}, được hiện thực hóa bởi N phương thức op_{ei} trong các lớp con $Cksi_M$, trong lớp trừu tượng Ci_M được tính bằng hợp tiền điều kiện của tất cả các phương thức op_{ei} trong các lớp con $Cksi_M$.*

Giả sử $Pi(op_{ei})$ biểu diễn tiền điều kiện của phương thức op_{ei} trong các lớp con, khi đó chúng ta tính tiền điều kiện theo công thức $PRE_{op_{iM}} = \bigvee Pi(op_{ei})$ cho phương thức op_{ei} trong lớp trừu tượng Ci_M, ở đây $op_{ei} \in Cksi_M$ là các phương thức hiện thực hóa, và P là mệnh đề.

Định nghĩa 4.4 (Hậu điều kiện của phương thức trừu tượng)**.** *Hậu điều kiện $POST_{op_{iM}}$ của phương thức op_{ei}, được hiện thực hóa bởi N phương thức op_{ei} trong các lớp con $Cksi_M$, trong lớp trừu tượng Ci_M được tính bằng hợp hậu điều kiện của tất cả các phương thức op_{ei} trong các lớp con $Cksi_M$.*

Tương tự như cách tính tiền điều kiện của phương thức trong lớp trừu tượng, dễ thấy $POST_{op_{iM}} = \bigvee Pi(op_{ei})$, ở đây $op_{ei} \in Cksi_M$ là các phương thức hiện thực hóa, và P là các mệnh đề biểu diễn hậu điều kiện của các phương thức op_{ei} trong các lớp con.

Trong chương này, mỗi một kịch bản của hệ thống tương ứng với một biểu đồ tuần tự trong UML.

Định nghĩa 4.5 (Kịch bản). *Một kịch bản Si_M được biểu diễn bởi bộ-4*
$Si_M = \langle CI_{SiM}, PRE_{SiM}, E_{SiM}, POST_{SiM}\rangle$, trong đó $CI_{SiM} \subseteq Ci_M$ biểu diễn cho tập hợp các lớp tham gia vào kịch bản, PRE_{SiM} là tiền điều kiện của kịch bản, E_{SiM} là tập có thứ tự các phương thức trong các lớp được thực thi trong kịch bản, và $POST_{SiM}$ là hậu điều kiện của kịch bản đó.

Định nghĩa 4.6 (Phương thức của kịch bản). *Một phương thức của kịch bản là một bộ-4*
$Ek_{SiM} = \langle PRE_{Ek_{SiM}}, OP_{Ek_{SiM}}, POST_{Ek_{SiM}}, k\rangle$, trong đó $PRE_{Ek_{SiM}}$ là tiền điều kiện của phương thức, $OP_{Ek_{SiM}}$ là các phương thức công khai thực thi trong kịch bản, $POST_{Ek_{SiM}}$ là hậu điều kiện của các phương thức, và k là thứ tự thực hiện của phương thức trong kịch bản.

Trong luận án này, chúng ta xem xét trường hợp mà cả tiền và hậu điều kiện của một phương thức là sự kết hợp của các vị từ trên các thuộc tính của các lớp tham gia vào kịch bản, ví dụ, $PRE_{Ek_{SiM}} = \bigwedge P(A_{CijM})$, ở đây $A_{CijM} \in A_{CiM}$ là thuộc tính, $CiM \subseteq CI_{SiM}$ và P là vị từ. Một kịch bản bao gồm một chuỗi các hoạt động, do đó tiền và hậu điều kiện của một phương thức được hình thành bởi biểu thức tiền và hậu điều kiện của phương thức thực hiện ngay trước đó. Lưu ý rằng, một phương thức công khai của một lớp trong các kịch bản khác nhau có thể có tiền và hậu điều kiện khác nhau. Tiền và hậu điều kiện của một kịch bản được xác định dựa trên tiền và hậu điều kiện của tất cả các phương thức có tham gia và kịch bản. Và được định nghĩa như sau:

Định nghĩa 4.7 (Tiền điều kiện của kịch bản). *Tiền điều kiện của một kịch bản PRE_{SiM} được xác định bởi tiền điều kiện của phương thức đầu tiên thực thi trong kịch bản đó.*

Tiền điều kiện của phương thức đầu tiên trong kịch bản được đặc bởi các ràng buộc trên tất cả các thuộc tính công khai của kịch bản.

Định nghĩa 4.8 (Hậu điều kiện của kịch bản). *Hậu điều kiện của một kịch bản $POST_{SiM}$ được xác định bởi hợp của các ràng buộc trên các thuộc tính công khai A_{CijM} của hậu điều kiện của phương thức $POST_{Ek_{SiM}}$ diễn ra sau cùng trong E_{SiM}.*

Cho một kịch bản $S = (e_1, e_2, ..e_n)$, trong đó e_i, $i = \overline{1..n}$, là phương thức thứ i thực thi trong kịch bản. Theo định nghĩa 4.6, chúng ta có $e_i = (pre_{ei}, op_{ei}, post_{ei}, i)$ và $post_(ei) = \bigwedge P_k(A_k C)$, ở đây P_k là các logic vị từ $A_k C$, biểu diễn cho thuộc tính của các lớp C có liên quan đến kịch bản này. Giả sử rằng kịch bản S chỉ có duy nhất một thuộc tính công khai A_C cùng xuất hiện trong biểu thức hậu điều kiện của e_i và e_j sao cho $1 \leq i < j \leq n$. Như vậy, chúng ta sẽ có $post_{ei} = P_i(A_C)$ và $post_{ej} = P_j(A_C)$. Nếu e_i thực thi trước e_j, $P_j(A_C)$ cần phải được duy trì cho đến khi phương thức thứ j được thực thi.

Định nghĩa 4.9 (Tái cấu trúc). *Một tiến trình tái cấu trúc R sử dụng mẫu thiết kế D được biểu diễn như sau: $R : M \xmapsto{D(SUB_{MS})} M'$, trong đó M và M' theo thứ tự biểu thị cho các mô hình trước và sau khi thực hiện tái cấu trúc, D là tên của mẫu thiết kế được ứng dụng, và $SUB_{MS} \subseteq S_M$ là tập các kịch bản tham gia vào tiến trình tái cấu trúc này.*

Với mục tiêu giảm độ phức tạp cho quá trình kiểm chứng, các thuộc tính cần kiểm chứng sẽ được phân thành hai loại *(i) Các thuộc tính tĩnh (static)*, và *(ii) các thuộc tính động (dynamic)*. Đối với loại thuộc tính thứ nhất, phương pháp kiểm chứng dựa trên các bất biến của lớp, với loại thuộc tính thứ hai, quá trình kiểm chứng sẽ thao tác trên tiền/hậu điều kiện của các kịch bản. Chú ý rằng, trong Chương 3 luận án đã giải quyết bài toán kiểm chứng sự bảo toàn bất biến trong tái cấu trúc biểu đồ lớp của UML. Chương này, khái niệm về bất biến được hiểu một cách tương tự (các ràng buộc trên

các thuộc tính của lớp), tuy nhiên tiến trình tái cấu trúc lại được thực hiện thông qua việc sử dụng mẫu thiết kế Strategy. Nói cách khác, quá trình tái cấu trúc thực hiện ở Chương 3 và Chương 4 là hoàn toàn khác nhau. Do đó, luận án vẫn tiếp tục thực hiện kiểm chứng sự bảo toàn về bất biến trong tiến trình tái cấu trúc ở chương này.

Quá trình kiểm chứng sẽ được thực hiện một cách lần lượt, dựa trên sự tuân thủ các Mệnh đề chi tiết sau đây.

4.2.2.2 Kiểm chứng các thuộc tính tĩnh

Mệnh đề 4.1 (Bảo toàn các tính chất tĩnh). Quá trình tái cấu trúc một mô hình phần mềm được coi là bảo toàn các thuộc tính tĩnh nếu các bất biến của các lớp trên mô hình hiện tại tương đương logic với các bất biến của các lớp trên mô hình ban đầu.

Chúng ta có thể hình thức hóa mệnh đề trên cụ thể như sau. Cho $R : M \xmapsto{\mathrm{D}(SUB_{MS})} M'$ là mô hình được tái cấu trúc, R được coi là bảo toàn các tính chất tĩnh nếu $\forall\, Ci_M \mid Ci_M \in C_M \wedge Ci_M \in C'_M \Rightarrow I_{CiM} \equiv I_{CiM'}$

4.2.2.3 Kiểm chứng các thuộc tính động

Các thuộc tính động được hiểu là các ràng buộc về hành vi của mô hình phần mềm, trong luận án này, chúng tôi quan tâm đến sự bảo toàn hành vi của các kịch bản tham gia vào quá trình tái cấu trúc.

Mệnh đề 4.2. (Nhất quán toàn phần trên các thuộc tính động) Một tiến trình tái cấu trúc trên mô hình phần mềm được gọi là nhất quán toàn phần trên các thuộc tính động (với mô hình ban đầu), nếu với mỗi kịch bản trên mô hình hiện tại, tiền và hậu điều kiện của nó đều tương đương logic với tiền và hậu điều kiện của kịch bản tương ứng trên mô hình ban đầu.

Một cách hình thức hóa, cho $R : M \xmapsto{\mathrm{D}(SUB_{MS})} M'$ là một tiến trình tái cấu trúc, $S_{iM} \in SUB_{MS}$ gọi là nhất quán toàn phần trên các thuộc tính động, nếu $PRE_{SiM} \equiv PRE_{SiM'} \wedge POST_{SiM} \equiv POST_{SiM'}$. Trong phần 4.2.2.1, chúng ta đã chỉ ra tiền và hậu điều kiện của các kịch bản được suy dẫn bởi các phương thức tham gia vào kịch bản, bởi vậy nếu $PRE_{SiM} \equiv PRE_{SiM'} \wedge POST_{SiM} \equiv POST_{SiM'}$, tất cả các ràng buộc trên các thuộc tính công khai của các kịch bản sau tái cấu trúc sẽ được bảo toàn.

Mệnh đề 4.3 (Nhất quán bộ phận trên các thuộc tính động). Một tiến trình tái cấu trúc mô hình phần mềm được gọi là nhất quán bộ phận trên các thuộc tính động (so với mô hình ban đầu) nếu tiền điều kiện của các kịch bản trên mô hình hiện tại (mô hình sau khi đã tái cấu trúc) không hề thay đổi, hậu điều kiện của các kịch bản chỉ thỏa mãn một phần so với tiền/hậu điều kiện của các kịch bản tương ứng trong mô hình gốc.

Tương tự như trên, chúng ta có thể hình thức hóa tiến trình nhất quán bộ phận trên các thuộc tính động như sau. Cho $R : M \xmapsto{\mathrm{D}(SUB_{MS})} M'$ là một tiến trình tái cấu trúc, $S_{iM} \in SUB_{MS}$ được gọi là nhất quán một phần nếu như $PRE_{SiM} \equiv PRE_{SiM'} \wedge POST_{SiM} \Rightarrow POST_{SiM'}$. Trong trường hợp này, nếu $POST_{SiM} \Rightarrow POST_{SiM'}$, như vậy giá trị của các ràng buộc trên các kịch bản của các thuộc tính cộng khai vẫn nằm trong phạm vi mong muốn.

Mệnh đề 4.4 (Không nhất quán). Một tiến trình tái cấu trúc mô hình phần mềm được gọi là không nhất quán nếu nó đồng thời không thỏa mãn cả nhất quán toàn phần và nhất quán bộ phận trên các thuộc tính động.

Cho $R : M \xmapsto{\mathrm{D}(SUB_{MS})} M'$ là một tiến trình tái cấu trúc, $S_{iM} \in SUB_{MS}$ là không nhất quán nếu S_{iM} không thỏa mãn nhất quán một phần (đương nhiên nó cũng không

thể thỏa mãn nhất quán toàn phần). Trong trường hợp này, giá trị của các ràng buộc trên các thuộc tính công khai của kịch bản bị vi phạm sau tiến trình tái cấu trúc.

4.2.3 Kiểm chứng tính nhất quán trong tái cấu trúc chương trình phần mềm

Mệnh đề 4.5 (Bảo toàn sự thực thi của chương trình nguồn). Một chương trình P được gọi là bảo toàn thực thi nếu với mỗi kịch bản, trước và sau khi thực thi chương trình tiền và hậu điều kiện của nó đều được bảo toàn .

Chúng ta có thể biểu diễn một cách hình thức hóa mệnh đề này như sau,
$$PRE_{S_P}[S_P]POST_{S_P}.$$

Mệnh đề 4.6 (Sự bảo toàn thực thi của chương trình tái cấu trúc). Một chương trình tái cấu trúc P' được gọi là bảo toàn thực thi với chương trình nguồn P nếu với cùng một kịch bản, tiền và hậu điều kiện của nó đều được bảo toàn sau khi thực hiện tái cấu trúc.

Chúng ta biểu diễn mệnh đề này bằng công thức hình thức như sau,
$$PRE_{S_P}[S_{P'}]POST_{S_P}.$$

4.2.4 Kiểm chứng sự bảo toàn hành vi trong tái cấu trúc mô hình hệ thống ARTC

Hệ thống điều khiển giao thông đường bộ (Adaptive Road Traffic Control - ARTC) cho phép phối hợp hoạt động của các tín hiệu đèn trong một khu vực và phản ứng một cách thông minh với sự dao động của lưu lượng giao thông tại một thời điểm bất kỳ. Hệ thống ARTC hiện thời có một số hạn chế về khả năng thực thi cũng như sự hỗ trợ cho các tiến trình sử dụng lại. Bởi vậy, chúng tôi đã tiến hành tái cấu trúc hệ thống này bằng cách sử dụng mẫu Strategy. Không mất tính tổng quát, giả sử tại một hướng nào đó của giao lội, luận án xác định một số hành vi cần bảo toàn của hệ thống như sau:

- Khi giao thông đang ở trạng thái bị ách tắc (*heavyTraffic*), tín hiệu đèn xanh cần được bật lên (nếu nó đang ở trạng thái đỏ).
- Khi giao thông đang ở trạng thái bị ách tắc (heavyTraffic), tín hiệu đèn xanh đã được bật lên, thì thời gian (*time*) dành cho tín hiệu này cần được gia tăng.
- Tại một hướng giao lộ nào đó, khi giao thông đang ở trạng thái với lưu lượng cao (*highTraffic*), tín hiệu đèn xanh đang được bật, thì phương tiện giao thông nên lựa chọn theo hướng đi này.

Theo Mệnh đề 4.1, dễ thấy, các thuộc tính tĩnh (bất biến) của mô hình trước và sau khi áp dụng mẫu thiết kế Strategy không hề thay đổi. Như vậy, quá trình tái cấu trúc hoàn toàn bảo toàn các thuộc tính tĩnh.

Quá trình kiểm chứng sự bảo toàn hành vi thực hiện trên các ràng buộc về tiền/hậu của kịch bản *analyzeTraffic()*. Các ràng buộc này trong mô hình khởi đầu được mô tả như trong Đặc tả 4.1:

```
1 PRE_SiM = trafficFlow -> isEmpty()
2 POST_SiM = if (state = heavyTraffic) then ((signal = green) AND (
    greenTime > 60))
3             else if (state = lowTraffic) then ((signal = red) AND (
    greenTime <= 60))
4              else if (state = highTraffic) then (direction = CHOOSE
    )
5                       else (direction = NO_CHOOSE)
6                  endif
7              endif
8          endif
```

Đặc tả 4.1: Các ràng buộc trên kịch bản *optimizeTraffic()* trước khi tái cấu trúc

Một cách tương tự, chúng ta cũng chỉ ra được tiền và hậu điều kiện của kịch bản *optimizeTraffic()* trong mô hình tiến hóa như trong Đặc tả 4.2:

```
1 PRE_S'iM = trafficFlow -> isEmpty()
2 POST_S'iM = if (state = heavyTraffic) then ((signal = green) AND (
    greenTime > 60))
3             else if (state = lowTraffic) then ((signal = red) AND
    (greenTime <= 60))
4              else if (state = highTraffic) then (direction =
    CHOOSE)
5                       else (direction = NO_CHOOSE)
6                       endif
7              endif
8         endif
```

Đặc tả 4.2: Các ràng buộc trên kịch bản *optimizeTraffic()* sau khi tái cấu trúc

Từ các giá trị về tiền và hậu điều kiện tính toán được của các kịch bản trên. Theo Mệnh đề 4.2, quá trình tái cấu trúc sử dụng mẫu thiết kế Strategy trong hệ thống ARTC các thuộc tính động của mô hình ban đầu. Nói cách khác, hành vi của mô hình hệ thống ARTC trước và sau khi tái cấu trúc là nhất quán toàn phần với nhau. Quá trình kiểm chứng sự bảo toàn hành vi của hệ thống ARTC tại giai đoạn cài đặt được thực hiện một cách tương tự.

4.3 Kết chương

Trong chương này, luận án đã đề xuất phương pháp kiểm chứng tính nhất quán về mặt hành vi trong tái cấu trúc hệ thống phần mềm bằng cách áp dụng mẫu thiết kế, tại các giai đoạn thiết kế và cài đặt phần mềm. Để kiểm chứng tại giai đoạn thiết kế, chúng tôi sử dụng UML để mô hình hóa hệ thống; dùng OCL để biểu diễn các hành vi cần bảo toàn. Để kiểm chứng tại giai đoạn cài đặt, chúng tôi mã hóa hệ thống bằng ngôn ngữ lập trình Java, biểu diễn hành vi bằng JML và thực hiện kiểm chứng một cách tự động bằng phần mềm OpenJML tích hợp vào môi trường Eclipse. Chú ý rằng, đối tượng chính mà chúng tôi quan tâm đến sự bảo toàn hành vi là *các kịch bản tham gia vào quá trình tái cấu trúc*. Tại giai đoạn thiết kế quá trình kiểm chứng vẫn đang được thực hiện một cách thủ công, trong chương tiếp theo chúng tôi sẽ phát triển công cụ để hỗ trợ cho quá trình này.

Kết quả nghiên cứu nêu trên được công bố tại kỷ yếu có phản biện của Hội nghị quốc tế lần thứ 5 về *EAI International Conference on Context-Aware Systems and Applications 2016* (ICCASA) (công trình khoa học số 1) và Tạp chí*Mobile Networks and Applications journal (MONET)* (SCIE-indexed) Vol.22.2 (công trình khoa học số 2).

Chương 5

Công cụ kiểm chứng

Trong các chương trước, luận án đã đề xuất các phương pháp kiểm chứng sự nhất quán về bất biến và hành vi trong tái cấu trúc hệ thống phần mềm. Trong chương này, luận án tiếp tục xây dựng công cụ CVT hỗ trợ cho quá trình kiểm chứng trong tái cấu trúc mô hình phần mềm.

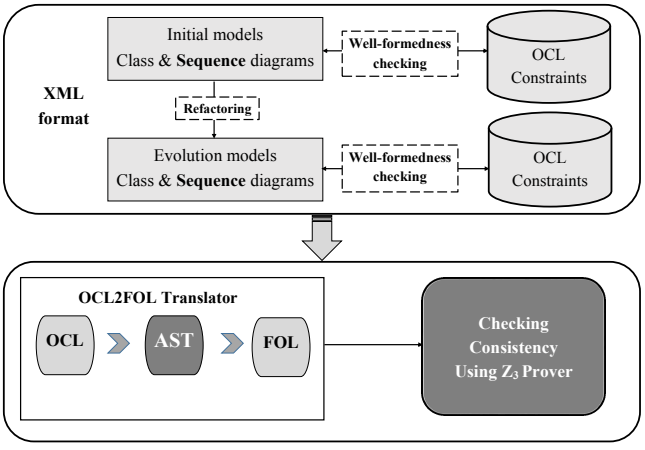
5.1 Giới thiệu

Tái cấu trúc hệ thống phần mềm mang lại một số ưu điểm như dễ dàng cho việc bảo trì, cải thiện chất lượng của hệ thống. Tuy nhiên, tiến trình này tiêu tốn khá nhiều thời gian và dễ phát sinh lỗi, đặc biệt khi thực hiện một cách thủ công. Trong chương này, luận án phát triển công cụ CVT, nhận đầu vào là các mô hình phần mềm trước và sau khi tái cấu trúc cùng với các biểu thức OCL biểu diễn hành vi của hệ thống và trả lại kết quả là khả năng nhất quán giữa các mô hình này.

5.2 Xây dựng công cụ kiểm chứng CVT

5.2.1 Kiến trúc của công cụ CVT

Từ góc nhìn của người thiết kế hệ thống, kiến trúc của CVT bao gồm hai phần chính, cụ thể là *môi trường định nghĩa* và *môi trường thực thi* được mô tả như trong Hình 5.1.



Hình 5.1: Kiến trúc của công cụ CVT

Môi trường định nghĩa chịu trách nhiệm về các tác vụ *tạo lập* các mô hình trước và sau khi tái cấu trúc, *soạn thảo* các tệp XML biểu diễn ràng buộc về hành vi của các mô hình trên và *kiểm tra* sự tương thích về kiểu giữa mô hình với ràng buộc của nó.

Môi trường thực thi là khối bên dưới của Hình 5.1, khối này bao gồm hai phần chính là *OCL2FOL Translator* và *Checking Consistency*. Phần *OCL2FOL Translator* ở phía

bên trái thực thi nhiệm vụ chuyển đổi biểu thức OCL sang các công thức FOL. Còn ở phía bên phải, chức năng *Checking Consistency* chỉ đơn giản là nhận vào công thức FOL (là sản phẩm đầu ra của chức năng *OCL2FOL Translator*) và so sánh sự tương đương giữa hai công thức này. Chú ý rằng, hai công thức FOL này vẫn tương ứng với các biểu thức OCL của hai mô hình trước và sau khi tái cấu trúc. Để thực thi chức năng *Checking Consistency*, luận án sử dụng lại một công cụ nổi tiếng được cung cấp bởi nhóm nghiên cứu của Microsoft là Z3. Kết quả cuối cùng về tính nhất quán giữa các hệ thống phần mềm cũng được thông báo bởi chức năng này. Nếu hai công thức FOL là tương đương nhau, điều này đồng nghĩa với việc hai mô hình trước và sau khi tái cấu trúc là nhất quán toàn phần với nhau, trong các trường hợp còn lại, hai mô hình có thể nhất quán một phần, hoặc không nhất quán với nhau.

5.2.2 Chuyển đổi biểu thức OCL sang công thức FOL

Các bước chuyển đổi từ OCL sang FOL được mô tả như sau:

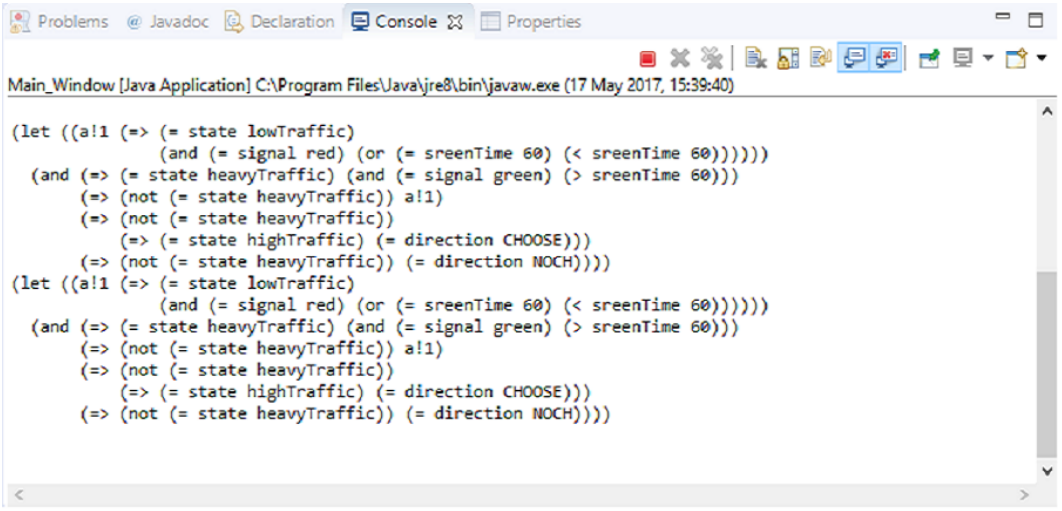
- Định nghĩa văn phạm phi chính quy G sản xuất các biểu thức OCL.
- Định nghĩa tập luật chuyển đổi kết hợp với các luật sản xuất của G.
- Sử dụng thuật toán LL(1) để sản sinh cây phân tích cú pháp P từ $C \in L(G)$ và G;
- Dựa trên tập luật của G và P đã có, đề xuất thuật toán xây dựng cây AST.
- Sử dụng thuật toán tìm kiếm theo chiều sâu (DFS) cho cây AST, và ánh xạ mỗi nút của cây AST sang các công thức $f_1, f_2, ..., f_n$ trong FOL.
- Kết hợp các công thức thành phần $f_1, f_2, ..., f_n$ sẽ thu được công thức F tương ứng với biểu thức C ban đầu.

5.3 Cài đặt và thực nghiệm

Công cụ CVT [1] được phát triển để hỗ trợ cho quá trình kiểm chứng tính nhất quán về mặt hành vi trong tái cấu trúc mô hình phần mềm. Công cụ CVT sẽ nhận đầu vào là *(i) các mô hình* (biểu đồ lớp và biểu đồ tuần tự) và *(ii) hành vi* (biểu diễn bằng các biểu thức OCL) của các hệ thống trước và sau khi tái cấu; sau quá trình kiểm tra, kết quả trả về là một trong ba khả năng sau: (1) hai hệ thống nhất quán toàn phần, (2) hai hệ thống nhất quán một phần hoặc (3) hai hệ thống không nhất quán. Sau khi nhập các tệp trên vào, công cụ CVT sẽ bắt đầu với chức năng *well-formedness checking* để kiểm tra sự tương thích về kiểu giữa các biểu thức OCL và các đặc tả thành phần của mô hình UML. Trong trường hợp biểu thức OCL không tương thích với các thành phần của mô hình UML, CVT sẽ có thông báo lần vết và yêu cầu nhập lại các tệp dữ liệu đầu vào. Hình 5.2 minh họa hai công thức FOL dạng tiền tố tương ứng với hai biểu thức OCL biểu diễn hành vi của các hệ thống ARTC trước và sau khi tái cấu trúc.

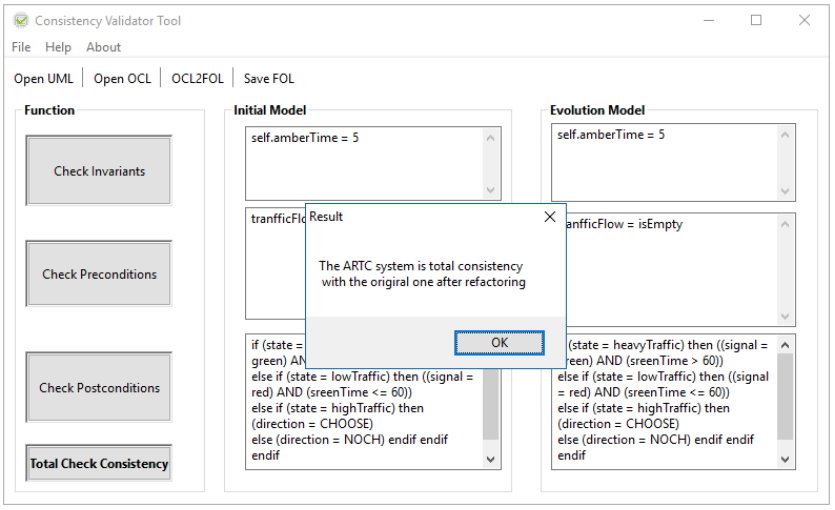
CVT có một số nút chức năng cơ bản bao gồm *Check Invariants*, *Check Pre-conditions*, *Check Post-conditions* và *Total Consistency*. Các nút chức năng này bắt nguồn từ việc các biểu thức OCL có ba thành phần chính (bất biến, tiền điều kiện và hậu điều kiện). Nếu mô hình tái cấu trúc bảo toàn đối với tất cả các ràng buộc trên thì nó được coi là nhất quán toàn phần (*Total Consistency*), nếu mô hình tái cấu trúc vi phạm một trong ba loại ràng buộc thì nó là nhất quán bộ phận (*Partial Consistency*), trong các trường hợp còn lại các mô hình phần mềm được coi là không nhất quán (*Inconsistency*). Trong quá trình thực thi, CVT cho phép người dùng lựa chọn một chức năng con để kiểm tra nhất quán bộ phận hoặc thực hiện đồng bộ tất cả các chức năng cùng lúc bằng nút **Total Check Consistency** để có được kết quả kiểm tra cuối cùng giữa hai mô hình

1. `Our code is open-source and available at: https://github.com/huongdt1901/ARTC`



Hình 5.2: Công thức FOL dạng tiền tố tương ứng với biểu thức OCL

phần mềm. Các kết quả thực nghiệm trên hệ thống ARTC được mô tả như trong Hình 5.3.



Hình 5.3: Kết quả kiểm chứng sự bảo toàn hành vi trong tái cấu trúc hệ thống ARTC

5.4 Kết chương

Công cụ CVT thể hiện tính ngữ cảnh, ứng dụng vào một tiến trình cụ thể trong giai đoạn cải tiến chất lượng phần mềm. CVT bao gồm các chức năng: Kiểm tra tính hợp lệ về dữ liệu, Kiểm tra tính nhất quán về mặt hành vi và chuyển đổi các biểu thức OCL phức hợp sang các công thức FOL. Dựa vào công cụ hỗ trợ Z3 có sẵn, quá trình kiểm tra thực hiện và trả lại kết quả về sự nhất quán giữa các mô hình trước và sau tái cấu trúc.

Kết quả nghiên cứu trên đã được công bố tại Kỷ yếu có phản biện của Hội nghị quốc tế lần thứ 9 về *Knowledge and Systems Engineering* - KSE 2017 (công trình khoa học số 3).

Chương 6

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Các đóng góp của luận án

Tái cấu trúc giúp người phát triển có được kiến trúc phần mềm chắc chắn, là nền tảng quan trọng cho các giai đoạn phát triển tiếp theo của chính phần mềm đó. Ý tưởng chính của tái cấu trúc là phân bố lại các lớp, các thuộc tính và các phương thức xung quanh mối quan hệ phân cấp giữa các thành phần của hệ thống.

Kiểm chứng nhất quán trong tái cấu trúc hệ thống phần mềm là một bài toán mở và mang tính chuyên biệt miền. Giải quyết bài toán này cần gắn liền với các yếu tố đặc trưng về miền ứng dụng hoặc mối quan tâm cụ thể của các nhà phát triển phần mềm. Sau một thời gian nghiên cứu, luận án đã đạt được một số đóng góp chính như sau:

- *Đề xuất phương pháp kiểm chứng tính bất biến của lớp trong tái cấu trúc mô hình phần mềm.* Để giải quyết bài toán này, luận án hình thức hóa biểu đồ lớp cùng với các ràng buộc về bất biến bằng các ký pháp toán học. Kế tiếp, luận án định nghĩa mẫu (template) mô tả các phép toán. Cuối cùng, bằng việc xây dựng các luật chuyển đổi áp dụng cho từng phép toán, quá trình tái cấu trúc trên biểu đồ lớp được tiến hành theo cách bảo toàn các bất biến.
- *Đề xuất phương pháp kiểm chứng sự bảo toàn hành vi trong tái cấu trúc hệ thống phần mềm có sử dụng mẫu thiết kế.* Quy trình tổng quan để giải quyết bài toán này bao gồm: *(i) đặc tả các hệ thống thống cùng với các ràng buộc về hành vi (bất biến, tiền/hậu điều kiện) bằng phương pháp hình thức*; (ii)*t*iến hành tái cấu trúc bằng mẫu Strategy; *(iii) tính toán các ràng buộc về hành vi của phiên bản tiến hóa sau tái cấu trúc* và *(iv) Xây dựng tập luật nhất quán là cơ sở lý thuyết cho quá trình kiểm chứng.* Quy trình trên được triển khai xuyên suốt từ giai đoạn thiết kế tới giai đoạn cài đặt phần mềm. Chú ý rằng, trong đóng góp này, chúng tôi đã đề xuất và tập trung đề xuất các định nghĩa và kiểm chứng tính nhất quán về mặt hành vi với đối tượng là các kịch bản (scenarios) tham gia vào tiến trình tái cấu trúc. Hệ thống phần mềm được coi là nhất quán với phiên bản tái cấu trúc của nó nếu các kịch bản này bảo toàn các ràng buộc về bất biến, tiền và hậu điều kiện.
- *Xây dựng công cụ CVT hỗ trợ kiểm chứng sự bảo toàn hành vi trong tái cấu trúc mô hình phần mềm.* CVT nhận đầu vào là mô hình các hệ thống trước và sau tái cấu trúc cùng với các ràng buộc về hành vi (bất biến, tiền và hậu điều kiện). Kết quả là kết luận về khả năng nhất quán giữa các mô hình này. CVT bao gồm ba chức năng chính: (i) *Kiểm tra tính hợp lệ về ràng buộc OCL trên mô hình*, (ii) *Chuyển đổi biểu thức OCL sang công thức FOL* và (iii) *Thực hiện kiểm tra.*
- *Minh họa các phương pháp đề xuất thông qua mô hình Hệ thống điều khiển giao thông đường bộ ARTC.* Các ràng buộc về hành vi đã được kiểm chứng dựa trên cơ sở lý thuyết ở Chương 4 và thực nghiệm ở Chương 5.

Luận án giải quyết bài toán kiểm chứng tính nhất quán trong tái cấu trúc bao phủ trên các ràng buộc về bất biến của lớp và hành vi của hệ thống phần mềm. Một cách tuần tự, ở Chương 3, luận án đề xuất phương pháp kiểm chứng tính bất biến, Chương 4 luận án tiếp tục giải quyết bài toán kiểm chứng về sự bảo toàn hành vi. Và để hoàn thiện hơn nữa cơ sở lý thuyết đã đề xuất, Chương 5 luận án xây dựng công cụ hỗ trợ

kiểm chứng một cách tự động. Như vậy, luận án đã xem xét bài toán kiểm chứng tính nhất quán trong tái cấu trúc một cách tương đối trọn vẹn, từ các đóng góp về lý thuyết đến các minh chứng bằng thực nghiệm và xuyên suốt từ giai đoạn thiết kế đến cài đặt trong vòng đời phát triển phần mềm. Có thể nói, các đóng góp của luận án có ý nghĩa trong việc bổ sung và hoàn thiện các kỹ thuật sử dụng trong lĩnh vực cải tiến chất lượng phần mềm.

Các kết quả của luận án đã công bố trong các công trình khoa học được đăng tải trên các tạp chí, hội nghị chuyên ngành trong nước và quốc tế có phản biện.

6.2 Hướng phát triển

Một số hướng nghiên cứu tiếp theo của luận án có thể được thực hiện là:

- Nghiên cứu bài toán kiểm chứng tính bất biến của lớp trong tái cấu trúc hệ thống phần mềm đối với nhiều tiến trình tái cấu trúc đa dạng hơn, xem xét kết hợp các tiến trình tái cấu trúc để tạo thành chiến lược tái cấu trúc có ý nghĩa hơn với người dùng. Xây dựng công cụ hỗ trợ kiểm chứng tính bất biến trong tái cấu trúc.
- Nghiên cứu bài toán kiểm chứng sự bảo toàn hành vi với nhiều mô hình hơn (biểu đồ ca sử dụng, biểu đồ trạng thái, biểu đồ hoạt động v.v.). Thực nghiệm phương pháp đề xuất với nhiều phân lớp bài toán đa dạng. Từ đó, có được những kết quả thực nghiệm tin cậy.
- Hoàn thiện và bổ sung các chức năng đối với công cụ CVT như sản sinh các biểu thức OCL một cách tự động từ mô hình.
- Xây dựng một bộ công cụ tích hợp kiểm chứng đầy đủ các bài toán về bảo toàn bất biến và bảo toàn hành vi trong tái cấu trúc.

DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN

1. Thi Huong Dao, Hong Anh Le, and Ninh Thuan Truong. *An approach to analyzing execution preservation in Java program refactoring.* International Conference on Context-Aware Systems and Applications, pp.101-110, Springer, 2016 (Scopus).

2. Hong Anh Le, Thi Huong Dao, and Ninh Thuan Truong. *A Formal Approach to Checking Consistency in Software Refactoring.* Mobile Networks and Application, Vol.22.2, pp.356–366, Springer, 2017. (SCIE IF 3.259)

3. Thi Huong Dao, Thanh Binh Trinh, and Ninh Thuan Truong. *A Tool Support for Checking Consistency in Model Refactoring.* In Proc. of the 9th International Conf. on Knowledge and Systems Engineering, pp. 108-113, Springer LNCS, 2017 (Scopus).

4. Thi Huong Dao, Xuan Truong Nguyen, and Ninh Thuan Truong. *Preservation of Class Invariants in Refactoring UML Models.* (Submitted).