ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Chu Thị Minh Huệ

KIỂM THỬ DỰA TRÊN MÔ HÌNH VỚI CÁCH TIẾP CẬN MÔ HÌNH HÓA CHUYÊN BIỆT MIỀN

TÓM TẮT LUẬN ÁN TIẾN SỸ CÔNG NGHỆ THÔNG TIN

Hà Nội - 2018

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Chu Thị Minh Huệ

KIỂM THỬ DỰA TRÊN MÔ HÌNH VỚI CÁCH TIẾP CẬN MÔ HÌNH HÓA CHUYÊN BIỆT MIỀN

Chuyên ngành: Kỹ thuật Phần mềm
Mã số: 9480103.01

TÓM TẮT LUẬN ÁN TIẾN SỸ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:
1. PGS.TS. Nguyễn Ngọc Bình
2. TS. Đặng Đức Hạnh

Hà Nội - 2018

Mục lục

1 Giới thiệu **1**
- 1.1 Đặt vấn đề . . . 1
- 1.2 Mục tiêu nghiên cứu và các đóng góp chính của luận án . . . 2
- 1.3 Bố cục của luận án . . . 2

2 Kiến thức cơ sở **3**
- 2.1 Kiểm thử dựa trên ca sử dụng . . . 3
- 2.2 Mô hình hóa chuyên biệt miền . . . 4
- 2.3 Chuyển đổi mô hình . . . 4
- 2.4 Ngôn ngữ ràng buộc đối tượng OCL . . . 4

3 Đặc tả ca sử dụng theo hướng mô hình hóa chuyên biệt miền **5**
- 3.1 Giới thiệu . . . 5
- 3.2 Xác định miền cho ngữ cảnh đặc tả ca sử dụng . . . 5
- 3.3 Cú pháp của USL . . . 6
  - 3.3.1 Cú pháp trừu tượng của USL . . . 6
  - 3.3.2 Các luật hợp lệ trên siêu mô hình của USL . . . 7
  - 3.3.3 Cú pháp cụ thể của USL . . . 8
- 3.4 Ngữ nghĩa hình thức của mô hình USL . . . 8
- 3.5 Chuyển đổi mô hình USL . . . 12
- 3.6 Kết chương . . . 12

4 Phương pháp sinh tự động các ca kiểm thử từ mô hình ca sử dụng **13**
- 4.1 Giới thiệu . . . 13
- 4.2 Tổng quan kỹ thuật đề xuất . . . 13
- 4.3 Ngôn ngữ đặc tả các ca kiểm thử TCSL . . . 13
  - 4.3.1 Xác định miền cho ngữ cảnh đặc tả ca kiểm thử chức năng . . . 13
  - 4.3.2 Định nghĩa siêu mô hình TCSL . . . 14
- 4.4 Chuyển đổi mô hình từ USL sang TCSL . . . 15
  - 4.4.1 Xác định tiêu chí phủ . . . 15
  - 4.4.2 Sinh các kịch bản ca sử dụng và các ràng buộc . . . 15
  - 4.4.3 Sinh các bộ dữ liệu đầu vào kiểm thử . . . 16
  - 4.4.4 Sinh mô hình TCSL . . . 16
- 4.5 Tổng kết chương . . . 18

5 THỰC NGHIỆM VÀ ĐÁNH GIÁ **20**
- 5.1 Giới thiệu . . . 20
- 5.2 Công cụ hỗ trợ USL . . . 20
- 5.3 Đánh giá . . . 20
  - 5.3.1 Đánh giá ngôn ngữ USL . . . 20
  - 5.3.2 Đánh giá phương pháp sinh các ca kiểm thử USLTG . . . 21

5.4 Kết chương 22

6 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN **23**

6.1 Các đóng góp của luận án 23

6.2 Hướng phát triển 23

Danh mục các công trình khoa học **25**

Chương 1

Giới thiệu

1.1 Đặt vấn đề

Trong quy trình phát triển phần mềm, các yêu cầu chức năng (*functional software requirements*) của phần mềm là đầu vào để xây dựng các ca kiểm thử chức năng, các mô hình cấu trúc (biểu đồ lớp), các mô hình hành vi (biểu đồ tuần tự, biểu đồ hoạt động) cho hệ thống, v.v. Trong thực tế, các yêu cầu phần mềm thường hay thay đổi trong suốt quy trình phát triển phần mềm. Khi các yêu cầu phần mềm thay đổi, các mô hình và các ca kiểm thử liên quan phải được xây dựng và thực thi lại (kiểm thử hồi quy). Vì vậy, những nỗ lực được yêu cầu để xác định, bảo trì, và thực thi các ca kiểm thử khi các yêu cầu thay đổi là rất lớn. Đối với hoạt động kiểm thử phần mềm, giải pháp sinh và thực thi các ca kiểm thử một cách tự động giúp tiết kiệm rất nhiều thời gian và các nỗ lực cũng như giảm được số lượng các lỗi và các sai sót trong hoạt động kiểm thử phần mềm. Do đó, đã có nhiều nghiên cứu đề xuất các giải pháp tăng tính tự động trong phát triển phần mềm cũng như hoạt động kiểm thử phần mềm.

Các công trình nghiên cứu về sinh tự động các ca kiểm thử chức năng từ các ca sử dụng rất đa dạng, phức tạp với với nhiều cách tiếp cận khác nhau. Trong khuôn khổ nghiên cứu, luận án nghiên cứu phương pháp kiểm thử kiểm thử dựa trên mô hình với cách tiếp cận mô hình hóa chuyên biệt miền để giải quyết bài toán sinh tự động các ca kiểm thử chức năng từ các ca sử dụng. Trong đó luận án chỉ tập trung vào hai pha chính trong phương pháp kiểm thử dựa trên mô hình là mô hình hóa và sinh tự động các ca kiểm thử, cụ thể như sau:

(i) Đặc tả mô hình ca sử dụng đủ chính xác để sinh tự động các chế tác phần mềm khác nhau trong đó có sinh các ca kiểm thử.

(ii) Bài toán sinh tự động các ca kiểm thử từ các ca sử dụng.

Vì vậy, mục tiêu của luận án hướng đến tập trung *đề xuất hai ngôn ngữ đặc tả chuyên biệt miền cho miền đặc tả ca sử dụng và miền đặc tả ca kiểm thử* và **phương pháp chuyển tự động từ các mô hình ca sử dụng vào trong mô hình ca kiểm thử chức năng** trong các ngôn ngữ mô hình hóa chuyên biệt miền đã đề xuất. Bài toán này có các đầu vào là các biểu đồ ca sử dụng trong UML, các mô tả ca sử dụng trong ngôn ngữ tự nhiên, và các mô hình lớp đặc tả các khái niệm miền của hệ thống.

Đối tượng nghiên cứu của luận án là kỹ thuật kiểm thử dựa trên mô hình, kỹ thuật xây dựng ngôn ngữ mô hình hóa chuyên biệt miền, và phương pháp sinh tự động các ca kiểm thử chức năng từ các ca sử dụng. Cụ thể, luận án quan tâm đến các biểu đồ ca sử dụng, các mô tả ca sử dụng, các mô hình lớp đặc tả các khái niệm miền của hệ thống, và các mô tả ca kiểm thử chức năng. Các mô hình lớp và mô hình ca sử dụng được xem xét tại mức đặc tả yêu cầu phần mềm và các ca kiểm thử chức năng được xem xét xác định từ các yêu cầu chức năng của phần mềm trong tài liệu đặc tả yêu cầu phần mềm.

1.2 Mục tiêu nghiên cứu và các đóng góp chính của luận án

Mục tiêu của luận án này là đặt ca sử dụng vào trong ngữ cảnh của quy trình kiểm thử dựa trên mô hình. Điều này cho phép tự động hóa hoạt động kiểm thử chức năng mà đầu vào là các yêu cầu chức năng của phần mềm.

Thứ nhất, luận án đề xuất một *phương pháp đặc tả ca sử dụng với cách tiếp cận mô hình hóa chuyên biệt miền.*

Thứ hai, luận án đề xuất một *phương pháp đặc tả ca kiểm thử.*

Thứ ba, luận án đề xuất một *phương pháp chuyển tự động từ các mô hình ca sử dụng trong USL sang mô hình đặc tả ca kiểm thử TCSL.*

Cuối cùng, luận án *xây dựng bộ công cụ hỗ trợ USL.* Công cụ này cho phép tích hợp ca sử dụng vào trong phương pháp phát triển hướng mô hình. Để minh chứng khả năng ứng dụng USL vào trong thực tế, luận án cũng trình bày các kết quả khi áp dụng USL cho một số ca sử dụng. Ngoài ra, luận án cũng đưa ra các đánh giá, so sánh phương pháp đặc tả ca sử dụng và phương pháp sinh các ca kiểm thử với các nghiên cứu khác liên quan.

Các kết quả nghiên cứu trên của luận án nhằm xây dựng một phương pháp hoàn chỉnh để sinh tự động các ca kiểm thử chức năng từ các ca sử dụng bằng phương pháp kiểm thử dựa trên mô hình với cách tiếp cận mô hình hóa chuyên biệt miền. Từ đó nghiên cứu hướng đến một phương pháp hoàn chỉnh cho phép tích hợp ca sử dụng vào trong phương pháp phát triển hướng mô hình.

1.3 Bố cục của luận án

Luận án bao gồm sáu chương. Trong đó, Chương 1 trình bày về các kiến thức nền được sử dụng trong luận án. Chương 2 trình bày một cách tóm tắt kiến thức cơ sở sử dụng trong các chương tiếp theo. Chương 3 đề xuất một ngôn ngữ đặc tả chuyên biệt miền cho miền đặc tả các ca sử dụng tên là USL. Chương 4 trình bày phương pháp sinh tự động các ca kiểm thử chức năng từ ca sử dụng. Công cụ hỗ trợ USL được trình bày ở Chương 5. Cuối cùng, Chương 6 kết luận và đưa ra các hướng nghiên cứu tiếp theo của luận án.

Chương 2

Kiến thức cơ sở

2.1 Kiểm thử dựa trên ca sử dụng

Kiểm thử (Testing) là một quy trình thực hiện một chương trình với ý định tìm kiếm các lỗi. Kiểm thử phần mềm bao gồm việc kiểm chứng động các hành vi của một chương trình trên một tập hữu hạn các ca kiểm thử (*test case*). Các ca kiểm thử được lựa chọn phù hợp từ các miền thực thi thường là vô hạn để có được hành vi được mong đợi.

Ca kiểm thử (Test case) là một tập của các dữ liệu kiểm thử (*test data*), các điều kiện thực hiện (*pre-condition*), các bước kiểm thử (*test steps*), và các kết quả đầu ra mong đợi (*expected output*) được phát triển cho một kịch bản kiểm thử (*test scenario*) cụ thể để kiểm chứng tuân thủ một yêu cầu xác định. Một ca kiểm thử định nghĩa một thử nghiệm đơn lẻ sẽ được thực hiện để đạt được mục tiêu kiểm thử phần mềm cụ thể, chẳng hạn như đi qua một đường thực thi của chương trình cụ thể hoặc kiểm chứng tuân thủ một yêu cầu cụ thể.

Dữ liệu kiểm thử (Test data) là tập các giá trị thực (thỏa mãn tiêu chuẩn bao phủ dữ liệu đã chọn) được xác định chỉ rõ là đầu vào để thực hiện các ca kiểm thử trong quá trình kiểm thử.

Kịch bản kiểm thử (Test scenario) là một ca kiểm thử trừu tượng và nó thường bao gồm nhiều ca kiểm thử liên quan nhau. Mục đích của kịch bản kiểm thử là kiểm tra việc thực hiện chức năng từ đầu đến cuối của một chức năng phần mềm và đảm bảo luồng logic đang hoạt động là đúng. Với mỗi kịch bản kiểm thử, kiểm thử viên có thể xác định được một hoặc nhiều bộ dữ liệu kiểm thử thỏa mãn kịch bản kiểm thử. Một ca kiểm thử là sự kết hợp của một kịch bản kiểm thử với một bộ dữ liệu kiểm thử thỏa mãn kịch bản.

Kiểm thử dựa trên mô hình (*Model-Based Testing - MBT*) là một kỹ thuật kiểm thử với mục đích để sinh các ca kiểm thử tự động từ mô hình mà đặc tả các khía cạnh liên quan của hành vi của hệ thống cần kiểm thử (*System Under Testing - SUT*).

Tiêu chuẩn bao phủ (Test coverage criteria) là một tập các quy tắc mà hướng dẫn quyết định các yếu tố thích hợp cần được đề cập để được phủ để tạo ra sự đầy đủ cho các ca kiểm thử được thiết kế. Mục đích để đánh giá mức độ hiệu quả của các ca kiểm thử, đo phần trăm độ bao phủ của đặc tả hoặc chương trình của các ca kiểm thử so với yêu cầu phần mềm, thông qua kiểm thử để loại trừ sai sót và tăng chất lượng phần mềm. Luận án trình bày một tiêu chí phủ kiểm thử mà luận án áp dụng để sinh các ca kiểm thử trong Chương 4.

Tiêu chí phủ đường hoạt động. Kundu và Samanta đã đề xuất một tiêu chí phủ kiểm thử tên là *phủ đường hoạt động* (*activity path coverage*). Tiêu chí này mục đích để sử dụng cho kiểm thử cả vòng lặp và kiểm thử đồng thời giữa các hành động trong đồ thị hoạt động.

Ca sử dụng (use case). Ca sử dụng được sử dụng rộng rãi như là một phương tiện để nắm bắt các yêu cầu chức năng của phần mềm. Một ca sử dụng là một mô tả của một chức năng sử dụng cụ thể của hệ thống bởi một tác nhân (*actor*). Mỗi ca sử dụng mô tả các tương tác của tác nhân với hệ thống để đạt được một nhiệm vụ cụ thể (hoặc ít nhất cung cấp một cái gì đó có giá trị cho người dùng). Các ca sử dụng là chế tác trung tâm trong phát triển phần mềm. Chúng được sử dụng như là đầu vào để xây dựng các

chế tác khác nhau của phần mềm, như các mô hình hành vi, mô hình cấu trúc, và các ca kiểm thử hệ thống.

Mô hình ca sử dụng (use case model). Một mô hình ca sử dụng trong UML thường được thể hiện bằng các biểu đồ ca sử dụng liên kết lỏng lẻo với các mô tả của các ca sử dụng trong ngôn ngữ tự nhiên được trình bày trong một mẫu có cấu trúc. Trong đó các biểu đồ ca sử dụng cung cấp tổng quan về các ca sử dụng, người dùng của hệ thống và các mối quan hệ giữa các ca sử dụng và người dùng. Các mô tả của mỗi ca sử dụng mô tả các chuỗi tương tác giữa môi trường và hệ thống. Mô tả ca sử dụng trong ngôn ngữ tự nhiên cho phép người dùng cũng như các bên tham gia phát triển phần mềm dễ dàng hiểu được được yêu cầu của hệ thống.

2.2 Mô hình hóa chuyên biệt miền

Ngôn ngữ chuyên biệt miền (*Domain-Specific Language* - DSL) là một ngôn ngữ chương trình hoặc ngôn ngữ đặc tả thực thi, bằng cách tích hợp các khái niệm trừu tượng của tri thức miền vào trong ngôn ngữ dưới dạng các ký hiệu có tính biểu cảm cao. DSL tăng mức độ trừu tượng bằng cách sử dụng các khái niệm quen thuộc với các chuyên gia miền và thường được giới hạn trong một miền vấn đề cụ thể nào đó.

Ngôn ngữ mô hình hóa chuyên biệt miền (*Domain-Specific Modeling Language* - DSML) là một ngôn ngữ chuyên biệt miền cụ thể. DSML được sử dụng để xây dựng các mô hình đồ họa cho một hệ thống phần mềm.

Mô hình hóa chuyên biệt miền (*Domain-Specific Modelling* - DSM) là sử dụng DSML để tạo các mô hình, và sinh mã từ các mô hình với một bộ sinh mã.

Phát triển hướng mô hình (*Model-Driven Development* - MDD) là một mô hình phát triển mà sử dụng các mô hình như là các chế tác chính trong các quy trình phát triển phần mềm. Thông thường, trong MDD triển khai được sinh tự động hoặc bán tự động từ các mô hình.

2.3 Chuyển đổi mô hình

Một phép chuyển đổi mô hình là một chương trình để tạo tự động các mô hình hoặc văn bản đầu ra từ các mô hình đầu vào. Chuyển đổi mô hình có ba dạng: Chuyển mô hình sang mô hình (Model to Model - M2M), mô hình sang văn bản (Model to Text - M2T), và văn bản sang mô hình (Text to Model - T2M).

2.4 Ngôn ngữ ràng buộc đối tượng OCL

Ngôn ngữ ràng buộc đối tượng OCL (*Object Constraint Language*) ra đời nhằm mục đích khắc phục các hạn chế của UML trong việc biểu diễn chính xác các khía cạnh chi tiết của một thiết kế hệ thống. OCL được phát triển lần đầu vào năm 1995 bởi IBM và được tích hợp vào UML năm 1997.

Đầu tiên, OCL chỉ được sử dụng như là một ngôn ngữ ràng buộc cho UML, tuy nhiên phạm vi của nó đã được mở rộng nhanh chóng và bây giờ OCL trở thành một thành phần quan trọng trong bất kỳ kỹ nghệ hướng mô hình MDE (*Model-Driven Engineering*). OCL được sử dụng như là ngôn ngữ mặc định cho trình diễn tất cả các truy vấn mô hình hoặc siêu mô hình, thực hiện và đặc tả yêu cầu. OCL cũng thường được sử dụng để trình diễn các chuyển mô hình (như là một phần của các mô hình nguồn và mô hình đích của các luật chuyển mô hình), các luật hợp lệ (như là một phần của định nghĩa của ngôn ngữ chuyên biệt miền mới), hoặc các mẫu sinh mã nguồn (như một cách để mô tả các mẫu và các luật sinh).

Chương 3

Đặc tả ca sử dụng theo hướng mô hình hóa chuyên biệt miền

3.1 Giới thiệu

Ca sử dụng là một chế tác của phần mềm mà được sử dụng chủ yếu để nắm bắt và cấu trúc các yêu cầu chức năng của phần mềm. Mô hình ca sử dụng được đặc tả chủ yếu bằng một biểu đồ ca sử dụng và các mô tả dạng văn bản được cấu trúc lỏng lẻo. Lợi ích chính của đặc tả ca sử dụng trong ngôn ngữ tự nhiên là dễ dàng cho các bên liên quan không có kỹ thuật có thể hiểu được. Tuy nhiên, các mô hình ca sử dụng được trình bày trong dạng này thường chứa các phần thông tin mập mờ và không chính xác. Điều này ngăn cản các mô hình ca sử dụng được sử dụng trực tiếp trong các cách tiếp cận hướng mô hình, như một nguồn chuyển để cung cấp các mô hình cấu trúc, các mô hình hành vi, và các ca kiểm thử. Trong chương này, luận án tập trung nghiên cứu xây dựng một ngôn ngữ mô hình hóa chuyên biệt cho miền [1] đặc tả các thông tin được mô tả trong ca sử dụng. Luận án khắc phục những hạn chế đã được nêu ở trên trong các nghiên cứu khác. Cú pháp trừu tượng của USL được định nghĩa bằng cách mở rộng siêu mô hình của biểu đồ ca sử dụng và biểu đồ hoạt động trong UML.

3.2 Xác định miền cho ngữ cảnh đặc tả ca sử dụng

Bảng 3.1 là một mô tả của ca sử dụng *Lend book*, mô tả này có định dạng theo cấu trúc mẫu mô tả ca sử dụng thông thường. Một mẫu mô tả ca sử dụng thường gồm hai phần: các phần tử thông tin tổng quan và mô tả chi tiết của các luồng. Phần thông tin tổng quan bao gồm: tên ca sử dụng (*use case name*), mô tả tóm lược (*brief description*) mô tả tóm lược về ca sử dụng, các tác nhân (*actors*) mô tả các tác nhân thực hiện ca sử dụng, tiền điều kiện (*pre-condition*) và hậu điều kiện (*postcondition*) mô tả tiền và hậu điều kiện của ca sử dụng, kích hoạt (*trigger*) mô tả sự kiện kích hoạt ca sử dụng, và yêu cầu đặc biệt (*special requirement*) mô tả các yêu cầu phi chức năng của ca sử dụng. Phần thứ hai của ca sử dụng là thông tin mô tả hai loại luồng: luồng chính (*basic flow*) và các luồng thay thế (*alternative flows*). Nội dung luồng chính mô tả các bước thực hiện ca sử dụng trong điều kiện thông thường khi ca sử dụng thực hiện. Một ca sử dụng chỉ có duy nhất một luồng chính. Các luồng thay thế mô tả các hành vi tùy chọn hoặc ngoại lệ cũng như các hành vi thông thường khác nhau. Luồng chính và các luồng thay thế thường được cấu trúc trong các bước (*steps*) hoặc các luồng con (*subflows*). Tuy nhiên, chúng ta có thể làm mịn các luồng để chỉ chứa một luồng chính và một số luồng thay thế.

Mỗi một bước trong ca sử dụng mô tả một hoặc nhiều hành động được thực hiện bởi hệ thống hoặc tác nhân. Mỗi một bước trong các luồng chứa các hành động được thực hiện bởi hệ thống hoặc các tác nhân.

Các hành động được mô tả trong các bước có thể được chia thành chín loại hành động như sau:

Actor-Input là một hành động của tác nhân để gửi dữ liệu tới hệ thống.

1. miền công việc đặc tả ca sử dụng

Bảng 3.1: Mô tả của ca sử dụng *Lend book*

Use case name: *Lend Book* **Brief description:** The Librarian processes a book loan. **Actors:** Librarian. **Precondition:** The librarian has logged into the system successful. **Postcondition:** If the use case successfully ends, the book loan is saved and a complete message is shown. In the other case, the system displays an error message. **Trigger:** The Librarian requests a book-loan process. **Special requirement:** There is no special requirement.
Basic flow
1. The Librarian selects the *Lend Book* function. 2. The system shows the Lend-book window, gets the current date and assigns it to the book-loan date. 3. The Librarian enters a book copy id. 4. The system checks the book copy id. If it is invalid, it goes to step 4a.1 5. The Librarian enters a borrower id. 6. The system validates the borrower id. If it is invalid, it goes to step 6a.1 7. The Librarian clicks the save-book-loan button. 8. The system validates the conditions to lend book. If it is invalid, the system goes to step 8a.1 9. The system saves the book loan record, then executing two steps 10 and 11 concurrently. 10. The system shows a complete message. 11. The system prints the borrowing bill.
Alternate flows
E1. request searched book 1. The Librarian selects the search function after step 4a.1. 2. The system executes the extending use case Search book. 4a. The book copy id is invalid 1. The system shows an error message, then going to step 3. 6a. The Borrower id is invalid 1. The system shows an error message, then going to step 5. 8a. The lending condition is invalid 1. The system shows an error message. 2. The system ends the use case.

Actor-Request là hành động của tác nhân gửi yêu cầu tới hệ thống.

System-Display là một hành động của hệ thống mà hệ thống thực hiện các hoạt động với giao diện người dùng.

System-Operation là một hành động hệ thống để thẩm định một yêu cầu và dữ liệu, hoặc xử lý và tính toán dữ liệu.

System-State là hành động của hệ thống để truy vấn hoặc cập nhật các trạng thái bên trong của hệ thống.

System-Output là hành động của hệ thống để gửi đầu ra cho các tác nhân.

System-Request là hành động của hệ thống để gửi yêu cầu tới tác nhân phụ (secondary actor).

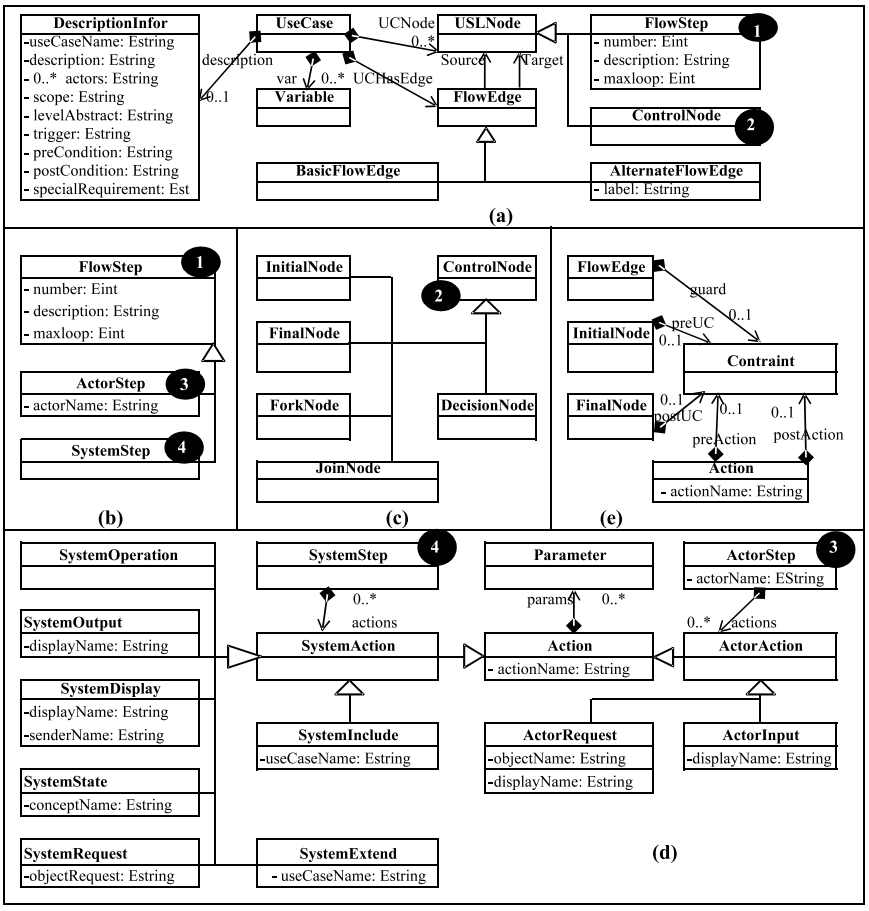
System-Include là hành động hệ thống để gọi một ca sử dụng khác có quan hệ *«include»* với ca sử dụng.

System-Extend là hành động của hệ thống để gọi một ca sử dụng khác có quan hệ *«extend»* với ca sử dụng.

3.3 Cú pháp của USL

3.3.1 Cú pháp trừu tượng của USL

Hình 3.1 thể hiện siêu mô hình của USL. Để ngắn gọn, luận án chia siêu mô hình này vào bốn khối (a), (b), (c), và (d) và đánh nhãn số thứ tự trên các khái niệm để chia khối. Hình 3.1-a có nghĩa là khối (a)) trình bày các khái niệm ở mức trên. Hình 3.1-b



Hình 3.1: Siêu mô hình USL.

trình bày các phân cấp khái niệm `FlowStep`. Hình 3.1-c trình bày các phân cấp khái niệm `ControlNode`. Hình 3.1-d trình bày các phân cấp khái niệm `Action` và cách nó được liên kết với `FlowStep`. Hình 3.1-e trình bày khái niệm `Constraint` và cách nó được sử dụng để đặc tả các ràng buộc của `Action`, `InitialNode`, `FinalNode`, và `FlowEdge`.

3.3.2 Các luật hợp lệ trên siêu mô hình của USL

Để đảm bảo tính đúng đắn của các mô hình USL tạo ra, luận án định nghĩa một tập cách luật hợp lệ OCL như các hạn chế trên siêu mô hình của USL. Các luật này cho phép kiểm chứng lại tính tính đắn của các mô hình USL khi người mô hình hóa xây dựng chúng. Những luật này được định nghĩa trong ngữ cảnh của khái niệm `UseCase` như danh sách dưới đây.

Luật 1. Một mô hình USL có duy nhất một `InitialNode`.

Luật 2. Một mô hình USL có ít nhất một `FinalNode`.

Luật 3. Một mô hình USL có ít nhất một `FlowStep`.

Luật 4. Một `InitialNode` có một `BasicFlowEdge` đi ra và không có bất kỳ `FlowEdge` đi vào.

Luật 5. Một `FinalNode` có một `FlowEdge` đi vào và không có bất kỳ `FlowEdge` đi ra.

Luật 6. Một `DecisionNode` có một `FlowEdge` đi vào và ít nhất hai `FlowEdge` đi ra.

Luật 7. Một `ForkNode` có duy nhất một `FlowEdge` đi vào và ít nhất `FlowEdge` đi ra.

Luật 8. Một `JoinNode` có ít nhất hai `FlowEdge` đi vào và duy nhất một `FlowEdge` đi ra.

Luật 9. Một `SystemStep` hoặc `ActorStep` có ít nhất một `FlowEdge` đi vào và chỉ có một `FlowEdge` đi ra.

Luật 10. Một mô hình USL là hợp lệ nếu các `FlowEdge` mà kết nối các `USLNode` của mô hình là hợp lệ, tức là, kiểu và nhãn của các `FlowEdge` được định nghĩa hợp lệ.

Luật 11. Thuộc tính `number` của mỗi `FlowStep` trong **Basic flow** là duy nhất.

Luật 12. Thuộc tính `number` của mỗi `FlowStep` trong một **Alternate flow** là duy nhất:

3.3.3 Cú pháp cụ thể của USL

Để giúp người dùng có thể tạo được các mô hình USL một cách trực quan và dễ dàng, luận án đề xuất xây dựng cú pháp thể cho USL với các ký hiệu (notation) như được thể hiện trong Hình 3.2.

3.4 Ngữ nghĩa hình thức của mô hình USL

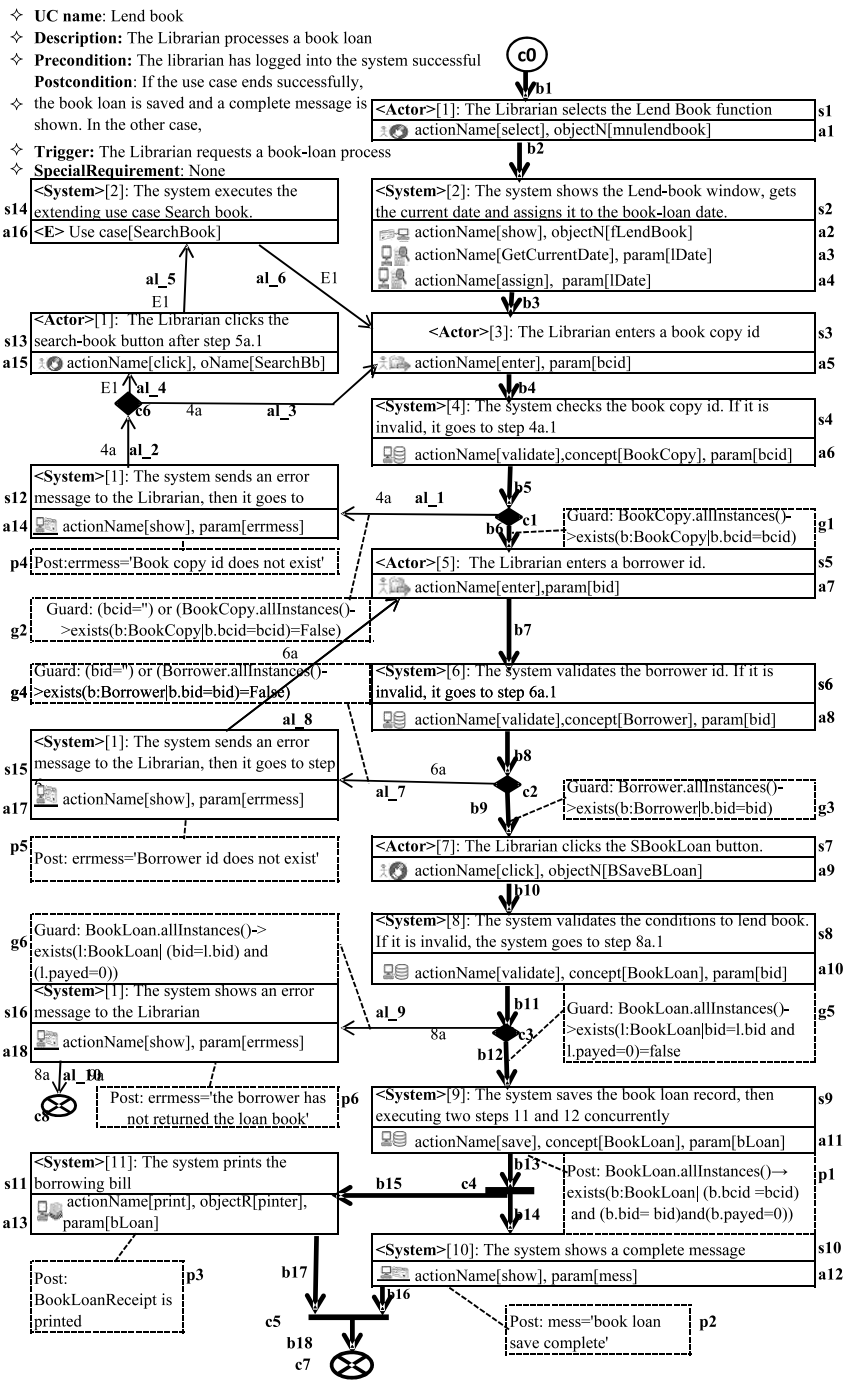
Phần này, luận án cung cấp ngữ nghĩa hình thức cho các mô hình USL. Mục đích là để cung cấp một ngữ nghĩa thực thi cho ngôn ngữ USL bằng cách sử dụng một hệ thống chuyển trạng thái được gán nhãn. Ngữ nghĩa thực thi của ngôn ngữ USL sẽ là cơ sở lý thuyết để luận án xây dựng các thuật toán sinh các ca kiểm thử tự động từ mô hình USL.

Định nghĩa 3.1 (Mô hình USL) Một mô hình USL của một ca sử dụng là một bộ $D = \langle D_C, A, E, C, Act, F_e, F_c, F_f \rangle$ với:

- D_C là một biểu đồ lớp trình diễn các khái niệm miền của hệ thống;
- $A = A_{cNode} \cup A_f$ là tập của các nút (`USLNode`);
- $E = E_b \cup E_a$ là tập của các cạnh (`FlowEdge`);
- $C = G \cup \{c_{preUC}\} \cup C_{postUC} \cup C_{preA} \cup C_{postA}$ là tập các ràng buộc (`Constraint`);
- $Act = Act_a \cup Act_s$ là tập của hành động (`Action`);
- $F_e \subseteq \{A \times G \times E \times A\}$ là mối quan hệ giữa các nút, điều kiện gác G, và các cạnh;
- $F_c \subseteq \{C_{preA} \times Act \times C_{postA}\}$ là các quan hệ giữa các hành động với các tiền và hậu điều kiện của các hành động;
- $F_f \subseteq \{N_f \times C_{postUC}\}$ là mối quan hệ giữa các nút kết thúc (`FinalNode`) và hậu điều kiện của ca sử dụng.

Trong đó:

- $A_{cNode} = N_I \cup N_f \cup N_d \cup N_j \cup N_k$, với:
  - $N_I = \{c_{it}\}$ là tập các nút bắt đầu (*InitialNode*),
  - $N_f = \{c_{f1}, \ldots, c_{fn}\}$ là tập các nút kết thúc (*FinalNode*),
  - $N_d = \{c_{d1}, \ldots, c_{dn}\}$ là tập các nút quyết định (*DecisionNode*),
  - $N_j = \{c_{j1}, \ldots, c_{jn}\}$ là tập các nút nối (*JoinNode*),
  - $N_k = \{c_{k1}, \ldots, c_{kn}\}$ là tập các nút chia (*ForkNode*).
- $A_f = A_a \cup A_s$ là tập các bước (*FlowStep*), khi đó:
  - $A_a = \{s_{a1}, \ldots, s_{an}\} (\forall\, s_a \in A_a, s_a = (s_d, Ac_a))$ là tập các bước của tác nhân (*ActorStep*),



Hình 3.2: Đặc tả ca sử dụng *Lend Book* bằng một mô hình USL.

+ $A_s = \{s_{s1}, \ldots, s_{sn}\}(\forall\, s_s \in A_s, s_s = (s_d, Ac_s))$ là tập các bước của hệ thống (*SystemStep*), trong đó:
  • $s_d \in$ `String` là mô tả của bước s_a và s_s,
  • $Ac_a =< a_{a1}, \ldots, a_{an} > (Ac_a \subset Act_a)$ là tập các hành động của tác nhân (`ActorAction`) có thứ tự trong s_a,
  • $Ac_s =< a_{s1}, \ldots, a_{sn} > (Ac_s \subset Act_s)$ là tập các hành động của hệ thống (`SystemAction`) có thứ tự trong s_s,
  • $\texttt{actions}(\texttt{s}_\texttt{a}) = Ac_a$ và $actions(s_s) = Ac_s$ là một ánh xạ từ s_a và s_s tới Ac_a và Ac_s,

• $firstAct(s_a) = a_{a1}, firstAct(s_s) = a_{s1}, lastAct(s_a) = a_{an}$, và $lastAct(s_s) = a_{sn}$ là bốn ánh xạ từ s_a và s_s tới a_{a1}, a_{s1}, a_{an}, và a_{sn},
• $\texttt{Actions(D)} = \bigcup_{s \in A_f} actions(s) \equiv Act$.

+ $\forall\, s_i, s_j \in A_f, actions(s_i) \cap actions(s_j) = \varnothing$.

- $E_b = \{e_{b1}, \ldots, e_{bn}\}$ là tập các cạnh BasicFlowEdge;
- $E_a = \{e_{a1}, \ldots, e_{an}\}$ là tập các cạnh AlternateFlowEdge;
- $G = \{g_{c1}, \ldots, g_{cn}\}$ là tập các điều kiện gác trên các cạnh;
- c_{preUC} là tiền điều kiện của ca sử dụng;
- $C_{postUC} = \{p_{u1} \ldots p_{un}\}$ là hậu điều kiện của ca sử dụng;
- $C_{preA} = \{p_{e1}, \ldots, p_{en}\}$ là tập các tiền điều kiện của các hành động;
- $C_{postA} = \{p_{o1}, \ldots, p_{on}\}$ là tập các hậu điều kiện của các hành động;
- $Act_a = Act_{ai} \cup Act_{ar}$ là tập các hành động của tác nhân, với:
  + Act_{ai} là tập các hành động `ActorInput`,
  + Act_{ar} là tập các hành động `ActorRequest`;
- $Act_s = Act_{sd} \cup Act_{sp} \cup Act_{ss} \cup Act_{so} \cup Act_{sr} \cup Act_{si} \cup Act_{se}$ là tập các hành động của hệ thống, với:
  + Act_{sd} là tập các hành động `SystemDisplay`,
  + Act_{sp} là tập các hành động `SystemOperation`,
  + Act_{ss} là tập các hành động `SystemState`,
  + Act_{so} là tập các hành động `SystemOutput`,
  + Act_{sr} là tập các hành động `SystemRequest`,
  + Act_{si} là tập các hành động `SystemInclude`,
  + Act_{se} là tập các hành động `SystemExtend`.

Trên mô hình D, luận án định nghĩa một số hàm như trong Bảng 3.2.

Định nghĩa 3.2 (Thực thi mô hình USL) Cho một mô hình USL $D = \langle D_C, A, E, C, Act, F_e, F_c, F_f \rangle$, một `LTS` mô tả thực thi của D là một bộ năm $\langle \Sigma(\mathcal{V}), \mathbb{P}(\mathcal{G} \times \mathcal{A} \times \mathcal{P}), \mathcal{T}, \alpha_{init}, \mathcal{F} \rangle$ trong đó:

- $\mathcal{V}$ là một tập hữu hạn của các biến mà kiểu của nó là các kiểu cơ bản và các lớp của D_C;
- $\Sigma(\mathcal{V})$ là tập của các trạng thái (α), mỗi trạng thái là một tập của các gán giá trị tới tập con của các biến trong $\mathcal{V}$;
- $\mathcal{P} \subseteq C_{postA} \cup C_{postUC}$ là tập của các ràng buộc mà là các hậu điều kiện của D;
- $\mathcal{A} = A_{cNode} \cup Act$ là tập của các hành động;
- $\mathcal{G} \subseteq G \cup C_{preUC} \cup C_{preA}$ là tập các điều kiện gác của các chuyển;
- $\alpha_{init} \in \Sigma(\mathcal{V})$ là trạng thái bắt đầu;
- $\mathcal{F} \subset \Sigma(\mathcal{V})$ là tập các trạng thái kết thúc;
- $\mathcal{T} \subseteq \Sigma(\mathcal{V}) \times \mathbb{P}(\mathcal{G} \times \mathcal{A} \times \mathcal{P}) \times \Sigma(\mathcal{V})$ là mối quan hệ chuyển được định nghĩa như sau:
  Một chuyển $t = (\alpha, (g, a, r), \alpha') \in \mathcal{T}$, hoặc ký hiệu là $\alpha \xrightarrow{g|a|r} \alpha'$, với:
  + $a \in \mathcal{A}$ là hành động để xảy ra chuyển t,
  + $\alpha, \alpha' \in \Sigma(\mathcal{V})$ là các trạng thái đầu và cuối của chuyển t mà α' thỏa mãn r,
  + $r \in \mathcal{P}$ là hậu điều kiện sau khi thực hiện hành động a, $g = \texttt{defGuard}(a) \in \mathcal{G}$ là điều kiện gác để thực thi hành động a,

Khi đó: defGuard(a) được định nghĩa như sau:

$$= \begin{cases} \texttt{preC}(D), \texttt{if } a \in N_I. \\ \texttt{guardE}(e)(e \in D.E, \texttt{target}(e) = a), \texttt{if } a \in A_{cNode} \setminus N_j. \\ \bigwedge_{(e \in D.E, \texttt{target}(e)=a)} \texttt{isCompleted}(e) \wedge \texttt{guardE}(e), \text{if } a \in N_j. \\ \texttt{preC}(D_I) \wedge \texttt{preA}(a) \wedge \texttt{guardE}(e)(e \in D.E, \texttt{target}(e) = a), \text{if } a \in Act_{si}. \\ \texttt{preC}(D_X) \wedge \texttt{preA}(a) \wedge \texttt{guardE}(e)(e \in D.E, \texttt{target}(e) = a), \text{if } a \in Act_{se}. \\ \texttt{preA}(a) \wedge \texttt{guardE}(e)(s \in A_f, \texttt{target}(e) = s), \texttt{if } ((a \in A_{act}) \wedge \ (a = \texttt{firstAct}(s)). \\ \texttt{preA}(a)(s \in A_f, a \in actions(s)), \texttt{if } otherwise \end{cases}$$

Định nghĩa 3.3 (Các hàm trong LTS) Cho một LTS L của một mô hình USL D, một trạng thái hiện thời của L là α, và một chuyển $t = \alpha \xrightarrow{g|a|r} \alpha' \in L.\mathcal{T}$. Luận án định nghĩa các thuật ngữ như sau:

- $\texttt{preT}(t) = \alpha$, $\texttt{postT}(t) = \alpha'$, $\texttt{guard}(t) = g$, $\texttt{postC}(t) = r$, và $\texttt{act}(t) = a$.
- $\texttt{eval}(g)$ là giá trị của ràng buộc g.
- $\texttt{reachable}(\alpha) = \{t \mid \texttt{preT}(t) = \alpha\}$ là tập của các chuyển mà bắt đầu từ α.

Bảng 3.2: Danh sách các hàm được định nghĩa trong D

Hàm	**Điều kiện**	**Mô tả**
$\texttt{firstAct(c)} = c$	$c \in A_{cNode}$	Trả về hành động đầu tiên chứa trong một *ControlNode*.
$\texttt{lastAct(c)} = c$	$c \in A_{cNode}$	Trả về hành động cuối cùng chứa trong một *ControlNode*.
$\texttt{source(e)} = n_s$	$e \in E, n_s \in A,$ $n_t \in A, g \in G,$ $(n_s, g, e, n_t) \in F_e$	Trả về một USLNode nguồn của một FlowEdge.
$\texttt{target(e)} = n_t$		Trả về một USLNode đích của một FlowEdge.
$\texttt{guardE(e)} = g$		Trả về điều kiện gác của một FlowEdge.
$\texttt{guardE}(\texttt{n}_\texttt{s}, \texttt{n}_\texttt{t}) = g$		Trả về điều kiện gác của cạnh có nút nguồn và nút đích truyền vào.
$\texttt{preA(a)} = p_e$	$a \in Actions(D),$ $(p_e, a, p_o) \in F_c$	Trả về tiền điều kiện của một hành động.
$\texttt{postA(a)} = p_o$		Trả về hậu điều kiện của một hành động.
$\texttt{preA(c)} = \texttt{True}$	$c \in (A_{cNode} \setminus N_I)$	Trả về tiền điều kiện của một ControlNode không phải là InitialNode
$\texttt{postA(c)} = True$	$c \in (A_{cNode} \setminus N_f)$	Trả về hậu điều kiện của một ControlNode không phải là FinalNode.
$\texttt{postA(c)} = True$	$c \in (A_{cNode} \setminus N_f)$	Trả về hậu điều kiện của một ControlNode không phải là FinalNode
$\texttt{postC(c)} = p_u$	$c \in N_f, (c, p_u) \in F_f$	Trả về hậu điều kiện của một FinalNode
$\texttt{preC(D)} = \texttt{c}_{\texttt{preUC}}$		Trả về tiền điều kiện của một mô hình USL.
$\texttt{isCompleted(e)} = True$	$e \in E,$ $lastAct(target(e))$ đã hoàn thành thực thi	Trả về trạng thái hoàn thành của một cạnh.
$\texttt{isCompleted(e)} = False$	$e \in E,$ $lastAct(target(e))$ chưa hoàn thành thực thi	

- $\texttt{firable}(\alpha) = \{t \in \texttt{reachable}(\alpha), \texttt{eval}(\texttt{guard}(t)) = \texttt{true}\}$ là tập của các chuyển mà có thể được kích hoạt (fired) từ α.

Định nghĩa 3.4 (Chuyển đồng thời) Cho một LTS L của một mô hình USL D và trạng thái hiện tại α của. Một chuyển trạng thái đồng thời $\tau \in L.\mathcal{T}$ là một tập của các chuyển $t_1, t_2, \ldots, t_n \in \texttt{firable}(\alpha)$.

Định nghĩa 3.5 (Thực thi kịch bản ca sử dụng) Cho một kịch bản ca sử dụng của một mô hình USL D bao gồm một chuỗi các hành động $(a_0, \ldots, a_{n-1})$. Sự thực thi của kịch bản này là một đường trong LTS L của D: $p = \alpha_0 \overset{t_0}{\rightarrow} \alpha_1 \overset{t_1}{\rightarrow} \ldots \overset{t_{n-1}}{\rightarrow} \alpha_n$, khi đó $t_i = \alpha_i \overset{g_i|a_i|r_i}{\longrightarrow} \alpha_{i+1}$ $(\forall\, i = 0, \ldots, n-1)$, $\alpha_0 = L.\alpha_{init}$, $\alpha_n \in L.\mathcal{F}$, và $t_i \in L.\mathcal{T}$.

3.5 Chuyển đổi mô hình USL

USL được xây dựng để đặc tả chính xác các thành phần thông tin mô tả về hành vi và cấu trúc trong ca sử dụng với mục đích tích hợp ca sử dụng vào trong phương pháp phát triển hướng mô hình. Vì vậy, các mô hình ca sử dụng trong USL có thể được sử dụng là đầu vào cho các bộ chuyển đổi mô hình để chuyển sang các mô hình cấu trúc như biểu đồ lớp trong UML, mô hình hành vi như biểu đồ hoạt động và biểu đồ tuần tự trong UML, các ca kiểm thử chức năng mức hệ thống, và các tài liệu mô tả ca sử dụng trong ngôn ngữ tự nhiên theo mẫu. Để minh họa cho khả năng sinh tự động sang các mô hình và tài liệu khác từ các mô hình USL, luận án đã xây dựng một chuyển USL2TUCD trong ngôn ngữ chuyển mô hình M2T Acceleo để chuyển tự động một mô hình USL sang một mô tả ca sử dụng ở dạng ngôn ngữ tự nhiên dựa theo mẫu (*Template Use Case Description - TUCD*).

3.6 Kết chương

Trong chương này, luận án đề xuất một ngôn ngữ mô hình hóa chuyên biệt miền tên là USL để đặc tả chính xác các ca sử dụng bằng các mô hình. Để đạt được mục tiêu này, đầu tiên luận án nghiên cứu về miền đặc tả ca sử dụng và mục đích sử dụng của nó trong quy trình phát triển phần mềm. Kế tiếp, luận án phát triển cú pháp trừu tượng và cú pháp cụ thể cho ngôn ngữ, cũng như định nghĩa một tập các luật hợp lệ cho ngôn ngữ. Ngoài ra, ngữ nghĩa thực thi cho các mô hình USL cũng được định nghĩa bằng cách ánh xạ sang hệ thống chuyển trạng thái được gán nhãn. Tiếp theo, để đánh giá khả năng tích hợp ngôn ngữ USL vào trong phương pháp phát triển phần mềm hướng mô hình, luận án thảo luận về các chuyển mô hình có thể được áp dụng để sinh tự động các chế tác phần mềm khác nhau từ mô hình USL. Cuối cùng, luận án đánh giá khả năng diễn tả của ngôn ngữ USL với các ngôn ngữ khác đã tồn tại.

Kết quả nghiên cứu trên được công bố tại kỷ yếu Hội nghị quốc tế lần thứ 8 *Information and Communication Technology 2017 (SoICT)* (công trình khoa học số (4)), tạp chí *International Journal of Computing and Informatics 2018 (Informatica)* (công trình khoa học số (2)), kỷ yếu hội nghị trong nước lần thứ 9 *Fundamental and Applied Information Technology Research (Fair)* (công trình khoa học số (5)).

Chương 4

Phương pháp sinh tự động các ca kiểm thử từ mô hình ca sử dụng

Trong chương này, luận án đề xuất phương pháp sinh tự động các ca kiểm thử chức năng từ các mô hình ca sử dụng, phương pháp có tên là USLTG. Đầu tiên, luận án đề xuất xây dựng ngôn ngữ đặc tả các ca kiểm chức năng tên là TCSL. Sau đó, phương pháp USLTG sẽ đọc các mô hình USL đặc tả các ca sử dụng để sinh tự động các ca kiểm thử được đặc tả trong một mô hình TCSL. Cụ thể, luận án đã xây dựng ba thuật toán chính để sinh các kịch bản kiểm thử, dữ liệu kiểm thử, và chuyển các ca kiểm thử vào một mô hình TCSL.

4.1 Giới thiệu

Trong kiểm thử chức năng, thiết kế các ca kiểm thử thường được thực hiện thủ công, dựa vào tài liệu đặc tả yêu cầu phần mềm. Tuy nhiên, các yêu cầu phần mềm thường hay thay đổi trong suốt quy trình phát triển phần mềm. Do đó, các ca kiểm thử liên quan phải được xây dựng lại và được thực thi lại. Vì vậy, các nỗ lực được yêu cầu để xác định, bảo trì, và thực thi các ca kiểm thử cho các ca sử dụng là rất lớn. Việc đề xuất và phát triển kỹ các thuật kiểm thử tự động trở nên cấp thiết. Giải pháp sinh và thực thi các ca kiểm thử chức năng từ các ca sử dụng tự động sẽ giúp tiết kiệm được rất nhiều thời gian và nỗ lực cũng như giảm được số lượng các lỗi và các sai sót trong hoạt động kiểm thử phần mềm. Vì vậy, trong chương này luận án tập trung vào nghiên cứu và đề xuất một ngôn ngữ đặc tả các ca kiểm thử chức năng và xây dựng quy trình để sinh tự động được các ca kiểm thử chức năng từ mô hình ca sử dụng.

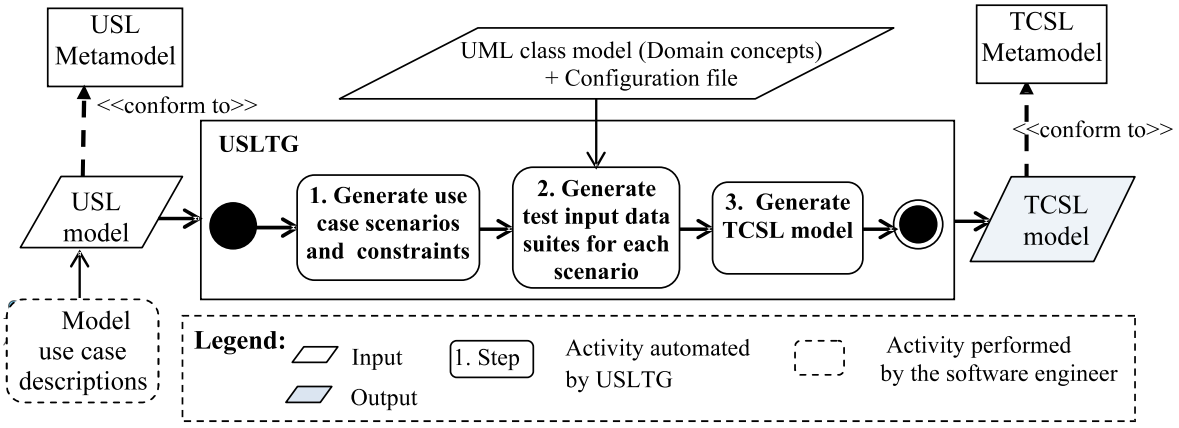
4.2 Tổng quan kỹ thuật đề xuất

Hình 4.1 biểu diễn các bước chính của cách tiếp cận USLTG của luận án. USLTG mục đích để sinh tự động các ca kiểm thử từ các ca sử dụng.

4.3 Ngôn ngữ đặc tả các ca kiểm thử TCSL

4.3.1 Xác định miền cho ngữ cảnh đặc tả ca kiểm thử chức năng

Theo Raghavendra để các ca kiểm thử có thể sinh tự động được các tập lệnh thực thi kiểm thử bằng các công cụ thực thi kiểm thử tự động, các ca kiểm thử phải gồm bốn phần thông tin khác nhau là: các bước kiểm thử (*step*), đối tượng kiểm thử của mỗi bước kiểm thử (*test object*), hành động bên trong của đối tượng kiểm thử (*test action*), và dữ liệu kiểm thử (*test data*).



Hình 4.1: Tổng quan của cách tiếp cận USLTG.

Bảng 4.1: Hai ca kiểm thử của ca sử dụng *Lend book*

STT	Bước	Đối tượng	Hành động	Dữ liệu 1	Dữ liệu 2
1	Librarian selects the Lend-book function	lendbookF	select		
2	Librarian enters a book copy id	bcid	enter	"bc_01"	"bc_03"
3	Librarian enters a borrower id	bid	enter	""	"b_03"
4	Librarian enters a borrower id	bid	enter	"b_02"	"b_02"
5	Librarian clicks the save-book-loan button	SaveBLoan	click		
6	The system save the book loan record	BookLoan	verify	a Book loan is recorded	
7	the system shows a complete message	message	verify	"Book loan save complete"	
8	the system prints the borrowing bill	bBill	verify	"Book loan receipt is printed"	

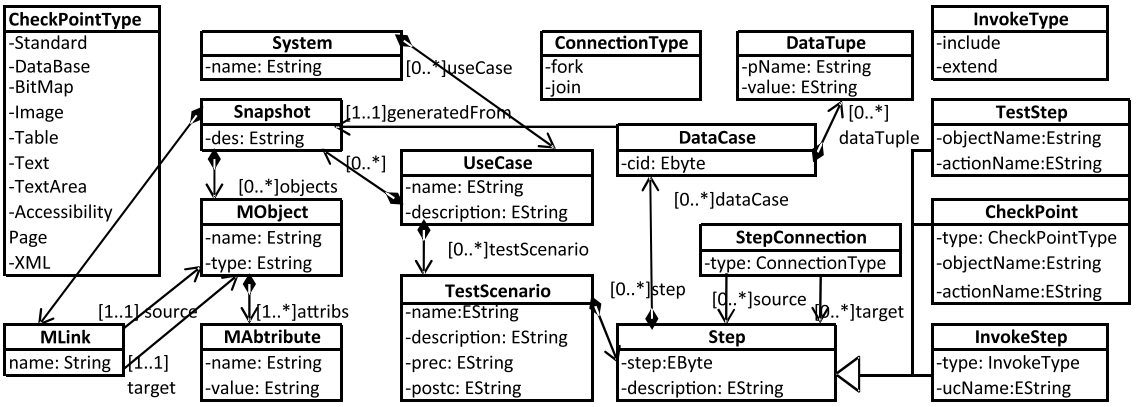
4.3.2 Định nghĩa siêu mô hình TCSL

Luận án đề xuất một ngôn ngữ TCSL (*Test Case Specification Language*) để đặc tả các ca kiểm thử chức năng mức. Một mô hình TCSL với mục đích để nắm bắt các thông tin của các ca kiểm thử của một hệ thống. Một hệ thống bao gồm một tập các ca sử dụng. Mỗi ca sử dụng bao gồm một hoặc nhiều snapshot mà nắm bắt các trạng thái bên trong của hệ thống trước khi thực hiện ca sử dụng và một tập các ca kiểm thử mà được tạo từ các kịch bản kiểm thử kết hợp với các bộ dữ liệu kiểm thử. Cụ thể, các dữ liệu kiểm thử của một ca sử dụng được xác định dựa vào trạng thái bên trong của hệ thống. Những trạng thái này là các thể hiện của các khái niệm miền của hệ thống mà liên quan tới ca sử dụng. Hình 4.2 hiển thị siêu mô hình của ngôn ngữ TCSL mà được định nghĩa bằng dự án EMF trong Eclipse.

Luận án cũng định nghĩa các luật hợp lệ trên siêu mô hình TCSL để thẩm định các mô hình TCSL khi chúng được tạo.

Luật 1: Một `TestScenario` có ít nhất hai `Step`.

Luật 2: Tất cả các `Step` trong một `TestScenario` có cùng số `DataCase` như nhau.



Hình 4.2: Siêu mô hình của TCSL

Luật 3: Nếu một hành động mà được thực hiện ở hai `TestStep` hoặc hai `CheckPoint` khác nhau trong một kịch bản kiểm thử `TestScenario`, thì các `DataTupe` của các `DataCase` ở hai `Steps` phải có cùng thuộc tính.

Luật 4: Tất cả các `DataCase` trong các `Step` của một `TestScenario` được sinh cùng từ một `SnapShot`.

4.4 Chuyển đổi mô hình từ USL sang TCSL

4.4.1 Xác định tiêu chí phủ

USL định nghĩa các khái niệm dựa trên cấu trúc của biểu đồ hoạt động để đặc tả luồng điều khiển đi qua các bước mô tả trong các luồng sự kiện của ca sử dụng. Vì vậy, Luận án sử dụng tiêu chí phủ *đường hoạt động* được đề xuất bởi Kundu và các cộng sự để sinh các ca kiểm thử từ mô hình USL. Cụ thể, một đường USL là một đường từ nút bắt đầu `InitialNode` tới một nút kết thúc `FinalNode` của mô hình USL. Đường này tương ứng với một đường hoạt động của một đồ thị hoạt động.

Định nghĩa 4.1 (Ca kiểm thử USL) Một kịch bản ca sử dụng của một mô hình USL D mà gồm chuỗi các hành động $(a_1, \ldots, a_n)$. Với một tập dữ liệu đầu vào cụ thể thỏa mãn tập các tiền điều kiện thực hiện của các hành động trong kịch bản, khi thực thi kiểm thử ứng với một đường trong một hệ thống chuyển trạng thái được gán nhãn LTS L của D : $p = \alpha_0 \xrightarrow{t_1} \alpha_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} \alpha_n$, Khi đó $t_i = \alpha_{i-1} \xrightarrow{g_i|a_i|r_i} \alpha_i$ $(\forall\, i = 1, \ldots, n)$, $\alpha_0 = L.\alpha_{init}$, $\alpha_n \in L.\mathcal{F}$, và $t_i \in L.\mathcal{T}$.

Kết quả của ca kiểm thử này là "Pass" nếu tập tất cả các hậu điều kiện của các hành động được thỏa mãn trong tất cả các chuyển, ngược lại kết quả là "Fail".

Định nghĩa 4.2 (Tiêu chí phủ đường USL) Cho P_u là tập các đường USL của một ca sử dụng u từ một mô hình USL, T là tập các ca kiểm thử, T được gọi là thỏa mãn tiêu chí phủ đường USL khi vào chỉ khi $\forall\, p_i \in P_u$, $\exists\ t \in T$ sao cho kịch bản tương ứng với t được thực hiện theo đường USL p_i.

4.4.2 Sinh các kịch bản ca sử dụng và các ràng buộc

USLTG mục đích để sinh các kịch bản kiểm thử mà thỏa mãn tiêu chí đầy đủ và phủ kiểm thử. Đầu tiên, USLTG sinh tất cả các đường USL theo tiêu chí phủ đường USL từ một nút bắt đầu `InitialNode` tới một nút kết thúc `FinalNode` trong một mô hình USL. Chú ý rằng với mỗi đường được sinh, các điều kiện gác trên một cạnh `FlowEdge` được kết hợp với tiền điều kiện của hành động tiếp theo và chuyển thành tiền điều kiện

của hành động này. Mỗi đường được xem như là một kịch bản ca sử dụng. Sau đó, mỗi kịch bản ca sử dụng được thăm lại để sinh một kịch bản kiểm thử

Thuật toán 4.1 GenScenarios

```
GenScenarios(D)
        Input: D is a USL model
Output: paths is a set of constrained use case scenarios returning from D
1  BEGIN
   pt ← ∅; sc ← ∅; x ← the InitialNode of D; GenerateScenarios(pt,sc,x);
   END.
   Procedure GenerateScenarios (pt,sc,x)
   prex = the last node in pt; push(pt,x)   if x is FinalNode then
2  |  sc ← sc ∪ x; paths ← paths ∪ sc; exit
3  end
4  if x is DecisionNode then
5  |  if ∀ e ∈ outGoing(x) was visited then
6  |  |  bf ← the BasicFlowEdge in outGoing(x)  if ∃ bf then  GenerateScenarios(pt, sc, targetNode(bf));
7  |  |  else  exit ;
8  |  else
9  |  |  foreach t in outGoing(x) that was not visited do
10 |  |  |  ptn ← copy objects from pt; scn ← copy objects from sc  GenerateScenarios(ptn, scn, targetNode(e))
11 |  |  end
12 end
13 else if x is ForkNode then
14 |  sc ← sc ∪ x; jn ← null; allsubSteps ← ∅; allsubpaths ← ∅   foreach t in outGoing(x) do
15 |  |  subSteps ← null; subPaths ← null;
   |  |    //subSteps, a set of USLNode sequences,
   |  |    //subpaths, a set of Action sequences and jn, a JoinNode
   |  |    VisitSubpath(allsubSteps, allsubPaths, subSteps, subPaths, targetNode(e), jn)
16 |  end
17 |  //maxLength(allsubSteps) returns the max size of subpaths in allsubSteps
   |    for i=0; i<maxLength(allsubSteps) do
18 |  |  ptn ← copy objects from pt; scn ← copy objects from sc  for j=0;j <allsubSteps.size() do
19 |  |  |  vt ← 0  if allsubSteps[j].size()<i then  vt=i ;
20 |  |  |  else  vt ← random from 0 to allsubSteps[j].size()-1;
21 |  |  |  ptn ← ptn ∪ allsubSteps[j][vt]; scn ← scn ∪ allsubpaths[j][vt];
22 |  |  end
23 |  |  GenerateScenarios(ptn,scn,jn)
24 |  end
25 end
26 else if x is FlowStep then
27 |  if prex is DecisionNode then
28 |  |  act ← copy the object actions(x)[0]   act.setPre(defGuard(guardE(prex, x),act)); sc ← sc ∪ act
29 |  end
30 |  else  sc ← sc ∪ actions(x)[0] ;
31 |  for i=1; i<actions(x).zise() do  sc ← sc ∪ actions(x)[i] ;
32 |  GenerateScenarios(pt, sc, targetNode(outGoing(x)[0]))
33 end
34 else  sc ← sc ∪ x; GenerateScenarios(pt, sc, targetNode(outGoing(x)[0])) ;
35 End;
```

4.4.3 Sinh các bộ dữ liệu đầu vào kiểm thử

Luận án triển khai một thuật toán sinh tên là *GenTestInputData* như được mô tả trong Thuật toán 4.2 để sinh các bộ dữ liệu đầu vào kiểm thử.

4.4.4 Sinh mô hình TCSL

Thuật toán 4.3 hiển thị Thuật toán *GenTCSLModel* để chuyển các kịch bản và các bộ dữ liệu đã được sinh vào trong một mô hình TCSL.

Hình 4.3 hiển thị một phần của mô hình TCSL của ca sử dụng *Lend book* trong hệ thống *Library*.

Thuật toán 4.2 GenTestInputData

```
GenTestInputData(paths, CD, CONF)
        Input: paths, a set of use case scenarios;
       CD, the class diagram of system domain concept;
       CONF, the configuration file;
Output: TCs, a list of pairs <scenario, EVObjects>;
       OMpi, a snapshot before executing the use case.
36 BEGIN
   OMpi ← GenerateSnapshot(CONF);
   foreach sc in paths do
37 |  EV ← processScenario(sc); CM ← combine from EV and CD;
   |   INVs ← ⋀ all pre-conditions of actions in sc;
   |   OMf ← OCLSolver(CM, CONF, INVs, OMp);
   |   a ← FindFirstStateModifingAction(a, sc); OMp ← OMpi;
   |   while (a <>null) and (∃ an action affter a that has pre-conditon) do
38 |  |  OMp ← UpdateOMp(OMp, a)  OMp ← CombineEnterdInputsOMp(OMp,OMf,a)  OMf ← OCLSol-
   |  |    ver(EV, CD, INVs, OMp)  a ← FindFirstStateModifingAction(a, sc);
   |  |
39 |  end
40 |  TCs ← TCs ∪ <sc, get Objects of EV from OMf>
41 end
42 END.
   Function processScenario(sc)
   Output: EV, a class captures test input variables in test steps of a scenario
   sc, its OCL expressions was replaced by new variable names.
   Create class EV;
   create attributes of EV from variables of the use case;
   foreach a in sc do
43 |  if a is ActorAction then  processAction(a, EV) ;
44 |  else if  a is SubSenario then
45 |  |  foreach sa in a do
46 |  |  |  if sa is ActorAction then processAction(sa, EV);
47 |  |  end
48 |  end
49 end
50 return EV;
   End;
   Procedure processAction(a, EV)
   ord ← the iteration order of a in sc  if ord >0 then
51 |  for p in a.getParameter() do
52 |  |  if isOutParam(p) then
53 |  |  |  CreateAttributeEV(p,ord, EV); na ← a; ReplaceOCL(na,p,ord,"post");
   |  |  |   while true do
54 |  |  |  |  na ← next of na  if !isEqual(na,a) then ReplaceOCL(na,p,ord,"all");
55 |  |  |  |  else ReplaceOCL(na,p,ord,"pre"); break;
56 |  |  |  |  if isFinalNode(na) then break;
57 |  |  |  end
58 |  |  end
59 |  end
60 else
61 |  for p in a.getParameter() do
62 |  |  if isOutParam(p) then CreateAttributeEV(p,ord, EV);
63 |  end
64 end
65 End;
```

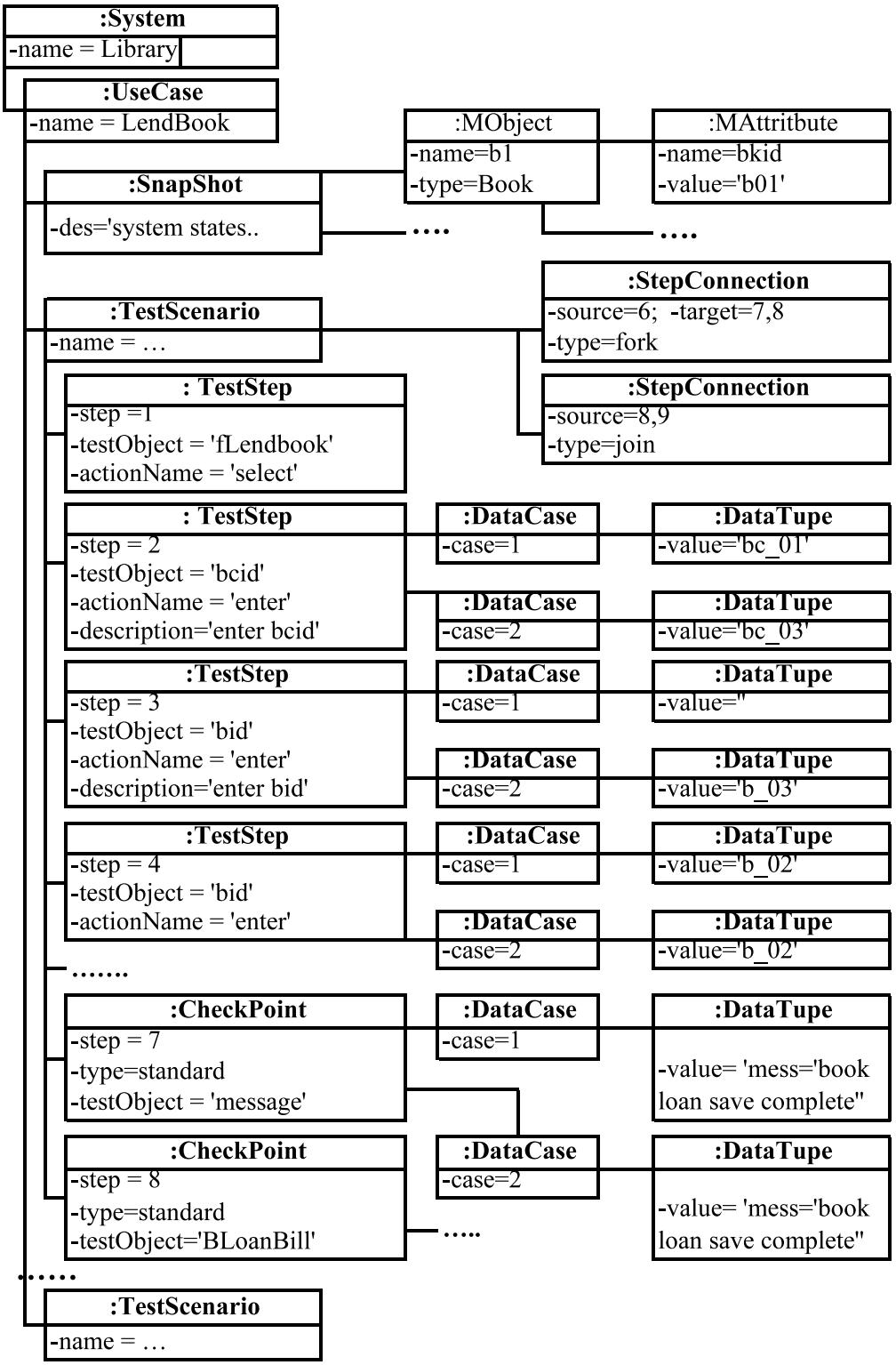
Thuật toán 4.3 GenTCSLModel

```
GenTCSLModel(lucs)
        Input: lucs, a list of use cases of the system
Output: TCSL, a TCSL model
BEGIN
  Create a TCSL model;
  for uc in lucs do
    paths ← GenScenarios(D of uc);

    TCs and OMpi ← GenTestInputData(paths, CD, CONF);
      //TCs is a set of pair<scenario, EVObjects>
      //OMpi is a snapshot of CD Create a snaphhot of uc from OMpi;
      for p in TCs do
        Create a test scenario;
          Transform p.scenario[0].pre to the pre-condition of the test scenario;
          Transfrom p.scenario[p.scenario.size()-1].post to the postcondition of the test scenario;
          a=sce[1];
          while a is not FinalNode do
            if a is Action then MapAction(p,a) ;
            else if a is ForkNode then
              Create StepConection CF;
                CF.type ← fork type; CF.source ← Step that was created before a;
                Create StepConection CJ; CJ.type ← join type; a ← next a;
                while a is not JoinNode do
                  foreach sa in a do MapAction(p,sa) ;
                  CF.target ← CF.taget ∪ the firt step is created from sa;
                    CJ.source ← CJ.source ∪ the last step is created from sa; a ← next a;
              end
              CJ.target ← the first step is created after JoinNode;

            end
            a ← next a;
        end
    end
end
END.
  Procedure MapAction (p, a)
  if a is ActorAction then
    Create TestSteps from a;
      for i=0; i<p.eVObjects.size() do Create DataCase from p.eVObjects[i] ;
end
if a is SystemAction then
    if a is SystemExtend or SystemInclude then Create InvokeStep from a ;
    else if a has postCondition then
        Create checkPoints from a for i=0; i<p.eVObjects.size() do
            pc ← a.post that its input variables is replaced by values in p.eVObjects[i] Create DataCase from
              pc
        end
    end
end
```

4.5 Tổng kết chương

Trong chương này, luận án đã đề xuất ngôn ngữ đặc tả các ca kiểm thử chức năng TCSL. Luận án cũng đã đề xuất phương pháp USLTG để sinh các ca kiểm thử từ ca sử dụng. Trong đó, các ca sử dụng được đặc tả bằng các mô hình USL, còn các ca kiểm thử sinh ra được đặc tả bằng một mô hình TCSL.

Kết quả nghiên cứu trên được công bố tại kỷ yếu Hội nghị quốc tế lần thứ 10 *Knowledge and Systems Engineering 2018 (KSE)* (công trình khoa học số (3)), và submit vào tạp chí quốc tế *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)* (ISI-indexed) (công trình khoa học số (1)), kỷ yếu hội nghị trong nước lần thứ 8 *Fundamental and Applied Information Technology Research (Fair)* (công trình khoa học số (6)).



Hình 4.3: Một phần các ca kiểm thử được sinh được đặc tả trong mô hình TCSL.

Chương 5

THỰC NGHIỆM VÀ ĐÁNH GIÁ

Trong chương này, luận án sẽ cài đặt các công cụ hỗ trợ, đưa thêm thực nghiệm, và các đánh giá cho cho ngôn ngữ USL và phương pháp USLTG.

5.1 Giới thiệu

Mục tiêu của phương pháp đặc tả chính xác ca sử dụng là để có thể tích hợp các mô hình ca sử dụng vào trong phương pháp phát triển phần mềm hướng mô hình thông qua các chuyển mô hình. Vì vậy, một công cụ cho phép tích hợp USL vào trong phương pháp phát triển phần mềm hướng mô hình giúp hiện thực hóa cách tiếp cận USL trong phát triển phần mềm.

5.2 Công cụ hỗ trợ USL

Hình 5.1 hiển thị các thành phần chính của bộ công cụ USL. Bộ công cụ USL được xây dựng từ bốn thành phần cơ bản khác nhau: (1) USL Tool là giao diện chính của công cụ hỗ trợ USL, (2) USL Editor để tạo các mô hình USL, (3) USLTG để sinh tự động các ca kiểm thử được đặc tả trong mô hình TCSL từ các mô hình USL, (4) Metamodel TCSL là cú pháp trừu tượng của TCSL cho phép đặc tả các ca kiểm thử được sinh từ USLTG. Ngoài ra luận án cũng triển khai một chuyển USL2TUCD để sinh tệp văn bản mô tả ca sử dụng theo mẫu TUCD.

5.3 Đánh giá

Trong mục này, chúng tôi sẽ đưa ra những đánh giá cho hai đề xuất của luận án là ngôn ngữ đặc tả ca sử dụng và phương pháp sinh các ca kiểm thử hệ thống từ các sử dụng. Các đề xuất sẽ được so sánh với các công việc tương tự dựa trên một số các tiêu chí so sánh.

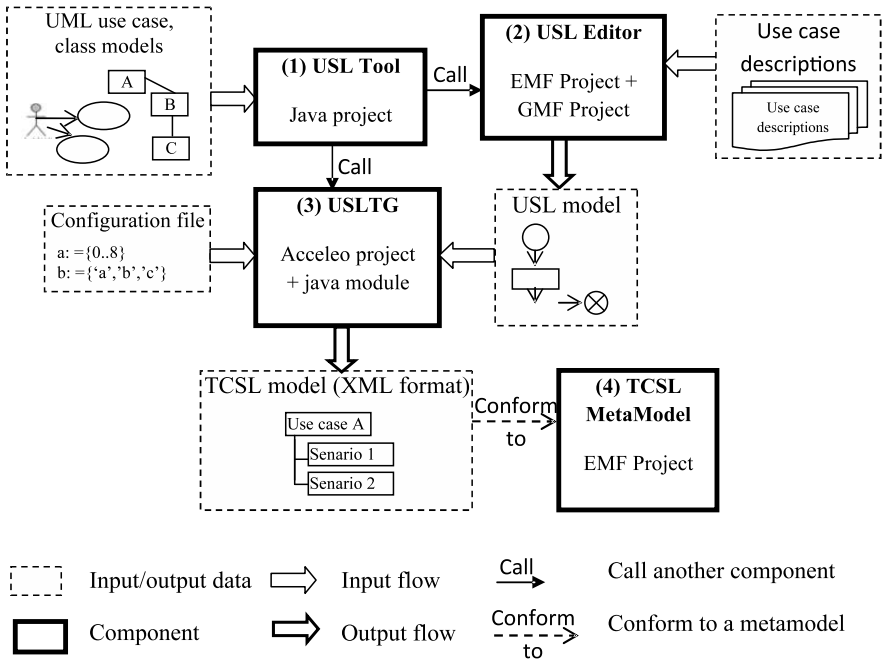
5.3.1 Đánh giá ngôn ngữ USL

Phần này trình bày đánh giá khả năng diễn tả của USL, được so sánh với sáu ngôn ngữ khác: RUCM, UC-B, ADA-M [1], MBD-L [2], SiLabReqvà RSL. Luận án sử dụng bốn tiêu chí con của khả năng diễn tả:

C1. Diễn tả dựa trên mẫu của các mô tả ca sử dụng.
C2. Diễn tả dựa trên luồng điều khiển của hành vi ca sử dụng.
C3. Đặc tả các hành động.
C4. Diễn tả các ràng buộc ca sử dụng.

Bảng 5.2 liệt kê các kết quả đánh giá cho các tiêu chí ở trên. Trong bảng, luận án sử dụng ba ký tự 'F', 'I', 'N' để ký hiệu phương pháp đặc tả mà được sử dụng cho mỗi

1. 'ADA' là viết tắt của biểu đồ hoạt động hướng tác nhân, 'M' là phương pháp (method).
2. 'MBD' là viết tắt của tên tác giả, 'L' là ngôn ngữ (language).



Hình 5.1: Các thành phần chính của công cụ USL

ngôn ngữ: 'F' ký hiệu cho phương pháp đặc tả chính thức (*formal specification method*), 'I' ký hiệu cho phương pháp đặc tả không chính thức (*informal specification method*), và 'N' ký hiệu cho thông tin không được thảo luận đặc tả.

Bảng 5.1: So sánh khả năng diễn tả giữa các ngôn ngữ đặc tả ca sử dụng

Thông tin ca sử dụng (UC)	RUCM	UC-B	ADA-M	MBD-L	SelabReq	RSL	USL
(c1) Thông tin chung	I	N	N	F	N	N	F
(c1) Các luồng của UC	I	N	I	F	N	N	F
(c1) Các kịch bản UC	N	N	N	N	F	F	N
(c2) Các luồng điều khiển	I	N	F	F	N	F	F
(c2) Các hành động đồng thời	N	N	F	N	N	N	F
(c3) Các kiểu hành động	I	N	N	F	F	F	F
(c4) Tiền và hậu điều kiện của các kịch bản UC	I	F	N	F	N	I	F
(c4) Các điều kiện gác luồng	I	F	I	F	N	I	F
(c4) Tiền và hậu điều kiện của hành động	N	F	N	N	N	N	F

5.3.2 Đánh giá phương pháp sinh các ca kiểm thử USLTG

Để so sánh với các nghiên cứu khác, luận án đưa ra bảy tiêu chí con dựa trên thông tin của các ca kiểm thử và ngôn ngữ đặc tả các ca kiểm thử. Bảng 5.2 liệt kê các kết quả so sánh trên các tiêu chí này. Trong bảng này, luận án sử dụng sáu ký tự 'Y', 'N', V', 'D', 'NL', 'M' để ký hiệu các thông tin các ca kiểm thử và ngôn ngữ đặc tả các ca kiểm thử mà đã được sinh và được đề xuất bởi mỗi phương pháp: 'Y' ký hiệu rằng thông tin được sinh, 'N' ký hiệu rằng thông tin không được xác định hoặc không thảo luận, 'V' ký

hiệu rằng thông tin đã xác định là giá trị cụ thể, 'D' ký hiệu rằng thông tin đã xác định là các mô tả chẳng hạn như các mô tả điều kiện, 'NL' ký hiệu là ngôn ngữ tự nhiên, và 'M' ký hiệu như là một ngôn ngữ đặc tả mà được xây dựng bằng kỹ thuật siêu mô hình hóa. Các tiêu chí này được thảo luận chi tiết như dưới đây.

Bảng 5.2: Sự so sánh các thông tin được xác định và ngôn ngữ đặc tả kiểm thử của các phương pháp

Thông tin ca kiểm thử	Wang	Sarmiento	Nebut	Boghdady	Tiwari	Gutierrez	Straszak	USLTG
(c1) Kịch bản kiểm thử	Y	Y	Y	Y	Y	Y	Y	Y
(c2) Đầu vào kiểm thử	V	N	N	D	N	N	D	V
(c2) Đầu ra mong đợi	V	N	N	D	N	N	D	V
(c3) Kiểu của các bước kiểm thử	N	N	N	N	N	Y	Y	Y
(c4) Tên hành động và tên đối tượng	N	N	N	N	N	N	N	Y
(c5) Các hành động đồng thời	N	Y	N	N	Y	N	N	Y
(c6) Các mối quan hệ với các ca kiểm thử khác	N	N	N	N	Y	N	Y	Y
(c7)Ngôn ngữ đặc tả ca kiểm thử	NL	NL	NL	NL	NL	M	M	M

5.4 Kết chương

Trong chương này, luận án đã xây dựng bộ công cụ hỗ trợ USL cho phép tích hợp ca sử dụng vào trong phương pháp phát triển hướng mô hình. Công cụ hỗ trợ bao gồm một trình soạn thảo để tạo các mô hình ca sử dụng trong ngôn ngữ đặc tả USL một cách trực quan, các bộ sinh tự động để chuyển đổi các mô hình USL tới các chế tác khác nhau trong phát triển phần mềm. Luận án chủ yếu tập trung vào xây dựng bộ sinh tự động các ca kiểm thử. Tiếp theo, để minh chứng cho khả năng áp dụng của phương pháp và công cụ đã đề xuất, luận án trình bày các kết quả trả về khi áp dụng phương pháp USL cho một ví dụ trong thực tế. Cuối cùng, luận án đưa ra các đánh giá cho các đề xuất của luận án với các phương pháp của các nghiên cứu khác trong phương pháp đặc tả ca sử dụng và sinh tự động các ca kiểm thử.

Kết quả nghiên cứu trên là một phần trong các công bố tại kỷ yếu Hội nghị quốc tế lần thứ 10 *Knowledge and Systems Engineering 2018 (KSE)* (công trình khoa học số (3)), hội nghị quốc tế lần thứ 8 *Information and Communication Technology 2017 (SoICT)* (công trình khoa học số (4)), tạp chí *International Journal of Computing and Informatics 2018 (Informatica)* (công trình khoa học số (2)), và submit vào tạp chí *Int'l Journal of Software Engineering and Knowledge Engineering (IJSEKE)* (ISI-indexed) (công trình khoa học số (1)).

Chương 6

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Các đóng góp của luận án

Kiểm thử phần mềm là một hoạt động quan trọng để đảm bảo chất lượng phần mềm. Hoạt động này phải được tiến hành lại khi phần mềm thay đổi. Vì vậy, tự động hóa hoạt động kiểm thử phần mềm giúp tăng chất lượng phần mềm và làm giảm chi phí trong phát triển phần mềm. Các ca sử dụng thường được mô hình hóa bằng mô hình ca sử dụng trong UML và các mô tả ca sử dụng trong ngôn ngữ tự nhiên. Vì vậy để giải quyết bài toán sinh tự động này, cần phải đặc tả chính xác được các ca sử dụng và các ca kiểm thử. Sau một thời gian nghiên cứu và giải quyết bài toán này, luận án đã có một số những đóng góp nhất định như sau.

- Đề xuất một ngôn ngữ USL để đặc tả chính xác các thông tin mô tả trong các ca sử dụng theo hướng tiếp cận mô hình hóa chuyên biệt miền.
- Xây dựng ngôn ngữ mô hình hóa chuyên biệt miền TCSL để đặc tả các ca kiểm thử chức năng.
- Đề xuất phương pháp USLTG để sinh tự động các ca kiểm thử được đặc tả trong một mô hình TCSL từ các mô hình ca sử dụng trong USL.
- Xây dựng công cụ hỗ hỗ trợ USL cho phép tích hợp ngôn ngữ USL đã đề xuất vào trong phương pháp phát triển phần mềm hướng mô hình.
- Đóng góp cuối cùng của luận án là minh họa các phương pháp đề xuất thông qua một số ca sử dụng hoàn chỉnh như *Lend book*, *Withdraw*. Ngoài ra, luận án cũng đánh giá các ngôn ngữ và phương pháp sinh kiểm thử đã đề xuất với kết quả của các nghiên cứu khác để đánh giá khả năng đặc tả của các ngôn ngữ và khả năng sinh các ca kiểm thử. Điều này phần nào thể hiện được tính khả thi của các phương pháp đề xuất.

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

6.2 Hướng phát triển

Bước đầu, các phương pháp đề xuất đã đạt được một số kết quả nhất định. Một số hướng nghiên cứu tiếp theo của luận án có thể được thực hiện là:

1) Đối với bài toán xây dựng ngôn ngữ đặc tả ca sử dụng USL. Cần tiếp tục xem xét, giải quyết các vấn đề sau:
   - Xây dựng thêm cho ngôn ngữ USL cần cung cấp thêm cú pháp và công cụ hỗ trợ soạn thảo các mô hình ở dạng văn bản gần với cấu trúc trong ngôn ngữ tự nhiên dựa trên mẫu của mô tả ca sử dụng;
   - Đưa ra giải pháp để kiểm tra tính đúng đắn của các biểu thức OCL khi xây dựng các ràng buộc cho các mô hình USL.
2) Đối với bài toán sinh kiểm thử tự động từ các mô hình USL. Cần tiếp tục xem xét, giải quyết các vấn đề sau:

- Ngôn ngữ TCSL cần cung cấp một cú pháp cụ thể cũng như công cụ soạn thảo trực quan cho kiểm thử viên dễ dàng thao tác trên mô hình.
- USLTG cần cấp các thuật toán *GenScenarios* khác nhau để sinh các kịch bản kiểm thử đạt các tiêu chí phủ kiểm thử khác nhau. Trong thuật toán *GenTestInputData*, cần đề xuất phương pháp để xác định được các trạng thái mới của hệ thống dựa trên ràng buộc hậu điều kiện của các hành động `SystemState`;

3) Đối với bộ công cụ USL cần bổ sung thêm các chức năng như: bộ soạn thảo các mô hình USL dạng văn bản, kiểm tra tính đúng đắn của các biểu thức ràng buộc OCL trong các mô hình USL, bộ soạn thảo mô hình TCSL, và các bộ sinh tự động các đầu ra khác nhau từ mô hình USL.
4) Ngoài ra, luận án sẽ thực hiện nghiên cứu đề xuất và xây dựng các bộ sinh các chế tác này từ các mô hình USL.

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

(1). Chu Thi Minh Hue, Dang Duc Hanh, and Nguyen Ngoc Binh *USLTG: A Test Case Automatic Generation by Transforming Use Cases*. (Submitted International Journal of Software Engineering and Knowledge Engineering Journal (IJSEKE - ISI indexed)).

(2). Chu Thi Minh Hue, Dang Duc Hanh, Nguyen Ngoc Binh, and Le Minh Duc. *USL: A Domain-Specific Language for Precise Specification of Use Cases and Its Transformations*. Informatica, Vol 42(3), pages 323-343, 2018. ISSN 0350-5596. (Scopus indexed)

(3). Chu Thi Minh Hue, Dang Duc Hanh, and Nguyen Ngoc Binh
A Transformation-Based Method for Test Case Automatic Generation from Use Cases. Proc. 10th Int. Conf. Knowledge and Systems Engineering (KSE), pages 252-257. IEEE Computer Society 2018. ISBN 978-1-5386-6113-0.

(4). Minh-Hue Chu, Duc-Hanh Dang, Ngoc-Binh Nguyen, Minh-Duc Le, and Thi-Hanh Nguyen. *USL: Towards Precise Specification of Use Cases for Model-Driven Development*. Proc. 8th Int. Symp. Information and Communication Technology (SoICT), pages 401-408. ACM 2017. ISBN 978-1-4503-5328-1 (Scopus indexed).

(5). Chu Thi Minh Hue, Nguyen Ngoc Binh, and Dang Duc Hanh. *A Method to Specify Software Functional Requirements for System Test Case Generation*. Proc. 9th National Conf. Fundamental and Applied Information Technology Research, pp.1-8, 2016.

(6). Chu Thị Minh Huệ, Đặng Đức Hạnh, and Nguyễn Ngọc Bình. *Phương Pháp Sinh Tự Động Ca Kiểm Thử từ Mô Hình Ca Sử Dụng*. Proc. 8th National Conf. Fundamental and Applied Information Technology Research, pp.590-599, 2015.