

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ



LÂM VĂN TÙNG

**NGHIÊN CỨU SINH CHẾ TÁC PHẦN MỀM
TỪ ĐẶC TẢ YÊU CẦU HƯỚNG MÔ HÌNH**

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

HÀ NỘI - 2023

MỤC LỤC

LỜI CẢM ƠN	i
LỜI CAM ĐOAN	ii
MỤC LỤC	iii
DANH SÁCH KÝ HIỆU, CHỮ VIẾT TẮT.....	v
DANH SÁCH BẢNG BIỂU, HÌNH VẼ.....	vi
TÓM TẮT	1
CHƯƠNG 1. MỞ ĐẦU	2
CHƯƠNG 2. KIẾN THỨC NỀN TẢNG.....	5
2.1 Tổng quan về phát triển hướng mô hình	5
2.1.1 Chuyển đổi từ mô hình sang mô hình.....	8
2.1.2 Chuyển đổi từ mô hình sang bǎn văn bản	10
2.1.3 Ngôn ngữ chuyên biệt miền.....	13
2.2 Sinh chế tác từ đặc tả yêu cầu	15
2.2.1 Giới thiệu yêu cầu.....	15
2.2.2 Đặc tả yêu cầu chức năng với mô hình FRSL	20
2.2.3 Một số tình huống chuyển đổi từ mô hình FRSL	25
2.3. Đặc tả kịch bản với công cụ plugin USE/Filmstrip.....	26
2.3.1 Đặc tả và kiểm chứng mô hình UML/OCL với USE	26
2.3.2. Mô hình Filmstrip	31
2.3.3 Chuyển đổi từ mô hình ứng dụng sang mô hình Filmstrip.....	32
2.4. Tổng kết chương.....	35
CHƯƠNG 3. PHƯƠNG PHÁP SINH CA KIỂM THỬ TỪ MÔ HÌNH FRSL.....	36
3.1 Giới thiệu.....	36
3.2 Tổng quan về phương pháp	37
3.3 Xây dựng mô hình ứng dụng Filmstrip4FRSL	39
3.3.1 Các thành phần chính của mô hình ứng dụng Filmstrip4FRSL	39
3.3.2 Sinh đặc tả lớp ứng dụng	42

3.3.3 Sinh đặc tả lớp ca sử dụng	43
3.3.4 Sinh tệp cấu hình	47
3.4 Sinh dữ liệu ca kiểm thử từ mô hình ứng dụng Filmstrip4FRSL	52
3.5. Tổng kết chương.....	57
CHƯƠNG 4. CÀI ĐẶT VÀ THỰC NGHIỆM	58
4.1 Giới thiệu.....	58
4.2 Công cụ hỗ trợ	58
4.3 Thực nghiệm 1: Sinh mô hình ứng dụng	61
4.3.1 Xây dựng mô hình FRSL.....	64
4.3.2 Xây dựng mô hình ứng dụng Filmstrip4FRSL.....	67
4.4 Thực nghiệm 2: Sinh ca kiểm thử.....	69
4.4.1 Xây dựng mô hình FRSL.....	70
4.4.2 Xây dựng mô hình ứng dụng Filmstrip4FRSL.....	72
4.4.3 Sinh các ca kiểm thử.....	74
4.5 Đánh giá kết quả thực nghiệm	75
4.6 Tổng kết chương.....	78
CHƯƠNG 5. KẾT LUẬN	79
5.1 Các đóng góp của luận văn.....	79
5.2 Hướng phát triển	79
TÀI LIỆU THAM KHẢO.....	80

DANH SÁCH KÝ HIỆU, CHỮ VIẾT TẮT

Từ viết tắt	Tiếng Anh	Tiếng Việt
ANTLR	Another Tool for Language Recognition	Công cụ nhận dạng ngôn ngữ
ATL	ATLAS Transformation Language	Ngôn ngữ chuyển đổi ATLAS
DSL	Domain Specific Language	Ngôn ngữ chuyên biệt miền
DSML	Domain Specific Modelling Language	Ngôn ngữ mô hình hóa chuyên biệt miền
EMF	Eclipse Modeling Framework	Khung mô hình hóa Eclipse
FRSL	Functional Requirements Specification Language	Ngôn ngữ đặc tả chức năng yêu cầu
MDSE	Model-Driven Software Engineering	Kỹ nghệ phần mềm hướng mô hình
M2T	Model To Text	Mô hình sang văn bản
M2M	Model To Model	Mô hình sang mô hình
OCL	Object Constraint Language	Ngôn ngữ ràng buộc đối tượng
UML	Unified Modeling Language	Ngôn ngữ mô hình hóa thống nhất

DANH SÁCH BẢNG BIẾU, HÌNH VẼ

Hình 2.1. Tổng quan phương pháp luận MDSE [14].....	6
Hình 2.2. Vai trò và định nghĩa chuyển đổi giữa các mô hình [14].	7
Hình 2.3. Chuyển đổi ngoại sinh (a) và nội sinh (b).	8
Hình 2.4. Metal Model của sWML và sMVCML [14].....	9
Hình 2.5. Mã chuyển đổi ATL.	10
Hình 2.6. Acceleo 3 – Quy trình tạo mã [5].....	11
Hình 2.7. Cấu trúc tệp module .mtl.	11
Hình 2.8. Ví dụ khai báo template	12
Hình 2.9. Ví dụ các truy vấn trong Acceleo [5].	13
Hình 2.10. Ngữ pháp được định nghĩa trong tệp .xtext.....	14
Hình 2.11. Tổng hợp phân loại yêu cầu theo Jeffrey O. Grady [18].	16
Hình 2.12. Biểu đồ ca sử dụng Hệ thống mua hàng trực tuyến.	19
Hình 2.13. Biểu đồ biểu diễn siêu mô hình của FRSL.....	23
Hình 2.14. Các bộ luật cú pháp triều tượng FRSL được xây dựng bằng Xtext.	23
Hình 2.15. Đặc tả ca sử dụng ProcessSales bằng cú pháp thẻ FRSL	24
Hình 2.16. Một số tình huống chuyển đổi từ mô hình FRSL.	25
Hình 2.17. Cấu trúc các loại biểu đồ UML 2.0.....	26
Hình 2.18. Tổng quan luồng đặc tả với USE [6].	28
Hình 2.19. Ví dụ minh họa cú pháp đặc tả USE [6].	28
Hình 2.20. Kết quả biên dịch đặc tả USE với công cụ USE [6].	29
Hình 2.21. Tiến trình xử lý của USE Model validator để sinh ca kiểm thử [13].....	29
Hình 2.22. Tệp cấu hình không gian giá trị cho bộ giải SAT trong USE.....	30
Hình 2.23. Biểu đồ đối tượng được tìm thấy sau khi bộ giải SAT thực hiện [8].....	31
Hình 2.24. Ví dụ minh họa biểu đồ lớp của CarRental.....	33
Hình 2.25. Biểu đồ lớp của mô hình filmstrip CarRental.	35
Hình 2.26. Công cụ filmstriping của USE.....	35
Hình 3.1. Luồng xử lý tự động sinh dữ liệu ca kiểm thử.	37
Hình 3.2. Tập lệnh trong quá trình tạo biểu đồ đối tượng trong USE.....	39
Hình 3.3. Đặc tả ca sử dụng BuyTicket bằng ngôn ngữ FRSL.	40
Hình 3.4. Luật chuyển các lớp ứng dụng trong FRSL model thành lớp trong Filmstrip4FRSL.	43
Hình 3.5. Luật chuyển bước FRSL model thành operations trong Filmstrip4FRSL	44
Hình 3.6 Mô sinh các thuộc tính tham chiếu của lớp ca sử dụng sang quan hệ.....	45
Hình 3.7. Hàm truy vấn lớp ca sử dụng để sinh ra các ràng buộc unchanged tất cả các bước.	46
Hình 3.8. Khuôn mẫu sinh bắt biến của các lớp trong ca sử dụng.....	46

<i>Hình 3.9. Khuôn mẫu để sinh ra preSnapshot.</i>	47
<i>Hình 3.10. Khuôn mẫu để sinh ra postSnapshot.</i>	47
<i>Hình 3.11. Khuôn mẫu tệp cấu hình được sinh ra tự động.</i>	48
<i>Hình 3.12. Tệp cấu hình cho ca sử dụng BuyTicket.</i>	49
<i>Hình 3.13. Khuôn mẫu sinh luật chuyển cho lớp ca sử dụng.</i>	50
<i>Hình 3.14. Khuôn mẫu sinh luật chuyển cho lớp tham gia ca sử dụng.</i>	51
<i>Hình 3.15. Luật chuyển cấu hình cho lớp hình chụp, lớp filmstrip và lớp thao tác OpClass.</i>	52
<i>Hình 3.16. Biểu đồ lớp của mô hình ứng dụng Filmstrip4FRSL.</i>	53
<i>Hình 3.17. Biểu đồ lớp của mô hình Filmstrip.</i>	54
<i>Hình 3.18. Tệp cấu hình được sinh ra tự động cả ca sử dụng BuyTicket.</i>	54
<i>Hình 3.19. Kết quả kiểm tra mô hình của plugin Model Validator trong USE.</i>	55
<i>Hình 3.20. Biểu đồ đối tượng.</i>	56
<i>Hình 3.21. File excel lưu trữ kịch bản kiểm thử dữ liệu.</i>	56
<i>Hình 4.1. Các thành phần của phương pháp để sinh dữ liệu cho ca kiểm thử.</i>	58
<i>Hình 4.2. Các package của công cụ Filmstrip4FRSL tool trên Eclipse IDE.</i>	59
<i>Hình 4.3. Công cụ Filmstripping trong USE.</i>	60
<i>Hình 4.4. Cấu hình không gian tìm kiếm cho Model Validator.</i>	60
<i>Hình 4.5. Biểu đồ UML ca sử dụng rút gọn của hệ thống POS.</i>	62
<i>Hình 4.6. Biểu đồ lớp của hệ thống POS.</i>	62
<i>Hình 4.7. Ca sử dụng ProcessSales đặc tả bằng cú pháp cụ thể FRSL.</i>	66
<i>Hình 4.8. Chuyển đổi cú pháp cụ thể FRSL sang cú pháp trừu tượng FRSL.</i>	66
<i>Hình 4.9. Công cụ sinh mô hình ứng dụng Filmstrip4FRSL.</i>	67
<i>Hình 4.10. Các đặc tả mô hình ứng dụng Filmstrip4FRSL và tệp cấu hình sau khi được sinh ra.</i>	68
<i>Hình 4.11. Đặc tả mô hình ứng dụng Filmstrip4FRSL dưới dạng UML/OCL.</i>	68
<i>Hình 4.12. Biểu đồ lớp của mô hình ứng dụng Filmstrip4FRSL của hệ thống POS.</i>	69
<i>Hình 4.13. Đặc tả mô hình ứng dụng Filmstrip4FRSL dưới dạng UML/OCL của ca sử dụng BuyTicket.</i>	73
<i>Hình 4.14. Các biến của mô hình kịch bản Filmstrip.</i>	73
<i>Hình 4.15. Cấu hình không gian giá trị cho Model Validator Plugin.</i>	74
<i>Hình 4.16. Biểu đồ đối tượng ca sử dụng BuyTicket được sinh ra thể hiện trạng thái của các đối tượng qua từng snapshot.</i>	74
<i>Hình 4.17. Kịch bản dữ liệu ca kiểm thử được mô tả trong một tệp.</i>	75
<i>Hình 4.18. Tệp cấu hình theo từng kịch bản của ca sử dụng BuyTicket.</i>	79

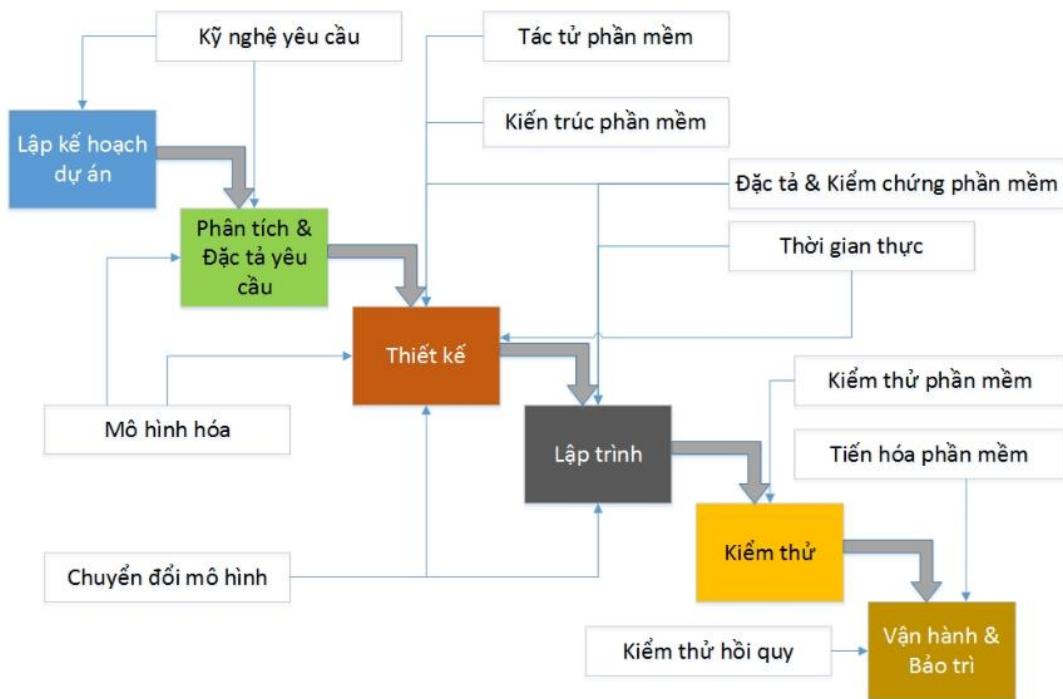
<i>Bảng 2.1. Cấu trúc đặc tả ca sử dụng</i>	<i>19</i>
<i>Bảng 2.2. Tổng quan chuyển đổi mô hình ứng dụng sang mô hình filmstrip [8].</i>	<i>33</i>
<i>Bảng 3.1. Cấu trúc cú pháp cụ thể của mô hình ứng dụng Filmstrip4FRSL.....</i>	<i>49</i>
<i>Bảng 3.2. Nhóm các luật chuyển từ FRSL sang mô hình ứng dụng Filmstrip4FRSL.....</i>	<i>42</i>
<i>Bảng 4.1. Mô tả văn bản ca sử dụng của hệ thống POS.....</i>	<i>62</i>
<i>Bảng 4.2. Ca sử dụng BuyTicket mô tả bằng văn bản.....</i>	<i>71</i>
<i>Bảng 4.3. Ca sử dụng BuyTicket đặc tả bằng FRSL.....</i>	<i>71</i>
<i>Bảng 4.4. Bảng so sánh thời gian xử lý của Model Validator.....</i>	<i>78</i>

TÓM TẮT

Kỹ nghệ phần mềm hướng mô hình (MDSE) dựa trên nền tảng mô hình hóa các khía cạnh của hệ thống đang trở thành một phương pháp phổ biến. Trong đó kỹ thuật chuyển đổi mô hình nhằm tạo ra các chế tác phần mềm đóng vai trò quan trọng. Các chế tác phần mềm bao gồm giao diện bản mẫu, các mô hình thiết kế UML, mã nguồn chương trình, bộ ca kiểm thử... Luận văn hướng tới một trong những chế tác phần mềm đó là ca kiểm thử. Việc sinh tự động ca kiểm thử từ đặc tả yêu cầu chức năng có một số khó khăn, bao gồm: (1) Đặc tả chính xác các yêu cầu chức năng; (2) Biến diễn chính xác các ca kiểm thử; (3) Phát triển các chuyển đổi mô hình để sinh các ca kiểm thử. Theo đó, luận văn giới thiệu một phương pháp sinh tự động ca kiểm thử theo tiếp cận hướng mô hình, cụ thể như sau.

Đầu tiên, các yêu cầu chức năng tương ứng với mô hình ca sử dụng sẽ được đặc tả chính xác dựa vào ngôn ngữ đặc tả FRSL. Tiếp đó, luận văn tập trung xây dựng bộ chuyển đổi để chuyển từ mô hình FRSL sang mô hình Filmstrip. Ở đây, mô hình Filmstrip được sử dụng để biểu diễn mô hình ca kiểm thử. Bằng cách sử dụng kỹ thuật tìm kiếm mô hình, từ mô hình Filmstrip này các ca kiểm thử có thể được sinh tự động.

Từ khóa: System testing, functional testing, generation of test case



CHƯƠNG 1. MỞ ĐẦU

1.1 Đặt vấn đề

Hiện nay, trong quy trình phát triển phần mềm truyền thống đang phụ thuộc nhiều vào các yếu tố con người cũng như quy trình của một tổ chức doanh nghiệp. Với các hệ thống ngày càng đa dạng và phức tạp thì không tránh khỏi những phát sinh ngoài mong muốn như sai sót hay hiệu năng, chi phí để phát triển một hệ thống hoàn chỉnh. Kỹ nghệ hướng mô hình (Model-Driven Engineering – MDE) là phương pháp hữu ích để giải quyết các vấn đề này.

Kỹ nghệ hướng mô hình đề cập đến các phương pháp tiếp cận dựa trên mô hình cho quá trình phát triển các chế tác phần mềm. Quy chuẩn hóa các góc nhìn, mô hình và công cụ để thu hẹp khoảng cách giữa yêu cầu nghiệp vụ và quá trình triển khai công nghệ thông tin.

Trong giai đoạn phát triển phần mềm, kiểm thử phần mềm là một bước quan trọng nhằm xác minh lại hệ thống được phân phối đúng với yêu cầu ban đầu hay chưa. Việc lập kế hoạch kiểm thử, thiết kế và chuẩn bị dữ liệu ca kiểm thử chủ yếu được thực hiện thủ công. Điều này gây ra tốn kém chi phí, nguồn lực cho khâu phát triển ca kiểm thử và thực thi kiểm thử. Ngoài ra, việc tạo thủ công ca kiểm thử cũng tiềm ẩn rủi ro sai sót khi phụ thuộc vào kinh nghiệm của từng người viết kịch bản và khả năng đọc hiểu đặc tả yêu cầu nghiệp vụ.

Hiện nay, cũng đã có một số nỗ lực trong việc tạo tự động ca kiểm thử từ mô hình ca sử dụng. Cơ bản nhất là dựa vào ký hiệu biểu đồ hoạt động để biểu diễn kịch bản ca sử dụng rồi tạo ra kịch bản kiểm thử bằng một số ký hiệu mô hình hóa khác như mô hình tương tác của Gutierrez và cộng sự [1]. Hay cách tiếp cận của El-Attar, M., Miller [8] dựa trên văn bản mô tả ca sử dụng kết hợp với mô hình miền và mô hình lớp. Điểm hạn chế của các phương pháp là mới ở mức hình thức, chỉ biểu diễn kịch bản ca kiểm thử ở mức biểu đồ gây khó khăn cho người dùng cuối. Các phương pháp này cũng chủ yếu tạo ra các kịch bản kiểm thử. Ngoài ra, vấn đề đặc tả chính thức yêu cầu cũng chưa được đề cập, các yêu cầu được mô tả bằng ca sử dụng hoặc các ký hiệu biểu đồ tuy dễ hiểu nhưng chưa rõ ràng và không diễn đạt hết hành vi của thống kê cũng gây khó khăn cho việc phát triển theo hướng mô hình. Nghiên cứu trong [19] đề xuất ngôn ngữ đặc tả USL để mô tả chi tiết ca sử dụng bằng cách: Mở rộng biểu đồ hoạt động của UML, đưa vào các siêu khái niệm cho mục đích đặc tả ca sử dụng. Ngoài ra, bổ sung cấu trúc mô tả hành động và điều kiện gác trên luồng. Tuy nhiên, ngôn ngữ USL có một số điểm hạn chế như khá phức tạp, cú pháp cụ thể dưới dạng đồ họa khó người sử dụng tiếp cận. Việc xác minh tính đúng

đắn, chính xác của các hành vi, ràng buộc trong ca sử dụng cũng không được mô tả, đây cũng là một vấn đề khá phức tạp.

1.2 Mục tiêu và phạm vi nghiên cứu

Với các vấn đề trên, luận văn đề xuất phương pháp Sinh tự động chế tác phần mềm từ đặc ta ca sử dụng. Bằng cách kết hợp với các công cụ hướng mô hình từ đó tạo ra dữ liệu cho ca kiểm thử. Phương pháp cho phép đặc tả ca sử dụng với một ngôn ngữ mô hình hóa FRSL chuyên biệt miền giúp yêu cầu được mô tả chính xác đầy đủ gồm các điều kiện ràng buộc, tiếp theo, qua các bước biến đổi mô hình, mô hình thu được sau biến đổi được kiểm chứng xác minh tính đúng đắn bằng công cụ hướng mô hình (USE tool) trước khi sinh ra được tập dữ liệu ca kiểm thử. Để đạt được mục đích, luận văn tập trung vào mục tiêu nghiên cứu sau:

- Nghiên cứu các kiến thức nền tảng của Kỹ nghệ hướng mô hình như các phép biến đổi từ mô hình sang mô hình (M2M), mô hình sang văn bản (M2T), cách đặc tả yêu cầu với ngôn ngữ mô hình hóa FRSL. FRSL là một ngôn ngữ chuyên biệt miền cho ca sử dụng gồm cú pháp trừu tượng dưới dạng một siêu mô hình (metal-model) và cú pháp cụ thể dưới dạng văn bản. FRSL có khả năng biểu diễn đầy đủ ca sử dụng dưới dạng mô hình và các hành vi của ca sử dụng đó. Khâu biểu diễn ca sử dụng bằng ngôn ngữ chuyên biệt miền FRSL là bước quan trọng làm đầu vào cho quá trình sinh tự động các chế tác phần mềm như ca kiểm thử, mã nguồn phần mềm, giao diện phần mềm.
- Nghiên cứu kỹ thuật sinh dữ liệu cho ca kiểm thử. Việc sinh tự động các ca kiểm thử được thực hiện qua một số bước chuyển đổi mô hình, từ mô hình ca sử dụng đến mô hình ứng dụng, cuối cùng là mô hình kịch bản (filmstrip). Từ mô hình kịch bản (filmstrip) sẽ tạo ra được biểu đồ đổi tượng thể hiện chuỗi trạng thái của hệ thống qua từng hình chụp (snapshot). Biểu đồ đổi tượng biểu diễn một kịch bản cụ thể của ca sử dụng tương ứng với một ca kiểm thử.
- Nghiên cứu kỹ thuật tìm kiếm mô hình với công cụ USE. Việc kiểm chứng mô hình đã thỏa mãn các ràng buộc hay chưa là rất phức tạp và đòi hỏi nhiều phương pháp kết hợp. Trong phạm vi luận văn chỉ trình bày về ModelValidator và kỹ thuật tìm kiếm mô hình thỏa mãn ràng buộc với Filmstrip model.

1.3 Các đóng góp chính của luận văn

Luận văn đã đề xuất được một phương pháp cho việc tạo dữ liệu ca kiểm thử tự động, phục vụ cho mục đích kiểm thử chức năng mức hệ thống. Giúp giảm bớt chi phí nguồn lực khi chuẩn bị dữ liệu ca kiểm thử ở giai đoạn tạo kịch bản. Tập dữ liệu ca kiểm

thử này được dùng cho cả giai đoạn kiểm thử tích hợp (System Integration Testing) hoặc kiểm thử chấp nhận (User Acceptance Testing-UAT). Các đóng góp chính như sau:

- Đề xuất phương pháp tổng quan trong việc sinh tự động dữ liệu ca kiểm thử.
- Xây dựng được công cụ chuyển đổi từ mô hình sang văn bản. Công cụ áp dụng các lý thuyết của Kỹ nghệ hướng mô hình: phép biến đổi mô hình sang văn bản (M2T) với các tập luật chuyển mà luận văn đề xuất cụ thể.
- Đánh giá hiệu quả phương pháp dựa trên công cụ. Áp dụng công cụ chuyển đổi được xây dựng và kỹ thuật tìm kiếm mô hình trong công cụ USE để sinh tự động dữ liệu ca kiểm thử. Từ đó, đưa ra các kết luận chung về tính hiệu quả của phương pháp cũng như tính khả thi khi ứng dụng vào thực tế.

1.4 Cấu trúc của luận văn

Cấu trúc của luận văn được tổ chức theo năm chương như sau:

Chương 1. Mở đầu: Đặt vấn đề, giới thiệu chung về kỹ nghệ hướng mô hình, mục tiêu và phạm vi của khóa luận.

Chương 2. Kiến thức nền tảng: Trình bày các khái niệm cơ bản, cùng các kiến thức và kỹ thuật nền tảng cho phương pháp được đề xuất trong luận văn.

Chương 3. Phương pháp sinh ca kiểm thử từ mô hình FRSL: Chi tiết nội dung phương pháp đề xuất.

Chương 4. Cài đặt và thực nghiệm: Cài đặt, hiện thực hóa phương pháp trên ca sử dụng cụ thể và công cụ USE. Đồng thời, tiến hành đánh giá thực nghiệm phương pháp đề xuất.

Chương 5. Kết luận: Đánh giá và tổng kết lại kết quả đạt được. Đưa ra các hướng phát triển tiếp theo.

CHƯƠNG 2. KIẾN THỨC NỀN TẢNG

Trong chương này, luận văn sẽ tập trung vào giới thiệu các kiến thức nền tảng của kỹ nghệ hướng mô hình, một số phương pháp chuyển đổi mô hình và công cụ đặc tả mô hình kịch bản. Đây là những kiến thức cơ sở mà luận văn cung cấp trước khi đi vào phương pháp và giải quyết bài toán.

2.1 Tổng quan về phát triển hướng mô hình

Phát triển phần mềm hướng mô hình [14] có thể hiểu đơn giản là một phương pháp phát triển dựa trên nền tảng là việc mô hình hóa các khía cạnh của một hệ thống, sau đó áp dụng chuyển đổi mô hình thành các tạo tác phần mềm hữu dụng. Dưới góc nhìn MDSE, việc phát triển phần mềm xoay quanh mô hình hóa và phép chuyển đổi mô hình. Chuyển đổi mô hình gồm hai dạng chính: từ mô hình sang mô hình (M2M) và từ mô hình sang văn bản (M2T). Chuyển đổi từ mô hình sang mô hình là đưa mô hình về dạng mô hình khác, ví dụ chuyển đổi biểu đồ lớp thành mô hình quan hệ, và các phép biến đổi có thể là một – một, một – nhiều hoặc nhiều – nhiều. Dạng còn lại, chuyển đổi từ mô hình sang văn bản, có thể chuyển đổi từ mô hình sang một số chế tác phần mềm như các ca kiếm thử, tập lệnh triển khai.

Sự cần thiết phải dựa vào các mô hình bởi một số lý do [14]: Các chế tác phần mềm ngày càng trở nên phức tạp hơn và do đó chúng cần được thảo luận ở các mức độ trừu tượng tùy thuộc vào các bên liên quan, giai đoạn của quá trình phát triển và mục tiêu của công việc. Nhu cầu về các phần mềm mới, sự phát triển của những phần mềm hiện có sẽ liên tục tăng lên với những kỳ vọng lớn hơn.

Kỹ thuật phát triển phần mềm theo hướng mô hình (Model Driven Software Engineering - MDSE) là mô hình phát triển phần mềm nhấn mạnh tầm quan trọng của các mô hình trong toàn bộ quy trình phát triển phần mềm. Phương pháp này áp dụng các ưu điểm của mô hình hóa vào các hoạt động của kỹ thuật phần mềm, bao gồm các khía cạnh sau:

- Các khái niệm: Các thành phần xây dựng nên phương pháp luận, từ tạo tác ngôn ngữ đến các tác nhân
- Các ký hiệu: Cách thức biểu diễn các khái niệm, là ngôn ngữ được sử dụng trong phương pháp luận
- Quy trình và quy tắc: Các hoạt động để tạo ra sản phẩm cuối cùng, các quy tắc phối hợp và kiểm soát chúng, và các khẳng định về các đặc tính mong muốn (tính đúng đắn, tính nhất quán ..) của sản phẩm hoặc của quy trình.
- Công cụ: Các ứng dụng để thực hiện, điều phối bao quát toàn bộ quy trình sản phẩm và hỗ trợ người phát triển trong việc sử dụng các ký hiệu.

Dưới đây là công thức kinh điển của Niklaus Wirth trong xây dựng chương trình:

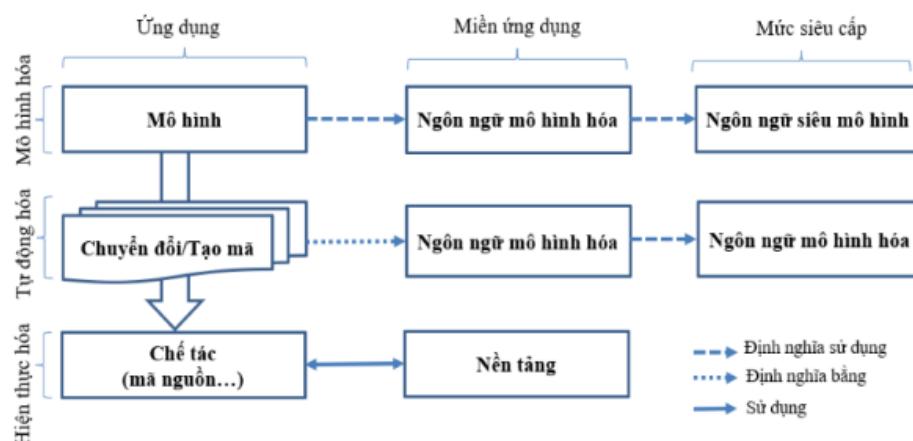
$$\text{Algorithms} + \text{Data Structures} = \text{Programs} [14]$$

Trong ngữ cảnh MDSE, công thức tương ứng sẽ là:

$$\text{Models} + \text{Transformations} = \text{Software} [14]$$

Nghĩa là, một phần mềm là quá trình chuyển đổi giữa các mô hình. Các Mô hình (**Models**) và Phép biến đổi (**Transformations**) cần được biểu diễn thông qua các ký hiệu mà trong MDSE gọi là Ngôn ngữ mô hình hóa. Kết hợp môi trường phát triển tích hợp với IDE (Integrated Development Environment) làm cho MDSE trở nên khả thi hơn trong thực tế, các Mô hình và Phép biến đổi được biên dịch, thông dịch qua IDE và tạo ra các phần mềm cuối cùng cho người dùng.

MDSE cung cấp tầm nhìn toàn diện để phát triển hệ thống trong Hình 2.1. Các cột thể hiện khía cạnh Khái niệm hóa, các hàng thể hiện khía cạnh Thực hiện



Hình 2.1. Tổng quan phương pháp luận MDSE [14].

- Mức mô hình (*Modeling*): Nơi các mô hình được xác định [14].
- Mức độ hiện thực hóa (*Realization*): Các giải pháp được thực hiện thông qua các chế tác phần mềm đang thực sự được sử dụng trong các hệ thống đang chạy.
- Mức độ tự động hóa (*Automation*): Ánh xạ từ mô hình hóa đến mức độ hiện thực hóa được đưa ra.

Vấn đề khái niệm hóa được định hướng để xác định các mô hình khái niệm để mô tả thực tế. Thể hiện ở các mức:

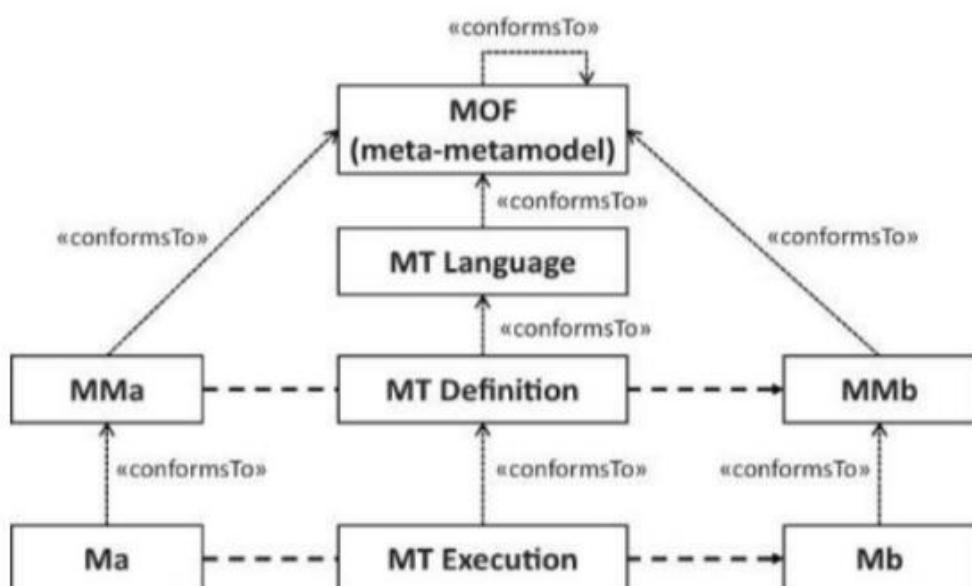
- Mức ứng dụng (*Application*): Mô hình của các ứng dụng được xác định, các quy tắc chuyển đổi được thực hiện và các thành phần đang chạy thực tế được tạo ra.
- Mức miền ứng dụng (*Application domain*): Xác định định nghĩa của ngôn ngữ mô hình hóa, phép biến đổi và nền tảng triển khai cho một miền cụ thể.
- Siêu cấp (*Meta*): Xác định khái niệm về các mô hình và các phép biến đổi.

Ngoài các mô hình, các phép chuyển đổi mô hình cũng là một thành phần quan trọng của MDSE. Các phép chuyển đổi mô hình cho phép xác định ánh xạ giữa các mô hình khác nhau. Chuyển đổi mô hình có thể được định nghĩa chung như sau:

Chuyển đổi từ một hoặc nhiều mô hình nguồn sang một hoặc nhiều mô hình đích [16].

Các phép biến đổi được thực hiện ở cấp siêu mô hình (meta model) và sau đó được áp dụng ở cấp mô hình dựa trên các mô hình phù hợp với các siêu mô hình đó. Việc chuyển đổi được thực hiện giữa mô hình nguồn và mô hình đích, nhưng nó thực sự được xác định dựa trên các siêu mô hình tương ứng.

Hình 2.2 thể hiện luồng chuyển đổi giữa các mô hình. Mô hình đầu vào Ma, bộ chuyển đổi mô hình MT để chuyển Ma thành mô hình đầu ra Mb. Ở cấp siêu mô hình (meta model) chúng ta có tương ứng là MMa, MMb và MMt. Các siêu mô hình này cũng tuân theo cùng một siêu -siêu mô hình (meta - metamodel).



Hình 2.2. Vai trò và định nghĩa chuyển đổi giữa các mô hình [14].

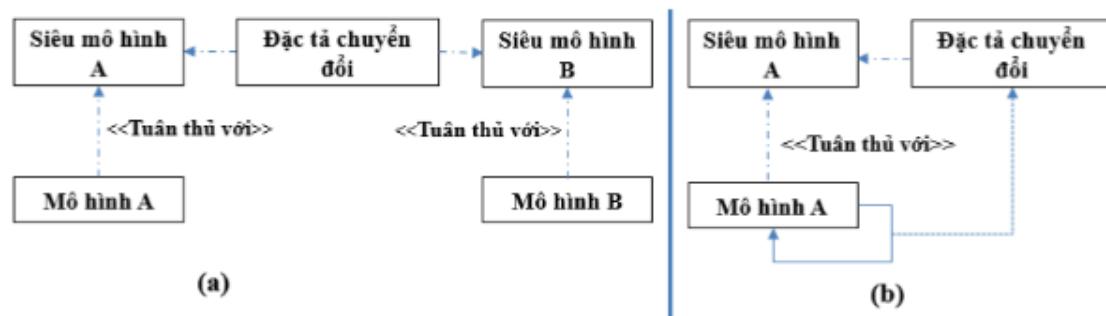
Có nhiều loại chuyển đổi mô hình khác nhau, trong phạm vi luận văn này chỉ nghiên cứu hai phép chuyển đổi là Mô hình sang mô hình (Model to Model) và Mô hình sang văn bản (Model to Text).

2.1.1 Chuyển đổi từ mô hình sang mô hình

Mô hình không phải là thực thể tinh, mà là một phần của tiến trình MDSE. Mô hình được dùng cho các mục đích khác nhau như chuẩn hóa giữa các hệ thống, đảm bảo tính nhất quán các góc nhìn khác nhau hoặc để chuyển đổi sang một ngôn ngữ khác ví dụ như mã nguồn. Nhìn chung, phép biến đổi M2M lấy một hay nhiều mô hình làm đầu vào để tạo ra một hay nhiều mô hình đầu ra, ví dụ chuyển đổi mô hình lớp sang mô hình quan hệ.

Ngoài việc phân loại dựa trên số lượng đầu vào đầu ra, phép chuyển đổi mô hình sang mô hình còn dựa trên cơ sở cùng ngôn ngữ giống nhau gọi là nội sinh, ngược lại gọi là ngoại sinh. Ví dụ chuyển đổi ngoại sinh từ mô hình UML sang mô hình cụ thể như ngôn ngữ C++, hay nội sinh chính mô hình ngôn ngữ đó nhằm cải thiện chất lượng mô hình bằng cách tái cấu mô hình sử dụng phép chuyển đổi.

Hình 2.3 mô tả các phép chuyển đổi ngoại sinh (a) đầu vào là một mô hình và đầu ra là một mô hình khác, phép biến đổi nội sinh (b) thực hiện chuyển đổi bên trong mô hình bằng cách cập nhật, xóa các phần tử.



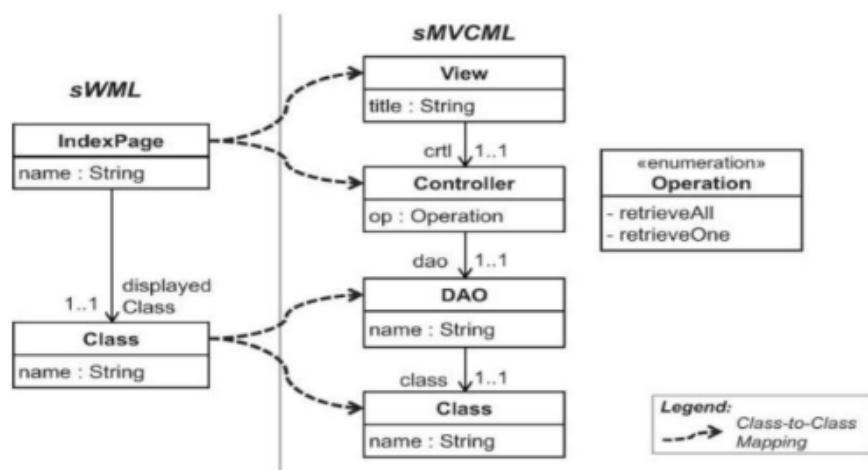
Hình 2.3. Chuyển đổi ngoại sinh (a) và nội sinh (b).

Biến đổi ngoại sinh sử dụng ngôn ngữ chuyển đổi Atlas (ATL - ATLAS Transformation Language) và biến đổi nội sinh sử dụng ngôn ngữ biến đổi đồ thị [14].

ATL là một ngôn ngữ được xây dựng dựa trên các nguyên tắc của OCL, nhưng cung cấp các tính năng ngôn ngữ dành riêng cho các phép biến đổi mô hình mà trong OCL còn thiếu [14]. Nó có khả năng tạo ra các phần tử mô hình bằng cách cung cấp các loại quy tắc khác nhau.

Ví dụ chúng ta cần thực hiện chuyển đổi một phần mô hình sWML [14] sang mô hình phụ thuộc nền tảng (PSMs), theo mẫu Model - View - Controller (MVC). Để biểu diễn PSMs, một ngôn ngữ mô hình dựa trên MVC gọi là sMVCML (simple Model View Controller Modeling Language) được sử dụng. Hình 2.4 thể hiện siêu mô hình nguồn và đích của phép chuyển đổi này. Bộ chuyển đổi cần thỏa mãn hai yêu cầu sau:

1. Yêu cầu R01 [14]: Với mỗi đối tượng Class trong sWML tương ứng với đối tượng DAO và Class trong sMVCML. Tên của lớp sWML phải trở thành tên của lớp sMVCML và tên của DAO là tên lớp sWML được ghép với - DAO. Cuối cùng, DAO phải bao gồm một liên kết đến lớp sMVCML.
2. Yêu cầu R02 [14]: Mỗi đối tượng IndexPage trong sWML ứng với đối tượng Controller và View trong sMVCML. Trong Controller có một hành động Operation. View chứa các thành phần giao diện. View liên kết với Controller, Controller được liên kết với DAO. IndexPage được liên kết với Class thông qua tham chiếu displayedClass.



Hình 2.4. Metal Model của sWML và sMVCML [14].

Một phép biến đổi được định nghĩa trong ATL được biểu diễn dưới dạng một mô-đun được chia thành phần tiêu đề và phần nội dung trong Hình 2.5. Phần tiêu đề của phép biến đổi ATL mô tả tên của mô-đun chuyển đổi và khai báo các mô hình nguồn, đích được nhập bởi siêu mô hình. Có thể có nhiều hơn một mô hình đầu vào và mô hình đầu ra cho một phép biến đổi ATL.

```
--tungly 2023-----
module SHML2MVC;
create OUT : MVC from IN : sHML;
--xử lý requirement 1-----
@rule Class2Class.DAO {
from
    c1: sHML!Class
to
    c2: MVC!Class(name <- c1.name),
    d : MVC!DAO(
        name <- c2
    )
}
--xử lý requirement 2-----
@rule IndexPage2Controller_View {
from
    p: sHML!IndexPage
to
    c: MVC!Controller (
        op <- #retrieveAll,
        dao <-thisModule.resolveTemp(p.displayedClass, 'd'),
        v: MVC!View (
            title <-p.title,
            ctrl <-c
        )
)
helper context sHML!IndexPage def: title : String = 'Show all ' + self.displayedClass.name + ' entries';
}
```

Hình 2.5. Mã chuyển đổi ATL.

2.1.2 Chuyển đổi từ mô hình sang văn bản

Trong mục này, luận văn trình bày kiến thức cơ bản về chuyển đổi từ mô hình sang văn bản, ngôn ngữ chuyển đổi và nguyên lý hoạt động của công cụ chuyển đổi.

2.1.2.1 Khái niệm M2T (Model To Text)

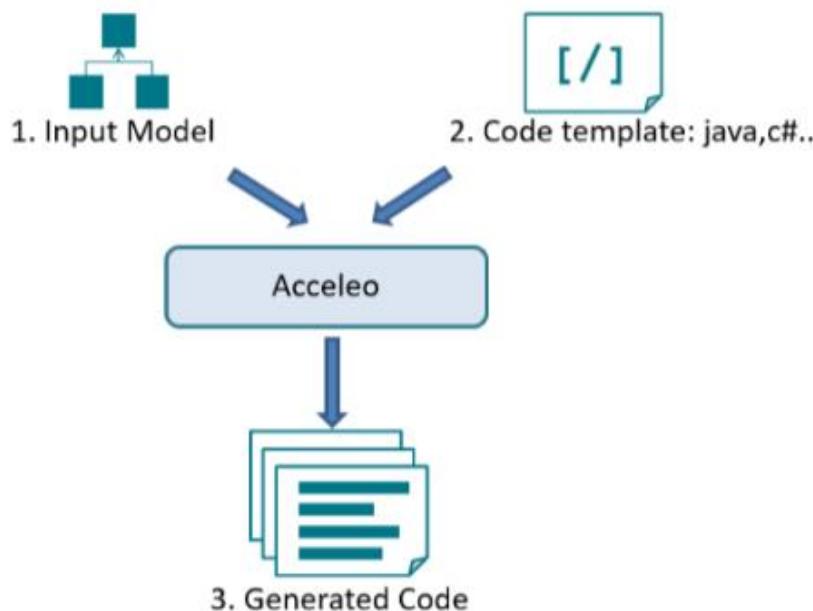
Quá trình chuyển đổi mô hình sang văn bản (M2T) là phương pháp chuyển đổi một mô hình thành văn bản. Các phép chuyển đổi từ mô hình sang văn bản được dùng để sinh mã nguồn và một số chế tác khác như các tài liệu, danh mục tác vụ, các script triển khai ứng dụng. Các phép biến đổi M2T là cầu nối cho các nền tảng thực thi và công cụ phân tích. Đối với các framework biên dịch, việc sinh mã là quá trình chuyển từ mã nguồn thành mã máy, còn đối với M2T là quá trình chuyển từ mô hình thành mã nguồn.

Để phát triển một trình tạo mã dựa trên mô hình cần xác định các yếu tố như Số lượng mã sinh ra đầy đủ hay một phần của ứng dụng; cách sinh mã thủ công hoặc tự động; kết quả đầu ra của việc sinh mã để lựa chọn ngôn ngữ phù hợp. Từ đó, sử dụng cách thức cho việc sinh mã từ Ngôn ngữ mục đích chung (General-Purpose Language GPL) hay Ngôn ngữ đặc tả chuyên biệt miền (Domain Specific Language – DSL).

2.1.2.2 Ngôn ngữ chuyển đổi Acceleo

Acceleo là một dự án mã nguồn mở nhằm mục đích cung cấp trình tạo mã để chuyển đổi mô hình thành mã cho đối tượng sử dụng phương pháp tiến cận hướng mô hình, để tăng năng suất phát triển phần mềm. Acceleo được cấp phép của Eclipse Public License (EPL), được coi là một công cụ tạo mã tốt nhất hiện nay cho phát triển dựa trên

mô hình MDD (Model Driven Development) và Kỹ nghệ hướng mô hình MDE (Model Driven Engineering). Acceleo được phát triển bởi công ty OBEO trong nhiều năm, bản đầu tiên là Acceleo 1.0.0 năm 2006 và phiên bản hiện thời là 3.7.



Hình 2.6. Acceleo 3 – Sinh tự động mã nguồn bởi chuyển đổi mô hình M2T.

Bước đầu tiên của Quy trình tạo mã Acceleo trong Hình 2.6 là một mô hình nguồn được xây dựng từ các công cụ mô hình hóa. Module ở bước 2 là tập hợp các khuôn mẫu (template) được người dùng định nghĩa tuân theo chuẩn MOFM2T, nhằm hỗ trợ cho trình tạo mã Acceleo. Mỗi module bao gồm nhiều template, template là tệp có đuôi mở rộng “.mtl” mô tả dữ liệu cần thiết để tạo ra mã nguồn từ siêu mô hình (meta-model). Acceleo engine sẽ thực hiện chuyển đổi tập các template này để sinh ra mã mong muốn.

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/uml2/5.0.0/UML')]

[template public generateElement(aClass : Class)]
[comment @main/]
[file (aClass.name, false, 'UTF-8')]

[/file]
[/template]
```

Hình 2.7. Cấu trúc tệp module .mtl.

Hình 2.7 minh họa về cấu trúc của tệp module. Bắt đầu với từ khóa **module** và một URI thể hiện meta-model mà module sinh mã này sử dụng. Một số siêu mô hình (meta-model) được sử dụng với các URI tương ứng như UML Metamodel¹, Pivot Metamodel², DSL Metamodel³, Ecore Metamodel⁴.

Các nội dung nằm trong từ khóa [template]...[/template] là tập hợp các câu lệnh Acceleo để tạo ra mã nguồn mong muốn. GenerateElement là tên của template giống như tên của một hàm/phương thức trong ngôn ngữ lập trình.

2.1.2.3 Nguyên lý chuyển đổi trong Acceleo

Template

Biểu thức *template* khai báo các thành phần sẽ thực hiện chuyển đổi theo mẫu được khớp.

Cú pháp :

```
[template public myGenerator(instance : myType)]  
[/template]
```

Trong đó :

- myType: Loại của thể hiện sẽ thực hiện phân tích cú pháp (Package, Class, Properties..) ví dụ lớp FRSLModel
- template có thể gọi lẫn nhau giống như *functions*

Trong Hình 2.8 minh họa khai báo template : tham số đầu vào là một đối tượng *uc* của lớp *UseCase*. Hàm *myGenerator* thực hiện chuyển đổi các khối lệnh bên trong *template* với mục đích tạo ra một file .txt có nội dung cố định như dưới.

```
[template public myGenerator(uc : UseCase)]  
[file (uc.name.concat('.txt'), false)]  
These are all the instructions for the script  
"generate".  
[/file]  
[/template]
```

Hình 2.8. Ví dụ khai báo template.

¹ <http://www.eclipse.org/uml2/2.0.0/UML>

² <http://www.eclipse.org/ocl/2015/Pivot>

³ <http://www.obeonetwork.org/dsl/cinematic/3.0.0>

⁴ <http://www.eclipse.org/emf/2002/Ecore>

Truy vấn (query)

Các truy vấn được sử dụng để trích xuất dữ liệu từ các mô hình được thao tác bằng cách sử dụng OCL. Chúng phải có giá trị trả về hoặc tập hợp các giá trị [5].

Ví dụ, Hình 2.9 minh họa truy vấn trong Acceleo. Nội dung truy vấn được xác định trong thẻ “query.../”. Truy vấn này trả về giá trị kiểu Boolean, có ý nghĩa: kiểm tra type của đối tượng p (kiểu Property) có thuộc loại nguyên thủy của OCL hay không (PrimitiveType).

```
[query public isPrimitive(p: Property):Boolean =  
p.type.oclIsKindOf(PrimitiveType)  
]
```

Hình 2.9. Ví dụ truy vấn trong Acceleo [5].

2.1.3 Ngôn ngữ chuyên biệt miền

Việc phát triển các ứng dụng phần mềm cho các miền cụ thể có thể trở thành một nhiệm vụ khó khăn do đòi hỏi sự hiểu biết đầy đủ về cả không gian miền và không gian triển khai. MDSE cho phép phát triển các ứng dụng dựa trên định nghĩa của các mô hình gần với miền vấn đề hơn là miền triển khai, làm giảm bớt sự phức tạp của các nền tảng. MDSE sử dụng Ngôn ngữ mô hình hóa chuyên biệt miền (DSML), là ngôn ngữ mô hình hóa được xác định cho một miền cụ thể cho phép người dùng nhận thức được họ làm việc trực tiếp với các khái niệm miền.

Ngôn ngữ mô hình hóa chuyên biệt miền (DSML) là các ngôn ngữ được thiết kế có mục đích cho một miền, ngữ cảnh hoặc công ty cụ thể để giảm bớt nhiệm vụ của những người cần mô tả mọi thứ trong miền đó [13]. Ngôn ngữ mô hình hóa chuyên biệt miền chính thức hóa cấu trúc ứng dụng, hành vi và các yêu cầu trong các miền cụ thể. DSML tuân theo các khái niệm trừu tượng và ngữ nghĩa của miền, cho phép các nhà lập mô hình tự nhận thức được họ đang làm việc trực tiếp với các khái niệm miền. Các thành phần của Ngôn ngữ mô hình hóa chuyên biệt miền:

- Cú pháp trừu tượng: được xác định bởi siêu mô hình. Cú pháp trừu tượng là một mô hình, mô tả các khái niệm về ngôn ngữ, các mối quan hệ giữa chúng. Ngoài ra, cú pháp trừu tượng bao gồm các quy tắc cấu trúc ràng buộc của các thành phần trong mô hình và các kết hợp giữa chúng để tôn trọng các quy tắc miền.
- Cú pháp cụ thể: Hiện thực hóa cú pháp trừu tượng dưới dạng ánh xạ giữa các khái niệm trong siêu mô hình và biểu diễn bằng văn bản hoặc đồ họa.
- Các ngữ nghĩa: Ngữ nghĩa của ngôn ngữ mô hình hóa nắm bắt thông tin cần thiết của các mô hình dưới dạng một ngôn ngữ được xác định rõ ràng. Miền cú

pháp mô tả tất cả các mô hình được mô tả, miền ngữ nghĩa nắm bắt tất cả thông tin cần thiết mà mô hình có thể mô tả và ánh xạ ngữ nghĩa liên kết cấu trúc cú pháp của mô hình với miền ngữ nghĩa.

Công cụ hỗ trợ Xtext

Xtext là một framework mã nguồn mở dùng để phát triển ngôn ngữ mô hình hóa chuyên biệt miền (DSL). Xtext chứa các thư viện, trình phân tích cú pháp, tạo mã và biên dịch cần thiết để tạo ra một ngôn ngữ mô hình hóa chuyên biệt miền (DSL). Xtext được tích hợp chặt chẽ với Khung mô hình hóa Eclipse (EMF) và tận dụng nền tảng của Eclipse để cung cấp một môi trường phát triển tích hợp (IDE) dành riêng cho ngôn ngữ chuyên biệt. Trái ngược với các trình tạo trình phân tích cú pháp phổ biến (như JavaCC hoặc ANTLR), Xtext có được nhiều thứ hơn là chỉ một trình phân tích cú pháp và trình phân tích từ vựng (lexer) từ ngữ pháp đầu vào [17]. Ngôn ngữ ngữ pháp dùng để mô tả và tạo ra:

- Trình phân tích cú pháp và từ vựng dựa trên ANTL để đọc các mô hình của từ văn bản.
- Ecore model.
- Bộ chuyển đổi để viết mô hình thành văn bản.
- Trình liên kết để thiết lập các liên kết chéo giữa các phần tử mô hình.
- Triển khai các giao diện (interfacea) EMF để tải và lưu mô hình EMF sau đó tích hợp ngôn ngữ vào Eclipse IDE.

Người sử dụng có thể định nghĩa các ngữ pháp trong một tệp có đuôi mở rộng .xtext (xem Hình 2.10). Ngữ pháp có hai mục đích: đầu tiên dùng để mô tả cú pháp cụ thể, thứ hai, chứa thông tin về cách trình phân tích cú pháp sẽ tạo một mô hình trong quá trình phân tích cú pháp.

```
Addition returns Expression:  
    Multiplication ({Addition.left=current} '+' right=Multiplication)*;  
  
Multiplication returns Expression:  
    Primary ({Multiplication.left=current} '*' right=Primary)*;  
  
Primary returns Expression:  
    NumberLiteral |  
    '(' Addition ')';  
  
NumberLiteral:  
    value=INT;
```

Hình 2.10. Ngữ pháp được định nghĩa trong tệp .xtext [27].

Hình 2.10 minh họa ngữ pháp cho biểu thức toán học gồm phép cộng (Addition) và phép nhân (Multiplication) có dạng $(A+B)*C$. Các quy tắc được định nghĩa như Addition, Multiplication, Primary và NumberLiteral mô tả giá trị trả về Expression hoặc NumberLiteral. Trong Xtext, mỗi quy tắc đều trả về một giá trị. Quy tắc NumberLiteral diễn đạt một phép gán ‘value’ với giá trị tương thích INT. Quy tắc Primary gồm phần tử NumberLiteral hoặc Addition. Sau đó, trình phân tích ngữ pháp trong Xtext sẽ thực hiện đọc lần lượt các quy tắc từ trên xuống dưới, từ trái qua phải để phân tích biểu thức toán học.

2.2 Sinh chép tác từ đặc tả yêu cầu

Để tạo được chép tác từ một yêu cầu thì khâu đặc tả yêu cầu là rất quan trọng nhằm mô tả và xác định chức năng, hành vi của hệ thống. Trong phần này, luận văn sẽ nêu tổng quan về yêu cầu, cách đặc tả yêu cầu bằng ca sử dụng và bằng ngôn ngữ chuyên biệt miền FRSL. Cuối cùng là giới thiệu các tình huống chuyển đổi mô hình để có được chép tác phần mềm.

2.2.1 Giới thiệu yêu cầu

Trong mục này, luận văn sẽ trình bày tổng quan khái niệm yêu cầu và cách để đặc tả yêu cầu với ca sử dụng.

2.2.1.1 Khái niệm yêu cầu

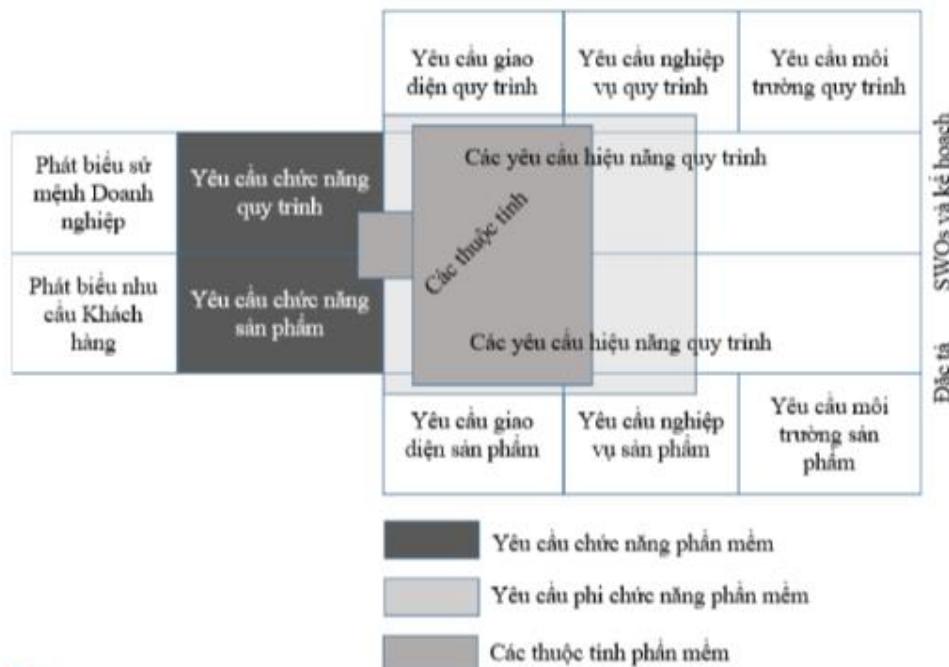
Yêu cầu là những phát biểu về những gì hệ thống cần thực hiện (functional requirements) hoặc những ràng buộc về thao tác hệ thống hoặc về quá trình phát triển (non-functional requirements). Yêu cầu là sự thống nhất giữa các bên liên quan. Yêu cầu cần được kỹ nghệ (thu thập và phân tích yêu cầu). Yêu cầu là chép tác đầu vào cho thiết kế và cài đặt phần mềm.

Để đảm bảo rằng một giải pháp phần mềm giải quyết chính xác một vấn đề cụ thể, trước tiên chúng ta phải hiểu và xác định vấn đề cần được giải quyết. Tuy nhiên, việc tìm ra đúng vấn đề gặp nhiều khó khăn trong thực tế. Chúng ta cần hiểu, xây dựng, phân tích và thống nhất xem vấn đề nào cần được giải quyết, tại sao vấn đề đó cần được giải quyết và ai nên tham gia vào trách nhiệm giải quyết vấn đề đó.

Đặc tả yêu cầu là bước đầu tiên trong một chu trình phát triển phần mềm, đó là sự kết hợp giữa các bên liên quan, khách hàng và nhà phát triển. Tùy theo các góc nhìn mà chúng ta phân loại yêu cầu theo từng loại khác nhau.

Theo Jeffrey O. Grady [18], các loại yêu cầu được chia thành phần cứng và phần mềm, như minh họa trong Hình 2.11. Các yêu cầu về phần cứng được đặc trưng và ràng buộc dưới dạng các yêu cầu về hiệu năng mà hệ thống đó cần phải đáp ứng, giao diện kết nối, môi trường và các đặc tả kỹ thuật. Các yêu cầu phần mềm gồm yêu cầu chức năng và

phi chức năng. Yêu cầu chức năng xác định các hành động mà phần mềm cần đáp ứng, các chức năng mà phần mềm tương lai (system-to-be) bắt buộc phải có. Ví dụ, phần mềm quản lý nhân sự có các chức năng quản lý thông tin cá nhân, quản lý ngày nghỉ phép, quản lý hợp đồng của người lao động trong một doanh nghiệp. Yêu cầu phi chức năng đặc tả các thuộc tính của hệ thống như độ tin cậy và an toàn, xác định các ràng buộc mà phần mềm tương lai phải thỏa mãn chức năng của nó.



Hình 2.11. Tổng hợp phân loại yêu cầu theo Jeffrey O. Grady [18].

Lớp trên cùng tương ứng với các yêu cầu phát triển, thường được gọi là yêu cầu thiết kế và phải được làm rõ, hiểu rõ trước khi thiết kế hệ thống. Lớp dưới tương ứng với các yêu cầu của sản phẩm, thường được gọi là yêu cầu xây dựng; các yêu cầu sản phẩm được nêu bút trong đặc tả yêu cầu của chương trình, mô tả tất cả những gì khách hàng cần. Lớp ở giữa là các yêu cầu về quy trình được ghi lại trong các tuyên bố về công việc và kế hoạch; các yêu cầu này nằm trong kế hoạch và thủ tục chương trình.

Sau khi đã xác định và phân loại được các loại yêu cầu, tiếp theo chúng ta đến giai đoạn đặc tả yêu cầu thành văn bản. Giai đoạn này đan xen với giai đoạn khảo sát yêu cầu.

Đầu vào của giai đoạn đặc tả và tài liệu hóa là các phát biểu (statement) khác nhau: Mục tiêu chung, yêu cầu hệ thống, yêu cầu phần mềm, các giả định về môi trường, thuộc tính miền liên quan và định nghĩa các khái niệm.

Đầu ra của giai đoạn đặc tả yêu cầu là phiên bản đặc tả yêu cầu đầu tiên đáp ứng tất cả các phát biểu đầu vào. Tài liệu đặc tả cần dễ hiểu, dễ truy xuất các đầu mục giúp cho kỹ thuật và các bên liên quan giao tiếp được với nhau. Tài liệu có thể được xây dựng bằng cách sử dụng các phương pháp: Thứ nhất, sử dụng *Ngôn ngữ tự nhiên phi cấu trúc* nhằm ghi lại tất cả những tuyên bố đã được thống nhất. Đây là cách đơn giản không mất chi phí đào tạo cho người viết, không có giới hạn trong khả năng diễn đạt tài liệu tuy nhiên dễ gặp phải những khiếm khuyết đặc biệt là sự mơ hồ, nội dung bị nhiễu, các phát biểu không đo lường được. Thứ hai, sử dụng *Ngôn ngữ tự nhiên có cấu trúc* để khắc phục những hạn chế của việc mô tả tài liệu với Ngôn ngữ tự nhiên phi cấu trúc bằng cách đưa vào các quy tắc chung. Đó là, sử dụng các quy tắc văn phong trong khi đặc tả, sử dụng bảng quyết định cho các tổ hợp điều kiện phức tạp và sử dụng mẫu biểu tài liệu. Thứ ba, *Sử dụng biểu đồ ký hiệu để đặc tả yêu cầu*. Các ký hiệu có thể bao hình thức mô tả một khía cạnh của yêu cầu. Một số loại biểu đồ: Biểu đồ ngữ cảnh, Biểu đồ khung, Biểu đồ quan hệ, Biểu đồ hoạt động...

2.2.1.2 Đặc tả yêu cầu với ca sử dụng

Ca sử dụng (Use Case) mô tả tập hợp chuỗi các hành động mà một hệ thống thực hiện mang lại kết quả có giá trị có thể quan sát được cho một tác nhân cụ thể [22].

Ca sử dụng thể hiện mối liên hệ giữa các bên liên quan của hệ thống, mô tả hành vi và tương tác của hệ thống trong các điều kiện khác nhau và phản hồi cho các bên liên quan, tác nhân chính, cho biết mục tiêu của tác nhân chính được thực hiện như thế nào. Ca sử dụng tập hợp tất cả các kịch bản liên quan với mục tiêu chính của các tác nhân. Ca sử dụng thường được viết bởi người phân tích yêu cầu và sử dụng trong giai đoạn lập kế hoạch yêu cầu hệ thống, thiết kế hệ thống hay kiểm thử phần mềm, giúp các nhóm phát triển xác định và hiểu được yêu cầu chức năng của hệ thống.

Một đặc tả ca sử dụng bao gồm các thành phần chính sau:

Tác nhân (Actor)

Là một cái gì đó có hành vi, có thể là máy móc, hệ thống máy tính, con người, tổ chức hoặc một tổ hợp. Tác nhân trao đổi thông tin với hệ thống hoặc sử dụng chức năng của hệ thống. Một use case được kích hoạt bởi một tác nhân, khi một use case thực thi nó sẽ gửi thông tin đến các tác nhân khác nhau

Tác nhân chính (Primary actor) và Tác nhân phụ (Secondary actor)

Tác nhân chính là các tác nhân sử dụng hệ thống để đạt được mục tiêu, tác nhân chính sử dụng những chức năng chính của hệ thống. Ca sử dụng mô tả các tương tác giữa

hệ thống và các tác nhân để đạt được mục tiêu của tác nhân chính. Tác nhân phụ là tác nhân mà hệ thống cần hỗ trợ để đạt được mục tiêu của tác nhân chính.

Ví dụ, trong *Hình 2.12*: Tác nhân chính là *Khách hàng*. Tác nhân phụ là *Hệ thống Paypal*.

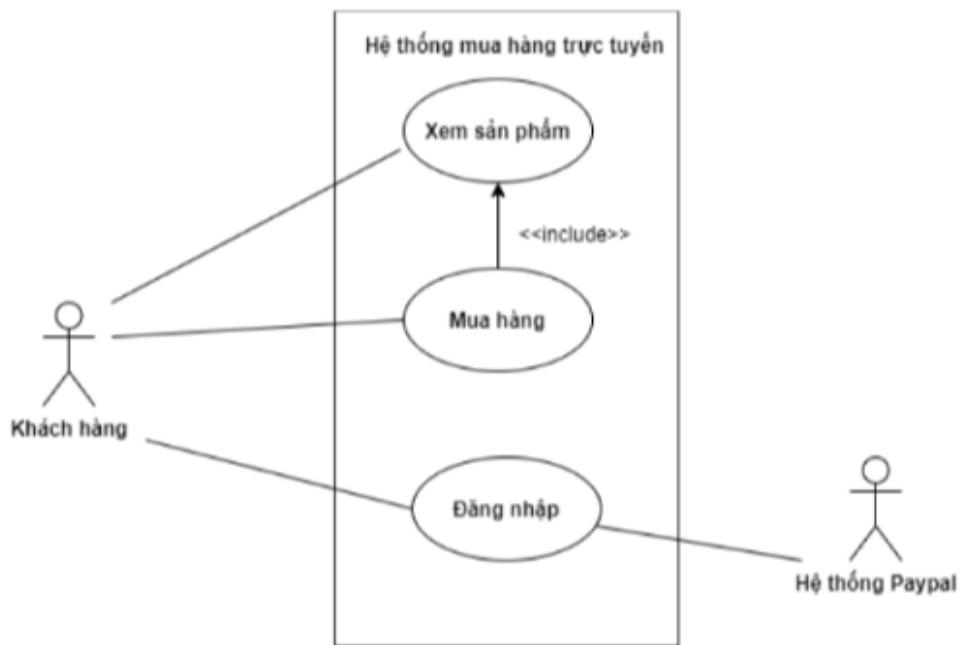
Các liên kết

Sự tham gia của một tác nhân trong một ca sử dụng được thể hiện bằng cách kết nối một tác nhân với một ca sử dụng bằng một liên kết vững chắc. Các tác nhân có thể được kết nối với các ca sử dụng bằng các liên kết, cho biết rằng tác nhân và ca sử dụng giao tiếp với nhau bằng các thông điệp.

Dường bao của hệ thống

Dường bao hệ thống có thể là toàn bộ hệ thống như được xác định trong tài liệu yêu cầu. Đối với các hệ thống lớn và phức tạp, mỗi mô-đun có thể là ranh giới hệ thống.

Hình 2.12 thể hiện một ca sử dụng trong Hệ thống mua hàng trực tuyến. Trong đó, hình vuông bao ngoài thể hiện đường bao của hệ thống, các đối tượng “Khách hàng”, “Hệ thống Paypal” là các tác nhân được minh họa bằng hình người. Các hình elip như “Xem sản phẩm”, “Mua hàng”, “Đăng nhập” là các ca sử dụng thể hiện chức năng của hệ thống. Các đường liên kết từ tác nhân đến ca sử dụng thể hiện mối quan hệ giữa các đối tượng. Trong đó quan hệ <> là loại bao gồm, thể hiện ca sử dụng này có các tính năng của ca sử dụng khác.



Hình 2.12. Biểu đồ ca sử dụng Hệ thống mua hàng trực tuyến.

Để đặc tả ca sử dụng chúng ta sử dụng biểu đồ ca sử dụng trong UML và đi kèm theo văn bản giải thích mục đích của ca sử dụng cũng như chức năng nào được thực hiện khi một ca sử dụng thực thi. Phần lớn mô hình ca sử dụng trên thực tế được trình bày ở dạng văn bản, với văn bản được ghi lại trong **Đặc tả ca sử dụng** được liên kết với mỗi phần tử của mô hình ca sử dụng. Các thông số đặc tả này mô tả luồng sự kiện của ca sử dụng. Cấu trúc của đặc tả ca sử dụng gồm 3 phần chính A) **Summary** mô tả tổng quan ca sử dụng; B) **Flow** mô tả các luồng trong ca sử dụng; C) **Additional Information** mô tả thông tin bổ sung (xem Bảng 2.1).

Bảng 2.1. Cấu trúc đặc tả ca sử dụng

A.Summary

Use Case Name: Tên Use Case

Use Case ID: Mã Use Case

Use Case Description: Tóm gọn nhanh *sự tương tác* được thể hiện trong Use Case là gì.

Actor: Những đối tượng thực hiện sự tương tác trong Use Case.

Priority: Mức độ ưu tiên của Use Case so với các Use Case còn lại trong dự án.

Trigger: Điều kiện kích hoạt Use Case xảy ra.

Pre-Condition: Điều kiện cần để Use Case thực hiện thành công.

Post-Condition: Những thứ sẽ xuất hiện sau khi Use Case được thực hiện thành công.

B. Flow

Basic Flow: luồng tương tác CHÍNH giữa các Actor và System để Use Case thực hiện thành công.

Alternative Flow: luồng tương tác THAY THẾ giữa các Actor và System để Use Case **thực hiện thành công**.

Exception Flow: luồng tương tác NGOẠI LỆ giữa các Actor và System mà Use Case **thực hiện thất bại**.

C. Additional Information

Business Rule: các quy định về mặt Business mà hệ thống bắt buộc phải nghe theo, làm theo.

Non-Functional Requirement: Vì Use Case chỉ dùng để thể hiện Functional Requirement, nên phải bổ sung các yêu cầu về Non-Functional ở đây luôn.

2.2.2 Đặc tả yêu cầu chức năng với mô hình FRSL

Các yêu cầu chức năng và hành vi của một hệ thống được phản ánh thông qua mô hình ca sử dụng. Ca sử dụng mô tả tương tác của các thành phần tham gia trong hệ thống, thực hiện các mục đích cụ thể và trong một môi trường cụ thể. Tuy nhiên, đặc tả yêu cầu với ca sử dụng có những hạn chế nhất định như không nhất quán về biểu mẫu trong quá trình đặc tả, không diễn đạt được hết các điều kiện ràng buộc của hệ thống như các yêu cầu về dữ liệu thuộc tính hay luồng dữ liệu giữa các quy trình. Do đó, chỉ đủ để trao đổi ở mức tổng quan mô hình trừu tượng của hệ thống trong giai đoạn đầu của quá trình phát triển phần mềm. Trong khi đó, việc sinh chép tác phần mềm dựa vào yêu cầu chức năng đòi hỏi các yêu cầu chức năng được đặc tả chính xác, đầy đủ và ở mức mô hình hóa. Để giải quyết vấn đề này, ngôn ngữ đặc tả ca sử dụng FRSL (Functional Requirement Specification Language) được xây dựng và phát triển cho miền vấn đề ca sử dụng. Từ các khái niệm miền ca sử dụng, FRSL được xây dựng với bộ cú pháp trừu tượng và cú pháp cụ thể

giúp người sử dụng đặc tả đầy đủ các thành phần trong mô hình ca sử dụng Ca sử dụng (usecase), Tác nhân (actor), Kịch bản (scenario), Các quan hệ của ca sử dụng, Luồng chính (basic flow), Luồng thay thế (alternative flows).

2.2.2.1 Cú pháp trừu tượng của FRSL

Cú pháp trừu tượng của ngôn ngữ FRSL được xây dựng theo hướng tiếp cận siêu mô hình. Hình 2.13 thể hiện siêu mô hình FRSL dưới dạng biểu đồ, mục đích để nắm bắt và biểu diễn miền ca sử dụng hay mô hình miền kỹ thuật. Siêu mô hình FRSL trừu tượng hóa và tạo ra cú pháp trừu tượng.

Mô hình FRSL cung cấp mô tả các khía cạnh của ca sử dụng [13]: (1) mô hình miền ở dạng một biểu đồ lớp UML/OCL; (2) cấu trúc ca sử dụng xác định bởi các quan hệ, bao gồm và mở rộng; (3) kịch bản ca sử dụng; (4) các mẫu hình chụp (snapshot pattern) thể hiện tiền và hậu điều kiện của ca sử dụng hoặc các bước trong ca sử dụng; (5) điều kiện bảo vệ của luồng thay thế (alternative) hoặc bước rejoin.

Các thành phần của mô hình FRSL được ánh xạ tương ứng với các khái niệm của miền vấn đề ca sử dụng.

- Mô tả cấu trúc ca sử dụng: bao gồm các siêu khái niệm (1) UseCase – thể hiện mô hình ca sử dụng; (2) Actor – thể hiện tác nhân chính và tác nhân phụ; (3) Include – thể hiện mối quan hệ bắt buộc giữa các ca sử dụng, tức là một ca sử dụng chứa chức năng của ca sử dụng khác như một phần xử lý của nó; (4) Extend – thể hiện mối quan hệ mở rộng giữa các ca sử dụng, mỗi quan hệ là không bắt buộc và (5) ExtensionPoint – điểm thể hiện xảy ra ca sử dụng mở rộng.
- Mô tả kịch bản ca sử dụng: Gồm các siêu khái niệm Step, ActStep, UCStep, RejoinStep, AltFlow, Action, ActorAction và SystemAction. Một bước (Step) có thể là bước hành động (ActStep) hoặc bước lặp lại (RejoinStep) hoặc bước ca sử dụng (UCStep). Một ActStep bao gồm hành động của tác nhân (ActorAction) và hành động của hệ thống (SystemAction). Một UCStep sẽ gọi ca sử dụng khác giống như quan hệ bao gồm (include). Thuộc tính *firstStep* và *nextStep* xác định bước trước và sau của một bước trong luồng ca sử dụng.
- Mô tả các hình chụp (snapshot): Bao gồm các siêu khái niệm thể hiện điều kiện và trạng thái của ca sử dụng: SnapshotPattern, ObjVar, VarLink và Constraint. Một SnapshotPattern mô tả trạng thái của hệ thống trong luồng ca sử dụng giống như các hình chụp của hệ thống, các hình chụp gồm tập hợp đối tượng (objVar) và liên kết (varLink) giữa chúng.

- objVar: Một objVars đại diện cho một đối tượng trong hình chụp hệ thống hiện thời, objVars diễn đạt đối tượng trong miền vấn đề hoặc miền phần mềm. Miền vấn đề mô tả các đối tượng thế giới thực, xác định chức năng nhiệm vụ của hệ thống tương lai (system to be). Miền phần mềm phản ánh đối tượng của miền vấn đề, xác định phần mềm tương lai (software to be) là một phần của hệ thống tương lai (system to be). Một đối tượng trong miền vấn đề được theo dõi bởi một đối tượng miền phần mềm qua quan hệ _Track. _Track là liên kết của lớp miền _DomainClass với chính nó, mọi lớp miền đều kế thừa _DomainClass.
- varLink: Xác định liên kết giữa hai đối tượng được định nghĩa bởi objVar, được khởi tạo bởi một liên kết trong mô hình miền.
- Constraint: Các ràng buộc của hình chụp hiện thời biểu thị dưới dạng điều kiện OCL trên đối tượng objVars.

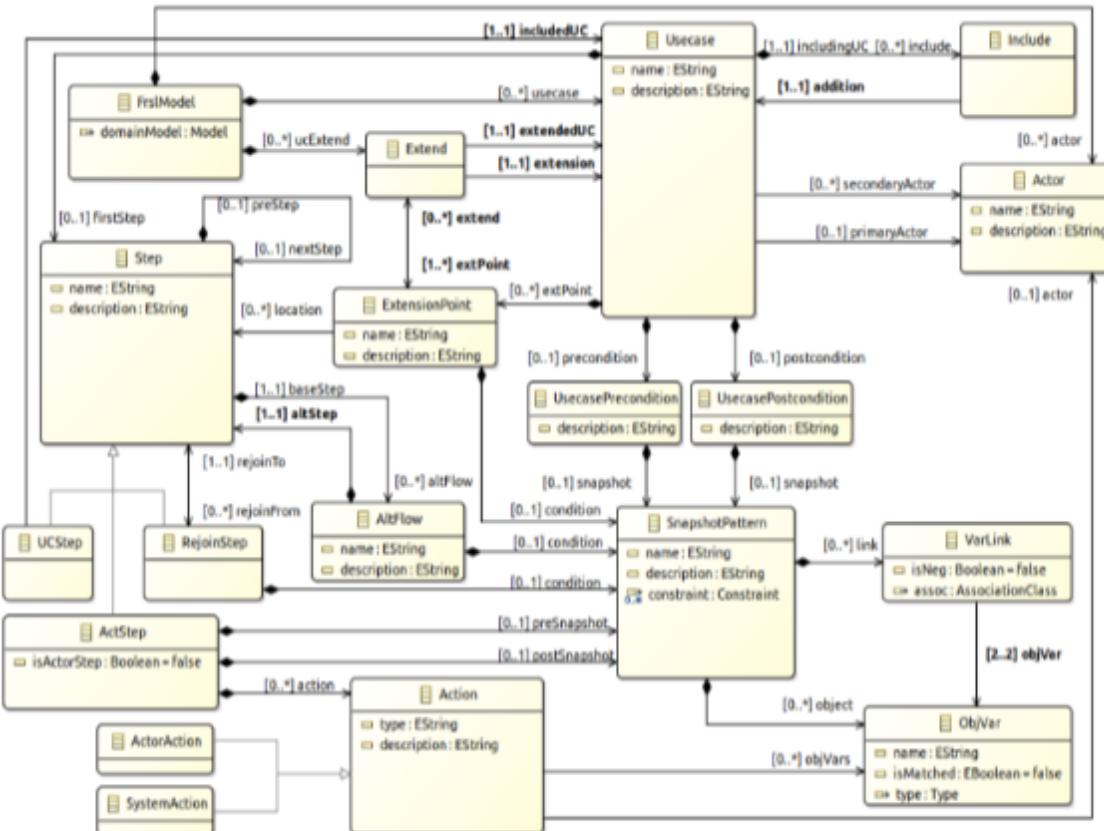
Ngoài ra, siêu mô hình của FRSL được xây dựng với một số luật giới hạn, mục đích để ràng buộc đảm bảo siêu mô hình đúng đắn và chính xác hơn như: Ca sử dụng chỉ có một tác nhân chính, tên ca sử dụng phải là duy nhất, bước thực thi phải có số bước.

Việc đặc tả ca sử dụng bằng FRSL sẽ tuân thủ theo cú pháp trừ tượng FRSL để từ đó chuyển thành mô hình FRSL làm đầu vào cho quá trình chuyên đổi, sinh chế tác như mã nguồn, dữ liệu kịch bản test...

2.2.2.2 Cú pháp cụ thể của FRSL

Ngoài cú pháp trừ tượng, FRSL còn bao gồm cú pháp cụ thể dạng văn bản để giúp người xây dựng mô hình dễ dàng hơn trong việc đặc tả thông tin ca sử dụng. Cú pháp cụ thể của FRSL được cải tiến từ khuôn mẫu và bộ luật RUCM (Restricted Use Case Modeling), được thiết kế trên EMF (Eclipse Modeling Framework) và ANTL (Another Tool for Language Recognition) để phân tích cú pháp. Trong triển khai thực tế, cú pháp cụ thể của FRSL được xây dựng dựa trên công nghệ Xtext.

Cú pháp cụ thể của FRSL bao gồm các bộ luật để định nghĩa và mô tả các khái niệm miền, quan hệ giữa các lớp đối tượng. Hình 2.14 thể hiện các luật được xây dựng bằng Xtext cho cú pháp cụ thể FRSL.



Hình 2.13. Biểu đồ biểu diễn siêu mô hình của FRSL.

```

grammar org.eclipse.sme.frsl.FRSL with org.eclipse.ocl.xtext.oclinecore.OCLinEcore

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "platform:/resource/org.eclipse.ocl.pivot/model/Pivot.ecore" as pivot
import "platform:/resource/org.eclipse.ocl.xtext.base/model/BaseCS.ecore" as base
import "platform:/resource/org.eclipse.ocl.xtext.essentialocl/model/EssentialOCLCS.ecore" as essentialocl
import "platform:/resource/org.eclipse.ocl.xtext.oclinecore/model/OCLinEcoreCS.ecore" as oclinEcore
import "platform:/resource/org.eclipse.sme.frsl/model/FRSLLCS.ecore"

//generate frsl "http://www.eclipse.org/sme/frsl/Frsl"

·FrslModelCS:
    domainModel = TopLevelCS
    (assocs += AssociationCS)*
    (actors += ActorCS)*
    (usecases += UsecaseCS)*
    (ucExtends += UcExtendCS)*
}

·UsecaseCS:
    'usecase' name = UnrestrictedName
    ('description' '=' description = StringLiteral)?
    'primaryActor' '=' primaryActor = ActorRefCS
    ('secondaryActors' '=' '(' secondaryActors += ActorRefCS (',' secondaryActors += ActorRefCS)* ')')?
    (precondition = UsecasePreconditionCS)?
    (postcondition = UsecasePostconditionCS)?
    firstStep = StepCS
    (extPoints += ExtensionPointCS)*
'end'

```

Hình 2.14. Một số bộ luật cú pháp trừu tượng FRSL được xây dựng bằng Xtext.

Hình 2.14 biểu diễn một phần đặc tả cú pháp dạng văn bản của FRSL. Đầu tiên, luật FrslModelCS xác định miền vấn đề được định nghĩa (sử dụng luật TopLevelCS), mô tả tập các ca sử dụng (sử dụng luật UsecaseCS), tập các tác nhân (luật ActorCS) và quan hệ giữa chúng (luật AssociationCS). Tiếp đó, luật UsecaseCS xác định cách thông tin đặc tả ca sử dụng như Tên ca sử dụng (từ khóa *usecase*) với, mô tả ca sử dụng (từ khóa *description*), tác nhân chính (từ khóa *primaryActor*), các tác nhân phụ (từ khóa *secondaryActors*). Tiền và hậu điều kiện của ca sử dụng được mô tả thông qua luật UsecasePreconditionCS và UsecasePostconditionCS. Cuối cùng kết thúc ca sử dụng với từ khóa ‘*end*’

```

usecase ProcessSales
    description = 'This use-case describes the process sale of Cashier'
    primaryActor = Cashier
    secondaryActors = {InventorySystem, AccountingSystem}

    ucPrecondition
        description = 'Cashier is identified and authenticated. Customer is ready to buy.'
        _cashier: Cashier;
        _customer: Customer;
        _sale: Sale;
        _pos: Register;
        _items: Set(Item);
        store: Store;
        pos: Register;
        cashier: Cashier;
        (_sale, _customer): IsFor;
        (_cashier, _pos): WorksOn;
        (_pos, pos): Tracks;
        (_cashier, cashier): _Tracks;
        (store, pos): Houses;
        [_sale.isComplete = false]
        [_items->isEmpty() = false]
    end

    ucPostcondition
        description = 'Sale is saved. Tax is correctly calculated. Accounting and Inventory
                      are updated. Commissions recorded. Receipts are generated.'
        sale: Sale;
        (_sale, sale):_Tracks;
        (store, sale): LogsCompleted;
        !(sale, pos): CapturedOn;
        !(_sale, _pos): CapturedOn;
        [sale.isComplete = true]
        [sale.total.oclIsUndefined() = false]
        [_sale.isComplete = true]
    end

    actStep step01
    description = '1. Customer arrives at POS checkout with goods and/or services to purchase'
    from

```

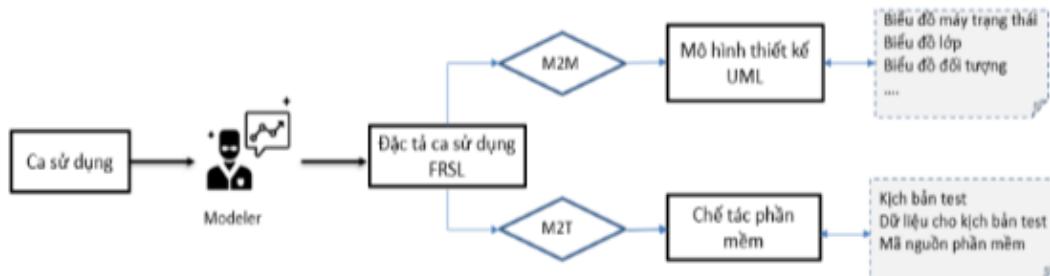
Hình 2.15. Đặc tả ca sử dụng *ProcessSales* bằng cú pháp cụ thể FRSL.

Hình 2.15 là đặc tả FRSL cho ca sử dụng *ProcessSale*, ứng với nghiệp vụ bán hàng của nhân viên thu ngân. Tác nhân chính (primaryActor) tham gia ca sử dụng là Nhân viên thu ngân (Cashier), các tác nhân phụ (secondaryActors) bao gồm Hệ thống quản lý hàng

tồn kho (InventorySystem) và Hệ thống kế toán (AccountingSystem). Tiền và hậu điều kiện của ca sử dụng được đặc tả tương ứng với các từ khóa ucPrecondition và ucPostcondition bao gồm các yêu cầu: Nhân viên thu ngân được định danh, Khách hàng sẵn sàng để mua và đối tượng Sale được tạo ra, Thuế (Tax) tính toán đúng, Kế toán (Accounting) và Hàng tồn kho (Inventory) được cập nhật các giá trị... Các đối tượng _cashier, _customer, _sale, _pos, _items thể hiện thực thể vật lý khi tham gia vào ca sử dụng. Quan hệ giữa các đối tượng như IsFor, WorksOn, _Tracks. Ký pháp ! thể hiện không tồn tại quan hệ giữa hai đối tượng.

2.2.3 Một số tình huống chuyển đổi từ mô hình FRSL

Sau khi có được mô hình đặc tả bằng FRSL như trình bày trong Mục 2.2.2, tiếp theo chúng ta sử dụng các phương pháp chuyển đổi mô hình để thu được các chế tác phần mềm như mã nguồn, kịch bản test, giao diện chương trình hay sinh mô hình thiết kế dạng UML như mô hình biểu đồ máy trạng thái, biểu đồ hoạt động, mô hình Filmstrip... Tùy vào kết quả đầu ra mong muốn, chúng ta sử dụng phương pháp chuyển đổi khác nhau như thực hiện bằng ngôn ngữ chuyển đổi mô hình sang mô hình (M2M) Atlas Transformation Language (ATL), Chuyển đổi đồ thị dựa trên cú pháp cụ thể (CGT - Concrete syntax-based graph transformation), Ngữ pháp đồ thị được phân bố (AGG - Attributed Graph Grammar) hay mô hình sang văn bản (M2T) như Acceleo.



Hình 2.16. Một số tình huống chuyển đổi từ mô hình FRSL.

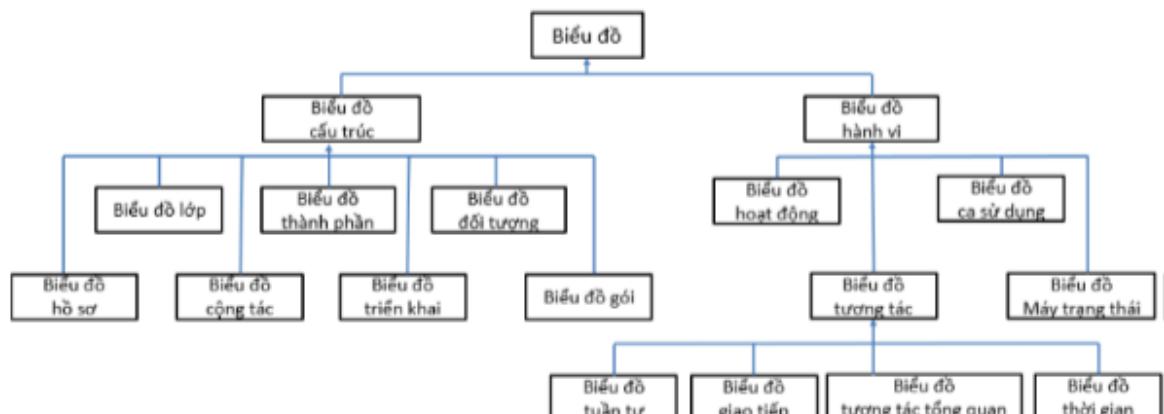
Hình 2.16 minh họa cho một số tình huống chuyển đổi mô hình từ đặc tả FRSL. Cụ thể, với các bộ chuyển đổi M2M cho phép chúng ta chuyển từ mô hình FRSL sang các dạng mô hình thiết kế UML như Biểu đồ máy trạng thái, Biểu đồ lớp, Biểu đồ đối tượng..Với bộ chuyển đổi M2T cho phép chuyển đổi sang các chế tác phần mềm như ca kiểm thử hay mã nguồn phần mềm. Trong phạm vi luận văn sử dụng phương pháp M2T để sinh chế tác phần mềm là dữ liệu cho ca kiểm thử sẽ được trình bày chi tiết ở chương tiếp theo (Chương 3).

2.3. Đặc tả kịch bản với công cụ plugin USE/Filmstrip

Trong phần này, luận văn tập trung trình bày diễn giải cách kiểm chứng mô hình với công cụ USE. Các plugin trong công cụ sẽ giúp kiểm tra và thẩm định hành vi của mô hình. Tiếp theo, giới thiệu mô hình kịch bản filmstrip để biểu diễn chuỗi trạng thái của hệ thống. Các luật chuyển đổi từ mô hình ứng dụng sang mô hình kịch bản filmstrip để từ đó việc kiểm chứng mô hình filmstrip hợp lệ bằng công cụ USE giúp tạo ra biểu đồ đối tượng làm nguyên liệu để tạo tập dữ liệu ca kiểm thử.

2.3.1 Đặc tả và kiểm chứng mô hình UML/OCL với USE

UML. Đây là một ngôn ngữ mô hình hóa được sử dụng rộng rãi nhất hiện nay [5]. Siêu mô hình bao gồm một tập hợp các kỹ thuật ký hiệu đồ họa, được sử dụng để tạo ra các mô hình trực quan của các hệ thống phần mềm hướng đối tượng. Chẳng hạn, một biểu diễn cơ bản của biểu đồ lớp, các thuộc tính, phương thức và mối quan hệ của chúng có thể được định nghĩa bởi UML. UML có thể được sử dụng để định nghĩa một số lượng đáng kể các khái niệm. UML là một ngôn ngữ mô hình hóa trực quan phổ biến, giàu ngữ nghĩa và cú pháp cho kiến trúc, thiết kế và triển khai các hệ thống phần mềm phức tạp cả về mặt cấu trúc và hành vi.



Hình 2.17. Cấu trúc các loại biểu đồ UML 2.0.

Hình 2.17 thể hiện các biểu đồ trong UML chia theo 2 nhóm chính: Biểu đồ cấu trúc và Biểu đồ hành vi. Trong mỗi nhóm là các loại biểu đồ tương ứng như Biểu đồ lớp, biểu đồ đối tượng, biểu đồ thành phần... thuộc nhóm Biểu đồ cấu trúc. Biểu đồ hoạt động, biểu đồ biểu đồ ca sử dụng.. thuộc nhóm Biểu đồ hành vi.

OCL. Đây là một ngôn ngữ ràng buộc đối tượng, được tạo ra với mục đích bổ sung các cú pháp ký hiệu cho UML, khắc phục những hạn chế của ngôn ngữ UML. UML nói chung hay một ngôn ngữ mô hình hóa nói riêng không thể mô tả hết các khía cạnh chi tiết

của một hệ thống, do đó OCL ra đời như một thành phần chính của kỹ thuật dựa trên mô hình (MDE) để thể hiện tất cả các loại truy vấn, thao tác và yêu cầu đặc tả mô hình.

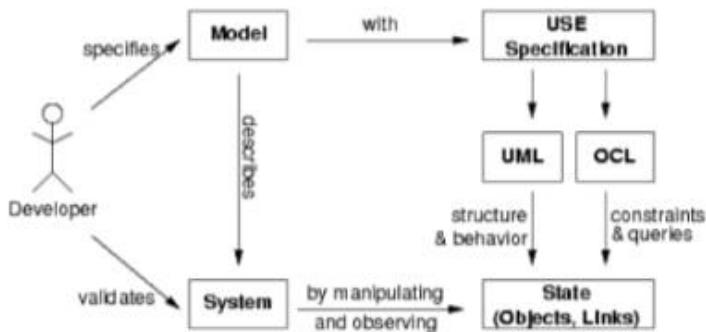
Ngôn ngữ ràng buộc đối tượng (OCL) xuất hiện như một nỗ lực để khắc phục những hạn chế của UML khi chỉ định chính xác các khía cạnh chi tiết của thiết kế hệ thống. OCL được phát triển lần đầu tiên vào năm 1995 bên trong IBM như một sự phát triển của ngôn ngữ biểu thức trong phương pháp Syntropy [3].

OCL được dùng cho những mục đích sau [2]:

- Như một ngôn ngữ truy vấn.
- Để chỉ định các biến trên các lớp và các loại trong mô hình lớp.
- Để chỉ định loại biến cho Bản mẫu.
- Để mô tả các tiền và hậu điều của một hoạt động (operation) hoặc phương thức (method).
- Để mô tả các điều kiện bảo vệ.
- Để chỉ định mục tiêu cho thông điệp và hành động.
- Để xác định các ràng buộc về hoạt động.
- Để chỉ định các quy tắc dẫn xuất cho các thuộc tính cho bất kỳ biểu thức nào trên mô hình UML.

USE. Đây là một hệ thống dùng để đặc tả hệ thống thông tin. Nó dựa trên một tập hợp con của Ngôn ngữ mô hình hóa thống nhất UML [6]. Công cụ USE được dùng cho mục đích đặc tả các mô hình, cung cấp cho người sử dụng một giao diện trực quan, thể hiện các đặc tả ở các dạng biểu đồ khác nhau như Biểu đồ lớp, Biểu đồ đối tượng, Biểu đồ trạng thái... Ngoài ra USE có khả năng kiểm tra tính hợp lệ/dúng đắn của mô hình.

Đặc tả USE là một mô tả bằng văn bản của một mô hình sử dụng các tính năng được tìm thấy trong biểu đồ lớp UML như các lớp (classes), liên kết (associations).. Biểu thức trong USE được viết bằng Ngôn ngữ ràng buộc đối tượng (OCL) để đảm bảo tính toàn vẹn trên mô hình. Đối với mỗi hình chụp (snapshot), các ràng buộc OCL sẽ tự động được kiểm tra. Các mô hình UML và OCL trong USE được đặc tả và lưu file .use. Các thuộc tính của mô hình sẽ được lưu ở file .properties.



Hình 2.18. Tổng quan luồng đặc tả với USE [6].

Hình 2.18 mô tả tổng quan luồng đặc tả USE, những nhà phát triển sẽ mô hình hóa hệ thống bằng một mô hình, mô hình này sau đó được đặc tả với cú pháp của USE. Đặc tả USE mô tả đầy đủ cấu trúc và hành vi của một mô hình bằng UML và các ràng buộc, truy vấn bằng ngôn ngữ OCL để chuyển sang trạng thái mới là các đối tượng và liên kết. Sau đó người phát triển thay vì kiểm tra hệ thống sẽ chuyển sang hành động thao tác và quan sát các trạng thái, mô hình được diễn đạt bởi USE. Trong thực tế, công cụ USE cung cấp một plugin cho phép người dùng có thể kiểm chứng tính hợp lệ của mô hình được tạo ra bởi đặc tả USE, mà luận văn sẽ trình bày trong phần tiếp theo Model Validator.

```

model Cars

class Car
    attributes
        mileage : Integer
    operations
        increaseMileage(kilometers : Integer)
    end

```

Hình 2.19. Ví dụ minh họa cú pháp đặc tả USE [6].

Hình 2.19 là một ví dụ minh họa đặc tả mô hình Cars với cú pháp USE. Trong mô hình Cars có một lớp *Car* (từ khóa class), thuộc tính *mileage* có kiểu Integer và một thao tác (operations) *increaseMileage* nhận giá trị tham biến *kilometers* có kiểu Integer.

Đặc tả sau đó được biên dịch qua công cụ USE, nếu mô tả là hợp lệ thì sẽ trả ra kết quả tìm thấy một mô hình với các thông tin bao nhiêu lớp, quan hệ, bắt biến, thao tác, tiền và hậu điều kiện như Hình 2.20.

```

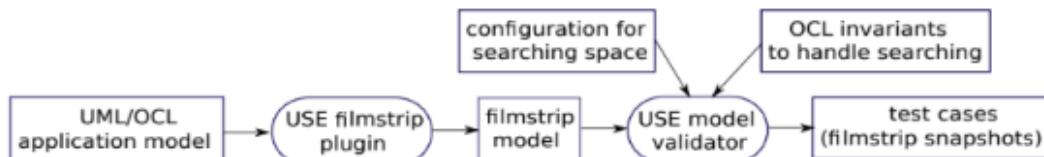
loading properties from: /home/opti/use/etc/use.properties
loading properties from: /home/opti/.userc
use version 2.3.1, Copyright (C) 1999-2006 University of Bremen
compiling specification...
Model Cars (1 class, 0 associations, 0 invariants, 1 operation, 0 pre-/postconditions)
Enter 'help' for a list of available commands.
use>

```

Hình 2.20. Kết quả biên dịch đặc tả USE với công cụ USE [6].

Kiểm chứng mô hình với USE Model Validator (model finding). Trong các cách tiếp cận dựa trên mô hình, ngôn ngữ ràng buộc đối tượng (OCL) được sử dụng để thể hiện các ràng buộc lớp và hoạt động trừu tượng. OCL và UML đóng vai trò trung tâm trong MDE. Mỗi quan tâm đối với một mô hình UML và OCL chính là kiểm tra và xác minh thuộc tính mô hình trong giai đoạn thiết kế trước khi đi vào triển khai phần mềm thực tế. Có nhiều phương pháp để xác thực một mô hình, tuy nhiên các phương pháp này thường tập trung vào khía cạnh cấu trúc ví dụ là tính nhất quán giữa mô hình lớp UML và các bất biến của lớp OCL. Ngoài ra có một cách tiếp cận là dựa trên các hoạt động trừu tượng OCL ở dạng tiền và hậu điều kiện.

Một trong những tính năng chính của công cụ USE là xác minh (verification) và kiểm tra (validate) mô hình. Ban đầu thì tính năng kiểm tra các biểu thức OCL là tính năng cốt lõi và mãnh mẽ của USE, tính năng này cũng được liên tục cập nhật để phù hợp với những thay đổi của ngôn ngữ OCL. Bên trong công cụ USE cũng được tích hợp nhiều phương pháp xác minh và kiểm tra mô hình khác nhau như ASSL và một plugin Model Validator. Ngôn ngữ kiểm tra mô hình ASSL ở trong USE cho phép kiểm tra tính nhất quán của mô hình từ một siêu mô hình được mô tả từ ngôn ngữ chuẩn UML/OCL.



Hình 2.21. Tiến trình xử lý của USE Model validator để sinh ca kiểm thử [13].

Hình 2.21 thể hiện tiến trình làm việc của USE Model Validator. Bắt đầu từ một mô hình ứng dụng được đặc bởi UML/OCL, sử dụng công cụ USE filmstrip plugin để biến đổi sang mô hình Filmstrip. Từ một mô hình gồm các lớp, bất biến, tiền và hậu điều kiện sẽ chuyển sang mô hình kịch bản Filmstrip chỉ bao gồm các lớp, quan hệ (associations) và bất biến. Về cơ bản, tính chất của mô hình kịch bản cũng giống như mô hình

ứng dụng chỉ khác là các hành động của mô hình sẽ chuyển thành các thành phần tĩnh. Đến bước này, phần mô tả vấn đề đã hoàn tất, bước tiếp theo là giải quyết các thể hiện của vấn đề. Vì tất cả các khía cạnh hành vi của mô hình ứng dụng đã được loại bỏ trong mô hình filmstrip nên có thể kiểm chứng mô hình bằng các kỹ thuật phân tích cấu trúc. Model Validator giải quyết vấn đề bằng cách sử dụng logic quan hệ, chuyển mô hình filmstrip thành các logic đại số quan hệ biểu diễn bằng Kodkod và giải quyết vấn đề bằng bộ giải SAT như Sat4j, MiniSat or Glucose [8]. Dựa vào không gian các giá trị tìm kiếm được cấu hình trong file properties.use (Hình 2.22), bộ giải SAT trả ra kết quả thỏa mãn hay không, nếu thỏa mãn một biểu đồ đối tượng được tạo ra tương ứng để thể hiện giải pháp mà Kodkod cung cấp (Hình 2.23).

```

[default]
Integer_min = -10
Integer_max = 10

String_max = 10

Real_min = -2.0
Real_max = 5.0
Real_step = 0.5

# ----- OperationCall

# ----- SnapshotItem

# PredSucc (pred:SnapshotItem, succ:SnapshotItem) -----
PredSucc_min = 1
PredSucc_max = 1

# SnapElement (snapshotItem:SnapshotItem, snapshot:Snapshot) -----
SnapElement_min = 1
SnapElement_max = 1

# ----- raiseSalary_EmployeeOpC
raiseSalary_EmployeeOpC_min = 1
raiseSalary_EmployeeOpC_max = 1

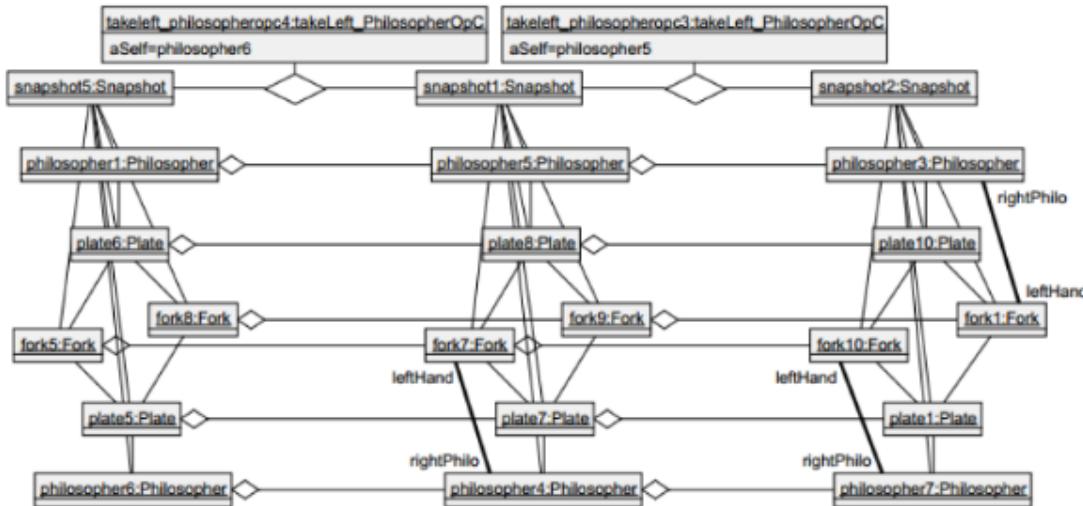
raiseSalary_EmployeeOpC_amount_min = -1
raiseSalary_EmployeeOpC_amount_max = -1
raiseSalary_EmployeeOpC_result_min = -1
raiseSalary_EmployeeOpC_result_max = -1

# ----- EmployeeOpC

```

Hình 2.22. Tệp cấu hình không gian giá trị cho bộ giải SAT trong USE.

Các đối tượng được tạo ra tối đa theo tệp cấu hình không gian giá trị và phân chia giữa các hình chụp (snapshot) của mô hình filmstrip. Việc phân chia này được ràng buộc bởi OCL.



Hình 2.23. Biểu đồ đối tượng được tìm thấy sau khi bộ giải SAT thực hiện [8].

Biểu đồ đối tượng của mô hình kịch bản filmstrip thể hiện giá trị của mỗi đối tượng qua từng trạng thái tĩnh của hệ thống, là ảnh chụp (snapshot) hệ thống tại một thời điểm nhất định. Dựa vào kết quả của biểu đồ này, chúng ta có thể xây dựng dữ liệu cho ca kiểm thử sẽ được trình bày ở chương sau.

2.3.2. Mô hình Filmstrip

Các kỹ thuật kiểm tra mô hình có hiệu quả rất tốt trong việc phân tích các hệ thống có cấu trúc tĩnh như biểu đồ lớp UML và các bất biến OCL. Tuy nhiên, các mô hình UML và OCL nói chung có thể liên quan đến các khía cạnh động dưới dạng tiền điều kiện và hậu điều kiện. Từ một mô hình UML và OCL với các bất biến, tiền và hậu điều kiện có thể chuyển đổi thành một mô hình tương đương chỉ với các bất biến. Mô hình đầu gọi là mô hình ứng dụng, mô hình thứ hai (chỉ với các bất biến) là mô hình cuộn phim (filmstrip). Một mô hình filmstrip nhằm mục đích mô tả một chuỗi các chuyển đổi trạng thái hệ thống từ mô hình ứng dụng dưới dạng một biểu đồ đối tượng duy nhất [8]. Bởi vì một chuỗi các trạng thái hệ thống trong mô hình ứng dụng trở thành một trạng thái duy nhất trong mô hình filmstrip. Trạng thái duy nhất này giống như một hình chụp (snapshot) của mô hình ứng dụng với các mốc thời gian khác nhau. Các tiền điều kiện và hậu điều kiện từ mô hình ứng dụng trở thành các bất biến ở mô hình filmstrip. Mô hình filmstrip này có thể được sử dụng để xây dựng tự động các kịch bản kiểm thử và kiểm tra các thuộc tính tạm thời.

2.3.3 Chuyển đổi từ mô hình ứng dụng sang mô hình Filmstrip

Mô hình ứng dụng. Một mô hình UML thông thường bao gồm một biểu đồ lớp, trong đó có số lượng các lớp, thuộc tính, quan hệ kết hợp (associations) và các thao tác (operations). Biểu đồ này được diễn đạt phong phú hơn nhờ các ràng buộc OCL ở dạng bất biến lớp, các tiền và hậu điều kiện của thao tác. Các bất biến để mô tả ràng buộc các thuộc tính trong mô hình, bên cạnh đó các tiền và hậu điều kiện thể hiện sự chuyển đổi trạng thái sau mỗi thao tác của hệ thống.

Hình 2.24 minh họa biểu đồ lớp của mô hình ứng dụng cho bài toán CarRental. Mô hình gồm 09 classes là *Person*, *Customer*, *Employee*, *Branch*, *Rental*, *CarGroup*, *Car*, *ServiceDepot*, *Check*. Trong đó *Customer* và *Employee* kế thừa lớp *Person*. Mỗi lớp lại có những thuộc tính riêng và kiểu giá trị thuộc tính. Mô hình gồm 11 quan hệ như bên trái hình, ví dụ quan hệ kết hợp *Management* giữa *Employee* và *Branch* thể hiện một *Employee* quản lý một chi nhánh. Các bất biến cũng được định nghĩa để ràng buộc các đối tượng ví dụ **context self : Person inv Person1:** (*self.age > 0*) để ràng buộc tuổi của đối tượng *Person* trong lớp lớn hơn 0. Hoặc **context self : Branch inv Branch1:** *self.employee->includes(self.manager)* thể hiện ràng buộc mỗi quan hệ tổng quát rằng *employee* trong *Branch* bao gồm cả đối tượng *manager*.

Tiền điều kiện của mô hình:

context Employee::raiseSalary(amount : Real) : Real

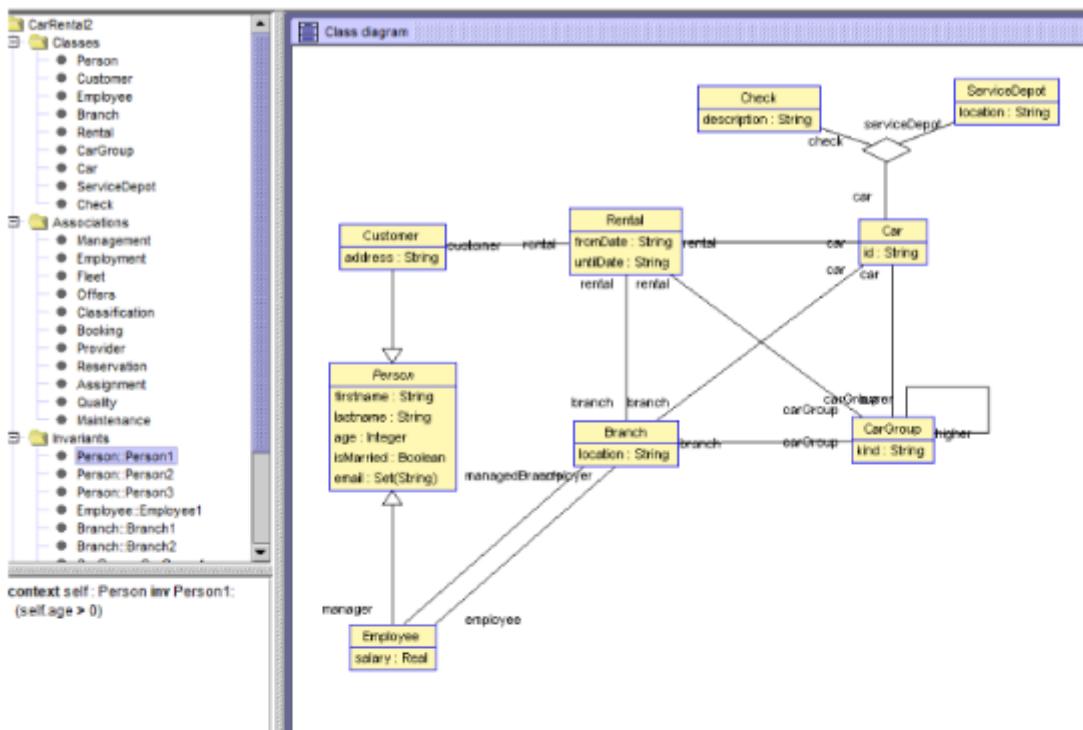
pre pre1: (amount > 0) thể hiện lương ban đầu của nhân viên lớn hơn 0.

Hậu điều kiện:

context Employee::raiseSalary(amount : Real) : Real

post post1: ((self.salary = (self.salary@pre + amount)) **and** (result = self.salary)) thể hiện lương mới bằng lương trước cộng với số tiền lương được tăng.

Chuyển đổi từ mô hình ứng dụng sang mô hình filmstrip. Các luật chuyển đổi mô hình ứng dụng sang filmstrip được mô tả như Bảng 2.25. Việc chuyển đổi này được xử lý tự động nhờ plugin filmstriping của công cụ USE.



Hình 2.24. Ví dụ minh họa biểu đồ lớp của CarRental.

Bảng 2.2. Tổng quan chuyển đổi mô hình ứng dụng sang mô hình filmstrip [8]

Application model	Filmstrip model						
class	$\rightarrow 1 : 1 \rightarrow$ application class *						
attribute	$\rightarrow 1 : 1 \rightarrow$ attribute						
operation (no return value)	$\rightarrow \Delta \rightarrow$ operation call class						
operation self object	$\rightarrow \Delta \rightarrow$ operation call class attribute						
operation parameter	$\rightarrow \Delta \rightarrow$ operation call class attribute						
operation (with return value)	$\rightarrow 1 : 1 \rightarrow$ operation in application class						
association	$\rightarrow 1 : 1 \rightarrow$ application association * composition (Snapshot, application class) * composition (Snapshot, operation call class) * composition (operation call class, Snapshot) * aggregation (application class, application class)						
operation definition	$\rightarrow 1 : 1 \rightarrow$ operation definition						
class invariant	$\rightarrow 1 : 1 \rightarrow$ application class invariant * operation self object and parameter invariants * filmstrip invariants						
operation precondition	$\rightarrow \Delta \rightarrow$ operation call class invariant						
operation postcondition	$\rightarrow \Delta \rightarrow$ operation call class invariant						
Symbol explanation:	<table border="1"> <tr> <td>$\rightarrow 1 : 1 \rightarrow$</td><td>model element is included without changes</td></tr> <tr> <td>*</td><td>new model element is created</td></tr> <tr> <td>$\rightarrow \Delta \rightarrow$</td><td>model element is included with changes applied</td></tr> </table>	$\rightarrow 1 : 1 \rightarrow$	model element is included without changes	*	new model element is created	$\rightarrow \Delta \rightarrow$	model element is included with changes applied
$\rightarrow 1 : 1 \rightarrow$	model element is included without changes						
*	new model element is created						
$\rightarrow \Delta \rightarrow$	model element is included with changes applied						

Luận văn sẽ mô tả minh họa một vài luật chuyển đổi: Thứ nhất, luật ***Chuyển đổi các lớp*** xác định mỗi lớp và thuộc tính ở mô hình ứng dụng sẽ chuyển thành lớp và thuộc tính trong mô hình filmstrip. Chỉ có một lớp mới sinh ra thêm ở mô hình filmstrip là Snapshot [8]. Các thao tác (operation) trong mô hình ứng dụng sẽ chuyển thành một lớp trong mô hình filmstrip là operation call (OperationCall) đại diện cho sự chuyển tiếp giữa các snapshot. Các phương thức này sẽ kế thừa lớp OperationCall được đặt tên có hậu tố Opc. Ví dụ Hình 2.25 thể hiện lớp *raiseSalary_EmployeeOpC* được sinh ra từ thao tác trong mô hình ứng dụng và kế thừa lớp *EmployeeOpc*. Lớp này lại kế thừa lớp trừu tượng *OperationCall*. Thứ hai, luật ***Chuyển đổi các quan hệ kết hợp (associations)*** xác định tất cả các quan hệ của mô hình ứng dụng trở thành một phần trong mô hình filmstrip [8]. Sinh ra thêm một số quan hệ hợp thành (composition) và sở hữu (aggregation). Đó là quan hệ hợp thành của lớp Snapshot và lớp gọi hàm Opc thể hiện kết quả tiếp theo của snapshot sau lời gọi hàm đó. Một quan hệ tiếp theo giữa lớp trong mô hình ứng dụng và lớp snapshot được đặt tên là Snapshot+tên lớp.

Ví dụ quan hệ *SnapshotBranch* được đặc tả trong mô hình filmstrip như sau:

```

association SnapshotBranch between
  Branch[*]
    Snapshot[1] role snapshotBranch redefines snapshot
  end

```

Mỗi lớp sẽ có một quan hệ PredSucc với chính lớp đó thể hiện trạng thái trước và sau của đối tượng trong lớp. Đối tượng pred đóng vai trò trước khi chuyển trạng thái thuộc snapshot trước, đối tượng succ thể hiện đối tượng sau chuyển trạng thái. Ví dụ quan hệ *PredSuccCustomer* được đặc tả trong mô hình filmstrip

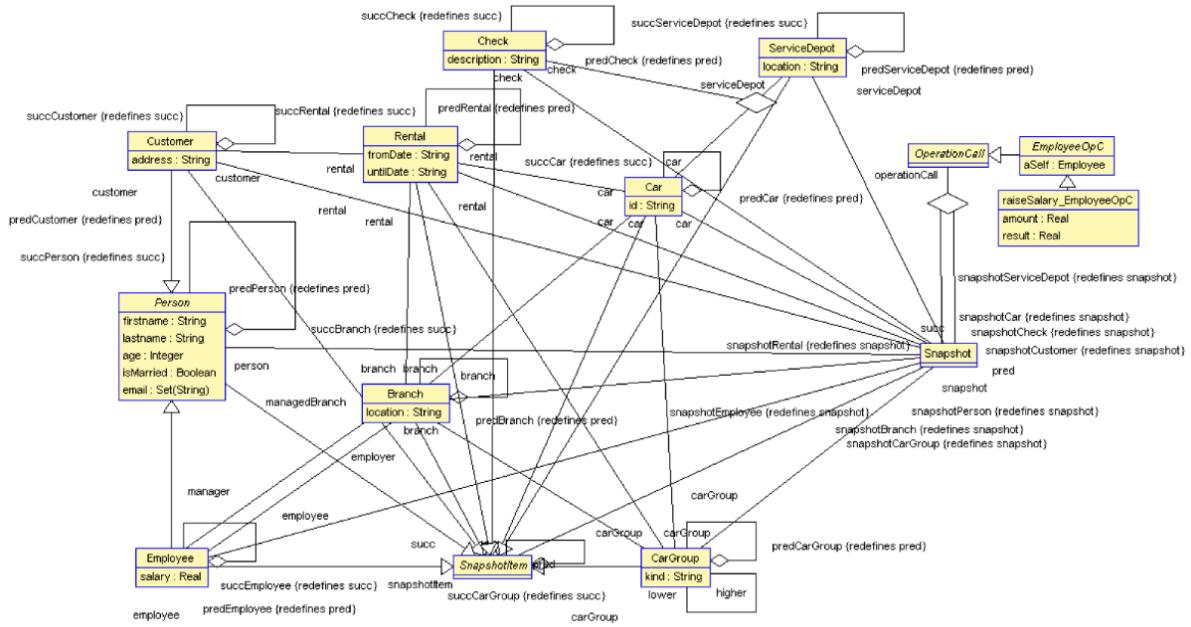
```

aggregation PredSuccCustomer between
  Customer[0..1] role predCustomer redefines pred
  Customer[0..1] role succCustomer redefines succ
  end

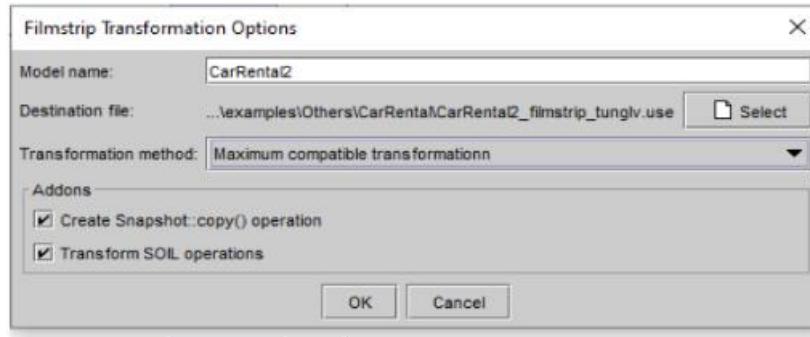
```

Thứ ba, luật ***Chuyển đổi các ràng buộc OCL*** bao gồm (1) Xử lý các thao tác cho các thao tác có giá trị trả về và các bắt biến (2) Xử lý tiền điều kiện của thao tác (3) Xử lý hậu điều kiện của thao tác. Nhìn chung, các tiền và hậu điều kiện sẽ trở thành các bắt biến trong mô hình filmstrip.

Công cụ USE sử dụng plugin filmstriping (Hình 2.26) để chuyển từ mô hình ứng dụng sang mô hình filmstrip, kết quả của việc chuyển đổi thành công là một mô hình filmstrip được biểu diễn dưới dạng biểu đồ lớp (Hình 2.25). Từ đây sẽ là tiền đề cho bước tiếp theo của quá trình sinh dữ liệu ca kiểm thử dựa trên mô hình filmstrip.



Hình 2.25. Biểu đồ lớp của mô hình filmstrip CarRental.



Hình 2.26. Công cụ filmstriping của USE.

2.4. Tổng kết chương

Như vậy các kiến thức nền tảng cho khóa luận đã được giới thiệu chi tiết trong *Chương 2*. Từ các phương pháp tiếp cận hướng mô hình, chuyển đổi mô hình, cách diễn đạt một ca sử dụng với ngôn ngữ đặc tả FRSL đến khái niệm mô hình kịch bản filmstrip là cơ sở để chương tiếp theo luận văn áp dụng vào phương pháp để xuất sinh dữ liệu ca kiểm thử ở *Chương 3*. Một số công cụ USE để kiểm chứng mô hình và chuyển đổi mô hình như Model Validator, Filmstrip cũng được giới thiệu để làm nguyên liệu cho quá trình thực nghiệm về sau.

CHƯƠNG 3. PHƯƠNG PHÁP SINH CA KIỂM THỬ TỪ MÔ HÌNH FRSL

Trong chương này, luận văn đề xuất một phương pháp tiếp cận để sinh được dữ liệu ca kiểm thử từ đặc tả ca sử dụng bằng mô hình FRSL. Bao gồm luồng xử lý tổng quan, chi tiết các bước thực hiện, các công cụ áp dụng và các luật chuyển đổi mô hình. Cuối cùng kết quả đạt được là bộ dữ liệu ca kiểm thử được cụ thể hóa trong tệp excel mô tả giá trị đầu vào, kết quả mong đợi đầu ra theo từng kịch bản trong ca sử dụng.

3.1 Giới thiệu

Việc sinh tự động các ca kiểm thử có nhiều ý nghĩa trong giai đoạn phát triển phần mềm, giúp sớm tạo ra kịch bản kiểm thử cho giai đoạn kiểm thử tích hợp (SIT) ở mức hệ thống do đó giảm chi phí thiết kế ca kiểm thử. Hơn nữa, việc sinh tự động ca kiểm thử từ yêu cầu nghiệp vụ cũng làm giảm sai sót trong quá trình thiết kế ca kiểm thử mà trước đây các kiểm thử viên thường làm thủ công.

Hiện nay, quá trình sinh tự động ca kiểm thử có nhiều khó khăn vì các yêu cầu thường được đặc tả bằng mô hình ca sử dụng ở dạng biểu đồ UML/OCL khó để tự động hóa. Đã có một số nghiên cứu để sinh được các ca kiểm thử tự động dựa trên đặc tả ca sử dụng. Như trong [12] Clémentine Nebut đã đề xuất một cấu trúc Contract cho đặc tả ca sử dụng, các contract này mô tả tiền và hậu điều kiện của ca sử dụng từ đó sinh được kịch bản kiểm thử. Hay trong [11] Tomasz Straszak cũng trình bày khái niệm và công cụ kiểm thử phần mềm hướng yêu cầu (Requirements Driven Software Testing -ReDSeT) để tạo được kiểm thử tích hợp tự động dựa trên yêu cầu. Hướng tiếp cận sử dụng một ngôn ngữ đặc tả kiểm tra TSL dựa trên siêu mô hình được định nghĩa trong EMF từ đó sinh được kịch bản kiểm thử ở mức trừu tượng. Cũng có phương pháp sinh kịch bản test dựa vào đặc tả ở mức thiết kế chức năng để tìm đường đi qua các trạng thái bằng thuật toán nhị phân.

Như vậy, có thể thấy đa số các phương pháp đều sinh ra kịch bản kiểm thử và dừng ở mức đề xuất phương pháp từ một biểu đồ UML hay ngôn ngữ đặc tả chứ chưa xác định được bộ dữ liệu cho kiểm thử. Có một số thách thức trong các phương pháp: Đặc tả chính thức ca sử dụng để thể hiện hết các ràng buộc, hành vi của mô hình. Mô hình sau khi được đặc tả cần có phương pháp để kiểm tra, xác nhận lại tính đầy đủ chính xác của các hành vi. Việc sinh được tự động ca kiểm thử phụ thuộc vào giải các ràng buộc và sinh ra mô hình biểu diễn lời giải.

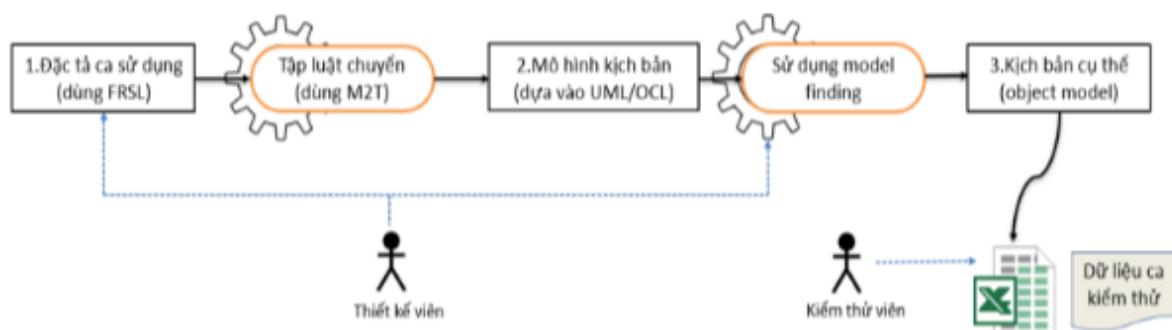
Để giải quyết thách thức trên và đạt được mục tiêu sinh dữ liệu cho ca kiểm thử, luận văn đưa ra phương pháp giải quyết: Sử dụng ngôn ngữ chuyên biệt miền FRSL để đặc tả yêu cầu; sử dụng mô hình kịch bản filmstrip được biến đổi từ mô hình FRSL để biểu diễn đầy đủ các hành vi, ràng buộc của mô hình; sử dụng plugin của công cụ USE để

xác thực mô hình và giải các ràng buộc (plugin Model Validator), tìm kiếm không gian các giá trị thỏa mãn mô hình. Kết quả của lời giải được biểu diễn dưới dạng mô hình đối tượng, thể hiện được trạng thái của các đối tượng qua mỗi bước theo từng khung cảnh sử dụng. Mỗi trạng thái này ứng với hình chụp (snapshot) trong mô hình khung cảnh filmstrip. Từ đây, bóc tách được bộ dữ liệu ca kiểm thử theo từng khung cảnh. Bộ dữ liệu ca kiểm thử này bao gồm giá trị đầu vào, giá trị đầu ra mong muốn tại mỗi bước giúp kiểm tra xem phần mềm có đáp ứng nhu cầu tính toán hay không (positive testing). Positive testing là kỹ thuật kiểm thử hệ thống bằng cách đưa vào các giá trị hợp lệ để xem phản hồi của hệ thống có hoạt động như mong đợi không. Đây là dữ liệu quan trọng cho cán bộ kiểm thử sử dụng trong quá trình thực thi kiểm thử chức năng của hệ thống ở giai đoạn kiểm thử tích hợp (System Integration Testing-SIT) hoặc kiểm thử chấp nhận (User Acceptance Testing-UAT) mức hệ thống (System Testing).

Phần còn lại của chương được tổ chức như sau: Mục 3.2 trình bày tổng quan về phương pháp sinh tự động dữ liệu cho ca kiểm thử. Mục 3.3 diễn giải bước xây dựng mô hình ứng dụng Filmstrip4FRSL gồm các luật sinh đặc tả. Mục 3.4 trình bày quá trình sinh dữ liệu ca kiểm thử từ mô hình ứng dụng Filmstrip4FRSL. Cuối cùng là phần tổng kết chương.

3.2 Tổng quan về phương pháp

Hình 3.1 biểu diễn sơ đồ luồng xử lý tổng quan của phương pháp để sinh tự động dữ liệu ca kiểm thử. Phương pháp tiếp cận theo hướng mô hình từ cấu trúc và hành vi của một mô hình nhất định. Để sinh tự động các ca kiểm thử các điều kiện như bắt biến, tiền và hậu điều kiện phải được định nghĩa chính xác, UML và OCL được sử dụng cho mục đích này. Tuy nhiên đặc thù các biểu thức OCL khá phức tạp do đó luận văn sử dụng một phương pháp tiếp cận khác là thể hiện các ràng buộc OCL dưới dạng các hình chụp (snapshot). Hành vi được xây dựng dưới dạng một chuỗi các hình chụp (snapshot) trong mô hình khung cảnh filmstrip.



Hình 3.1. Luồng xử lý tự động sinh dữ liệu ca kiểm thử.

Phương pháp để xuất sinh tự động ca kiểm thử gồm các bước sau. Đầu tiên, ca sử dụng được diễn đạt bằng ngôn ngữ FRSL. Thứ hai, mô hình ca sử dụng sau khi được diễn đạt bằng FRSL được biến đổi sang mô hình mô hình ứng dụng (Filmstrip4FRSL). Luận văn đề xuất các tập luật chuyển cho việc biến đổi này gồm *Luật chuyển từ mô hình FRSL sang mô hình ứng dụng Filmstrip4FRSL (Mục 3.3.2 và 3.3.3)* và *Luật chuyển tệp cấu hình (Mục 3.3.4)* cho việc tìm kiếm các giá trị về sau. Các luật chuyển này được cài đặt bằng ngôn ngữ chuyển đổi M2T Acceleo. Lưu ý rằng đối với dữ liệu tệp cấu hình sinh ra, người dùng cần điều chỉnh thêm (nếu cần) để không gian tìm kiếm mô hình là thỏa mãn. Thứ ba, sử dụng công cụ USE Filmstriping để chuyển từ mô hình ứng dụng Filmstrip4FRSL sang mô hình kịch bản filmstrip. Tại bước này, các đối tượng UML/OCL các tiền và hậu điều kiện mô tả ca sử dụng sẽ được biến đổi sang thể hiện ở mô hình kịch bản chỉ bao gồm các bất biến. Đây cũng là bước kiểm chứng việc chuyển đổi mô hình có hợp lệ hay không. Nếu hợp lệ, một mô hình kịch bản được sinh ra để làm đầu vào cho bước tiếp theo. Thứ tư, thực hiện xác minh mô hình kịch bản ở bước 03, kết hợp với tệp cấu hình ở bước 02 bằng công cụ Model Validator của USE. Kết quả SAT solver là thỏa mãn nếu các thuộc tính hành vi của mô hình là thỏa mãn, một kịch bản hợp lệ được chỉ ra trong không gian tìm kiếm giới hạn. Kịch bản này được biểu diễn dưới dạng một biểu đồ đổi tượng mô tả mô hình filmstrip. Biểu đồ này thể hiện giá trị, trạng thái của mỗi đối tượng trong ca sử dụng qua mỗi hình chụp (snapshot). Biểu đồ cũng thể hiện trình tự các bước được thực hiện bởi hệ thống.

Từ biểu đồ đổi tượng này, chúng ta sinh được dữ liệu cho ca kiểm thử mô tả kịch bản từng bước, các giá trị đầu vào đầu ra mong muốn tương ứng tại mỗi bước. Quá trình sinh dữ liệu có thể được thực hiện tự động hóa bằng phương pháp bóc tách dữ liệu từ quá trình chạy tập lệnh tạo biểu đồ (Hình 3.2) hoặc tùy chỉnh mã nguồn hàm tạo trạng thái hệ thống trong USE (MSystemState).

```

Command list
1. lnew step1_act_BuyTicketOpC('step1_act_buylticketopc1')
2. lnew match_step2_sys_BuyTicketOpC('match_step2_sys_buylticketopc1')
3. lnew step4_sys_BuyTicketOpC('step4_sys_buylticketopc1')
4. lnew match_step4_sys_BuyTicketOpC('match_step4_sys_buylticketopc1')
5. lnew match_step3_act_BuyTicketOpC('match_step3_act_buylticketopc1')
6. lnew step3_act_BuyTicketOpC('step3_act_buylticketopc1')
7. lnew Snapshot('snapshot1')
8. lnew Snapshot('snapshot2')
9. lnew Snapshot('snapshot3')
10. lnew Snapshot('snapshot4')
11. lnew Snapshot('snapshot5')
12. lnew Snapshot('snapshot6')
13. lnew Snapshot('snapshot7')
14. lnew Snapshot('snapshot8')
15. lnew step2_sys_BuyTicketOpC('step2_sys_buylticketopc1')
16. lnew Ticket('ticket1')
17. lnew Ticket('ticket2')
18. lnew Ticket('ticket3')
19. lnew Ticket('ticket4')
20. lnew Ticket('ticket5')
21. lnew Ticket('ticket6')
22. lnew Ticket('ticket7')

```

Hình 3.2. Tập lệnh trong quá trình tạo biểu đồ đối tượng trong USE.

3.3 Xây dựng mô hình ứng dụng Filmstrip4FRSL

Mục tiêu của phương pháp là xây dựng mô hình biểu diễn kịch bản, luận văn để xuất sử dụng mô hình filmstrip. Đặc tả ca sử dụng bằng ngôn ngữ FRSL mục đích chính là mô tả luồng nên không thể sử dụng để làm đầu vào cho chuyển đổi trực tiếp sang mô hình kịch bản filmstrip. Do đó, chúng ta cần một bước chuyển từ đặc tả FRSL về mô hình ứng dụng Filmstrip4FRSL ở dạng UML/OCL. Bước chuyển này sẽ được thực hiện tự động hóa sử dụng ngôn ngữ M2T Acceleo.

3.3.1 Các thành phần chính của mô hình ứng dụng Filmstrip4FRSL

Mô hình ứng dụng Filmstrip4FRSL gồm hai thành phần chính: thành phần thứ nhất thể hiện các lớp của đối tượng ca sử dụng, thành phần thứ hai mô tả hành vi gồm các biến, tiền và hậu điều kiện. Hai thành phần này được sinh tự động sử dụng công nghệ Acceleo (M2T). Đầu vào của mô hình ứng dụng Filmstrip4FRSL là một ca sử dụng được đặc tả bằng ngôn ngữ FRSL (*Hình 3.3*).

Với bài toán ví dụ BuyTicket, gồm các thành phần sau:

- Hai lớp gồm Customer đại diện cho đối tượng khách hàng mua vé và Ticket đại diện cho đối tượng vé.

- Một quan hệ Takes giữa 2 lớp Customer và Ticket.
- Một usecase BuyTicket gồm 4 bước chính và 1 bước thay thế. Tại mỗi bước có tiền điều kiện và hậu điều kiện tương ứng. Ví dụ Tiền điều kiện của usecase là chưa tồn tại quan hệ Takes giữa 2 đối tượng customer và ticket.

```

package ticketModel{
    class Ticket{
        attribute id: Integer;
        attribute value: Integer;
    }
    class Customer {
        attribute money: Integer;
    }
}
association Takes
    customer: Customer[0..1]
    ticket: Ticket[0..*]
end
actor Customer
end
usecase BuyTicket
    description = 'This use case describes a customer buying ticket'
    primaryActor = Customer

    ucPrecondition
    description = 'The customer wants to buy a ticket.'
        customer: Customer;
        ticket: Ticket;
        !(customer, ticket): Takes;
    end

    ucPostcondition
    description = 'The customer buys a ticket successfully.'
        (customer, ticket): Takes;
    end

    actStep step1
    description = '1. The customer wants to buy a ticket.'
    from
        customer: Customer;
        [customer.money > 0]
    to
    end
}

```

Hình 3.3. Đặc tả ca sử dụng BuyTicket bằng ngôn ngữ FRSL.

Một mô hình ứng dụng Filmstrip4FRSL sẽ có cú pháp cụ thể trong *Bảng 3.1*.

Bảng 3.1. Cấu trúc cú pháp cụ thể của mô hình ứng dụng Filmstrip4FRSL

```

model <tên model>
class <tên các đối tượng>
attributes
<tên thuộc tính của đối tượng> : <loại dữ liệu>
end
enum <tên xx>
class <Tên usecase>
attributes
<tên thuộc tính> : <loại dữ liệu>
curStep:String
matchedStep : String
operations
enter_<tên usecase>() : <tiền điều kiện của một usecase>
exit_<tên usecase>() : <hậu điều kiện của một usecase>
<tên bước>() //các bước trong usecase
match_<tên bước>()
isFinalStep(stepId:String) : <biểu thức logic>
isValidRejoinStep(rStepIndex: RejnStep) : <biểu thức logic>
isOk_ : <biểu thức logic> //hàm helper
enum2String : <biểu thức logic> //hàm helper
end
-----
-- associations -----
-----
Association <Tên quan hệ> between
<tên đối tượng1>[0...x] role <tên thẻ hiện1>
<tên đối tượng2>[0...x] role <tên thẻ hiện2>
End
-----
-- OCL constraints -----
-----
Constraints
Context tên usecase::<tên bước>
    pre preStep //Tiền điều kiện của một bước
    post nextStep //Hậu điều kiện của một bước
    post unChangedPart: //Các điều kiện ràng buộc bắt biến
    pre preSnapshot: //Tiền điều kiện của snapshot
    post postSnapshot: //Hậu điều kiện của snapshot

```

Tiếp theo là các diễn giải cho các từ khóa được mô tả như sau: Từ khóa *class*, *attributes* xác định các lớp trong ca sử dụng, lớp ca sử dụng và các thuộc tính của lớp. Thuộc tính có kiểu dữ liệu nguyên thủy như Integer, String, Boolean, Real, Enum.. Từ

khóa *operations* xác định thao tác của lớp ca sử dụng. Quá trình chuyển đổi trạng thái của hệ thống được thực hiện bởi một lệnh gọi *operations*. Từ khóa *Association* xác định các quan hệ giữa các đối tượng trong lớp. Số lượng lực lượng tham gia quan hệ là một-một hoặc một-nhiều. Từ khóa *Constraints* xác định các tiền điều kiện, hậu điều kiện, bất biến của lớp ca sử dụng và các thao tác trong ca sử dụng.

3.3.2 Sinh đặc tả lớp ứng dụng

Các nhóm luật chuyển đổi giữa đặc tả FRSL sang đặc tả mô hình ứng dụng được mô tả tổng quát trong Bảng 3.2.

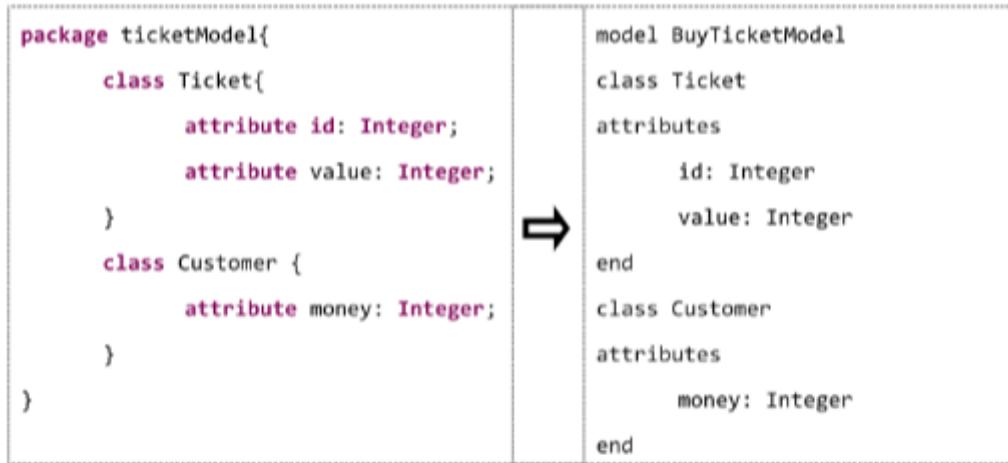
Bảng 3.2. Nhóm các luật chuyển từ FRSL sang mô hình ứng dụng Filmstrip4FRSL

STT	FRSL	Filmstrip4FRSL
1	Lớp thuộc package model	Lớp
2	Ca sử dụng	<ul style="list-style-type: none"> - Class ca sử dụng - Thêm thuộc tính <ul style="list-style-type: none"> • curStep • matchedStep
3	Các bước trong usecase	Operations
4	association	Association
5	constraints	Constraints usecase Constraint operations

Luật R1.1: Luật chuyển các lớp ứng dụng

- Các lớp ứng dụng trong mô hình FRSL sẽ chuyển thành các lớp tương ứng trong Filmstrip4FRSL: giữ nguyên tên Lớp và các thuộc tính của lớp tương ứng.

Ví dụ Hình 3.4 minh họa luật chuyển lớp ứng dụng. Các lớp *Ticket*, *Customer* được chuyển thành lớp tương ứng, các thuộc tính nguyên thủy cũng được giữ nguyên không thay đổi.



Hình 3.4. Luật chuyển các lớp ứng dụng trong FRSL model thành lớp trong Filmstrip4FRSL.

3.3.3 Sinh đặc tả lớp ca sử dụng

Luật chuyển lớp ca sử dụng từ đặc tả FRSL sang đặc tả mô hình ứng dụng bao gồm các luật được mô tả như sau:

Luật R2.1: Luật chuyển lớp ca sử dụng

- Đối tượng có kiểu usecase trong FRSL sẽ chuyển thành lớp trong mô hình ứng dụng giữ nguyên tên ca sử dụng.
- Bổ sung thêm các thuộc tính **curStep** với kiểu String để lưu trữ, để đánh dấu bước hiện thời của kịch bản ca sử dụng.
- Bổ sung thêm một thuộc tính **matchedStep (string)**, mục đích để chuẩn bị dữ liệu đầu vào cho hàm chuyển trạng thái tương ứng với bước tiếp theo. Với những biến được đánh dấu là \$ (tham biến) trong đặc tả FRSL cần phải chuẩn bị dữ liệu đầu vào.

Luật chuyển R2.2: Luật chuyển các thao tác (operation)

Chuyển các bước trong đặc tả FRSL thành các thao tác (operation) của lớp ca sử dụng trong Filmstrip4FRSL theo nguyên tắc sau:

- Bổ sung 2 thao tác *enter_TênUsecase()* và *exit_TênUsecase ()* có kiểu trả về là boolean để kiểm tra tiền và hậu điều của ca sử dụng.
- Mỗi bước của ca sử dụng trong đặc tả FRSL chuyển thành một thao tác của lớp ca sử dụng trong Filmstrip4FRSL. Với bước chuyển có tham biến cần chuẩn bị

dữ liệu đầu vào cho bước tiếp theo thì cần tạo thêm một thao tác match_ để xác định giá trị cho tham biến.

Ví dụ *Hình 3.5*: Trong ca sử dụng BuyTicket được đặc tả bằng FRSL có 4 bước chính. Trong đó tại bước 2 (step2) có biến \$avblTickets, khi chuyển sang mô hình Filmstrip4FRSL sẽ tương ứng là 4 phương thức và thêm một thao tác match_step2_sys để chuẩn bị dữ liệu đầu vào cho bước tiếp theo (step3_act).

sysStep step2 description = '2. The system displays a list of available tickets.' from $\$avblTickets: \text{Set}(Ticket);$ $[@oclCompile] = 'avblTickets = Ticket.allInstances()->select(t:Ticket t.customer.oclIsUndefined())'$ to actions $Customer <- ticketIds:$ Set(Integer) = $avblTickets->collect(id)->asSet();$ end		operations enter_BuyTicket() : Boolean = $(\text{not customer.oclIsUndefined()}) \text{ and } (\text{not ticket.oclIsUndefined()})$ $\text{and customer.ticket->excludes(ticket)}$ $\text{and curStep} = 's0'$ exit_BuyTicket() : Boolean = $\text{customer.ticket->includes(ticket)}$ $\text{and isFinalStep(curStep)}$ step1_act() step2_sys() match_step2_sys()
---	--	--

Hình 3.5. Luật chuyển bước FRSL model thành operations trong Filmstrip4FRSL.

Luật chuyển R2.3: Luật chuyển các quan hệ kết hợp (association)

- Quan hệ giữa các lớp trong đặc tả FRSL được chuyển sang quan hệ được đặc tả trong USE.
- Các thuộc tính trong lớp ca sử dụng có kiểu tham chiếu sẽ chuyển thành các quan hệ trong Filmstrip4FRSL. Tên của quan hệ này được kết hợp giữa tên lớp ca sử dụng và thuộc tính tham chiếu: tenlopcasudung_tenthuoctinhthamchieu. Lực lượng phía lớp ca sử dụng của quan hệ này là [0..1]. Với các kiểu thuộc tính

tập hợp (Set, Bag) thì lực lượng phía lớp ứng dụng [0..*]. Trường hợp còn lại thì lực lượng là [1..1].

```
[comment]-----[/comment]
[comment]-- 3.2. Generate associations for UC attributes -----[/comment]
[comment]-----[/comment]
[template public genAssoc4UcAttrs(usecase : Usecase) post(trim())]
[for (p: Property | usecase.getRefProps())]
association [usecase.name/]_[p.name/] between
[if (p.type.oclIsKindOf(CollectionType))]
    [usecase.name/]['/0..1['/] role [usecase.name.toLowerFirst()]/[p.name.toUpperCase()]/
    [p.getCollectElemType().name/]['/0..*['/] role [p.name/]
[else]
    [usecase.name/]['/0..1['/] role [usecase.name.toLowerFirst()]/[p.name.toUpperCase()]/
    [p.type.name/]['/0..1['/] role [p.name/]
[/if]
end
[/for]
[/template]
```

Hình 3.6. Mã sinh các thuộc tính tham chiếu của lớp ca sử dụng sang quan hệ.

Luật chuyển R2.4: Luật chuyển đổi các ràng buộc

Các ràng buộc ở mô hình ứng dụng Filmstrip4FRSL gồm ràng buộc chung của một ca sử dụng, ràng buộc tiền/hậu điều kiện của các thao tác (operations). Trong đó, ràng buộc của các thao tác thể hiện các điều kiện của bước hiện tại, bước tiếp theo, các thành phần không đổi (unchanged part), các tiền/hậu điều kiện của hình chụp (snapshot). Bao gồm các luật chuyển đổi: Thứ nhất, tiền và hậu điều kiện của mỗi bước trong đặc tả FRSL được chuyển thành tiền và hậu điều kiện của thao tác tương ứng của lớp ca sử dụng. Thứ hai, trong mỗi thao tác của mô hình ứng dụng có thêm các tiền và hậu điều kiện sau:

- **Pre preStep:** Đặc tả tiền điều kiện của bước hiện thời. Ràng buộc sao cho thuộc tính **curStep** trả đến bước hiện thời trước đó. Nếu là bước đầu tiên thì tiền điều kiện ca sử dụng phải thỏa mãn (bằng cách kiểm tra giá trị hàm `enter_`). Nếu bước tiếp theo này có chưa tham biến thì phải đảm bảo hàm `match_` vừa được gọi, tức `matchedStep` khác null.
- **Post nextStep:** Đặc tả hậu điều kiện của bước hiện thời. Xác định **curStep** trả đến bước hiện thời này. Nếu bước hiện tại là bước cuối của kịch bản ca sử dụng cần đảm bảo hậu điều kiện của ca sử dụng thỏa mãn (bằng cách kiểm tra giá trị hàm `exit_`).
- **Post unChangedPart:** Mô tả những ràng buộc cho những đối tượng không bị tác động bởi thao tác. Duyệt qua tất cả các thuộc tính của lớp ca sử dụng với những loại đối tượng có kiểu `isPrimitive` (là những kiểu dữ liệu nguyên thủy trong bộ dữ liệu chuẩn của OCL) được xác định là `unchanged` và thể hiện ở mô hình ứng dụng Filmstrip4FRSL.

```

9/ [comment]
10 [template public genConstraint4UnchangedPart{step : ActStep, objVars: Sequence(ObjVar), propNames: Sequence(Sequence(String))} post(trim()){
11   usecase: Usecase = step.getUsecase();
12   frslModel : FrslModel = usecase.eContainer(FrslModel);
13   usecaseName : String = usecase.name;
14   ucClass: Class = frslModel.getUsecaseClass(usecaseName);
15 }
16 [let unchangedProps: Set<Property> = ucClass.ownedProperties->select(p: Property | step.getNewObjVarNames()->excludes( p.name ) -> objVars->select(type.isPrimitive() or type.isPrimitiveSet())->collect(type)->asSet())
17 [comment][let unchangedProps: Set<Property> = ucClass.ownedProperties->select(p: Property | step.getNewObjVarNames()->excludes( p.name ) -> objV
18 [for(p: Property | unchangedProps)
19 [if ( i = 1 ) ]
20 post unChangedPart:
21   [p.name/] + [p.name/]@pre
22 [else]
23   and [p.name/] + [p.name/]@pre
24 [/if]
25 [/for]
26 [/let]
27 [let domainObjVars: Sequence(ObjVar) = objVars->select( ( not type.isPrimitive() and (not type.ocIsKindOf(CollectionType)) ) )
28 [for (class: Class | usecase.getDomClass()) before(`\tand `) separator(`\tand `)
29 [let excludedObjSet1: String =
30   if(domainObjVars->select( type.type.name = class.name )->size() > 0) then
31     ' + Set(' + domainObjVars->select(type.type.name = class.name)->collect(name)->sep(',')->toString() + ')'
32   else
33     ''
34 endif]

```

Hình 3.7. Hàm truy vấn lớp ca sử dụng để sinh ra các ràng buộc unchanged tất cả các bước.

Thứ ba, sinh ra các biến bắt tại mỗi bước ca sử dụng để mô tả ràng buộc ràng chỉ tồn tại một liên kết giữa lớp trong ca sử dụng và lớp ca sử dụng (Hình 3.8).

```

[for (class: Class | usecase.getDomClass()) before(`\tand `) separator(`\tand `)
[let excludedObjSet1: String =
  if(domainObjVars->select( type.type.name = class.name )->size() > 0) then
    ' + Set(' + domainObjVars->select(type.type.name = class.name)->collect(name)->sep(',')->toString() + ')'
  else
    ''
endif]
[comment]TODO: check it!
[let excludedObjSet2: String =
  if(objVars->select(type.isMany)->size() > 0) then
    ' + (' + objVars->select(type.isMany)->collect(name)->sep(',')->toString() + ')'
  else
    ''
endif]
[comment]/[comment]
{class.name/}.allInstances[excludedObjSet1]->forAll(x | x@pre=x[for (property : Property | class.getAllOneProps()) before(` and `) separator(` and `)
x.[property.name/]@pre=x.[property.name/][/for]
[/let]
[/for]

```

Hình 3.8. Khuôn mẫu sinh biến của các lớp trong ca sử dụng.

Thứ tư, các hành động xảy ra trước (from) và sau (to) trong đặc tả FRSL được chuyển thành tiền và hậu điều kiện của snapshot (Hình 3.9 và Hình 3.10).

```

[template private genPreSnapshotOCL(snapshot: SnapshotPattern) post(trim())]
[let andStr1: String = if( snapshot.genPreObjVarOCL()->size() = 0 ) then '' else 'and ' endif]
[let andStr2: String = if( snapshot.genLinkOCL()->size() = 0 ) then '' else 'and ' endif]
[let andStr3: String = if(andStr1 = '' and andStr2 = '') then '' else 'and ' endif]
[if(andStr1 <> '')]
[snapshot.genPreObjVarOCL()->sep(' and ')->toString()]
[/if]
[if(andStr2 <> '')]
[andStr1][snapshot.genLinkOCL()->sep(' and ')->toString()]
[/if]
[for (oclConstraint : Constraint | snapshot.constraint) before(andStr3) separator('and ')]
([oclConstraint.oclCompile()])
[/for]
[/let]
[/let]
[/let]
[/template]

```

Hình 3.9. Khuôn mẫu để sinh ra preSnapshot.

```

[template private genPostSnapshotOCL(step: ActStep) post(trim())]
[let andStr1: String = if( step.genPostObjVarOCL()->size() = 0 ) then '' else 'and ' endif]
[let andStr2: String = if( step.postSnapshot.genLinkOCL()->size() = 0 ) then '' else 'and ' endif]
[let andStr3: String = if(andStr1 = '' and andStr2 = '') then '' else 'and ' endif]
[if(andStr1 <> '')]
[step.genPostObjVarOCL()->sep(' and ')->toString()]
[/if]
[if(andStr2 <> '')]
[andStr1][step.postSnapshot.genLinkOCL()->sep(' and ')->toString()]
[/if]
[for (oclConstraint : Constraint | step.postSnapshot.constraint->select(oclCompile()<>'')) before(andStr3) separator('and ')]
([oclConstraint.oclCompile()])
[/for]
[/let]
[/let]
[/let]
[/template]

```

Hình 3.10. Khuôn mẫu để sinh ra postSnapshot.

3.3.4 Sinh tệp cấu hình

Ngoài file đầu vào là mô hình Filmstrip4FRLS dưới dạng file .use, plugin Model Validator trong công cụ USE cần một file cấu hình nữa để xác định không gian giá trị cho các thuộc tính của đối tượng. File này có đuôi . properties, chứa các thông tin cần thiết để plugin có thể nạp được và thực hiện kiểm tra.

Trong quá trình xác thực, tệp cấu hình này sẽ xác định các lớp và quan hệ liên kết như nào, là điều kiện cần thiết cho việc xây dựng các ca kiểm thử dưới dạng biểu đồ đối tượng. Tệp cấu hình bao gồm các đối tượng của mô hình ứng dụng và filmstrip, các phụ thuộc giữa chúng.

Tệp cấu hình là một danh sách các cấu hình theo các kịch bản khác nhau của ca sử dụng, bao gồm thông tin cấu hình cho các thành phần chính: **Lớp ca sử dụng, Thuộc tính của lớp ca sử dụng, Lớp tham gia ca sử dụng, Quan hệ các snapshot, Quan hệ trước trước/sau của một snapshot**. Mỗi một đối tượng trong tệp cấu hình có giá trị min & max thể hiện số lượng đối tượng tối thiểu và tối đa được sinh ra và tham gia vào ca sử dụng.

Số lượng vô hạn có giá trị bằng -1. *Hình 3.11* thể hiện khuôn mẫu chung cho tệp cấu hình và *Hình 3.12* là một ví dụ cụ thể tệp cấu hình cho ca sử dụng BuyTicket.

[scenario tên kịch bản]
Định nghĩa min max cho các kiểu dữ liệu nguyên thủy sử dụng trong ca sử dụng như Integer, String, Real
----- Usecase Class -----
Định nghĩa min max cho Lớp ca sử dụng, thuộc tính của Lớp ca sử dụng, hình chụp (snapshot) ca sử dụng, quan hệ của lớp ca sử dụng, quan hệ PredSuccess trong lớp ca sử dụng
--- DomainClass, SnapshotAssociation, PredSuccAssociation ---
Định nghĩa min max cho Lớp tham gia ca sử dụng, thuộc tính của Lớp tham gia ca sử dụng, hình chụp(snapshot) của đối tượng trong lớp tham gia ca sử dụng, quan hệ PredSuccess trong lớp tham gia ca sử dụng
----- Snapshot & Filmstrip & OpClass -----
Định nghĩa min max cho lớp hình chụp (<i>snapshot</i>), lớp <i>Filmstrip</i> , lớp <i>thao tác (operation ~ OpClass)</i>
----- Model Association -----
Định nghĩa min max cho các quan hệ của mô hình

Hình 3.11. Khuôn mẫu tệp cấu hình được sinh ra tự động.

```

-----[scenario 01234: step1_act->step2_sys->step3_act->step4_sys]
-----
Integer_min = -10
Integer_max = 10

String_max = 10

Real_min = -2.0
Real_max = 2.0
Real_step = 0.5
-----
----- Usecase Class -----
-----
BuyTicket_min = 8
BuyTicket_max = 8

BuyTicket_curStep = Set('s0','step1_act','step2_sys','step3_act','step4_sys')

BuyTicket_matchedStep = Set('match_null','match_step2_sys','match_step3_act','match_step4_sys')

BuyTicket_msg = Set('msg_null','The money is not enough!')

--BuyTicket_ticketIds=Set{ avblTickets->collect(id)->asSet() }

SnapshotBuyTicket_min = 8
SnapshotBuyTicket_max = 8

PredSuccBuyTicket_min = 7
PredSuccBuyTicket_max = 7

BuyTicket_customer_min = 8
BuyTicket_customer_max = 8

BuyTicket_ticket_min = 8
BuyTicket_ticket_max = 8

BuyTicket_avblTickets_min = 16
BuyTicket_avblTickets_max = 16
-----
--- DomainClass, SnapshotAssociation, PredSuccAssociation ---
-----
----- Ticket
Ticket_min = 16
Ticket_max = 16

```

Hình 3.12. Tệp cấu hình cho ca sử dụng BuyTicket.

Để sinh được tệp cấu hình tự động từ mô hình FRSL sử dụng phương pháp model2text, ngôn ngữ chuyển đổi Acceleo và áp dụng các luật chuyển như sau:

Luật chuyển R3.1: Luật chuyển cấu hình lớp ca sử dụng

- Mỗi cấu hình được xác định cho một kịch bản của ca sử dụng.
- Xác định khoảng giá trị số lượng đối tượng của lớp ca sử dụng. Giá trị này chính là số bước chuyển trạng thái của mô hình filmstrip (mỗi bước chuyển được thực hiện bởi một thao tác của lớp ca sử dụng).
- Xác định tập giá trị cho các thuộc tính trong ca sử dụng, trong đó: Tập giá trị của *curStep* là tất cả các bước của kịch bản đang xét. Tập giá trị của *matched-*

Step được xác định sao cho mỗi giá trị tương ứng với một bước của kịch bản ca sử dụng.

- Giá trị ngầm định của lực lượng của các Quan hệ kết hợp được xác định bằng số lượng đối tượng của lớp ca sử dụng.
- Số lượng hình chụp (snapshot) bằng với số lượng đối tượng của lớp ca sử dụng.

Hình 3.13 minh họa khuôn mẫu luật chuyển tệp cấu hình cho lớp ca sử dụng.

```
[template public genUsecasePropBoundary(usecase: Usecase, curTrace: Sequence<Step>) post(trim()){
    frslModel : FrslModel = usecase.eContainer(FrslModel);
    numberOfTransition : String = numberOfTransition(curTrace);

    [usecase.name/]_min = [numberOfTransition.add('1')/]
    [usecase.name/]_max = [numberOfTransition.add('1')/]
    [usecase.name/] = Set{[numberOfTransition.add('1')/]}

    [usecase.name/]_curStep = Set{'s0',[for (step : Step | curTrace) separator(',')][step.getId()]/[/for]}

    Snapshot[usecase.name/]_min = [numberOfTransition.add('1')/]
    Snapshot[usecase.name/]_max = [numberOfTransition.add('1')/]

    PredSucc[usecase.name/]_min = [numberOfTransition/]
    PredSucc[usecase.name/]_max = [numberOfTransition/]

    [for (p: Property | usecase.getRefProps()) separator ('\n')]
    [usecase.name/][p.name/]_min = 0
    [usecase.name/][p.name/]_max = [numberOfTransition.add('1')/]
    [/for]
}]/template]
```

Hình 3.13. Khuôn mẫu sinh luật chuyển cho lớp ca sử dụng.

Luật chuyển R3.2: Luật chuyển cấu hình cho các lớp miền

- Xác định số lượng đối tượng của lớp miền tham gia ca sử dụng. Giá trị này được xác định bằng tích của số bước chuyển trạng thái của kịch bản hiện thời và số thuộc tính có kiểu lớp miền đó. Ví dụ, ca sử dụng BuyTicket có hai thuộc tính kiểu lớp miền là Ticket và Customer và kịch bản gồm 4 bước step1_act->step2_sys->step3_act->step4_sys thì số lượng đối tượng của Ticket là $2^8 = 16$.
- Các thuộc tính trong lớp miền được xác định theo giá trị ngầm định.
- Số lượng liên kết của quan hệ kết hợp giữa hình chụp và mỗi lớp miền được xác định bằng số đối tượng lớp miền.
- Số lượng liên kết của quan hệ **PredSucc** cho mỗi lớp miền (quan hệ giữa đối tượng trước và đối tượng sau của bước hiện thời) được xác định bằng tích của số bước chuyển trạng thái của kịch bản hiện thời trừ đi 1 và số thuộc tính có kiểu lớp miền đó.

Hình 3.14 minh họa khuôn mẫu luật chuyển tệp cấu hình cho lớp tham gia ca sử dụng.

```

[comment]-----[/comment]
[comment]-- 5.1.3. Generate boundary values for domain classes -----[/comment]
[comment]-----[/comment]
[template public genDomClassPropBoundary(usecase: Usecase, numberOfTransition : String) post(trim()){
    frslModel : FrslModel = usecase.eContainer(FrslModel);
    ucClass: Class = frslModel.getUsecaseClass(usecase.name);
}]
[for (c : Class | frslModel.getDomClass()) separator ('\n')]
---- [c.name/]
[if (usecase.getDomClass()->includes(c))]
[let num : String = ucClass.getDomClassProps(c)->size().toString()]
[c.name/]_min = [num.mul(numberOfTransition.add('1'))/]
[c.name/]_max = [num.mul(numberOfTransition.add('1'))/]

Snapshot[c.name/]_min = [num.mul(numberOfTransition.add('1'))/]
Snapshot[c.name/]_max = [num.mul(numberOfTransition.add('1'))/]

PredSucc[c.name/]_min = [num.mul(numberOfTransition)/]
PredSucc[c.name/]_max = [num.mul(numberOfTransition)/]
[/let]
[else]
[c.name/]_min = 0
[c.name/]_max = 0

Snapshot[c.name/]_min = 0
Snapshot[c.name/]_max = 0

PredSucc[c.name/]_min = 0
PredSucc[c.name/]_max = 0
[/if]

```

Hình 3.14. Khuôn mẫu sinh luật chuyển cho lớp tham gia ca sử dụng.

Luật chuyển R3.3: Luật chuyển cấu hình cho lớp hình chụp (Snapshot)

- Xác định số lượng đối tượng của lớp hình chụp bằng với số lượng đối tượng ca sử dụng trong cùng một kịch bản.
- Xác định số lượng đối tượng của lớp Filmstrip bằng với số lượng đối tượng ca sử dụng trong cùng một kịch bản trừ đi 1
- Số đối tượng của lớp Operation tương ứng với mỗi thao tác của lớp ca sử dụng được xác định bằng 0 hoặc 1. Giá trị bằng 1 nghĩa là thao tác đó được gọi. Theo đó, kịch bản ca sử dụng hiện thời được xác định.

Hình 3.15 minh họa khuôn mẫu luật chuyển tệp cấu hình cho lớp hình chụp, lớp filmstrip và lớp thao tác OpClass.

```

[template public genFilmstripPropBoundary(usecase: Usecase, curTrace: Sequence(Step) ) post(trim()){
    numberofTransition : String = numberofTransition(curTrace);
}]
----- Snapshot
Snapshot_min = [numberofTransition.add('1')/]
Snapshot_max = [numberofTransition.add('1')/]

----- Filmstrip
Filmstrip_min = [numberofTransition/]
Filmstrip_max = [numberofTransition/]

----- Operation Calls
[for (step : Step | curTrace->asSet()->select(s | s.name = 's0'))]
[step.getStepId()/_][usecase.name/]OpC_min = 1
[step.getStepId()/_][usecase.name/]OpC_max = 1
[if(step.oclIsKindOf(ActStep) and step.oclAsType(ActStep).isMatchingStep())]
match_[step.getStepId()/_][usecase.name/]OpC_min = 1
match_[step.getStepId()/_][usecase.name/]OpC_max = 1
[/if]
[/for]
[for (step : Step | usecase.getSteps()->asSet() - curTrace->asSet())]
[step.getStepId()/_][usecase.name/]OpC_min = 0
[step.getStepId()/_][usecase.name/]OpC_max = 0
[if(step.oclIsKindOf(ActStep) and step.oclAsType(ActStep).isMatchingStep())]
match_[step.getStepId()/_][usecase.name/]OpC_min = 0
match_[step.getStepId()/_][usecase.name/]OpC_max = 0
[/if]
[/for]
[/template]

```

Hình 3.15. Luật chuyển cấu hình cho lớp hình chụp, lớp filmstrip và lớp thao tác OpClass.

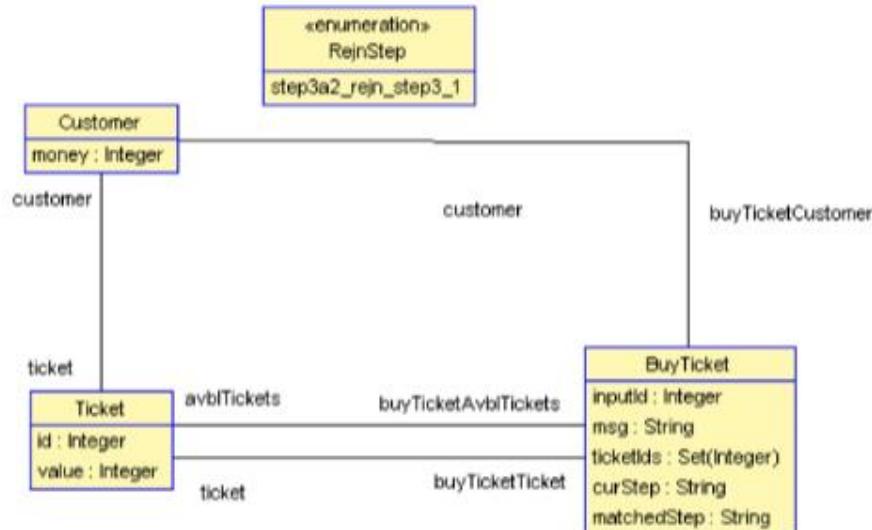
3.4 Sinh dữ liệu ca kiểm thử từ mô hình ứng dụng Filmstrip4FRSL

Từ đặc tả ca sử dụng bằng FRSL, qua các luật chuyển đổi được mô tả ở Mục 3.3.2 chúng ta thu được một mô hình ứng dụng Filmstrip4FRSL. Sử dụng mô hình ứng dụng này kết hợp với tệp cấu hình được sinh ra từ các luật chuyển (Mục 3.3.3) làm đầu vào cho đặc tả USE. Mô hình Filmstrip được tạo ra kết hợp với tệp cấu hình (xác định các đối tượng, liên kết và giá trị thuộc tính) trình xác nhận mô hình của công cụ USE (Model Validator) sẽ tự động tạo ra các ca kiểm thử trong dạng biểu đồ đối tượng (object diagram).

Sau khi đã chuyển đổi mô hình ứng dụng sang mô hình filmstrip, các thành phần động của mô hình ban đầu thể hiện các hành vi của mô hình đã được chuyển thành các thành phần tĩnh (các lời gọi thao tác chuyển thành các lớp OperationCall, tiền và hậu điều kiện chuyển thành các biến lối,...). Khi đó, ta có thể sử dụng plugin Model Validator để sinh giá trị cho mô hình filmstrip.

Hình 3.16 thể hiện mô hình ứng dụng Filmstrip4FRSL được biểu diễn dưới dạng biểu đồ lớp. Nạp mô hình này vào công cụ USE, sau đó sử dụng chức năng Chuyển đổi mô hình để biến đổi sang mô hình filmstrip. Mô hình filmstrip được biểu diễn dưới dạng biểu đồ lớp (*Hình 3.17*). Ví dụ với ca sử dụng BuyTicket, chúng ta có thể nhìn thấy các lớp mới được sinh ra như lớp hình chụp(snapshot), lớp OperationCall, BuyTicketOpC,

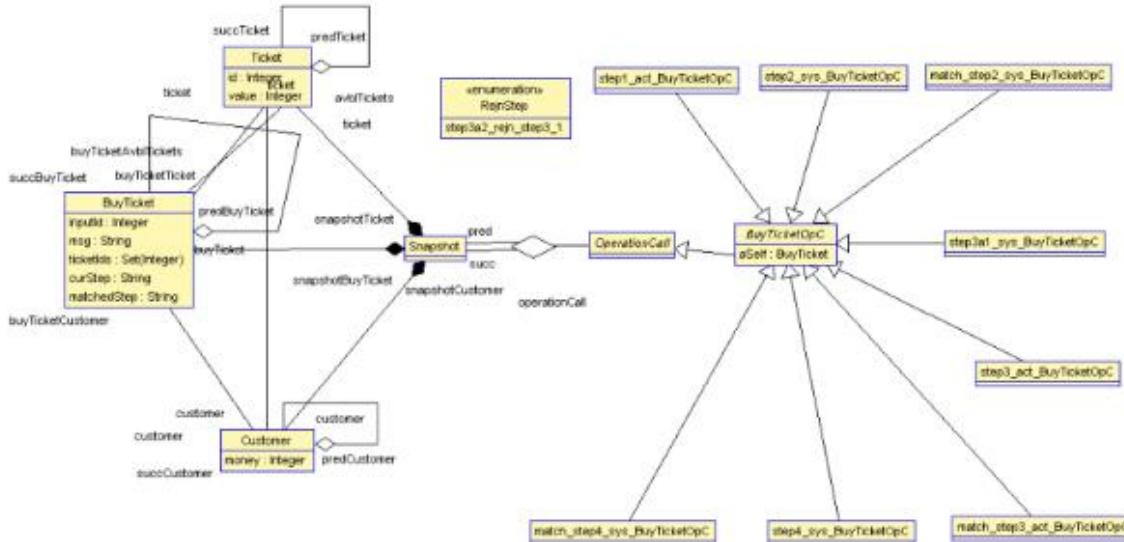
các lời gọi thao tác trong mô hình ứng dụng Filmstrip4FRSL cũng được chuyển thành các lớp như step1_act_BuyTicketOpC, step2_sys_BuyTicketOpC, match_step2_sys_BuyTicketOpC...kết thừa BuyTicketOpC. Các liên kết Pred,Succ giữa chính các lớp ca sử dụng, lớp ứng dụng và lớp các hình chụp cũng được sinh ra đầy đủ thể hiện liên kết với đối tượng trước và sau của một lớp.



Hình 3.16. Biểu đồ lớp của mô hình ứng dụng Filmstrip4FRSL.

Tiếp theo là quá trình sử dụng Model Validator để kiểm chứng mô hình và sinh dữ liệu phục vụ cho tạo các ca kiểm thử. Plugin này sử dụng dữ liệu đầu vào là mô hình film-strip được sinh ra ở trên, xác định khoảng tìm kiếm các đối tượng để đánh giá xem mô hình có thỏa mãn hay không. Người dùng cần cấu hình khoảng tìm kiếm này trước khi chạy plugin, việc cấu hình có thể làm thủ công trên giao diện USE, các giá trị phải được thay đổi sao cho kết quả trả ra là thỏa mãn do vậy sẽ tốn khá nhiều công sức, đòi hỏi kiến thức về cá công cụ lắn mô hình.

Để giải quyết vấn đề này, luận văn đã đưa ra phương pháp để tạo tự động tệp cấu hình dựa trên các luật chuyển được mô tả ở Mục 3.3.3. Thuật toán để sinh tệp này sao cho thỏa mãn không gian tìm kiếm của Model Validator cũng là một vấn đề khó và phức tạp, mặc dù đã sinh được dữ liệu cho tệp tuy nhiên vẫn cần chỉnh thủ công tại một số tham biến. Ví dụ Hình 3.18 là tệp cấu hình được sinh ra tự động của ca sử dụng BuyTicket.



Hình 3.17. Biểu đồ lớp của mô hình Filmstrip.

```

=====
[scenario 01234: step1_act->step2_sys->step3_act->step4_sys]
=====
Integer_min = -10
Integer_max = 10

String_max = 10

Real_min = -2.0
Real_max = 2.0
Real_step = 0.5
-----
Usecase Class -----
BuyTicket_min = 8
BuyTicket_max = 8

BuyTicket_curStep = Set('s0','step1_act','step2_sys','step3_act','step4_sys')
BuyTicket_matchedStep = Set('match_null','match_step2_sys','match_step3_act','match_step4_sys')
BuyTicket_msg = Set('msg_null','The money is not enough!')
--BuyTicket_ticketIds=Set{ avblTickets->collect(id)->asSet() }

SnapshotBuyTicket_min = 8
SnapshotBuyTicket_max = 8

PredSuccBuyTicket_min = 7
PredSuccBuyTicket_max = 7

BuyTicket_customer_min = 8
BuyTicket_customer_max = 8

```

Hình 3.18. Tệp cấu hình được sinh ra tự động cho ca sử dụng BuyTicket.

Sau khi đã có dữ liệu cấu hình, plugin Model Validator bắt đầu thực hiện quá trình kiểm tra và xác nhận mô hình. Nếu kết quả trả ra là SATISFIABLE có nghĩa là mô hình

đã thỏa mãn. *Hình 3.19* thể hiện kết quả giải mô hình ca sử dụng BuyTicket với thời gian là 1753ms.

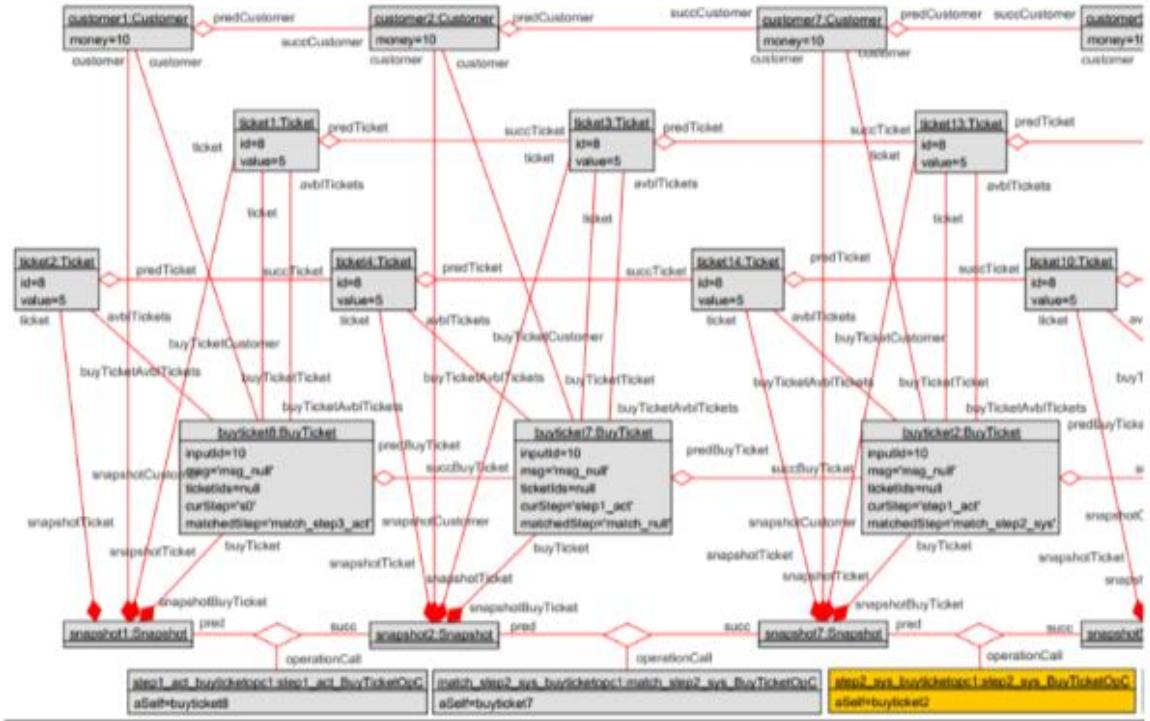
```
Log
compiling specification E:\8.cae\hccl2022\fsl-use10112022\use\use-validator\use-validator\test\cases\test0\itungv_BuyTicket\instro...
done.
Model BuyTicketModel (14 classes, 11 associations, 42 invariants, 21 operations, 1 pre-/postcondition, 0 state machines)
INFO: Start model transformation for 'BuyTicketModel'
INFO: Use 'modelvalidator-downloadsolvers' to automatically download and install additional solver libraries.
WARN: Only default SetSolver 'DefaultSAT4J' can be used! Failed to get field handle to set library path
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.avt$Tickets->collect|{Selem1 : Ticket | {Selem1.id}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
WARN: Collect operation 'self.a$elf.succBuyTicket.predBuyTicket.avt$Tickets->colecflested|{1 : Ticket | {1.succTicket}' results in unsupported type 'Bag'. It will be interpreted as 'Set'.
INFO: Invariant transformation successful
INFO: Model transformation successful
INFO: Translation time (USE to Kodkod): 939 ms
INFO: Model configuration successful
INFO: Searching solution with SetSolver 'DefaultSAT4J' and bitwidth 6...
INFO: SATISFIABLE
INFO: Translation time (Kodkod to SAT): 681 ms; Solving time: 1753 ms
```

Hình 3.19. Kết quả tìm kiếm mô hình của plugin Model Validator trong USE.

Các đối tượng sinh ra được biểu diễn dưới dạng biểu đồ đối tượng như *Hình 3.20*. Các bước trong kịch bản tệp cấu hình được sinh ra đầy đủ theo đúng miền giá trị lớn nhất, nhỏ nhất của từng đối tượng Lớp ca sử dụng, Lớp tham gia ca sử dụng, Lớp ảnh chụp. Mỗi bước chuyển trong kịch bản ca sử dụng đều gọi các thao tác tương ứng.

Ví dụ: để chuyển từ snapshot1 sang snapshot2 sẽ thực hiện lời gọi thao tác step1_act_buylticketop1. Giá trị thuộc tính value bằng 5 của đối tượng Ticket không đổi qua các bước do ràng buộc của bắt biến. Từ snapshot3 sang snapshot8 (tương ứng kịch bản từ bước 3 sang bước 4 của ca sử dụng) thì giá trị thuộc tính money của đối tượng Customer bị thay đổi do khách hàng đã mua một vé.

Biểu đồ đối tượng trong *Hình 3.20* tương ứng với đặc tả ca kiểm thử trong Excel ở *Hình 3.21*. Trong đó, mô tả chi tiết kịch bản kiểm thử với dữ liệu đầu vào và kết quả mong đợi đầu ra.



Hình 3.20. Biểu đồ đối tượng được tạo ra thể hiện một phần kịch bản của ca sử dụng BuyTicket .

Test Case	Test Map	Test Data	Expected Result	Description	Actual Result	Decision Test	Date	Last Test
Pre condition: Khi đang tồn tại trạng thái giá trị của Customer và Ticket là: là một đối tượng và chưa thuộc vào bất kỳ biến mới nào (tương ứng Customer)								
Standard TC1	abc...xyz...abc...xyz...abc...xyz							
(BuyTicket-1) snapshot1 (snap1,act)	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot1 = snap1,act	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot1 = snap1,act	Khách hàng muốn mua vé số (step1)		Pass			
(BuyTicket-2) snapshot2 (snap2,act)	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot2 = snap2,act	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot2 = snap2,act	Khách hàng đã bắt đầu mua vé số (step2)		Pass			
(BuyTicket-3) snapshot3 (snap3,act)	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot3 = snap3,act	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot3 = snap3,act	Khách hàng mua vé số thành công (step3)		Pass			
(BuyTicket-4) snapshot4 (snap4,act)	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot4 = snap4,act	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot4 = snap4,act	Khách hàng mua vé số thành công (step4)		Pass			
(BuyTicket-5) snapshot5 (snap5,act)	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot5 = snap5,act	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot5 = snap5,act	Khách hàng chưa mua vé số (step5)		Pass			
(BuyTicket-6) snapshot6 (snap6,act)	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot6 = snap6,act	Customer1.money = 10 Ticket1.id=8 Ticket1.value=5 Ticket1.quantity=40 snapshot6 = snap6,act	Khách hàng đã mua vé số (step6)		Pass	Success	2019-01-10	

Hình 3.21. Đặc tả kịch bản ca kiểm thử trong Excel.

3.5. Tổng kết chương

Trong chương này, luận văn đã trình bày chi tiết về phương pháp sinh ca kiểm thử cụ thể là dữ liệu ca kiểm thử. Ý tưởng của phương pháp là sự kết hợp giữa kỹ nghệ hướng mô hình và công cụ hướng mô hình. Cụ thể, để xuất các luật chuyển đổi mô hình, biểu diễn ca sử dụng với mô hình ứng dụng Filmstrip4FRSL và mô hình kịch bản filmstrip. Sử dụng công cụ USE để xác minh ràng buộc mô hình; kiểm chứng mô hình kịch bản filmstrip để tạo ra kết quả hợp lệ trong không gian tìm kiếm được cấu hình trong tệp .properties.

CHƯƠNG 4. CÀI ĐẶT VÀ THỰC NGHIỆM

Trong chương này, luận văn sẽ mô tả cài đặt công cụ và thực nghiệm phương pháp trong Chương 3 với bài toán cụ thể. Tiếp theo, luận văn sẽ đánh giá kết quả thực nghiệm, từ đó đưa ra những kết luận và hướng phát triển tiếp theo.

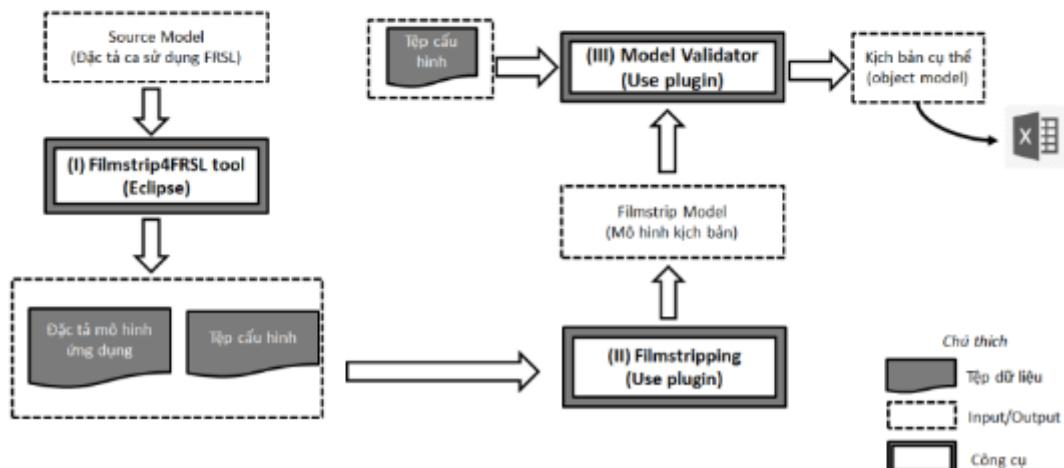
4.1 Giới thiệu

Từ những kiến thức nền tảng Chương 2 và tổng quan về phương pháp Chương 3, trước tiên chúng ta cần phát triển một công cụ để chuyển đổi đặc tả ca sử dụng FRSL sang mô hình ứng dụng (Filmstrip4FRSL) được mô tả ở Mục 3.3. Cấu trúc của công cụ được trình bày ở Mục 4.2.

Trong Mục 4.2, luận văn cũng trình bày chức năng của các công cụ hỗ trợ (USE tool). Tiếp theo, Mục 4.3 luận văn sẽ trình bày bài toán thực nghiệm cụ thể để đưa ra các kết quả đánh giá, kết luận tại Mục 4.4.

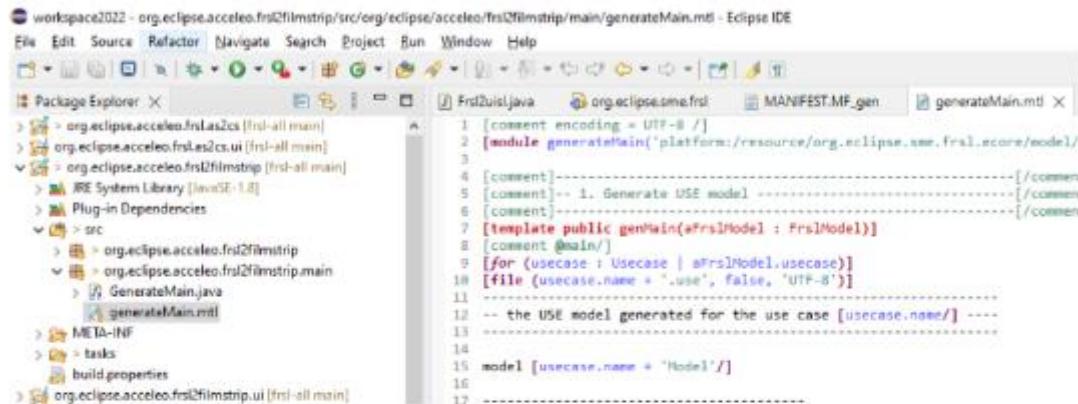
4.2 Công cụ hỗ trợ

Luận văn mô tả phương pháp để sinh được dữ liệu ca kiểm thử từ đặc tả yêu cầu chức năng biểu diễn bởi mô hình FRSL. Phương pháp sử dụng các công cụ hướng mô hình để cài đặt và đánh giá. Như mô tả trong Hình 4.1, các công cụ hỗ trợ cho phương pháp gồm ba thành phần chính : (I) Thành phần chuyển đổi đặc tả và sinh tệp cấu hình (Filmstrip4FRSL tool); (II) Thành phần chuyển đổi từ mô hình ứng dụng sang mô hình kịch bản filmstrip (Filmstripping); (III) Thành phần kiểm tra mô hình và giải bài toán SAT (Model Validator).



Hình 4.1. Các bước xử lý trong phương pháp để sinh dữ liệu cho ca kiểm thử.

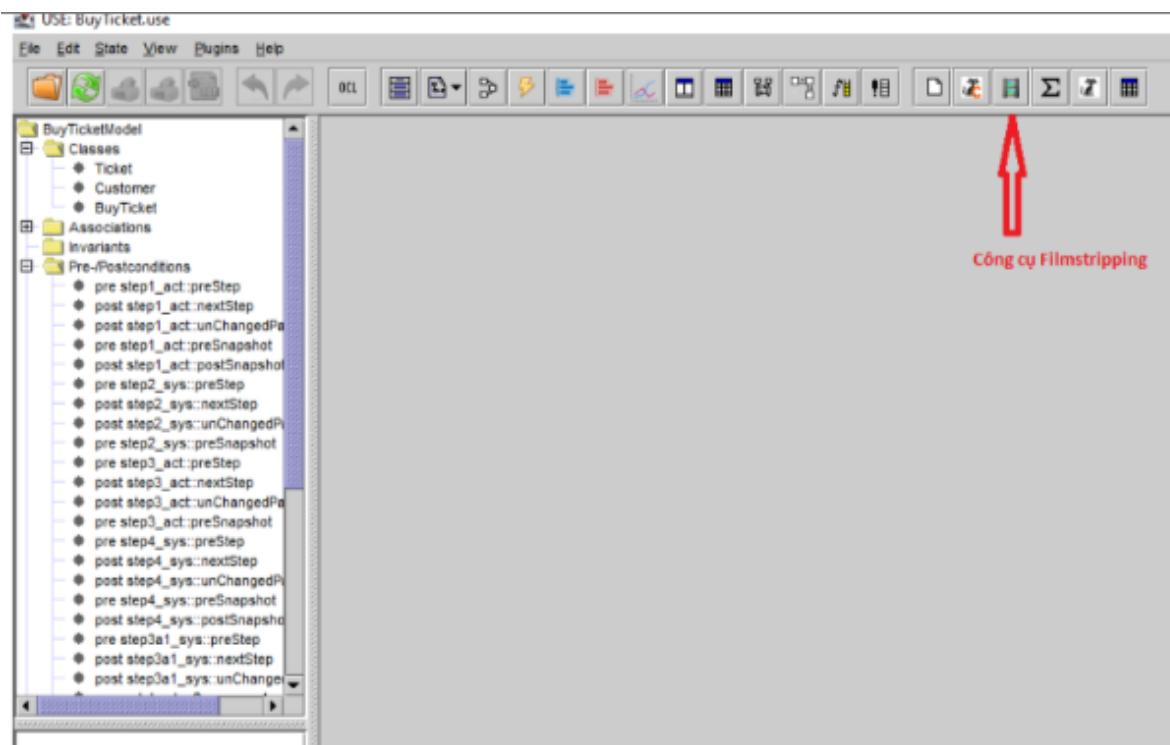
Thành phần thứ nhất, **mô đun Filmstrip4FRSL** cho phép chuyển đổi đặc tả ca sử dụng FRSL sang đặc tả USE gọi là mô hình ứng dụng. Công cụ này được phát triển trên Eclipse IDE, tích hợp với công cụ đặc tả ca sử dụng dạng mô hình FRSL. Chức năng tự động chuyển từ mô hình FRSL sang mô hình ứng dụng bằng công nghệ Acceleo. Bản chất của công cụ là một plugin, tích hợp với plugin của công cụ FRSL để sinh tự động sang đặc tả USE. Hình 4.2 thể hiện các package cần thiết để xây dựng công cụ, trong đó tệp generateMain.mtl để cài đặt luật chuyển mô hình.



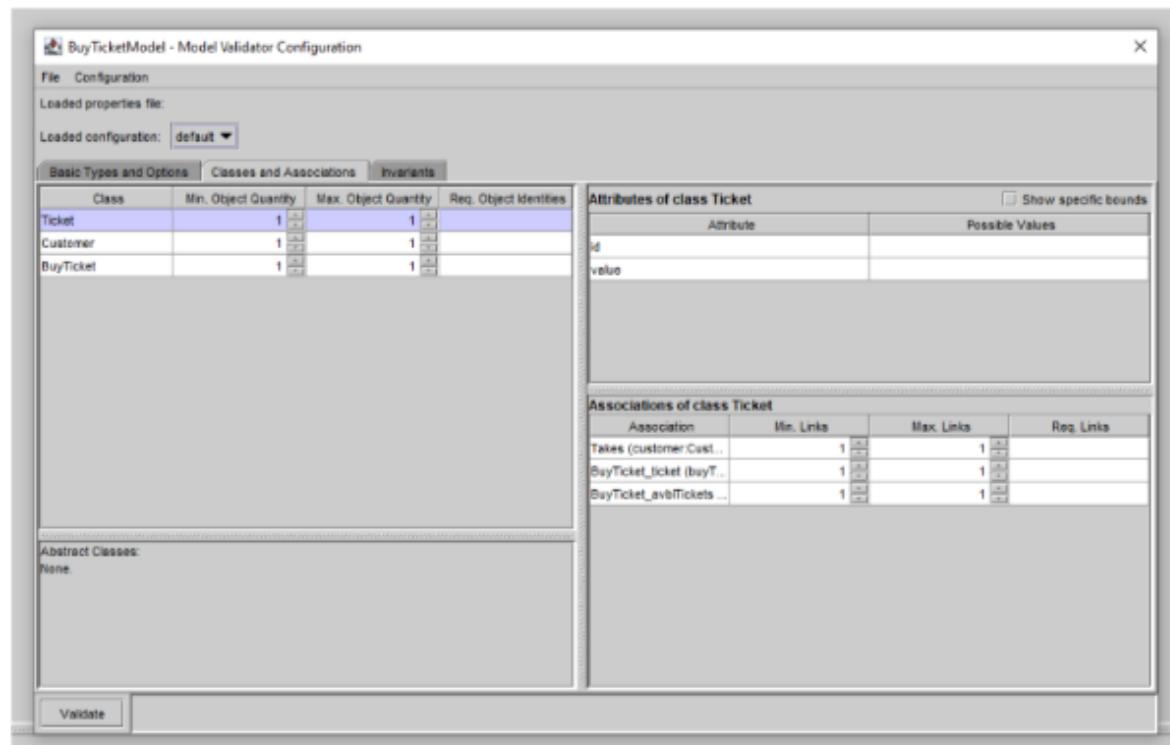
Hình 4.2. Các package của công cụ Filmstrip4FRSL tool trên Eclipse IDE.

Thành phần thứ hai, **Filmstripping** là một plugin của công cụ USE. Thành phần này cho phép chuyển đổi mô hình ứng dụng Filmstrip4FRSL sang mô hình kịch bản filmstrip. Sau khi mô hình ứng dụng Filmstrip4FRSL được tải vào công cụ USE, sử dụng plugin (Hình 4.3) để thực hiện chuyển đổi mô hình và lưu vào ổ đĩa.

Thành phần thứ ba, mô đun **Model Validator** là một plugin trong USE, thành phần này thực hiện xác minh thuộc tính hành vi trong mô hình filmstrip. Các bước gồm : (1) kiểm tra thuộc tính hành vi của mô hình filmstrip. Một thuộc tính hành vi được thể hiện như biểu đồ tuần tự mô hình ứng dụng và có thể phân tích bằng cách tự động xây dựng một kịch bản (biểu đồ đối tượng của mô hình filmstrip) [20]. Nếu các thuộc tính hành vi là thỏa mãn, một kịch bản hợp lệ được chỉ ra trong không gian tìm kiếm giới hạn. Bước này được thực hiện tự động sử dụng tệp cấu hình mô tả không gian tìm kiếm hữu hạn (Hình 4.4); (2) chuyển đổi biểu đồ đối tượng thành một biểu đồ trình tự thể hiện trạng thái của mỗi đối tượng qua từng kịch bản.



Hình 4.3. Công cụ Filmstripping trong USE.



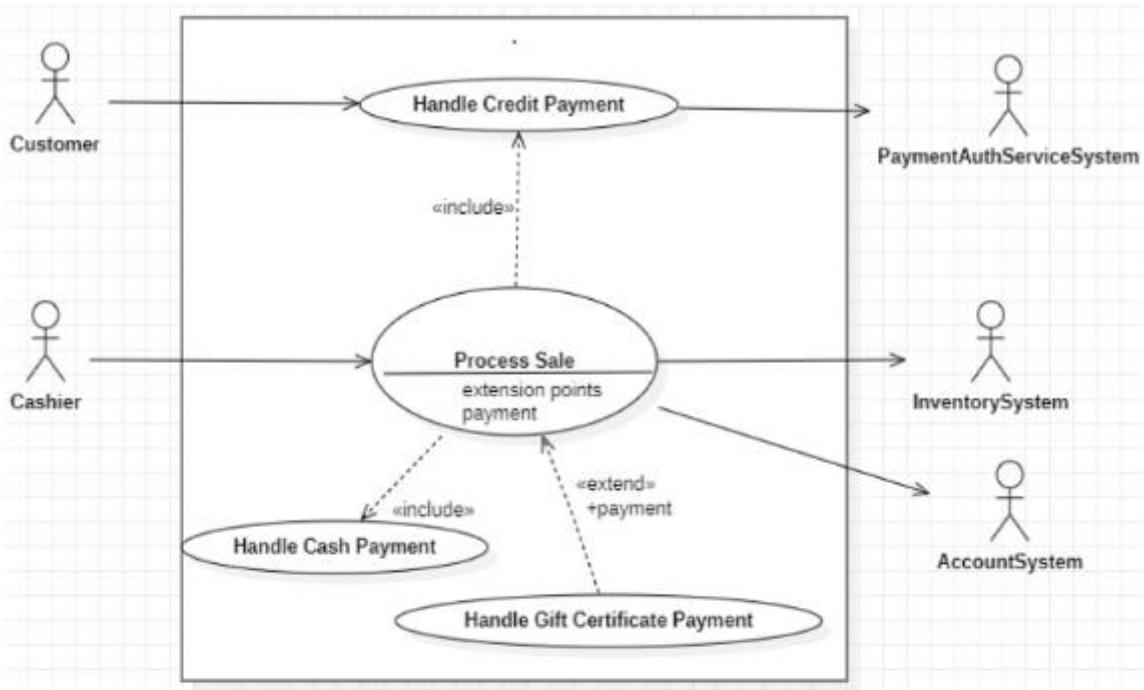
Hình 4.4. Cấu hình không gian tìm kiếm cho Model Validator.

4.3 Thực nghiệm 1: Sinh mô hình ứng dụng

Để thực nghiệm sinh mô hình ứng dụng từ công cụ xây dựng được, luận văn lựa chọn ca sử dụng của hệ thống Thiết bị bán hàng (Point of Sale – POS) (Hình 4.5). Ca sử dụng có ý nghĩa trong việc xử lý thanh toán giao dịch mua bán hàng, quy trình được mô tả như sau:

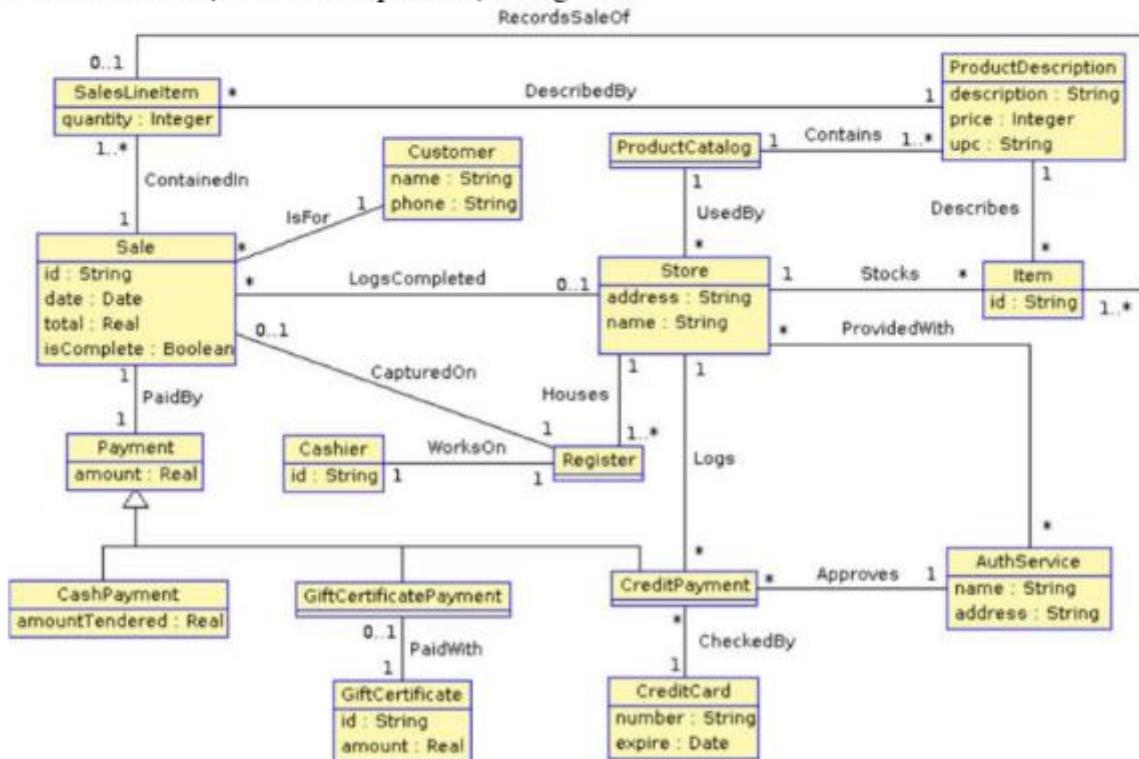
1. Khách hàng đến quầy thanh toán với các mặt hàng cần mua.
2. Nhân viên thu ngân (Cashier) sử dụng hệ thống POS để ghi lại từng mặt hàng.
3. Hệ thống hiện thị chi tiết mặt hàng, giá và tổng tiền.
4. Khách hàng nhập thông tin thanh toán. Tại bước này, Khách hàng có thể chọn phương thức thanh toán bằng tiền mặt (Handle Cash Payment) hoặc Thẻ tín dụng (Handle Credit Payment) và sử dụng phiếu quà tặng nếu có (Handle Gift Certificate Payment). Nếu sử dụng phương thức thanh toán Thẻ tín dụng, Khách hàng cần xác thực định danh với hệ thống thanh toán thẻ tín dụng (PaymentAuthServiceSystem)
5. Hệ thống cập nhật hàng tồn kho (InventorySystem) và hệ thống kế toán (AccountSystem).
6. Khách hàng ra về với hóa đơn và hàng hóa.

Tác nhân chính Cashier tham gia trực tiếp vào ca sử dụng ProcessSale để đạt được mục đích chính. Ngoài ra, hệ thống tương tác với các tác nhân phụ bao gồm AccountSystem, InventorySystem, PaymentAuthServiceSystem. Trong Bảng 4.1 minh họa ca sử dụng ở dạng văn bản.



Hình 4.5. Biểu đồ UML ca sử dụng rút gọn của hệ thống POS.

Hình 4.6 Minh họa biểu đồ lớp của hệ thống POS.



Hình 4.6. Biểu đồ lớp của hệ thống POS.

Bảng 4.1. Mô tả văn bản ca sử dụng của hệ thống POS

<p>Use Case UC1: Process Sale</p> <p>Tác nhân chính: Thu ngân</p> <p>Tiền điều kiện: Thu ngân được xác định và xác thực.</p> <p>Postconditions: Đơn hàng được lưu. Thông tin kê toán được cập nhật.</p> <p>Luồng chính:</p> <ol style="list-style-type: none"> 1. Khách hàng đến quầy POS với hàng hóa và dịch vụ cần mua. 2. Thu ngân khởi tạo đơn hàng. 3. Hệ thống sẵn sàng cho bán hàng. 4. Thu ngân nhập mã hàng. 5. Hệ thống ghi lại mã hàng và hiển thị mô tả sản phẩm, giá và tổng tiền. 4-5. Thu ngân lặp lại bước 4-5 cho đến khi hoàn thành. 6. Hệ thống hiển thị tổng đơn hàng bao gồm cả thuế. 7. Thu ngân thông báo với Khách hàng tổng số tiền cần thanh toán. 	<p>8. Ca sử dụng Handle Cash Payment được gọi.</p> <p>9. Hệ thống ghi lại lịch sử đơn hàng.</p> <p>10. Hệ thống hiển thị hóa đơn thanh toán.</p> <p>11. Khách hàng rời đi với hàng hóa và hóa đơn.</p> <p>Luồng thay thế:</p> <ol style="list-style-type: none"> 1a. Khách hàng hoặc Người quản lý chỉ định tiếp tục đơn hàng đã tạm dừng. 1. Nhân viên thu ngân tiếp tục thực hiện và nhập ID để truy vấn thông tin đơn hàng. 2. Hệ thống hiển thị trạng thái đơn hàng với tổng tiền. 2a. Đơn hàng không tìm thấy. 1. Hệ thống báo lỗi cho Thu ngân. 2. Thu ngân khởi tạo một đơn hàng mới. 8a. Khách hàng muốn thanh toán bằng Thanh toán tín dụng. 8a1. Ca sử dụng HandleCreditPayment được gọi. <p>Điểm mở rộng:</p> <p>E1. Thanh toán bằng quà tặng. Điểm mở rộng xảy ra ở Bước 8 khi Khách hàng muốn thanh toán bằng phiếu quà tặng.</p>
--	--

<p>UC2 : Handle Cash Payment</p> <p>Tác nhân chính: Thu ngân</p> <p>Tiền điều kiện : Khách hàng sẵn sàng thanh toán tiền hàng bằng tiền mặt.</p> <p>Hậu điều kiện : Đơn hàng được thanh toán bằng tiền mặt.</p> <p>Luồng chính:</p> <ol style="list-style-type: none"> 1. Khách hàng muốn thanh toán bằng tiền mặt. 2. Thu ngân nhập số tiền. 3. Hệ thống hiển thị số tiền cần trả và mở ngăn kéo tiền mặt. 4. Thu ngân thu tiền mặt và trả lại tiền thừa cho Khách hàng. 	<p>UC3: Handle Credit Payment</p> <p>Tác nhân chính: Khách hàng</p> <p>Tác nhân phụ Hệ thống dịch vụ xác thực thanh toán (PaymentAuthServiceSystem)</p> <p>Tiền điều kiện: Khách hàng sẵn sàng thanh toán hóa đơn bằng thanh toán tín dụng.</p> <p>Hậu điều kiện: Khách hàng mua hàng. Thu ngân lấy hóa đơn. Đơn hàng được xử lý.</p> <p>Luồng chính:</p> <ol style="list-style-type: none"> 1. Khách hàng nhập thông tin tài khoản tín dụng của mình. 2. Hệ thống gửi yêu cầu thanh toán tới Hệ thống dịch vụ ủy quyền thanh toán và yêu cầu phê duyệt thanh toán. 3. Hệ thống nhận duyệt thông tin và gửi kết quả đến Thu ngân 4. Hệ thống ghi lại thanh toán tín dụng và gồm cả thông tin phê duyệt. 5. Hệ thống hiển thị màn hình nhập chữ ký thanh toán tín dụng. 6. Thu ngân yêu cầu Khách hàng ký tên. <p>Khách hàng nhập chữ ký.</p>
<p>UC4: Handle Gift Certificate Payment</p> <p>Tác nhân chính : Thu ngân</p> <p>Tiền điều kiện : Khách hàng muốn thanh toán với phiếu quà tặng.</p> <p>Hậu điều kiện : Hóa đơn được thanh toán một phần với phiếu quà tặng.</p> <p>Luồng chính :</p> <ol style="list-style-type: none"> 1. Khách hàng đưa phiếu quà tặng cho Thu ngân. Thu ngân nhập mã ID phiếu quà tặng 2. Hệ thống hiển thị số dư của phiếu quà tặng. 3. Thu ngân nhập số tiền để thanh toán một phần với phiếu quà tặng. 2. Hệ thống nhận thanh toán với phiếu quà tặng. 	

4.3.1 Xây dựng mô hình FRSL

Ca sử dụng POS sau đó được đặc tả bằng FRSL, bước này được thực hiện thủ công bởi người thiết kế mô hình. Hình 4.7 minh họa ca sử dụng ProcessSale đặc tả bằng FRSL.

<pre> package posModel{ class Date{ attribute value:String; } class Customer { attribute name: String; } class Cashier { attribute name: String; } class Sale { attribute id: String; attribute date: Date; attribute total: Real; attribute isComplete: Boolean; } class Register { } class Store{ attribute address: String; attribute name: String; } class Item { attribute id: String; } class ProductDescription { attribute desc: String; attribute price: Real; attribute tax: Real; attribute upc: String; } class ProductCatalog{ } class SalesLineItem { attribute quantity: Integer; } class Payment { attribute amount: Real; } class CashPayment extends Payment { } } </pre>	<pre> @association ProvidedWith store: Store[*] authService: AuthService[*] end @association Stocks store: Store [1] item: Item [*] end @association RecordsSaleOf salesLineItem: SalesLineItem[0..1] item: Item[+] end @association ContainedIn sale: Sale[1] salesLineItem: SalesLineItem[+] end @association Describes prdDesc: ProductDescription[1] item: Item[*] end @association DescribedBy salesLineItem: SalesLineItem[*] productDesc: ProductDescription[1] end @association Contains prdtCtg: ProductCatalog[1] prdtDesc: ProductDescription[1..*] end @association UsedBy prdtCtg: ProductCatalog[1] </pre>
<pre> usecase ProcessSales description = 'This use-case describes the process sale of Cashier' primaryActor = Cashier secondaryActors = {InventorySystem, AccountingSystem} ucPrecondition description = 'Cashier is identified and authenticated. Customer is ready to buy.' _cashier: Cashier;--đối tượng nhân viên thu ngân _customer: Customer; _sale: Sale; _pos: Register; _items: Set(Item); _store: Store; pos: Register;--đối tượng có kiểu giao thu ngân cashier: Cashier;--đối tượng có kiểu nhân viên thu ngân (_sale, _customer): IsFor; (_cashier, _pos): WorksOn;--nhân viên thu ngân cần phải làm việc (_pos, pos): _Tracks; (_cashier, cashier): _Tracks; (store, pos): Houses; [_sale.isComplete = false] [_items->isEmpty() = false] end ucPostcondition description = 'Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipts are generated.' sale: Sale; (_sale, sale):_Tracks; (store, sale): LogsCompleted; !(sale, pos): CapturedOn; !(sale, pos): CapturedOn; [_sale.isComplete = true] [_sale.total.oclsIsUndefined() = false] [_sale.isComplete = true] </pre>	<pre> actStep step01 description = '1. Customer arrives at POS checkout with goods and/or services to purchase'. from _pos: Register; _customer: Customer; _sale: Sale; _items: Set(Item); (_sale, _customer): IsFor; [_items->isEmpty() = false] to end actStep step02 description = '2. Cashier starts a new sale'. from _pos: Register; _cashier: Cashier; _sale: Sale; (_cashier, _pos): WorksOn; to (_sale, _pos): CapturedOn; [_sale.total = 0] [_sale.isComplete = false] end sysStep step03 description = '3. The system creates a new sale and requires the cashier to enter items'. from _sale: Sale; _pos: Register; _cashier: Cashier; \$pos: Register; \$cashier: Cashier; \$curDate: Date; (cashier, pos): WorksOn; (_pos, pos): _Tracks; (_cashier, cashier): _Tracks; </pre>

```

to
    sale: Sale;
    (sale, pos): CapturedOn;
    (_sale, sale): _Tracks;
    [sale.oclIsUndefined() = false]
    [sale.total = 0]
    [sale.date = curDate]
actions
    Cashier <- saleInfor: Sequence<OclAny> = Sequence[sale.id, sale.total];
    Cashier <- cashierInfor: Sequence<OclAny> = Sequence[cashier.name];
end

actStep step04
description = '4. Cashier enters item identifier'
from
    _sales: Sale;
    _items: Set<Item>;
    $_item: Item;
    [_items->includes(_item)]
    [_item.id = itemId]
    [_item.salesLineItem->isEmpty()]
to
    _salesLineItem: SalesLineItem;
    (_salesLineItem, _sale): ContainedIn;
    (_salesLineItem, _item): RecordsSaleOf;
    [_salesLineItem.oclIsUndefined() = false]
    [_salesLineItem.quantity = 1]
actions
    Cashier -> itemId: String;
end

sysStep step05
description = '5. System records sale line item and presents
    item description, price, and running total with calculated tax.
    Price calculated from a set of price rules.'
from
    _item: Item;
    _salesLineItem: SalesLineItem;
    sale: Sale;

```

```

$Item: Item;
ProdDesc: ProductDescription;
(ProdDesc, Item): Describes;
(_Item, Item): _Tracks;
to
    salesLineItem: SalesLineItem;
    (salesLineItem, sale): ContainedIn;
    (salesLineItem, item): RecordsSaleOf;
    (salesLineItem, prodDesc): DescribedBy;
    (_salesLineItem, salesLineItem): _Tracks;
    [_salesLineItem.oclIsUndefined() = false]
    [_salesLineItem.quantity = 1]
    [sale.total = sale@pre.total + prodDesc.price * prodDesc.tax]
actions
    Cashier <- itemInfor: Sequence<OclAny>
    = Sequence[prodDesc.desc, prodDesc.price, prodDesc.upc];
    Cashier <- saleInfor: Sequence<OclAny>
    = Sequence[sale.total];
end

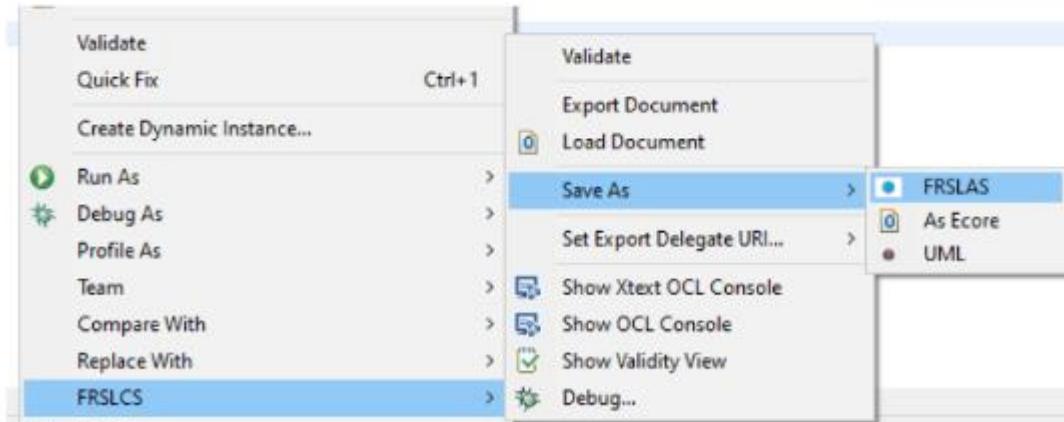
rejoinStep step04
description = '(*) Cashier repeats steps 4-5 until indicates done.'
when
    _items: Set<Item>;
    $_item: Item;
    [_items->includes($_item)]
    [_item.salesLineItem.oclIsUndefined()]
end

```

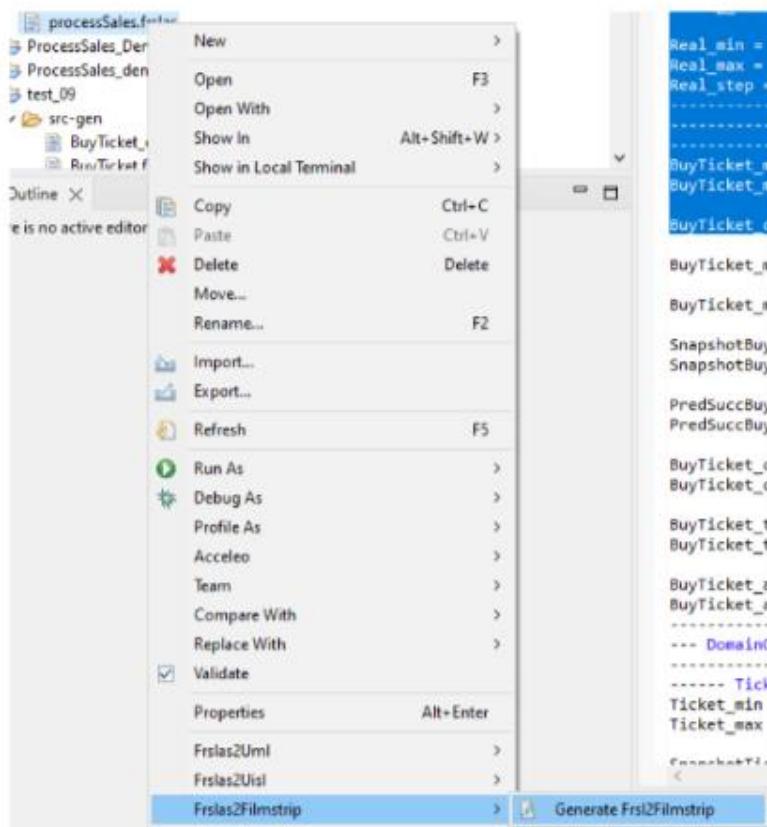
Hình 4.7. Ca sử dụng ProcessSales đặc tả bằng cú pháp cụ thể FRSL.

Sau khi đặc tả ca sử dụng bằng cú pháp cụ thể FRSL, sử dụng công cụ FRSL plugin để chuyển đổi sang cú pháp trùu tượng FRSLAS (Hình 4.8), chúng ta thu được tệp có đuôi .frslas : Chuột phải vào tệp .frls → FRSLCS → Save As → FRSLAS → Chọn nơi lưu tệp cùng thư mục với tệp .frsl.

Tiếp theo, chuột phải tệp .frslas → chọn Frslas2Filmstrip → Generate Frsl2Filmstrip để sinh mô hình ứng dụng Filmstrip4FRSL (Hình 4.9).



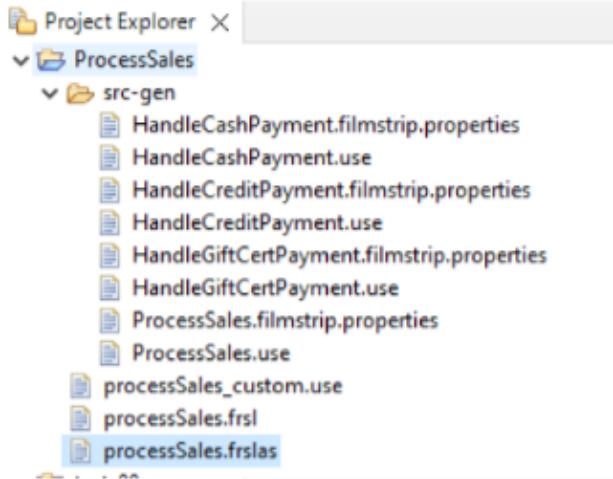
Hình 4.8. Chuyển đổi cú pháp cụ thể FRSL sang cú pháp trùu tượng FRSL.



Hình 4.9. Công cụ sinh mô hình ứng dụng Filmstrip4FRSL.

4.3.2 Xây dựng mô hình ứng dụng Filmstrip4FRSL

Sau khi sử dụng công cụ để chuyển đổi từ cú pháp trùu tượng FRSL sang mô hình ứng dụng, chúng ta thu được 04 tệp ProcessSales.use, HandleCashPayment.use, HandleGiftCertPayment.useHandleCreditPayment.use (Hình 4.10) là các ca sử dụng tương ứng của hệ thống POS. Các tệp này là đặc tả USE ở dạng UML/OCL mô tả mô hình ứng dụng Filmstrip4FRSL (Hình 4.11).



Hình 4.10. Các đặc tả mô hình ứng dụng Filmstrip4FRSL và tệp cấu hình sau khi được sinh ra.

```

---- the USE model generated for the use case ProcessSales ----
-----
model ProcessSalesModel
-----
-- classes --
-----
class ProductDescription
  attributes
    desc: String
    price: Real
    tax: Real
    upc: String
  end

class Store
  attributes
    address: String
    name: String
  end

class CreditPayment
  attributes
end

class GiftCertPayment
  attributes
end

class Sale
  attributes
    id: String
    total: Real
    isComplete: Boolean
  end

class CreditCard
-----
-- associations --
-----
association PaidBy between
  Sale[1] role sale
  Payment[1] role payment
end

association ProvidedWith between
  Store[0..*] role store
  AuthService[0..*] role authService
end

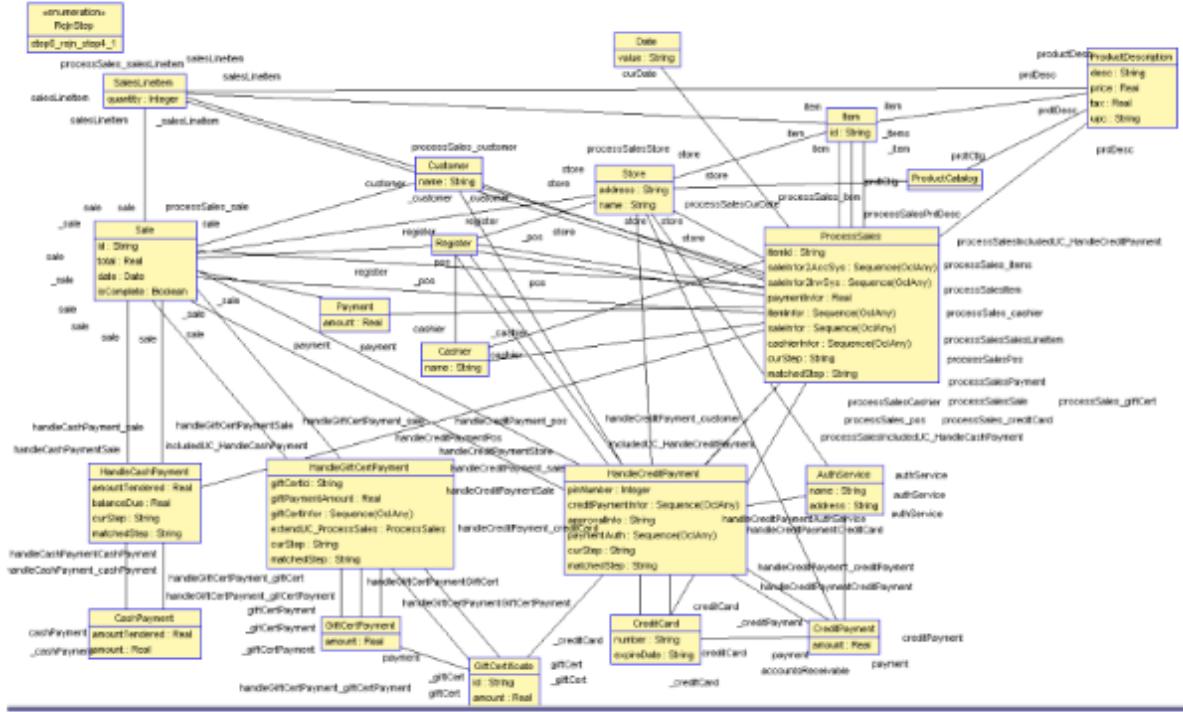
association Houses between
  Store[1] role store
  Register[0..*] role register
end

association CheckedBy between
  CreditPayment[0..*] role payment
  CreditCard[1] role creditCard
end

association DescribedBy between
  SalesLineItem[0..*] role salesLineItem
  ProductDescription[1] role productDesc
end

```

Hình 4.11. Đặc tả mô hình ứng dụng Filmstrip4FRSL dưới dạng UML/OCL.



Hình 4.12. Biểu đồ lớp của mô hình ứng dụng Filmstrip4FRSL của hệ thống POS.

4.4 Thực nghiệm 2: Sinh ca kiểm thử

Để sinh ca kiểm thử, luận văn thực nghiệm với ca sử dụng BuyTicket. Nội dung ca sử dụng BuyTicket được mô tả bằng văn bản trong Bảng 4.2. Ca sử dụng thể hiện chức năng một khách hàng mua vé. Gồm có tác nhân chính Customer, tác nhân phụ Ticket. Ca sử dụng gồm một luồng chính và một luồng thay thế. Trong đó, luồng chính gồm 04 bước, luồng thay thế xảy ra tại bước 3 khi khách hàng không đủ tiền mua vé.

Bảng 4.2. Ca sử dụng BuyTicket mô tả bằng văn bản

Use case ID	BuyTicket
Mô tả	Ca sử dụng mô tả một khách hàng mua vé
Tiền điều kiện	Khách hàng muốn mua một vé
Hậu điều kiện	Khách hàng mua vé thành công
Tác nhân chính	Khách hàng
Tác nhân phụ	Vé

Luồng chính	<ol style="list-style-type: none"> Khách hàng muốn mua một vé. Hệ thống hiện thị danh sách vé có sẵn. Khách hàng chọn một vé để mua. Hệ thống hiển thị số tiền và cập nhật thanh toán.
Luồng thay thế	3a. Khách hàng không đủ tiền. Hệ thống thông báo tiền không đủ. Khách hàng lặp lại bước 3 để chọn vé khác.

4.4.1 Xây dựng mô hình FRSL

Ca sử dụng BuyTicket được biểu diễn bằng FRSL (xem Bảng 4.3) gồm có hai lớp Ticket và Customer, đây là hai đối tượng cơ bản của ca sử dụng BuyTicket. Quan hệ giữa hai đối tượng của lớp được diễn đạt bằng cú pháp *association Takes*, thể hiện rằng : Một customer có thể mua nhiều ticket. Ca sử dụng BuyTicket với Tiền điều kiện (ucPrecondition) thể hiện chưa tồn tại quan hệ Takes giữa đối tượng customer và ticket. Hậu điều kiện (ucPostcondition) mô tả phát sinh một quan hệ Takes của đối tượng customer và ticket sau khi mua vé thành công. Các điều kiện ràng buộc được mô tả bằng ngôn ngữ OCL.

Bảng 4.3. Ca sử dụng BuyTicket đặc tả bằng FRSL

Đặc tả ca sử dụng với FRSL	Ý nghĩa
<pre>ucPrecondition description = 'The customer wants to buy a ticket.' customer: Customer; ticket: Ticket; !(customer, ticket): Takes;</pre>	Tiền điều kiện: Khách hàng muốn mua một vé. Các thể hiện của lớp Customer, Ticket. Giữa thể hiện customer và ticket không tồn tại quan hệ Takes .
<pre>ucPostcondition description = 'The customer buys a tick- et successfully.' (customer, ticket): Takes;</pre>	Hậu điều kiện: Khách hàng mua vé thành công. Giữa thể hiện customer và ticket tồn tại quan hệ Takes .
<pre>actStep step1 description = '1. The customer wants to buy a ticket.' from customer: Customer; [customer.money > 0] to</pre>	Bước 1: Khách hàng muốn mua một vé. SnapshotPattern: from: - Thể hiện customer của lớp Customer. - Biểu thức logic thể hiện thuộc tính money > 0.
<pre>sysStep step2 description = '2. The system displays a list of available tickets.' from \$avblTickets: Set(Ticket); ['@oclCompile' = 'avblTickets = Ticket.allInstances()->select(t:Ticket t.customer.oclIsUndefined())']</pre>	Bước 2: Hệ thống hiển thị danh sách các vé còn có thể đặt được. Đây là hành động của hệ thống (system). SnapshotPattern: from: - \$avblTickets: Tập hợp các Ticket kế thừa từ các bước trước, đây là một bắt biến. - Biểu thức logic gán giá trị của avblTickets (true

<pre> to actions Customer <- ticketIds: Set(Integer) = avbTickets->collect(id)->asSet(); </pre> <p>actStep step3</p> <pre> description = '3. The customer selects a ticket to buy.' from avbTickets: Set(Ticket); '@oclCompile' = 'avbTickets- >collect(id)->asSet()->includes(inputId)' to actions Customer -> inputId: Integer; </pre>	<p>hoặc fail).</p> <p>to:</p> <ul style="list-style-type: none"> - Gán ticketIDs vào lớp Customer. <p>Bước 3: Khách hàng chọn một vé để mua.</p> <p>SnapshotPattern:</p> <p>from:</p> <ul style="list-style-type: none"> - Biểu thức logic(true hoặc fail) : duyệt tất cả thuộc tính id của avbTicketes bao gồm cả inputId (là ticketId của customer). <p>to:</p> <ul style="list-style-type: none"> - Gán inputId của lớp Customer.
<pre> sysStep step4 description = '4. The system presents the money and updates the payment.' from customer: Customer; \$ticket: Ticket; !(customer, ticket): Takes; [ticket.id = inputId] to (customer, ticket): Takes; [custom- er.money=customer.money@pre - ticket.value] ['@changed' <> 'customer/{money}; ticket/{customer}'] </pre>	<p>Bước 4: Hệ thống hiển thị số tiền và cập nhật thanh toán.</p> <p>SnapshotPattern:</p> <p>from:</p> <ul style="list-style-type: none"> - \$Ticket: Đối tượng bất biến của lớp Ticket - Không tồn tại liên kết Takes giữa customer & ticket <p>to:</p> <ul style="list-style-type: none"> - Tồn tại một quan hệ Takes giữa customer & ticket - Số tiền của customer = số tiền customer ở trạng thái trước - ticket.value
<pre> altStep at step3 description = '3a. The customer does not have enough money.' when [customer.money < ticket.value] end sysStep step3a1 description = '3a.1. The system informs that the money is not enought.' from to actions Customer <- msg: String = 'The money is not enought!'; end rejoinStep step3 description = '3a.2 The customer repeats Step3 to re-select another ticket.' when [customer.money < ticket.value] </pre>	<p>Luồng thay thế ở Bước 3: Khách hàng không đủ tiền.</p> <p>Khi Khách hàng không đủ tiền, có một sysStep thay thế:</p> <p>Step3a1: Hệ thống thông báo tiền không đủ.</p> <ul style="list-style-type: none"> - Gán thông báo msg đến khách hàng. <p>RejoinStep: Quay lại Bước 3 để KH nhập lại vé khác</p> <ul style="list-style-type: none"> - Điều kiện: Số tiền của Khách hàng < value của ticket

4.4.2 Xây dựng mô hình ứng dụng Filmstrip4FRSL

Ca sử dụng BuyTicket đặc tả bằng FRSL làm đầu vào cho công cụ Generate Frsl2Filmstrip để tạo ra mô hình ứng dụng Filmstrip4FRSL. Mô hình ứng dụng sau khi được tạo ra có đặc tả minh họa trong Hình 4.13.

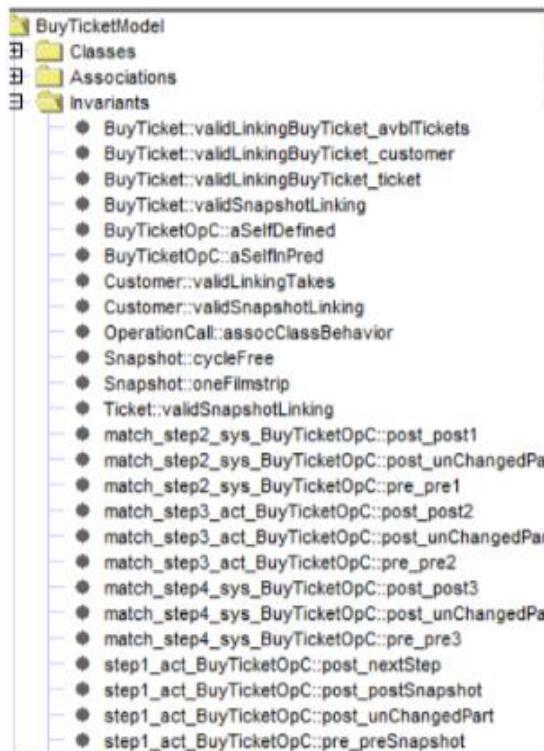
```
-- the USE model generated for the use case BuyTicket ----
model BuyTicketModel
class Ticket
attributes
    id: Integer
    value: Integer
end
class Customer
attributes
    money: Integer
end
enum RejnStep{step3a2_rejn_step3_1}
class BuyTicket
attributes
    inputId: Integer
    msg: String
    ticketIds: Set(Integer)
    curStep: String
    matchedStep: String
operations
    enter_BuyTicket(): Boolean =
        (not customer.oclIsUndefined()) and (not ticket.oclIsUndefined())
        and customer.ticket->excludes(ticket)
        and curStep = 's0'
    exit_BuyTicket(): Boolean =
        customer.ticket->includes(ticket)
        and isFinalStep(curStep)
    step1_act()
    step2_sys()
    match_step2_sys()
    step3_act()
    match_step3_act()
    step4_sys()
    match_step4_sys()
    step3a_alt()
    step3a1_sys()
    step3a2_rejn_step3_1()
    isFinalStep(stepId: String): Boolean =
        Set{ 'step4_sys', 'step3a2_rejn_step3_1' }->includes(stepId)
    isValidRejnStep(rStepIndex: RejnStep): Boolean =
```

```

        if ( rStepIndex = #step3a2_rejn_step3_1 ) then
            isOK_step3a2_rejn_step3_1()
        else false endif
    isOK_step3a2_rejn_step3_1(): Boolean =
        (customer.money < ticket.value)
    enum2String(rStepIndex: RejnStep): String =
        if ( rStepIndex = #step3a2_rejn_step3_1 ) then
            'step3a2_rejn_step3_1'
        else 'null' endif
    end

```

Hình 4.13. Đặc tả mô hình ứng dụng Filmstrip4FRSL dưới dạng UML/OCL của ca sú dụng BuyTicket.

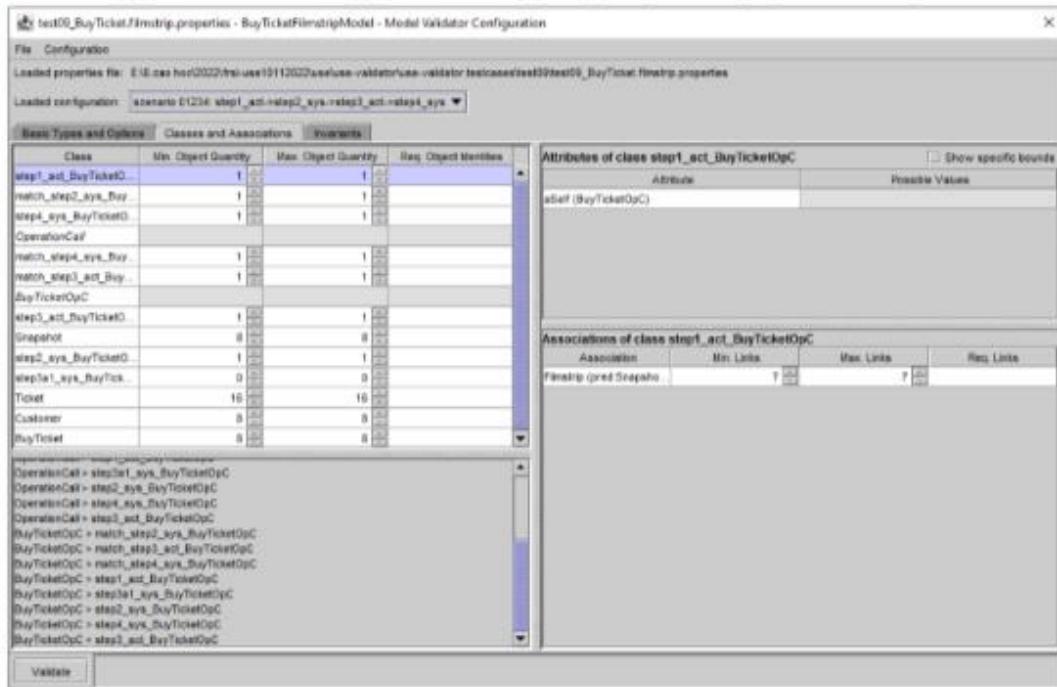


Hình 4.14. Các bắt biến của mô hình kịch bản Filmstrip.

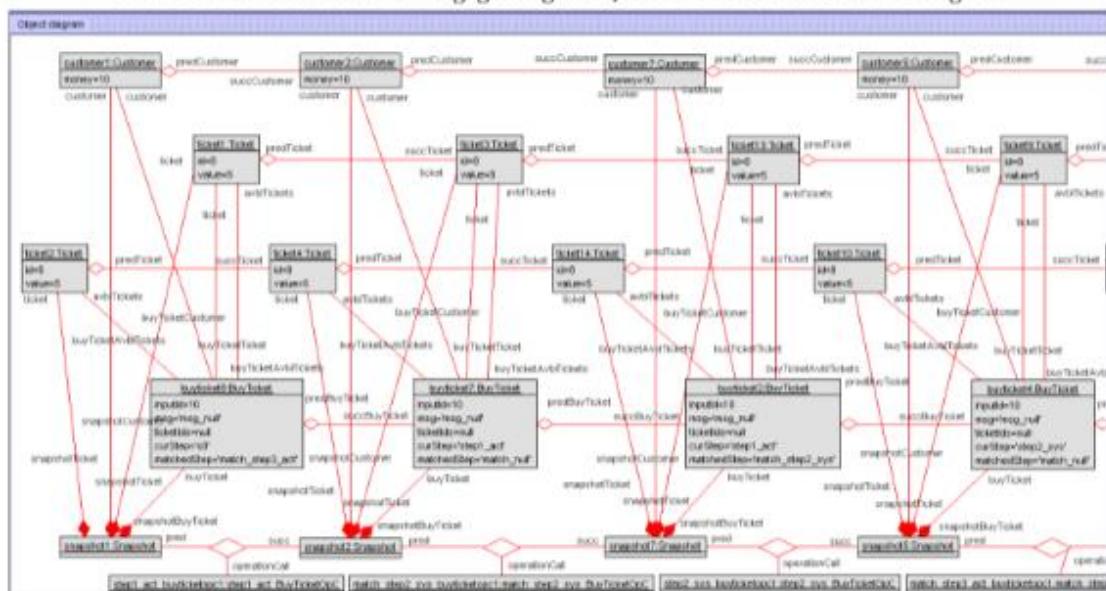
Sử dụng plugin **Filmstripping** của USE để chuyển đổi mô hình ứng dụng Filmstrip4FRSL sang mô hình kịch bản filmstrip, chúng ta thu được mô hình chỉ gồm các bắt biến (Hình 4.14).

4.4.3 Sinh các ca kiểm thử

Cuối cùng, sử dụng plugin Model Validator của USE để sinh mô hình thỏa mãn các ràng buộc được cấu hình trong tệp .properties (Hình 4.15). Mô hình sinh ra được biểu diễn dưới dạng biểu đồ đổi tượng thỏa mãn giá trị được cài đặt (Hình 4.16).



Hình 4.15. Cấu hình không gian giá trị cho Model Validator Plugin.



Hình 4.16. Biểu đồ đổi tượng ca sử dụng BuyTicket được sinh ra thể hiện trạng thái của các đổi tượng qua từng snapshot.

Từ biểu đồ đối tượng này, chúng ta có thể tổng hợp thành tệp mô tả dữ liệu kịch bản ca kiểm thử (Hình 4.17).

Mô testcase	Test Step	Test Data	Expected Result	Description
#-condition		Không tồn tại quan hệ giữa Customer và Ticket tức là một đối tượng về chia thuộc sở hữu của một đối tượng Customer		
Scenario01234	#0->step1_act->step2_sys->step3_act->step4_sys			
[BuyTicket-1]	snapshot1 (step1_act)	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=0 curStep = #0 matchedStep=match_step1 operationCall = step1_act.buyTicketstep1	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step1_act matchedStep=match_step1	Khách hàng muốn mua vé (step1)
[BuyTicket-2]	snapshot2 (match_step2_sys)	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step1_act matchedStep=match_null operationCall = match_step2_sys.buyTicketstep1	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step1_act matchedStep=match_step2_sys	Chuẩn bị dữ liệu đầu vào cho step2
[BuyTicket-3]	snapshot3 (step2_sys)	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step1_act matchedStep=match_step2_sys operationCall = step2_sys.buyTicketstep1	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step2_sys matchedStep=match_null	Hệ thống hiển thị danh sách các vé còn có thể đặt được. Đây là hành động của hệ thống (step2)
[BuyTicket-4]	snapshot4 (match_step3_act)	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step2_sys matchedStep=match_null operationCall = match_step3_act.buyTicketstep1	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step3_act matchedStep=match_step3_act	Chuẩn bị dữ liệu đầu vào cho step3
[BuyTicket-5]	snapshot5 (step3_act)	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step3_act matchedStep=match_step3_act operationCall = step3_act.buyTicketstep1	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step3_act matchedStep=match_step3_act	Khách hàng chọn một vé để mua (step3)
[BuyTicket-6]	snapshot6 (match_step4_sys)	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step3_act matchedStep=match_null operationCall = match_step4_sys.buyTicketstep1	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step4_act matchedStep=match_step4_sys	Chuẩn bị dữ liệu đầu vào cho step4
[BuyTicket-7]	snapshot7 (match_step4)	Customer L.money = 10 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step4_act matchedStep=match_step4_act operationCall = match_step4_sys.buyTicketstep1	Customer L.money = 5 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step4_sys matchedStep=match_step4_act	Hệ thống hiển thị số tiền và cập nhật thành toán
[BuyTicket-8]	snapshot8	Customer L.money = 5 Ticket1.id=0, Ticket1.value=4 Ticket2.id=0, Ticket2.value=4 curStep = step4_sys matchedStep=match_step4_act		Số tiền của khách hàng bị trừ đi sau khi đã thanh toán vé

Hình 4.17. Kịch bản dữ liệu ca kiểm thử được mô tả trong một tệp.

4.5 Đánh giá kết quả thực nghiệm

Trong mục này, luận văn đưa ra một số đánh giá về tính khả thi và hiệu quả của phương pháp dựa trên các tiêu chí: thời gian sinh dữ liệu ca kiểm thử, độ phủ ca kiểm thử và độ chính xác của dữ liệu ca kiểm thử.

Thời gian. Đầu tiên, *thời gian sinh dữ liệu ca kiểm thử* phụ thuộc vào thời gian tìm kiếm mô hình của Model Validator plugin. Thực nghiệm thay đổi không gian giá trị các đối tượng trong mô hình để so sánh thời gian giải của Model Validator plugin, chúng ta thu được kết quả như sau (xem Bảng 4.4).

Với kết quả thu được, có thể nhận thấy: Số lượng đối tượng càng lớn càng mất nhiều thời gian để SAT solver xử lý, tuy nhiên thời gian vẫn ở mức chấp nhận được.

Bảng 4.4. Bảng so sánh thời gian xử lý của Model Validator

STT	Thực nghiệm cấu hình	Translation time (USE to Kodkod)	Translation time (Kodkod to SAT)	Solving time
1	Integer : 1-10, String : 0-10, Real : -2-2 Ticket : 6, Customer : 3, BuyTicket : 3 Snapshot : 3	844 ms	27ms	21ms
2	Integer : 1-10, String : 0-10, Real : -2-2 Ticket : 12, Customer : 6, BuyTicket : 6 Snapshot : 6	844 ms	181ms	597ms
3	Integer : 1-10, String : 0-10, Real : -2-2 Ticket : 16, Customer : 8, BuyTicket : 8 Snapshot : 8	844 ms	729 ms	2105 ms
4	Integer : 1-10, String : 0-10, Real : -2-2 Ticket : 16, Customer : 16, BuyTicket : 16 Snapshot : 16	844 ms	1447 ms	326814 ms (5')

Độ phủ ca kiểm thử. Thứ hai, *độ phủ ca kiểm thử* được đánh giá theo nhiều tiêu chí khác nhau như bao phủ nhánh, bao phủ đường dẫn, bao phủ dòng lệnh, bao phủ điều kiện. Trong phạm vi luận văn đánh giá độ phủ ca kiểm thử theo tiêu chí bao phủ đường dẫn: là tỉ lệ bao phủ toàn bộ luồng xử lý chức năng trong đặc tả ca sử dụng. Luồng xử lý chức năng trong ca sử dụng tương đương với các kịch bản khác nhau của ca sử dụng. Tại bước chuyển đổi từ đặc tả FRSL sang mô hình ứng dụng sử dụng mô đun **Film-strip4FRSL**, chúng ta thu được tệp cấu hình .properties trong đó mô tả giá trị của các đối tượng theo từng kịch bản (Hình 4.18). Với mỗi kịch bản được cấu hình sẽ xác định dữ liệu ca kiểm thử tương ứng. Như vậy, ca kiểm thử sinh ra bao phủ được toàn bộ luồng xử lý chức năng trong ca sử dụng.

<pre> ===== [scenario 01234: step1_act->step2_sys->step3_act->step4_sys] ===== Integer_min = -10 Integer_max = 10 String_max = 10 Real_min = -2.0 Real_max = 2.0 Real_step = 0.5 ----- Usecase Class ----- BuyTicket_min = 8 BuyTicket_max = 8 BuyTicket_curStep= Set['s0','step1_act','step2_sys','step3_act','step4_sys'] =====</pre>	<pre> ===== [scenario 0123: step1_act->step2_sys->step3_act] ===== Integer_min = -10 Integer_max = 10 String_max = 10 Real_min = -2.0 Real_max = 2.0 Real_step = 0.5 ----- Usecase Class ----- BuyTicket_min = 6 BuyTicket_max = 6 BuyTicket_curStep=Set['s0','step1_act','step2_sys','step3_act'] ===== [scenario 012: step1_act->step2_sys] ===== Integer_min = -10 Integer_max = 10 String_max = 10 Real_min = -2.0 Real_max = 2.0 Real_step = 0.5 ----- Usecase Class ----- BuyTicket_min = 4 BuyTicket_max = 4 BuyTicket_curStep = Set['s0','step1_act','step2_sys']</pre>
--	---

Hình 4.18. Tệp cấu hình theo từng kịch bản của ca sử dụng BuyTicket.

Độ chính xác. Cuối cùng, *độ chính xác* của dữ liệu ca kiểm thử được xác định dựa trên kết quả kiểm tra, xác minh lại các ca kiểm thử theo từng kịch bản. Thực nghiệm kiểm tra thủ công cho thấy các giá trị đầu vào, đầu ra mong muốn tại mỗi bước trong từng ca kiểm thử thỏa mãn đặc tả yêu cầu ca sử dụng.

Như vậy, qua quá trình thực nghiệm với các ca sử dụng khác nhau, phương pháp đã đạt được mục tiêu đề ra: (1) Sinh dữ liệu cho ca kiểm thử từ đặc tả yêu cầu bằng ca sử dụng; (2) Xây dựng được công cụ chuyển đổi mô hình bằng phương pháp M2T; (3) Sử dụng và thực nghiệm được công cụ USE theo hướng mô hình. Ca sử dụng được đặc tả bằng ngôn ngữ FRSL giúp mô tả yêu cầu chính xác và đầy đủ hơn, làm đầu vào cho công cụ chuyển đổi mô hình. Việc phát triển công cụ để sinh mô hình ứng dụng Film-strip4FRSL từ cú pháp trùu tượng FRSL với các tập luật chuyển đổi được kết quả thành công. Mô hình ứng dụng sinh ra diễn đạt bằng UML/OCL phù hợp làm đầu vào cho công cụ USE. Công cụ USE cũng giúp xác minh tính đúng đắn của mô hình ca sử dụng ban đầu. Cuối cùng, bộ dữ liệu ca kiểm thử được tạo ra tự động thỏa mãn hành vi của ca sử

dụng. Dữ liệu ca kiểm thử này có thể sinh theo các kịch bản khác nhau theo luồng ca sử dụng ban đầu. Với dữ liệu ca kiểm thử được tạo ra, giúp giảm bớt thời gian và chi phí thiết kế ca kiểm thử, áp dụng được vào giai đoạn kiểm thử tích hợp hoặc chấp nhận người dùng ở mức kiểm thử hệ thống.

Tuy nhiên, phương pháp cũng có những hạn chế nhất định. Việc đặc tả ca sử dụng bằng ngôn ngữ FRSL cần đánh giá thêm ở các ca sử dụng rộng và phức tạp hơn. Mặc dù ngôn ngữ FRSL bổ sung mô tả ràng buộc bằng OCL, tiền điều kiện/hậu điều kiện nhưng mới đạt mức sấp xỉ với ngôn ngữ tự nhiên. Các luật chuyển được cài đặt trong công cụ chuyển đổi cần có phương pháp đảm bảo tính đúng đắn. Ngoài ra, phương pháp gồm một chuỗi hành động kết hợp giữa công cụ và con người khá phức tạp, cần có hiểu biết về công cụ để thực thi được.

4.6 Tổng kết chương

Trong chương này, luận văn đã trình bày cách cài đặt và sử dụng công cụ chuyển đổi mô hình M2T, công cụ USE để đặc tả và kiểm chứng mô hình. Sau đó, luận văn sử dụng các thực nghiệm khác nhau để minh họa quá trình sinh mô hình ứng dụng và sinh ca kiểm thử. Cuối cùng, luận văn đưa ra các đánh giá về kết quả đạt được của phương pháp và công cụ từ thực nghiệm.

CHƯƠNG 5. KẾT LUẬN

Việc sinh tự động ca kiểm thử từ đặc tả yêu cầu có nhiều ý nghĩa trong giai đoạn phát triển phần mềm. Mặc dù có nhiều nỗ lực trong việc tạo tự động ca kiểm thử từ mô hình ca sử dụng, tuy nhiên còn nhiều điểm hạn chế, khó khăn khi đặc tả chính thức yêu cầu và biểu diễn chính xác ca kiểm thử. Do đó, luận văn đề xuất một kỹ thuật Sinh tự động ca kiểm thử từ đặc ta ca sử dụng bằng cách kết hợp kỹ nghệ hướng mô hình và các công cụ hướng mô hình. Phương pháp cho phép đặc tả ca sử dụng với ngôn ngữ mô hình hóa FRSL chuyên biệt miền giúp yêu cầu được mô tả chính xác đầy đủ hơn. Tiếp theo qua một số bước chuyển đổi mô hình để thu được mô hình kịch bản filmstrip. Cuối cùng, sử dụng bộ công cụ USE để tìm kiếm mô hình và sinh mô hình thỏa mãn ràng buộc. Dữ liệu được sinh ra tương ứng với ca kiểm thử, giúp người dùng có thể sử dụng và thẩm định ngay trong giai đoạn thiết kế.

5.1 Các đóng góp của luận văn

Sau một thời gian nghiên cứu và tìm hiểu, luận văn đã có những đóng góp nhất định như sau.

Thứ nhất, đề xuất phương pháp tổng quan trong việc sinh dữ liệu ca kiểm thử từ đặc tả mô hình ca sử dụng. Phương pháp tiếp cận theo hướng mô hình, kết hợp với công cụ USE cho phép tạo ra ca kiểm thử chính xác thỏa mãn các ràng buộc của đặc tả yêu cầu. Bên cạnh đó, luận văn cũng giới thiệu các lý thuyết của Kỹ nghệ hướng mô hình cùng công cụ hướng mô hình USE.

Thứ hai, đề xuất tập luật chuyển đổi mô hình sang văn bản (đặc tả USE) và tập luật chuyển tệp cấu hình làm đầu vào cho công cụ USE.

Thứ ba, xây dựng được công cụ chuyển đổi từ mô hình sang văn bản để hiện thực hóa các tập luật chuyển.

Luận văn đã tiến hành thực nghiệm trên một số ví dụ cụ thể. Từ đó, đánh giá hiệu quả phương pháp và đưa ra các kết luận chung về tính khả thi khi ứng dụng vào thực tế.

5.2 Hướng phát triển

Với những kết quả thu được, luận văn xin được đề xuất một số hướng phát triển để hoàn thiện phương pháp hơn:

Thứ nhất, cải tiến phương pháp để tự động hóa hoàn toàn tiến trình, giảm tối đa sự tham gia của con người.

Thứ hai, cải tiến công cụ chuyển đổi mô hình, nâng cao kỹ thuật tạo mã để bao phủ được nhiều ca sử dụng hơn.

TÀI LIỆU THAM KHẢO

- [1] Gutierrez, J.J., Escalona, M.J., Mej'as, M., Torres, J. *An approach to generate test cases from use cases*. In: Proc. 6th International Conference on Web Engineering. pp. 113–114. ACM (2006).
- [2] Object Management Group , *Object Constraint Language*, 2014.
- [3] Jordi Cabot1, Martin Gogolla. *Object Constraint Language (OCL) a Definitive Guide*, 2012.
- [4] Roy Groma, Birger Moller-Pedersen, Goran K.Olsen. *Comparison of Three Model Transformation Languages*, 2009.
- [5] Ibrahim Bumin Kara. *Design and Implementation of the ModelicaML Code Generator Using Acceleo 3.X*, 2015-05-07.
- [6] Database Systems Group, Brement University. *USE A UML based Specification Environment, Preliminary Version 0.1*. May 16, 2007.
- [7] Object Management Group. *Unified Modeling Language Specification, Version 1.3*, June 1999.
- [8] M. Gogolla, L. Hamann1, F. Hilken, M. Kuhlmann, R. France. *From Application Models to Filmstrip Models:An Approach to Automatic Validation of Model Dynamics*. cs.colostate.edu, 2014.
- [9] Duc-Hanh Dang (2023). *FRSL: A Domain Specific Language to Specify Functional Requirements*. VNU Journal of Science: Comp. Science & Com. Eng., Vol. 39, No. N (2023) 1–20.
- [10] Frank Hilken, Philipp Niemann, Martin Gogolla, and Robert Wille. *Filmstripping and Unrolling: A Comparison of Verification Approaches for UML and OCL Behavioral Models*. University of Bremen, Computer Science Department D-28359 Bremen, Germany, 2014.
- [11] Tomasz Straszak, Michał Smiałek. *Automating Acceptance Testing with tool support*. Warsaw University of Technology Warsaw, Poland, DOI: 10.15439/2014F342.
- [12] Cle'mentine Nebut, Franck Fleurey, Yves Le Traon. *Automatic Test Generation:A Use Case Driven Approach*. DOI:10.1109/TSE.2006.22.
- [13] M. El-Attar and J. Miller, “Developing comprehensive acceptance tests from use cases and robustness diagrams,” Requir. Eng., vol. 15, no. 3, pp. 285–306, Sep. 2010.
- [14] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice, Second Edition*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017.
- [15] El-Attar, M., Miller, J. *Developing comprehensive acceptance tests from use cases and robustness diagrams*. Requirements Engineering 15(3), 285–306 (2010).

- [16] Benoit Combemale, Robert B.France, Jean-Marc, Berhard Rumpe, Jim Steel, Didier Vojtisek. *Engineering Modeling Languages*.
- [17] Kurtev F., Jouault I. *Transforming models with ATL*. 2006.
- [18] Ralph R. Young . *The Requirements Engineering Handbook*. ARTECH HOUSE, INC, 2004.
- [19] Chu Thi Minh Hue, Dang Duc Hanh, Nguyen Ngoc Binh and Truong Anh Hoang (2019), “USLTG: Test Case Automatic Generation by Transforming Use Cases”, *Int. Journal of Software Engineering and Knowledge Engineering*, volumne 29, pp. 1313–1345.
- [20] Khanh-Hoang Doan, Martin Gogolla, and Frank Hilken. *Towards a Developer-Oriented Process for Verifying Behavioral Properties in UML and OCL Models*. Federation of International Conferences on Software Technologies. December 2016.
- [21] Heiko Behrens, Michael Clay, Sven Efftinge, Moritz Eysholdt, Peter Friese, Jan Köhnlein, Knut Wannheden, Sebastian Zarnekow and contributors. *Xtext User Guide*. 2008-2009.
- [22] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual*, 2nd Edition, Addison-Wesley Professional, 2004.