

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ



QUÁCH ĐỨC BÌNH

**PHƯƠNG PHÁP SINH TỰ ĐỘNG GIAO DIỆN
NGƯỜI DÙNG TỪ MÔ HÌNH YÊU CẦU FRSL**

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

HÀ NỘI – 2023

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

QUÁCH ĐỨC BÌNH

PHƯƠNG PHÁP SINH TỰ ĐỘNG GIAO DIỆN
NGƯỜI DÙNG TỪ MÔ HÌNH YÊU CẦU FRSL

Ngành: Công nghệ thông tin

Chuyên ngành: Kỹ thuật phần mềm

Mã số: 8480103.01

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Cán bộ hướng dẫn: TS. Đặng Đức Hạnh

HÀ NỘI - 2023

LỜI CẢM ƠN

Lời đầu tiên tôi xin được gửi lời cảm ơn chân thành và sâu sắc nhất tới giảng viên hướng dẫn Tiến sĩ Đặng Đức Hạnh - hiện đang công tác tại Bộ môn Công nghệ phần mềm - Khoa Công nghệ thông tin - Trường Đại học Công nghệ - Đại học Quốc gia Hà Nội. Nhờ có thầy đã tận tâm hướng dẫn, định hướng và góp ý tôi trong suốt quá trình mà tôi tìm hiểu và nghiên cứu để tôi có thể vượt qua các khó khăn và có động lực để hoàn thành luận văn này. Một lần nữa tôi xin được gửi lời cảm ơn tới thầy.

Tôi cũng xin được gửi lời cảm ơn tới các thầy/cô trong Khoa Công nghệ thông tin - Trường Đại học Công Nghệ - Đại học Quốc gia Hà Nội đã dạy bảo tận tình, trang bị cho tôi những kiến thức quý báu, những kinh nghiệm thực tiễn bổ ích và tạo điều kiện thuận lợi trong suốt quá trình tôi học tập và nghiên cứu tại trường. Từ đó giúp tôi có khả năng tư duy, nghiên cứu và tổng hợp các vấn đề một cách khoa học và qua đó áp dụng vào quá trình nghiên cứu đề tài luận văn này.

Cuối cùng, tôi cũng muốn gửi lời cảm ơn đến những bạn trong lớp, những bạn trong nhóm làm luận văn và cả những người thân bên cạnh tôi đã luôn không ngừng động viên, cổ vũ và giúp đỡ tôi trong quá trình thực hiện luận văn.

Dù đã cố gắng hết sức, song do hạn chế về thời gian cũng như kiến thức nên luận văn không tránh khỏi những thiếu sót và hạn chế. Tôi rất mong được sự thông cảm và có được những góp ý quý báu của các thầy cô và các bạn để luận văn có thể được hoàn thiện hơn.

Tôi xin chân thành cảm ơn!

Hà Nội, ngày ... tháng ... năm 2023

Học viên thực hiện

Quách Đức Bình

LỜI CAM ĐOAN

Tôi là Quách Đức Bình, học viên khóa K27 chuyên ngành Công nghệ phần mềm thuộc chương trình đào tạo Thạc sĩ của Trường Đại học Công nghệ - Đại học Quốc gia Hà Nội. Tôi xin cam đoan đây là công trình nghiên cứu của tôi dưới sự giúp đỡ rất lớn của Giảng viên hướng dẫn là Tiến sĩ Đặng Đức Hạnh và các bạn trong nhóm Nghiên cứu. Những nội dung nghiên cứu và kết quả trong đề tài này là hoàn toàn trung thực. Các trích dẫn từ tài liệu bên ngoài tôi đều liệt kê rõ ràng ở cuối của luận văn.

Hà Nội, ngày ... tháng ... năm 2023

Học viên thực hiện

Quách Đức Bình

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.....	viii
CHƯƠNG 1: MỞ ĐẦU.....	1
1.1 Đặt vấn đề	1
1.2 Mục tiêu và phạm vi	2
1.3 Cấu trúc của luận văn.....	3
CHƯƠNG 2: KIẾN THỨC NỀN TẢNG.....	4
2.1 Kỹ nghệ hướng mô hình	4
2.1.1 Biểu diễn mô hình phần mềm	4
2.1.2 Chuyển đổi mô hình	7
2.2 Đặc tả yêu cầu chức năng với ngôn ngữ FRSL	14
2.2.1 Giới thiệu.....	14
2.2.2 Mô hình yêu cầu chức năng FRSL.....	17
2.3 Kỹ thuật tạo bản mẫu giao diện phần mềm	19
2.4 Biểu diễn mô hình tương tác với IFML	22
2.5 Tổng kết chương	28
CHƯƠNG 3: SINH GIAO DIỆN BẢN MẪU DỰA VÀO CHUYỂN ĐỔI MÔ HÌNH ..	29
3.1 Giới thiệu	29
3.2 Tổng quan phương pháp	30
3.3 Chuyển đổi từ mô hình yêu cầu sang mô hình bản mẫu.....	32
3.4 Sinh giao diện bản mẫu từ mô hình IFML.....	37
3.5 Tổng kết chương	39
CHƯƠNG 4: CÀI ĐẶT VÀ THỰC NGHIỆM.....	40
4.1 Công cụ hỗ trợ.....	40
4.2 Minh họa phương pháp	42

4.3	Đánh giá phương pháp và thảo luận	48
CHƯƠNG 5: KẾT LUẬN.....		49
5.1	Các đóng góp của luận văn	49
5.2	Hướng phát triển	50
TÀI LIỆU THAM KHẢO		51

DANH SÁCH KÝ HIỆU, CHỮ VIẾT TẮT

Ký hiệu	Nội dung
MDD	Model-Driven Development
M2M	Model-To-Model
M2T	Model-To-Text
FRSL	Functional Requirement Specification Language
HTML	HyperText Markup Language
ATL	ATLAS Transformation Language
MDE	Model-Driven Engineering
MOF	Meta Object Facility
OMG	Object Management Group
EMF	Eclipse Modeling Framework
MDSE	Model-Driven Software Engineering
OCL	Object Constraint Language
GUI	Graphical User Interface
RE	Requirements Engineering
IFML	Interaction Flow Modeling Language

DANH SÁCH BẢNG BIỂU, HÌNH VẼ

Hình 2.1: Mối quan hệ giữa mô hình và siêu mô hình [6].	5
Hình 2.2: Kim tự tháp mô hình của OMG.	6
Hình 2.3: Ví dụ về một chương trình chuyển với ATL.	9
Hình 2.4: Acceleo sinh tự động mã nguồn bởi chuyển đổi mô hình M2T.	11
Hình 2.5: Mô đun chỉnh sửa Acceleo với cú pháp.	12
Hình 2.6 Tổng hợp phân loại yêu cầu theo Ralph R. Young [9].	14
Hình 2.7: Biểu đồ biểu diễn siêu mô hình của FRSL.	18
Hình 2.8: Đặc tả ca sử dụng HandleCashPayment bằng cú pháp FRSL.	18
Hình 2.9: Metamodel của mô hình IFML.	23
Hình 2.10: Ví dụ về các thành phần của IFML.	23
Hình 2.11: Ví dụ về 3 thành phần List, Details, Forms trong IFML.	25
Hình 2.12: Mô tả một ví dụ về event trong IFML.	26
Hình 2.13: Mô tả ví dụ thành phần Action trong IFML.	27
Hình 2.14: Mô hình IFML mô tả chương trình Quản lý mail đã gửi.	27
Hình 3.1: Mô tả tổng quan cách tiếp cận để xây dựng ngôn ngữ đặc tả giao diện người dùng IFML từ chế tác FRSL.	31
Hình 3.2: Mô tả tóm lược luật FI1 - FrslModel2IfmlModel.	32
Hình 3.3: Mô tả tóm lược luật FI2 - Usecase2InteractionFlowModel.	33
Hình 3.4: Mô tả tóm lược luật FI2.1 - Step2ViewContainer.	33
Hình 3.5: Mô tả luật FI2.2 - InputActStep2Form.	34
Hình 3.6: Mô tả 2 luật NextStep2Flow và RejoinStep2Flow.	35

Hình 3.7: Mô tả luật chuyển đổi tạo ra mô hình miền IFML.	35
Hình 3.8: Mô tả đặc tả ca sử dụng bằng FRSL được chuyển đổi thành mô hình luồng tương tác IFML.	36
Hình 3.9: Mô tả luật chuyển tạo ra tệp đầu ra .html.	37
Hình 3.10: Mô tả luật chuyển thành phần của từng trang.	38
Hình 3.11: Mô tả tóm lược luật chuyển các thành phần trong ViewComponent.	38
Hình 3.12: Mô tả luật chuyển các thành phần chuyển trang NavigationFlow.	39
Hình 4.1: Kiến trúc tổng thể của VNU-FRSL [8].	40
Hình 4.2: Công cụ FRSL2IFML được tích hợp vào Eclipse IDE.	41
Hình 4.3: Các package của công cụ FRSL2IFML tool trên Eclipse IDE.	41
Hình 4.4: Các package của công cụ IFML2GUI tool trên Eclipse.	42
Hình 4.5: Đặc tả ca sử dụng dưới dạng FRSL.	43
Hình 4.6: Chuyển đổi từ FRSL sang FRSLAS.	44
Hình 4.7: Đặc tả ca sử dụng dưới dạng FRSLAS.	44
Hình 4.8: Sử dụng công cụ chuyển đổi mô hình FRSL2IFML.	45
Hình 4.9: Mô hình IFML dạng cú pháp trừu tượng được sinh từ đặc tả đầu vào FRSL.	45
Hình 4.10: Mô hình IFML được sinh ở dạng đồ họa.	46
Hình 4.11: Sử dụng công cụ chuyển đổi IFML2GUI.	46
Hình 4.12: Giao diện tự động bản mẫu được sinh tự động từ mô hình IFML.	47
Hình 4.13: Giao diện bản mẫu minh họa cho Step 3 trong Usecase ProcessSale.	47
Hình 4.14: Giao diện bản mẫu minh họa cho Step 4 có Form Component	48

TÓM TẮT

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. Bên cạnh đó, giao diện người dùng rất quan trọng trong thời đại ứng dụng web và di động ngày nay. Do đó, phương pháp tự động hóa trong việc tạo bản mẫu giao diện người dùng từ các đặc tả yêu cầu chức năng ban đầu là nhu cầu vô cùng thiết yếu. Bằng cách áp dụng phương pháp này, ta có thể biểu diễn và mô tả các yêu cầu và thiết kế của giao diện bằng các mô hình trừu tượng, không dựa vào mã nguồn cụ thể.

Tính tự động hóa trong việc sinh giao diện từ mô hình giúp giảm thiểu công sức viết mã và tăng tính tái sử dụng. Thay vì chỉnh sửa mã giao diện trực tiếp, chỉ cần cập nhật mô hình và chạy lại quá trình sinh mã, giúp giảm thiểu thời gian và công sức cần thiết để duy trì và nâng cấp giao diện.

Mô hình hóa trong việc phát triển giao diện hướng mô hình giúp diễn đạt và hiểu rõ hơn về yêu cầu và thiết kế của hệ thống trước khi bước vào việc viết mã nguồn. Điều này giúp giảm thiểu sai sót và tăng tính nhất quán trong quá trình phát triển. Tự động hóa giao diện hướng mô hình cũng giúp tăng tính linh hoạt và dễ dàng thay đổi khi cần thiết. Các thay đổi trong mô hình có thể dễ dàng được cập nhật và áp dụng vào quy trình phát triển mà không làm ảnh hưởng đến mã nguồn hiện tại, giúp tiết kiệm thời gian và công sức.

Chính vì lẽ đó, hướng tiếp cận của luận văn là áp dụng các luật chuyển đổi để biểu diễn giao diện phác thảo từ các thành phần của mô hình yêu cầu FRSL. Mô hình FRSL sẽ đóng vai trò mô hình hóa phân tích xử lý đặc tả các ca sử dụng, từ đó, xây dựng các đặc tả cho các chức năng, đối tượng theo yêu cầu của tình huống nghiên cứu cụ thể. Luận văn sẽ xây dựng các bộ luật để chuyển đổi, xây dựng các chức năng, biểu diễn các đối tượng tương ứng theo mô hình IFML. Sau đó, với phần đồ họa sử dụng công cụ tích hợp để sinh tự động bản mẫu giao diện người dùng theo đặc tả ca sử dụng.

Luận văn đưa ra mục tiêu là xây dựng một phương pháp tự động tạo bản mẫu giao diện người dùng từ các đặc tả yêu cầu. Phương pháp này dựa trên kỹ thuật mô hình hóa, nhằm phân tách sự phức tạp của lập trình khỏi nền tảng thực thi, đồng thời tối ưu hóa quá trình phát triển bằng việc tái sử dụng thiết kế hệ thống.

CHƯƠNG 1: MỞ ĐẦU

1.1 Đặt vấn đề

Trong thời đại số hóa bùng nổ, giao diện người dùng (User Interface - UI) đóng vai trò vô cùng quan trọng trong việc xây dựng các ứng dụng phần mềm và tương tác hiệu quả với người dùng. Một giao diện hấp dẫn, trực quan và dễ sử dụng không chỉ tạo nên trải nghiệm tuyệt vời cho người dùng mà còn đóng góp tích cực vào thành công của sản phẩm và sự cạnh tranh trong thị trường công nghệ. Tuy nhiên, việc phát triển giao diện người dùng truyền thống thường gặp phải nhiều thách thức, đòi hỏi công sức và tốn thời gian [1]. Việc xây dựng bản mẫu giao diện, viết mã nguồn và thực hiện kiểm thử là một quy trình phức tạp, đòi hỏi sự tập trung và công phu từ các nhà phát triển. Đồng thời, sự phức tạp của giao diện và sự thay đổi liên tục khiến việc duy trì và nâng cấp giao diện trở nên khó khăn và tốn kém.

Mặc dù có nhiều nghiên cứu về phát triển giao diện người dùng tự động, nhưng chủ yếu tập trung theo hướng hỗ trợ xây dựng mô hình giao diện bằng cách kết hợp các thành phần đã có. Tuy nhiên, phương pháp này thường không linh hoạt và gặp khó khăn khi giải quyết các vấn đề không chuẩn mực hoặc phức tạp. Trong các ứng dụng tương tác chuyên sâu về dữ liệu, yêu cầu về giao diện người dùng thường phức tạp và đòi hỏi sự linh hoạt cao. Điều này đặt ra thách thức về việc tự động sinh giao diện một cách hiệu quả và đáng tin cậy. Các phương pháp hiện tại chưa thể đáp ứng đủ nhu cầu này và có thể gặp khó khăn khi áp dụng vào các dự án thực tế.

Do đó, cần có nghiên cứu và phát triển thêm về phương pháp tự động sinh giao diện người dùng trong các ứng dụng tương tác chuyên sâu về dữ liệu. Các phương pháp này cần đảm bảo tính linh hoạt và đáng tin cậy, giúp giải quyết các vấn đề phức tạp một cách hiệu quả. Đồng thời, cần tập trung vào việc sử dụng các công nghệ và công cụ mới nhất để hỗ trợ quá trình phát triển giao diện người dùng một cách tự động và hiệu quả hơn. Tầm quan trọng của việc phát triển các phương pháp và công cụ tự động sinh giao diện người dùng trong các ứng dụng tương tác chuyên sâu về dữ liệu là cực kỳ cần thiết để giúp tăng tính hiệu quả và chất lượng trong quá trình phát triển phần mềm [2]. Điều này giúp tiết kiệm thời gian và công sức của các nhà phát triển, đồng thời đảm bảo tính nhất quán và linh hoạt trong việc xây dựng các giao diện người dùng.

Trong bối cảnh đó, sinh tự động giao diện bản mẫu theo hướng mô hình đã xuất hiện như một giải pháp tiên tiến, giúp giải quyết các vấn đề trên và tạo ra giao diện người dùng tự động từ các mô hình trừu tượng [3]. Phương pháp này kết hợp giữa phát triển hướng mô hình (Model-Driven Development - MDD) và các công nghệ tự động hóa để tạo ra các bản mẫu giao diện, giúp giảm thiểu công sức và thời gian trong quá trình phát triển.

Việc hiện thực hóa được tiếp cận sinh giao diện người dùng theo hướng mô hình đặt ra các thách thức chính sau. Trước hết, cần xây dựng kỹ thuật biểu diễn chính xác mô hình đầu vào tương ứng với đặc tả yêu cầu và mô hình kết quả tương ứng với giao diện người dùng. Tiếp đó, cần phát triển bộ chuyển đổi mô hình để thực hiện các bước chuyển đổi từ đặc tả yêu cầu sang bản mẫu giao diện người dùng. Đó cũng là những thách thức trọng tâm được xem xét trong luận văn.

1.2 Mục tiêu và phạm vi

Luận văn tập trung xây dựng phương pháp sinh tự động bản mẫu giao diện người dùng từ các đặc tả yêu cầu chức năng. Cụ thể, luận văn sử dụng ngôn ngữ đặc tả yêu cầu FRSL để biểu diễn mô hình đầu vào, ngôn ngữ mô hình hóa tương tác IFML để biểu diễn kết quả mô hình giao diện người dùng. Tiếp đó, luận văn tập trung phát triển các bộ chuyển đổi mô hình để hiện thực hóa phương pháp đề xuất.

Đóng góp chính của luận văn bao gồm:

- Tìm hiểu tổng quan kỹ thuật phát triển hướng mô hình (MDE), đặc biệt, (1) trình bày kỹ thuật đặc tả yêu cầu chức năng với ngôn ngữ FRSL và kỹ thuật xây dựng mô hình tương tác với ngôn ngữ IFML, (2) diễn giải các kỹ thuật chuyển đổi mô hình, bao gồm chuyển đổi sang mô hình (M2M) và sang văn bản (M2T).
- Xây dựng phương pháp sinh tự động bản mẫu giao diện người dùng từ đặc tả yêu cầu chức năng. Hai bộ chuyển đổi mô hình được luận văn phát triển để hiện thực hóa phương pháp: (1) Frsl2Ifml để chuyển đổi mô hình yêu cầu FRSL sang mô hình tương tác IFML và (2) Ifml2GUI để chuyển đổi mô hình tương tác IFML sang mã nguồn biểu diễn giao diện người dùng (GUI).
- Phát triển bộ công cụ hỗ trợ phương pháp ở dạng các plugin trên nền tảng Eclipse. Đồng thời, minh họa và đánh giá phương pháp với một số ứng dụng cụ thể.

1.3 Cấu trúc của luận văn

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

Chương 1. Mở đầu: Đặt vấn đề, mục tiêu, phạm vi và cấu trúc của luận văn.

Chương 2. Kiến thức nền tảng: Trình bày các khái niệm cơ bản về biểu diễn mô hình phần mềm, chuyển đổi mô hình, đặc tả yêu cầu chức năng với ngôn ngữ FRSL, kỹ thuật tạo bản mẫu giao diện phần mềm và giới thiệu mô hình luồng tương tác IFML.

Chương 3. Sinh giao diện bản mẫu dựa vào chuyển đổi mô hình: Nghiên cứu, giới thiệu tổng quan phương pháp, vận dụng Acceleo và ATL để chuyển đổi mô hình

Chương 4. Cài đặt và thực nghiệm: Vận dụng, thực nghiệm đánh giá kỹ thuật phát triển ngôn ngữ đặc tả giao diện.

Chương 5. Kết luận: Tổng kết về đề tài, đưa ra các đề xuất và hướng nghiên cứu tiếp theo cho phương pháp.

CHƯƠNG 2: KIẾN THỨC NỀN TẢNG

Trong chương này, luận văn sẽ tập trung trình bày các khái niệm cơ bản về biểu diễn mô hình phần mềm, chuyển đổi mô hình và kỹ thuật tạo bản mẫu giao diện phần mềm. 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 Kỹ nghệ hướng mô hình

Kỹ nghệ hướng mô hình (Model-driven Engineering, MDE) là một phương pháp tiếp cận trong lĩnh vực công nghệ phần mềm, nhằm tạo ra và sử dụng các mô hình để phát triển hệ thống phần mềm[4]. Mục đích chính của MDE là nâng cao hiệu quả và chất lượng của quy trình phát triển phần mềm bằng cách sử dụng mô hình để trừu tượng hóa và tự động hóa các giai đoạn quy trình phát triển. Các mô hình trong MDE có thể trải dài từ giao tiếp giữa mọi người (như là các mô hình hình thức, mô hình ngôn ngữ hướng dẫn người dùng) cho đến mô hình phần mềm (như là mô hình tương tác, mô hình thiết kế, mô hình triển khai).

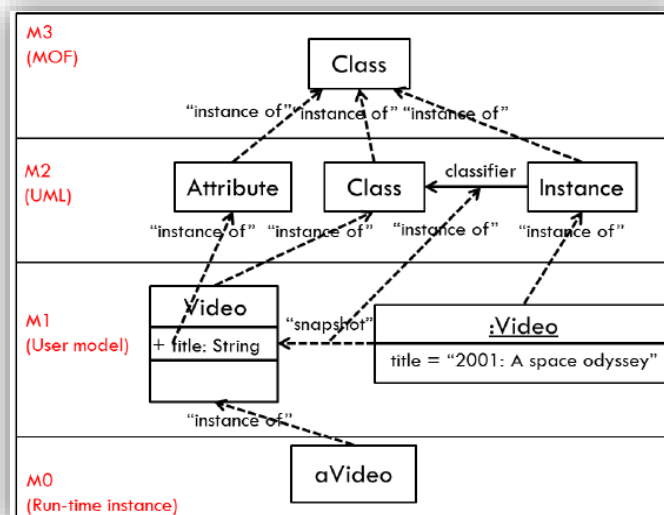
2.1.1 Biểu diễn mô hình phần mềm

Mô hình (Model) là sự trừu tượng hóa một khía cạnh của thực tế (hiện tại hoặc tương lai) được xây dựng cho một mục đích nhất định [5]. Các mô hình được thiết kế hoạt động giống như các hệ thống trong thực tế nhằm phục vụ một số hoạt động nhất định.

Trong kỹ thuật, người ta muốn chia nhỏ một hệ thống phức tạp thành nhiều mô hình khi cần thiết để chúng trở nên dễ hiểu, có thể phân tích và cuối cùng có thể được xây dựng. Một mô hình trong kỹ thuật phần mềm có thể có một hoặc nhiều mục đích. Đầu tiên, mô hình phục vụ như một sự khám phá các khả năng giải pháp. Tiếp theo, mô hình được sử dụng để xây dựng hệ thống cũng như tùy chỉnh hệ thống được thiết kế. Chính vì vậy, mô hình cũng được xem là tài liệu của hệ thống. Ngoài ra, mô hình được sử dụng để mô phỏng một hiện tượng trong thế giới thực hay một hệ thống chưa tồn tại. Cuối cùng, mô hình được sử dụng để tạo mã hay làm nguồn tài liệu giúp khách hàng có thể hiểu rõ hơn về sản phẩm.

Trong phát triển phần mềm, mô hình là một biểu diễn trừu tượng của hệ thống hoặc ứng dụng mà sẽ được xây dựng. Mô hình thể hiện cấu trúc, hành vi và các quan hệ giữa các thành phần của hệ thống, giúp người phát triển, nhà quản lý dự án và các bên liên quan có cái nhìn tổng quan và rõ ràng hơn về cách phần mềm sẽ hoạt động.

Để quá trình định nghĩa một mô hình chính xác, lúc này chúng ta cần đến siêu mô hình (metamodel). Siêu mô hình được định nghĩa là một mô hình mô tả cú pháp trừu tượng của một ngôn ngữ [5]. Trong thực tế, siêu mô hình đóng vai trò quan trọng trong việc định nghĩa của một ngôn ngữ mô hình hóa. Điều này bởi vì siêu mô hình cung cấp cách để mô tả toàn bộ tập hợp các lớp mô hình mà ngôn ngữ đó có thể biểu diễn. Vì vậy, người ta có thể định nghĩa các mô hình thực tế và sau đó xây dựng các mô hình mô tả siêu mô hình. Một cách cụ thể, một mô hình tuân theo siêu mô hình khi tất cả các phần tử của nó có thể được biểu thị dưới dạng các thể hiện của các lớp siêu mô hình, điều đó được biểu diễn ở Hình 2.1.



Hình 2.1: Mối quan hệ giữa mô hình và siêu mô hình [6].

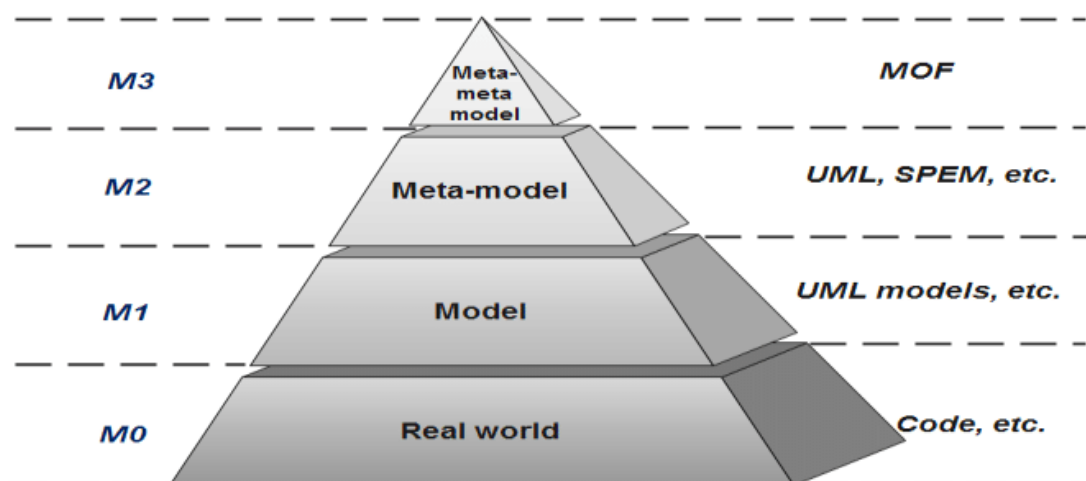
Siêu mô hình mô tả cú pháp trừu tượng của ngôn ngữ. MDE sử dụng cách tiếp cận hướng đối tượng để lập mô hình cấu trúc như vậy (ví dụ: MOF và Ecore) mô tả các yếu tố và liên kết có thể có giữa chúng. Do đó, tất cả các phần tử của mô hình được phân loại theo các khái niệm có thể được tạo trong mô hình và các mối quan hệ tổ chức chúng.

Mô hình hợp lệ đối với một ngôn ngữ được xác định bởi metamodel của ngôn ngữ đó, và mô hình phải tuân thủ các quy tắc và cấu trúc đã được định nghĩa trong metamodel. Có thể mô hình hóa metamodel thành meta-metamodel để mô tả các tính chất của metamodel. Tiếp tục như vậy, có thể mô hình hóa nhiều cấp độ (meta)model khác nhau, tạo ra mức độ trừu tượng tăng dần.

Mô hình từ lâu đã được sử dụng trong khoa học và kỹ thuật như một công cụ cơ bản để quản lý độ phức tạp. Mô hình hóa phân tách các mối quan tâm bằng cách trừu tượng hóa các khía cạnh cụ thể của thực tế cho các mục đích cụ thể. Cách tiếp cận này đã trở nên tương đối phổ biến trong thời gian gần đây nhằm giải quyết các mối quan tâm về phân tích và thiết kế, đặc biệt dựa vào ngôn ngữ mô hình hóa UML

Model-Driven Engineering (MDE) là một phương pháp hướng tới cải thiện quá trình phát triển các hệ thống phức tạp bằng cách tập trung vào trừu tượng hóa mối quan tâm hơn so với lập trình thông thường thông qua việc sử dụng các mô hình. MDE đề xuất tạo ra các mô hình trừu tượng, từ đó tự động sản xuất mã và các thành phần cần thiết cho hệ thống. Điều này giúp cải thiện tính tái sử dụng, hiệu quả và tính đáng tin cậy của hệ thống.

Trong MDE, mô hình hóa được sử dụng để định nghĩa các loại mô hình và quan hệ giữa chúng. Nó là một mô hình trừu tượng cao nhất trong một hệ thống mô hình hóa. OMG (Object Management Group) đã định nghĩa một mô hình hóa kim tự tháp để biểu diễn cấu trúc của mô hình hóa UML:



Hình 2.2: Kim tự tháp mô hình của OMG.

Hình 2.2 mô tả kim tự tháp mô hình của OMG bao gồm:

- Mức M0: Thế giới thực được biểu diễn ở đáy của Kim tự tháp; đó là thực tế mà người ta muốn mô hình hóa.
- Mức M1: Các mô hình đại diện cho thực tế này cấu thành cấp độ M1. Một mô hình là một sự trừu tượng hóa của một hệ thống, được mô hình hóa như một tập hợp các sự kiện được xây dựng với một ý định cụ thể. Nó có thể đại diện cho cả một hệ thống (cấu trúc, hành vi và thuộc tính phi chức năng) hoặc chỉ một khía cạnh của hệ thống bằng cách che khuất những khía cạnh khác.
- Mức M2: Khái niệm mô hình trong MDE một cách rõ ràng đề cập đến định nghĩa của các ngôn ngữ được sử dụng để xây dựng. Ngôn ngữ mà mô hình này được thể hiện phải được xác định rõ ràng. Các nhà nghiên cứu đã gọi đây là ngôn ngữ mô hình hóa một siêu mô hình. Siêu mô hình là một mô hình định nghĩa một ngôn ngữ mô hình hóa. Nó có thể xác định chính xác các khái niệm về ngôn ngữ mô hình hóa và thiết lập mối quan hệ giữa các khái niệm này.
- Cấp độ M3: Cơ sở siêu đối tượng (MOF) là một ngôn ngữ định nghĩa siêu mô hình tiêu chuẩn để tránh sự không phù hợp, sự gia tăng của các siêu mô hình khác nhau và không tương thích. Siêu mô hình phải được xác định từ một mô hình ngôn ngữ gọi là meta-meta-model. Để hạn chế số lượng mức độ trừu tượng, mô hình meta-meta sau đó phải có thuộc tính siêu tuần hoàn, nghĩa là khả năng tự mô tả.

Hiện nay có nhiều khung mô hình khác nhau được phát triển, trong đó khung mô hình hóa Eclipse Modeling Framework (EMF) đang trở nên nổi bật hơn để triển khai siêu mô hình. EMF là một công cụ mạnh mẽ và phổ biến trong việc phát triển các ứng dụng dựa trên mô hình. Công cụ này giúp đơn giản hóa việc tạo, hiển thị, lưu trữ và xử lý các mô hình trong ứng dụng Java. EMF sử dụng siêu ngôn ngữ Ecore để mô tả các mô hình. Ecore có thể được coi là một hình thức của Meta Object Facility (MOF) dành riêng cho EMF. Ecore sử dụng để định nghĩa cấu trúc của các mô hình và cung cấp các khái niệm như lớp, thuộc tính, quan hệ, cũng như ràng buộc và tính năng mở rộng.

2.1.2 Chuyển đổi mô hình

Chuyển đổi mô hình đóng vai trò linh hoạt, tùy thuộc vào việc ngôn ngữ nguồn và ngôn ngữ đích có cùng mức độ trừu tượng hay không, cùng sử dụng trong một miền hay không. Nó có thể ứng dụng để tạo ra các mô hình cấp thấp hơn từ những mô hình cấp cao hơn, cũng có thể tối ưu hóa, tái cấu trúc lại các mô hình hiện có hoặc tạo ra một mô hình ở miền khác.

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 (M2M) và Mô hình sang văn bản (M2T).

2.1.2.1. Chuyển đổi mô hình sang mô hình

Mô hình không phải là thực thể tĩnh, 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.

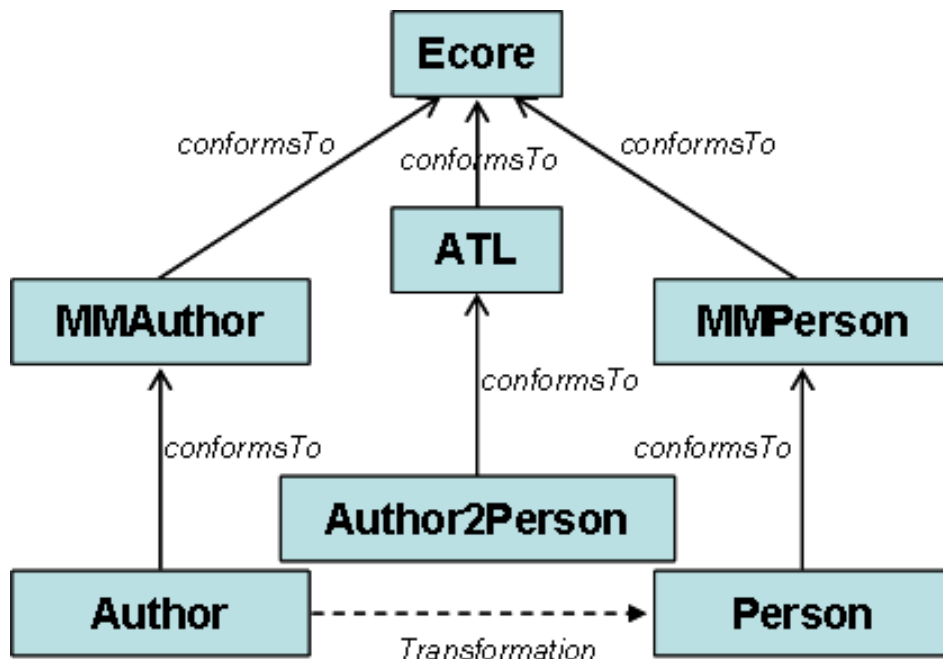
Trong ngôn ngữ M2M, ATL [7] là một trong những ngôn ngữ chuyên mô hình được sử dụng rộng rãi nhất trong cả nghiên cứu và công nghiệp:

- Ngôn ngữ dựa trên luật: ATL là một ngôn ngữ biến đổi dựa trên luật (rule-based transformation language), trong đó các luật biến đổi được xác định để ánh xạ các yếu tố trong mô hình nguồn sang mô hình đích. Các luật biến đổi trong ATL sử dụng cú pháp dễ hiểu và linh hoạt, giúp người lập trình dễ dàng định nghĩa các quy tắc biến đổi.
- Sử dụng OCL (Object Constraint Language): ATL sử dụng OCL để xác định các ràng buộc và điều kiện cho các luật biến đổi. OCL là một ngôn ngữ định nghĩa ràng buộc trên các đối tượng trong mô hình, giúp đảm bảo tính chính xác và đáng tin cậy của quá trình biến đổi.
- Cơ chế ánh xạ và quy tắc biến đổi: ATL cung cấp cơ chế ánh xạ (matching) giữa các yếu tố trong mô hình nguồn và mô hình đích, từ đó xác định các quy tắc biến đổi cụ thể. Các ánh xạ này định nghĩa mối quan hệ giữa các phần tử trong hai mô hình, giúp hiểu cách thức biến đổi được thực hiện.
- Hỗ trợ nhiều ngôn ngữ mô hình: ATL hỗ trợ nhiều ngôn ngữ mô hình phổ biến, bao gồm UML (Unified Modeling Language), Ecore (Eclipse Modeling Framework), XML (eXtensible Markup Language), và nhiều ngôn ngữ mô hình khác.
- Ứng dụng trong nghiên cứu và công nghiệp: ATL được sử dụng rộng rãi trong cả nghiên cứu và công nghiệp. Nó là một công cụ mạnh mẽ giúp giảm thiểu công việc lập trình thủ công và tăng tính tái sử dụng khi phát triển các ứng dụng dựa trên mô hình.

ATL thường được lựa chọn là ngôn ngữ chuyển đổi cho các phép biến đổi ngoại lai bởi vì đây là một trong những ngôn ngữ được trang bị nhiều công cụ hỗ trợ tối ưu hóa quá trình. ATL là một ngôn ngữ được xây dựng chủ yếu dựa trên nguyên tắc của OCL, nhưng nó cung cấp thêm các tính năng ngôn ngữ được tối ưu hóa cho các phép biến đổi mô hình mà có thể bị thiếu trong OCL.

ATL được thiết kế dưới dạng một ngôn ngữ chuyển đổi mô hình kết hợp với các cấu trúc khai báo và mệnh lệnh. Các phép biến đổi trong ATL được thực hiện theo hướng một chiều, tức là để chuyển đổi từ ngôn ngữ A sang ngôn ngữ B và từ ngôn ngữ B sang ngôn ngữ A, cần phải phát triển hai phép biến đổi riêng biệt. Các phép biến đổi trong ATL hoạt động dựa trên mô hình nguồn để đọc dữ liệu và mô hình đích để ghi dữ liệu. Trong quá trình chuyển đổi, các mô hình nguồn được truy vấn để lấy thông tin nhưng không bị thay đổi.

Hình 2.3 cung cấp cho chúng ta cái nhìn tổng quan về cách ATL hoạt động, ở đây ATL đóng vai trò là công cụ chuyển đổi (Author2Person) để sinh ra mô hình Person tuân theo metamodel MMPerson, từ mô hình Author tuân theo metamodel MMAuthor. Trong hình minh họa, cả ba metamodel MMAuthor, ATL và MMPerson đều được biểu diễn bằng ngữ nghĩa của metamodel Ecore.



Hình 2.3: Ví dụ về một chương trình chuyển với ATL.

Một phép biến đổi trong ATL được định nghĩa như một mô-đun, bao gồm phần đầu và phần thân. Phần đầu của phép biến đổi chứa tên của mô-đun chuyển đổi, khai báo mô hình nguồn và mô hình đích được nhập bằng metamodel của chúng. Phần thân của mô-đun bao gồm một tập hợp các luật và hàm trợ giúp được liệt kê theo thứ tự tùy ý sau phần đầu.

Rules

Mỗi luật (rule) mô tả cách một phần của mô hình nguồn được biến đổi thành một phần của mô hình đích. Trong ATL, có hai loại khai báo luật: *matched rules* và *lazy rules*. *Matched rules* là các quy tắc được tự động áp dụng lên mô hình nguồn bởi công cụ thực thi ATL. Còn *lazy rules*, chúng cần được gọi một cách rõ ràng từ một luật khác, cho phép nhà phát triển có khả năng kiểm soát quá trình biến đổi mô hình.

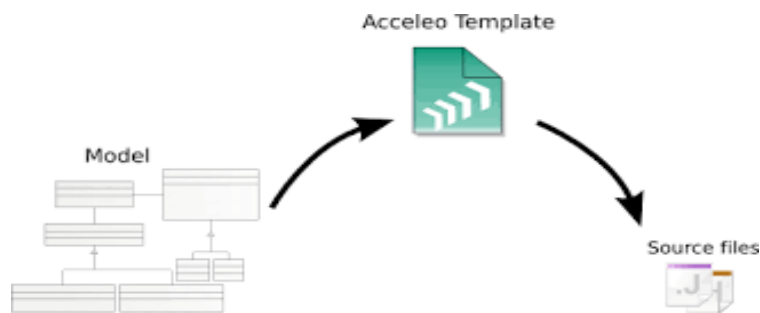
Helpers

Helpers là các hàm hỗ trợ cho phép khả năng phân tích mã nguồn ATL được sử dụng theo cách khác nhau trong quá trình biến đổi. Các helpers có thể đại diện cho một thuộc tính có khả năng truy cập trong suốt quá trình chuyển đổi hoặc đại diện cho một hoạt động tính toán giá trị cho một đối tượng trong ngữ cảnh cụ thể, được xác định bởi tham số đầu vào.

2.1.2.2. Chuyển đổi mô hình sang văn bản

Quá trình chuyển đổi mô hình sang văn bản (Model-to-Text - M2T) là một phương pháp trong lĩnh vực phát triển phần mềm dựa trên mô hình (model-driven development) nhằm tạo ra các tập tin văn bản từ các mô hình đã được xác định trước. Mục tiêu chính của các phép chuyển đổi M2T là tạo ra các tập tin văn bản hoặc mã nguồn dựa trên các mô hình đã được xác định trước. Điều này giúp tự động hóa việc sinh mã và giảm thiểu công việc lập trình thủ công. Phép chuyển đổi M2T rất hữu ích trong việc sinh mã nguồn từ các mô hình mô tả cấu trúc và logic ứng dụng. Điều này giúp tăng tính tái sử dụng và hiệu quả trong quá trình phát triển phần mềm. Các phép chuyển đổi M2T đóng vai trò cầu nối giữa các mô hình và nền tảng thực thi hoặc các công cụ phân tích khác. Nó giúp thực hiện việc biến đổi mô hình thành các đầu ra cụ thể mà các nền tảng hoặc công cụ có thể hiểu được.

Acceleo là một ngôn ngữ chuyển đổi mô hình sang văn bản (Model-to-Text - M2T) được sử dụng trong lĩnh vực phát triển phần mềm dựa trên mô hình (model-driven development). Nó cho phép tạo ra các tập tin văn bản, mã nguồn, tài liệu hoặc các đầu ra khác từ các mô hình đã được xác định trước. Acceleo sử dụng cú pháp dựa trên cấu trúc để xác định các phép chuyển đổi từ mô hình sang văn bản. Cú pháp này dựa trên các mẫu văn bản (text templates) được định nghĩa trong các file Acceleo (.mtl). Trong các mẫu này, người lập trình có thể sử dụng các câu lệnh, biểu thức và ràng buộc để ánh xạ các yếu tố trong mô hình thành văn bản. Acceleo hỗ trợ tích hợp với nhiều ngôn ngữ mô hình phổ biến như UML (Unified Modeling Language), Ecore (Eclipse Modeling Framework), và các ngôn ngữ mô hình khác. Điều này giúp người lập trình dễ dàng tạo ra các phép chuyển đổi từ các mô hình đã xác định trước. Công cụ này cung cấp các tính năng hỗ trợ việc biên dịch, gỡ lỗi, và tự động hoàn thành mã trong quá trình viết các phép chuyển đổi M2T. Acceleo cho phép mô-đun hóa các mẫu văn bản và tái sử dụng chúng trong nhiều văn bản khác nhau. Điều này giúp giảm thiểu sự lặp lại và tăng tính tái sử dụng trong việc tạo ra các đầu ra từ các mô hình.



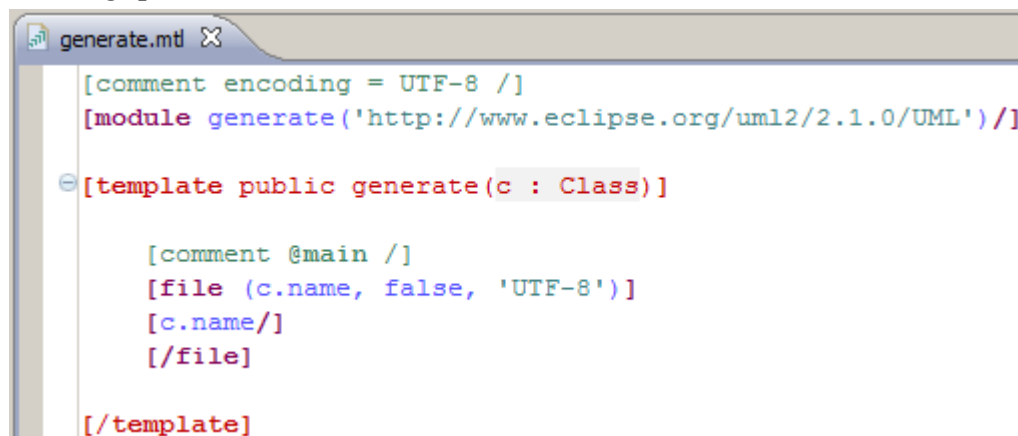
Hình 2.4: Acceleo sinh tự động mã nguồn bởi chuyển đổi mô hình M2T.

Hình 2.4 mô tả việc sinh tự động mã nguồn bởi chuyển đổi mô hình M2T. Acceleo được chọn vì đây là công cụ phổ biến để chuyển đổi mô hình M2T cung cấp tính phù hợp với thực tế và sự hỗ trợ đầy đủ. Acceleo cung cấp một ngôn ngữ dựa trên mẫu để xác định các mẫu tạo mã. Ngôn ngữ này đi kèm với một giao diện lập trình ứng dụng (API) mạnh mẽ hỗ trợ OCL cùng với các hoạt động bổ sung để làm việc với các tài liệu dựa trên văn bản, ví dụ như các hàm nâng cao để xử lý chuỗi. Acceleo cũng được kết hợp với các công cụ mạnh mẽ như trình chỉnh sửa với kiểm tra cú pháp, hoàn thành mã, tái cấu trúc, phát hiện lỗi, trình gỡ lỗi, trình biên dịch và các API truy xuất nguồn gốc, cho phép theo dõi các phần tử trong mô hình liên quan đến mã được tạo ra và ngược lại.

Trước khi có khả năng xác định mẫu trong Acceleo, cần tạo một mô-đun để hoạt động như một phạm vi chứa các mẫu. Mô-đun cũng định nghĩa meta-model mà các mẫu sẽ dựa trên. Điều này giúp cho các mẫu nhận biết các lớp meta-model hiện có và có thể sử dụng chúng như các loại dữ liệu trong mẫu. Mẫu trong Acceleo luôn được định rõ cho một lớp meta-model cụ thể. Bên cạnh loại phân tử trong mô hình, có thể xác định các điều kiện trước, tương tự như việc sử dụng điều kiện lọc trong ATL.

Acceleo cung cấp một số thẻ đánh dấu phổ biến trong các ngôn ngữ chuyển đổi M2T có sẵn khác. Một số khái niệm chính được mô tả trong mẫu Acceleo được dùng để chuyển đổi mô hình:

- File: Để tạo mã, tệp cần được mở, điền nội dung và sau đó đóng tệp, tương tự như cách chúng ta đã thấy khi tạo mã trong ngôn ngữ Java. Trong Acceleo, có một thẻ tệp đặc biệt được sử dụng để in nội dung được tạo ra giữa phần đầu và phần cuối của tệp dành cho một tệp cụ thể. Đường dẫn và tên của tệp được xác định thông qua thuộc tính của thẻ.



Hình 2.5: Mô-đun chỉnh sửa Acceleo với cú pháp.

- Cấu trúc điều khiển: Trong Acceleo, có các thẻ được sử dụng để định nghĩa cấu trúc điều khiển như vòng lặp, cho phép lặp qua các tập hợp các phân tử. Ví dụ, các thẻ này rất hữu ích để làm việc với các tham chiếu đa giá trị thu được khi điều hướng đến một tập hợp các phân tử, cũng như cho các nhánh điều kiện (nếu có) được gắn kèm.
- Truy vấn: Các truy vấn OCL có thể được xác định bằng cách sử dụng thẻ truy vấn, tương tự như cách trình trợ giúp được sử dụng trong ATL. Những truy vấn này có thể được gọi trong phạm vi toàn bộ mẫu và được sử dụng để tính toán mã định kỳ.

- **Biểu thức:** Các biểu thức phổ biến được sử dụng để tính toán giá trị và tạo ra các phần của văn bản đầu ra. Quá trình gọi các mẫu tương tự với việc gọi các phương thức trong ngôn ngữ Java.
- **Phạm vi truy cập:** Phạm vi truy cập giúp hỗ trợ các dự án mà chỉ có thể tạo một phần mã. Đặc biệt, cần hỗ trợ đặc biệt để bảo vệ mã được thêm thủ công khỏi các sửa đổi tệp trong các lần chạy trình tạo mã tiếp theo. Chức năng này giúp các khu vực được bảo vệ được sử dụng để đánh dấu các phần trong mã đã tạo mà sẽ không bị ghi đè lại bởi các lần chạy tiếp theo.

Modules

Modules trong Acceleo được biểu diễn dưới dạng các tệp ".mtl" và chứa các templates (để tạo mã) hoặc queries (để trích xuất thông tin từ các mô hình đang xử lý). Mỗi tệp phải bắt đầu với khai báo modules theo cú pháp:

```
[module <module_name>('metamodel_URI_1', 'metamodel_URI_2')]

import qualified::name::of::imported::module
```

Templates

Mẫu là tập hợp các câu lệnh Acceleo được sử dụng để tạo văn bản. Chúng được phân tách bằng các thẻ [template...][template].

```
[template public name(arg : Type)]

    template_expression

[/template]
```

Queries

Các câu truy vấn được áp dụng để trích xuất thông tin từ mô hình, và chúng có khả năng trả về một giá trị đơn hoặc một tập hợp giá trị. Việc sử dụng OCL để thực hiện truy vấn được thực hiện trong phạm vi thẻ [query ... /]. Các truy vấn được định nghĩa để luôn trả về cùng một kết quả mỗi khi chúng được gọi với các đối số giống nhau

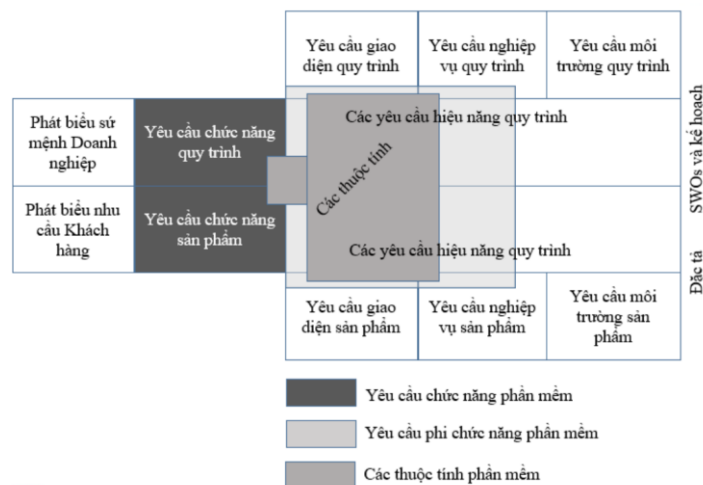
```
[query public name(arg : Type) : OclAny = self /].
```

2.2 Đặc tả yêu cầu chức năng với ngôn ngữ FRSL

Đề tạo chế tác từ một yêu cầu, bước quan trọng là đặc tả yêu cầu, nhằm mô tả và xác định chức năng cũng như hành vi của hệ thống. Phần này của luận văn sẽ cung cấp một cái nhìn tổng quan về yêu cầu, giải thích cách thực hiện việc đặc tả yêu cầu thông qua ca sử dụng và ngôn ngữ chuyên biệt miền FRSL [8]. Cuối cùng, luận văn sẽ trình bày một số tình huống chuyển đổi mô hình, qua đó tạo ra các chế tác phần mềm.

2.2.1 Giới thiệu

Yêu cầu có thể là các tuyên bố về nhiệm vụ mà hệ thống phải thực hiện (functional requirements), hoặc các ràng buộc liên quan đến cách hệ thống hoạt động hoặc quá trình phát triển (nonfunctional requirements). Yêu cầu thể hiện sự đồng thuận giữa các bên liên quan và cần phải được xử lý một cách chuyên nghiệp thông qua việc thu thập và phân tích. Yêu cầu chính là cơ sở cho việc thiết kế và triển khai phần mềm.



Hình 2.6 Tổng hợp phân loại yêu cầu theo Ralph R. Young [9].

Bước đầu tiên trong quá trình phát triển phần mềm là việc đặc tả yêu cầu, điều này đòi hỏi sự tương tác giữa các bên liên quan, bao gồm cả khách hàng và nhà phát triển. Tùy thuộc vào các góc độ khác nhau, yêu cầu có thể được phân loại theo nhiều tiêu chí khác nhau. Theo Ralph R. Young [9], các loại yêu cầu được phân thành hai nhóm chính là phần cứng và phần mềm, như được minh họa trong Hình 2.6 mô tả yêu cầu về phần cứng được đặc trưng và giới hạn bởi các yêu cầu về hiệu năng mà hệ thống phải đáp ứng, các kết nối giao diện, thông số môi trường và các mô tả kỹ thuật. Trong khi đó, yêu cầu về phần mềm bao gồm cả các khía cạnh chức năng và phi chức năng:

- Yêu cầu chức năng xác định các hành động mà phần mềm cần thực hiện, đặc thù là các chức năng cốt lõi mà phần mềm tương lai (hệ thống sẽ thành) phải có. Ví dụ, một phần mềm quản lý nhân sự có thể bao gồm các chức năng quản lý thông tin cá nhân, xử lý đơn xin nghỉ, và quản lý hợp đồng lao động trong một tổ chức.
- Yêu cầu phi chức năng mô tả các thuộc tính của hệ thống như độ tin cậy và bảo mật, đồng thời xác định các ràng buộc mà phần mềm tương lai phải tuân theo liên quan đến chức năng của nó.

Sau khi đã xác định và phân loại các loại yêu cầu, bước tiếp theo là giai đoạn đặc tả yêu cầu thành văn bản. Giai đoạn này kết hợp với quá trình khảo sát yêu cầu. Các thông tin đầu vào cho giai đoạn đặc tả và tài liệu hóa bao gồm nhiều phát biểu 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 các định nghĩa về các khái niệm.

Kết quả của giai đoạn đặc tả yêu cầu là phiên bản đầu tiên của tài liệu đặc tả yêu cầu, mà thỏa mãn tất cả các yêu cầu đầu vào đã được nêu. Tài liệu đặc tả cần phải dễ hiểu và dễ dàng truy xuất, để tạo điều kiện cho việc giao tiếp hiệu quả giữa các chuyên gia kỹ thuật và các bên liên quan

Ca sử dụng (Use Case) là một tập hợp chuỗi hành động mà hệ thống thực hiện để tạo ra kết quả có giá trị có thể quan sát được cho một tác nhân cụ thể. Ca sử dụng thể hiện sự tương tác giữa các thực thể liên quan trong hệ thống, mô tả hành vi và tương tác của hệ thống trong các ngữ cảnh khác nhau, và cung cấp phản hồi đối với các bên liên quan và tác nhân chính. Ca sử dụng bao gồm tất cả các kịch bản liên quan đến mục tiêu chính của các tác nhân. Thường được lập bởi người phân tích yêu cầu và được sử dụng trong các giai đoạn như lập kế hoạch yêu cầu hệ thống, thiết kế hệ thống và kiểm thử phần mềm. Việc này giúp các nhóm phát triển xác định và hiểu rõ yêu cầu chức năng của hệ thống, đồng thời cung cấp cơ sở để tạo ra thiết kế và triển khai hệ thống một cách hiệu quả. Một đặc tả ca sử dụng bao gồm các thành phần chính sau:

- **Tác nhân (Actor):** Tác nhân là một thực thể có khả năng thực hiện hành vi, có thể là máy móc, hệ thống máy tính, con người, tổ chức hoặc kết hợp của chúng. Tác nhân tương tác và trao đổi thông tin với hệ thống, hoặc sử dụng các chức năng của hệ thống. Mỗi ca sử dụng được kích hoạt bởi một tác nhân cụ thể, và khi một ca sử dụng được thực thi, nó có thể gửi thông tin đến các tác nhân khác trong hệ thống.

- **Tác nhân chính (Primary actor) và Tác nhân phụ (Secondary actor):** Tác nhân chính là những thực thể sử dụng hệ thống để thực hiện mục tiêu cụ thể, và chúng tận dụng những chức năng chính của hệ thống. Các ca sử dụng được sử dụng để miêu tả các tương tác giữa hệ thống và những tác nhân chính, nhằm đạt được mục tiêu của chính những tác nhân đó. Tác nhân phụ là những tác nhân mà hệ thống cần hỗ trợ để đảm bảo mục tiêu của các tác nhân chính được thực hiện thành công.
- **Các liên kết:** Sự tham gia của một tác nhân trong một ca sử dụng được biểu thị bằng việc liên kết tác nhân với ca sử dụng tương ứng qua một liên kết vững chắc. Các tác nhân có khả năng được kết nối với các ca sử dụng thông qua các liên kết, thể hiện rằng tác nhân và ca sử dụng tương tác với nhau thông qua trao đổi thông điệp.
- **Đường bao của hệ thống:** Đường bao hệ thống có thể ảnh hưởng đến toàn bộ phạm vi của hệ thống, như được định rõ trong tài liệu yêu cầu. Đối với các hệ thống lớn và phức tạp, mỗi mô-đun có thể là ranh giới của hệ thống.

Đặc tả ca sử dụng được tổ chức thành ba phần chính:

A) **Tóm tắt (Summary):** Đây là phần mô tả tổng quan về ca sử dụng.

- **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:** Điều kiện cần thỏa mãn để Use Case thực hiện thành công.

B) **Luồng (Flow):** Phần này mô tả chi tiết các luồng sự kiện trong ca sử dụng.

- **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) Thông tin bổ sung (*Additional Information*): Phần này cung cấp thông tin bổ sung liên quan đến ca sử dụng.
- **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 Mô hình yêu cầu chức năng FRSL

Thông qua mô hình ca sử dụng, các yêu cầu chức năng và hành vi của hệ thống được thể hiện. Ca sử dụng dùng để miêu tả tương tác giữa các thành phần tham gia trong hệ thống, thực hiện các mục tiêu cụ thể trong một môi trường cụ thể. Tuy nhiên, việc đặc tả yêu cầu bằng ca sử dụng gặp những hạn chế, như sự không nhất quán về biểu mẫu trong quá trình đặc tả, khả năng không thể diễn đạt đầy đủ các điều kiện ràng buộc của hệ thống, cũng như các yêu cầu về thuộc tính dữ liệu hoặc luồng dữ liệu giữa các quy trình. Do đó, phần đặc tả yêu cầu trong ca sử dụng thường chỉ đủ để trao đổi mô hình tổng quan của hệ thống trong giai đoạn phát triển ban đầu của phần mềm. Trong quá trình sinh chế tác phần mềm, yêu cầu chức năng phải được đặc tả chính xác, đầy đủ và ở mức mô hình hóa để thực hiện quá trình phát triển.

Để giải quyết vấn đề này, ngôn ngữ đặc tả yêu cầu chức năng FRSL (Functional Requirement Specification Language) [8] đã được phát triển dành riêng cho miền vấn đề của ca sử dụng. FRSL được xây dựng từ các khái niệm cơ bản trong miền ca sử dụng, với bộ cú pháp trừu tượng và cú pháp cụ thể giúp người sử dụng đặc tả một cách đầy đủ các thành phần trong mô hình ca sử dụng, bao gồm Ca sử dụng (*use case*), Tác nhân (*actor*), Kịch bản (*scenario*), Các mối quan hệ của ca sử dụng, Luồng chính (*basic flow*), và Luồng thay thế (*alternative flows*).

Mô hình FRSL mang lại mô tả đa dạng các khía cạnh của ca sử dụng. Đầu tiên, nó biểu thị mô hình miền dưới dạng biểu đồ lớp UML/OCL. Thứ hai, cấu trúc của các ca sử dụng được xác định thông qua các quan hệ, bao gồm và mở rộng. Thứ ba, nó mô tả kịch bản của các ca sử dụng. Thứ tư, nó sử dụng các mẫu hình chụp (snapshot pattern) để thể hiện tiền và hậu điều kiện của các ca sử dụng hoặc các bước trong chúng. Cuối cùng, nó định nghĩa các điều kiện bảo vệ cho luồng thay thế (alternative) hoặc các bước rejoin trong ca sử dụng.

Hình 2.8 là đặc tả cho ca sử dụng HandleCashPayment với tác nhân chính tham gia ca sử dụng là Cashier. Tiền và hậu điều kiện của ca sử dụng được đặc tả tương ứng với các từ khóa *ucPrecondition* và *ucPostcondtion* bao gồm các yêu cầu và tham số tương ứng.

2.3 Kỹ thuật tạo bản mẫu giao diện phần mềm

Trong việc phát triển các ứng dụng tương tác, khả năng sử dụng và trải nghiệm của giao diện người dùng là yếu tố quan trọng để đạt được sự chấp nhận và hài lòng của người dùng cuối. Điều này đặc biệt quan trọng vì trải nghiệm người dùng tốt sẽ tạo ra ấn tượng tích cực và khả năng giữ chân người dùng, trong khi trải nghiệm không tốt có thể dẫn đến sự từ chối và giảm hiệu quả của ứng dụng.

Các công cụ tạo mẫu cho phép tạo các bản mẫu giao diện trực quan, tương tác và kiểm tra trước hiệu quả của giao diện với người dùng. Nhờ vào việc tạo mẫu, các bên liên quan có thể thấy trước diễn biến và trải nghiệm của giao diện, đánh giá tính hợp lý và sử dụng của giao diện, từ đó điều chỉnh và cải thiện thiết kế để đáp ứng đúng nhu cầu của người dùng.

Tạo mẫu là một giai đoạn quan trọng trong mô hình vòng đời phần mềm, nó giúp hình dung rõ ràng về giao diện, cung cấp cơ hội cho phản hồi và điều chỉnh thiết kế trước khi tiến hành phát triển phần mềm hoàn chỉnh. Điều này giúp tối ưu hóa chất lượng và hiệu quả của giao diện người dùng và từ đó đóng góp vào thành công của toàn bộ dự án phát triển phần mềm.

- **Hiểu nhu cầu người dùng:** Tạo mẫu giúp đưa ra một hình dung rõ ràng về giao diện cuối cùng cho người dùng và cho phép giao tiếp hiệu quả với các bên liên quan. Khi có một phiên bản sơ bộ của giao diện, người dùng và các bên liên quan có thể dễ dàng hiểu và thẩm định các yêu cầu và chức năng của sản phẩm.
- **Đánh giá thiết kế:** Tạo mẫu giúp nhà phát triển đánh giá tính khả thi và tính hiệu quả của thiết kế giao diện trước khi đầu tư vào việc phát triển sản phẩm hoàn chỉnh. Những điều chỉnh và sửa đổi có thể được thực hiện dễ dàng trong giai đoạn này, giúp tiết kiệm thời gian và tối ưu hóa quá trình phát triển.
- **Giảm rủi ro:** Tạo mẫu giúp giảm rủi ro trong quá trình phát triển, bởi vì nó cho phép phát hiện và khắc phục các vấn đề thiết kế sớm hơn. Những lỗi hoặc không rõ ràng trong giao diện có thể được phát hiện và giải quyết trước khi tiến hành phát triển hoàn chỉnh, từ đó giảm khả năng xảy ra sai sót và lỗi trong sản phẩm cuối cùng.

- Tối ưu hóa trải nghiệm người dùng: Tạo mẫu giúp thử nghiệm và cải tiến trải nghiệm người dùng. Bằng cách có một phiên bản sơ bộ của giao diện, người phát triển có thể thu thập phản hồi từ người dùng và thay đổi thiết kế dựa trên những phản hồi này, từ đó tối ưu hóa trải nghiệm người dùng.
- Hỗ trợ quyết định: Tạo mẫu cung cấp một cái nhìn trực quan về giao diện, giúp đưa ra quyết định và chọn lựa thiết kế tốt nhất cho sản phẩm cuối cùng.

Việc tạo mẫu (prototyping) trong quá trình phát triển giao diện người dùng là một bước quan trọng và có tầm quan trọng cao. Mẫu (prototype) là một phiên bản sơ bộ hoặc mô phỏng của sản phẩm cuối cùng, được sử dụng để hiểu, kiểm tra và đánh giá các yếu tố thiết kế, tính năng và trải nghiệm của giao diện trước khi tiến hành phát triển sản phẩm hoàn chỉnh. Dưới đây là những lý do chính vì sao việc tạo mẫu quan trọng trong phát triển giao diện người dùng:

Có nhiều phương pháp và công cụ phát triển giao diện người dùng (UI) khác nhau để giúp người phát triển tạo và quản lý giao diện người dùng một cách hiệu quả. Dưới đây là một số phương pháp và công cụ phổ biến:

- WYSIWYG (What You See Is What You Get) các IDE và trình xây dựng UI định hướng: Đây là phương pháp sử dụng các công cụ tích hợp (Integrated Development Environment - IDE) và các trình xây dựng giao diện người dùng định hướng để tạo giao diện một cách trực quan. Ví dụ như Microsoft Visual Studio, Eclipse, NetBeans và NextStep đều cung cấp các công cụ WYSIWYG cho việc thiết kế và xây dựng giao diện người dùng.
- Ngôn ngữ đánh dấu: Đây là phương pháp sử dụng các ngôn ngữ đánh dấu như HTML, XML, UIML, XUL, XAML và nhiều ngôn ngữ khác để mô tả cấu trúc và giao diện người dùng của ứng dụng. Các ngôn ngữ này cho phép người phát triển xây dựng các giao diện phức tạp và nhiều tính năng.
- Môi trường phát triển giao diện người dùng dựa trên mô hình (Model-based UI Development): Đây là phương pháp sử dụng các môi trường phát triển dựa trên mô hình (MB-UIDE) để tạo và quản lý giao diện người dùng. Các môi trường này cung cấp các công cụ và mô hình trừu tượng để tạo giao diện một cách tự động và tái sử dụng.

Các phương pháp và công cụ này được sử dụng để giúp người phát triển xây dựng giao diện người dùng một cách dễ dàng, trực quan và hiệu quả. Việc lựa chọn phương pháp và công cụ phù hợp sẽ giúp tối ưu hóa quá trình phát triển và đảm bảo chất lượng của giao diện cuối cùng.

Giao diện người dùng có thể kết hợp nhiều kiểu giao diện khác nhau để đáp ứng nhu cầu và mục tiêu của ứng dụng. Giao diện người dùng đồ họa (GUI) là một kiểu giao diện kết hợp giữa phong cách giao diện WIMP (Windows, Icons, Menus, Pointers) với các yếu tố và tính năng từ các giao diện người dùng khác. GUI thường sử dụng các thành phần đồ họa như biểu tượng, cửa sổ, menu và các đối tượng trực quan để tạo ra môi trường tương tác trực quan và dễ sử dụng cho người dùng.

Thiết kế và bố cục của hộp thoại và màn hình giao diện cũng rất quan trọng trong việc tạo ra trải nghiệm tốt cho người dùng. Cách thông tin được trình bày và sắp xếp trên giao diện có ảnh hưởng đáng kể đến khả năng sử dụng của hệ thống. Một bố cục hợp lý và tổ chức thông tin hợp lý giúp người dùng dễ dàng tìm thấy thông tin cần thiết và thực hiện các tác vụ một cách hiệu quả. Bên cạnh đó, bố cục màn hình để nhập thông tin cũng phải được xây dựng sao cho dễ sử dụng và tránh gây nhầm lẫn hoặc lỗi cho người dùng.

Sự kết hợp và cân nhắc các yếu tố trên giao diện người dùng đóng vai trò quan trọng trong việc tạo ra giao diện hấp dẫn, trực quan và hiệu quả cho người dùng cuối. Quá trình thiết kế giao diện cần tập trung vào trải nghiệm người dùng và đảm bảo rằng hệ thống cung cấp môi trường tương tác thuận tiện và dễ sử dụng. Tương tác giữa người dùng và hệ thống diễn ra trên giao diện người dùng. Các kiểu giao diện phổ biến bao gồm:

- Giao diện dòng lệnh (Command Line Interfaces - CLI): Cung cấp một cách để người dùng giao tiếp và thực thi lệnh trực tiếp thông qua bộ lệnh văn bản. CLI thường được sử dụng cho các ứng dụng và hệ thống dòng lệnh, nhưng có thể khó sử dụng đối với người dùng không có kinh nghiệm.
- Ngôn ngữ tự nhiên (Natural Language Interfaces): Sử dụng ngôn ngữ tự nhiên để tương tác với hệ thống. Hiện tại, các hệ thống sử dụng ngôn ngữ tự nhiên chỉ hạn chế trong một số miền và cụm từ.
- Giao diện hướng menu (Menu-driven Interfaces): Cung cấp một tập hợp các tùy chọn cho người dùng để chọn bằng cách sử dụng menu, các phím số hoặc chữ cái. Các menu thường được sắp xếp theo thứ bậc.
- Hộp thoại hỏi/tra lời, truy vấn (Question/Answer Dialogs): Cung cấp cơ chế đơn giản để cung cấp đầu vào cho một ứng dụng trong miền cụ thể. Người dùng trả lời một loạt câu hỏi trong các hộp thoại điều khiển.
- Windows, Icons, Menu và Pointers (WIMP): Giao diện người dùng phổ biến nhất, sử dụng các cửa sổ, biểu tượng, menu và con trỏ chuột để tương tác với hệ thống.

- Point-and-click: Kiểu giao diện tương tự WIMP, nhưng dễ dàng sử dụng trên các màn hình cảm ứng và không bị ràng buộc bởi chuột.
- Giao diện 3D (3D Interfaces): Trải rộng từ các ứng dụng 3D đơn giản với yếu tố ngoại hình 3D đến không gian làm việc 3D phức tạp hoặc hệ thống thực tế ảo.

Tùy thuộc vào yêu cầu và mục đích của ứng dụng, các kiểu giao diện này có thể được sử dụng độc lập hoặc kết hợp để cung cấp trải nghiệm tốt nhất cho người dùng.

2.4 Biểu diễn mô hình tương tác với IFML

IFML (Interaction Flow Modeling Language) là một ngôn ngữ mô hình hóa tiêu chuẩn trong lĩnh vực công nghệ phần mềm [12]. IFML cho phép xác định các mô hình giao diện người dùng đồ họa (GUIs) độc lập với nền tảng của các ứng dụng phần mềm. Mục tiêu của IFML là mô tả cấu trúc và hành vi của các ứng dụng theo cảm nhận của người dùng cuối [9].

IFML mang lại một số lợi ích cho quá trình phát triển giao diện người dùng ứng dụng, bao gồm:

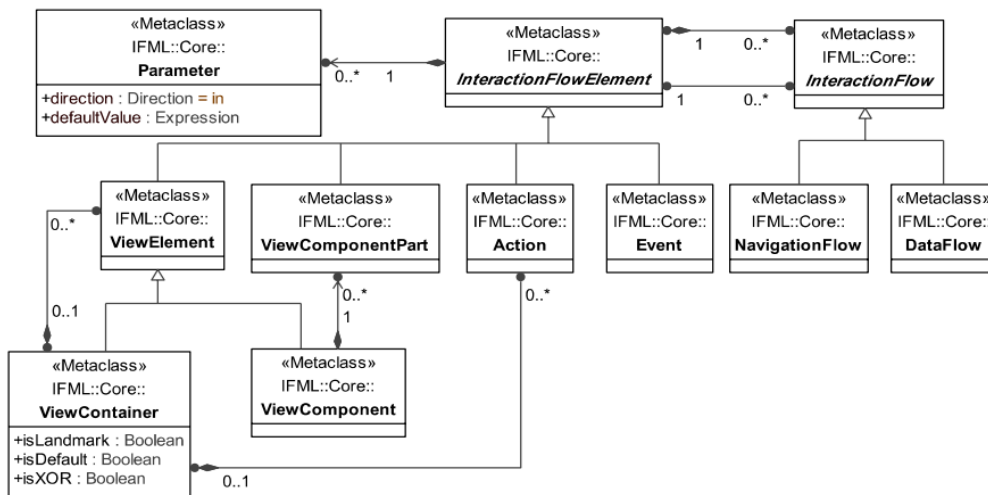
- Hỗ trợ đặc điểm kỹ thuật của giao diện người dùng ứng dụng với các quan điểm khác nhau, bao gồm kết nối với logic nghiệp vụ, mô hình dữ liệu và lớp trình bày đồ họa.
- Tách riêng thông số kỹ thuật giao diện người dùng khỏi các chi tiết triển khai cụ thể.
- Tách biệt mối quan tâm giữa các vai trò trong thiết kế tương tác.
- Cho phép giao tiếp thiết kế giao diện người dùng cho các bên liên quan không chuyên về kỹ thuật.

IFML được phát triển bởi WebRatio và lấy cảm hứng từ ký hiệu WebML và các kinh nghiệm khác trong việc mô hình hóa web. IFML nhằm giải quyết sự đa dạng của thiết bị phần cứng và nền tảng phần mềm, dẫn đến sự phức tạp trong thiết kế và phát triển các ứng dụng phần mềm.

IFML hỗ trợ đặc tả của các quan điểm như cấu trúc giao diện người dùng, đặc tả nội dung giao diện người dùng, sự kiện, đặc tả chuyển đổi sự kiện và ràng buộc tham số. Các đặc tả này giúp mô tả các thành phần và tương tác trong giao diện người dùng. Ví dụ, đặc tả cấu trúc giao diện người dùng mô tả các bộ chứa giao diện người dùng, trong khi đặc tả nội dung giao diện người dùng tập trung vào dữ liệu trong giao diện.

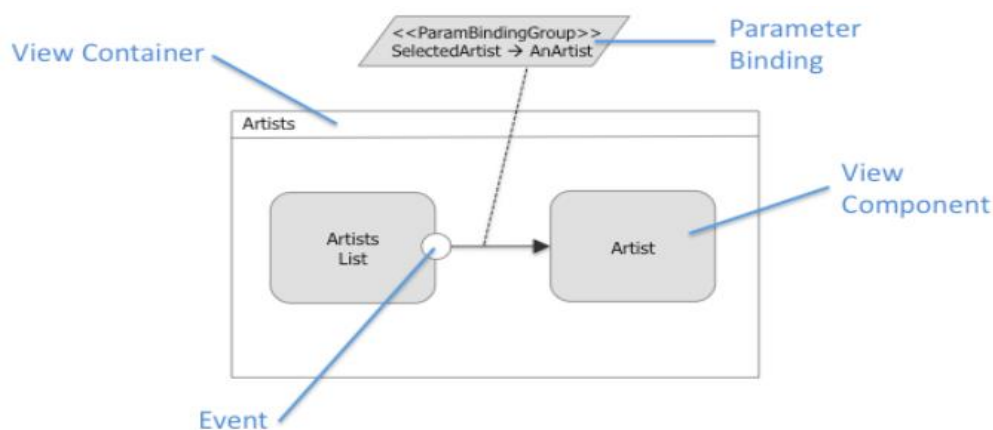
Đặc tả sự kiện giúp định nghĩa các sự kiện có thể ảnh hưởng đến giao diện người dùng, trong khi đặc tả chuyển đổi sự kiện xác định các thay đổi được áp dụng sau khi xảy ra sự kiện. Các đặc điểm kỹ thuật của liên kết tham số định nghĩa các phụ thuộc đầu vào - đầu ra giữa các thành phần và hành động trong giao diện người dùng.

IFML được xác định thông qua siêu mô hình xác định các thành phần để mô hình hóa các yêu cầu của người dùng. Hình 2.9 bên dưới trình bày siêu mô hình của IFML thể hiện các thành phần cấu trúc của mô hình và mối quan hệ giữa các thành phần này.



Hình 2.9: Metamodel của mô hình IFML.

Một số thành phần của IFML được đề cập trong luận văn:



Hình 2.10: Ví dụ về các thành phần của IFML.

View Container: là một thành phần được sử dụng để nhóm các thành phần giao diện người dùng liên quan lại với nhau trong một khối đơn vị. View Container giúp tổ chức và quản lý các thành phần giao diện người dùng và định nghĩa cách chúng tương tác với nhau.

View Container thường được sử dụng để đại diện cho các phần của giao diện người dùng, chẳng hạn như cửa sổ, trang web, trang con, khung nhìn, panel, v.v. Nó giúp xác định và phân vùng các phần khác nhau của giao diện người dùng và tạo ra sự cấu trúc hợp lý.

Một View Container có thể chứa các thành phần giao diện khác như các nút, hộp văn bản, bảng, danh sách, v.v. Ngoài ra, nó cũng có thể chứa các View Container con để tạo cấu trúc phân cấp và tổ chức phức tạp hơn cho giao diện người dùng.

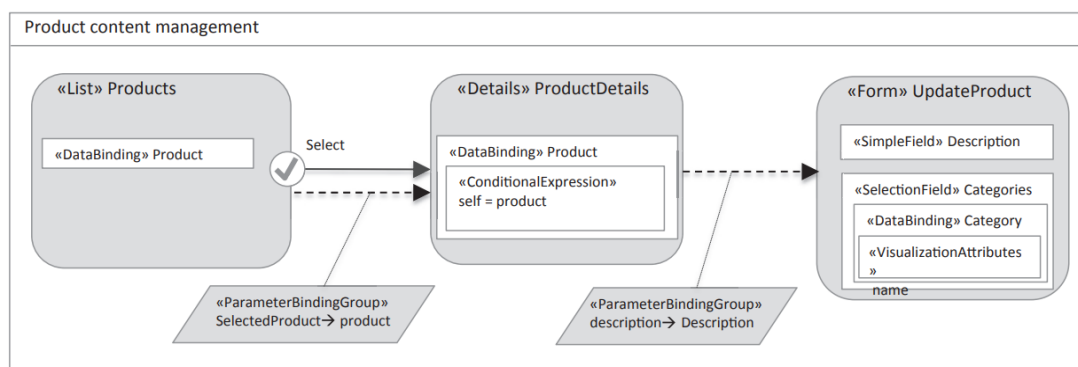
View Container trong IFML có thể có các thuộc tính và hành vi riêng, cho phép xác định các quy tắc, ràng buộc hoặc hành vi đặc biệt cho nhóm thành phần giao diện người dùng trong đó.

View Component: là 1 thành phần chính trong View Container. View Container trong IFML bao gồm một hoặc nhiều View Component. Nó cũng là một phần tử hiển thị ở cấp giao diện người dùng. Nó đã được bắt nguồn từ phần tử View Element. Nó biểu thị nội dung hoặc các thành phần giao diện để nhập dữ liệu (ví dụ: input form, v.v.), có thể có hành vi động, hiển thị nội dung hoặc chấp nhận đầu vào tương ứng với biểu mẫu, dữ liệu hoặc thư viện hình ảnh.

Một số thuộc tính của View Component là:

- Một tham số đầu vào và đầu ra có thể được liên kết với View Components.
- LIST, DETAIL and FORM là các chuyên biệt của một View Component.
- Một View Component có thể được liên kết với một View Element Event để đại diện cho một Sự kiện tương tác người dùng, trong đó có thể được kích hoạt bởi các View Elements (View Containers và View Components).
- Một View Component không bao giờ có thể tồn tại bên ngoài View Container.

Data Binding: là một khái niệm quan trọng giúp liên kết và quản lý dữ liệu giữa giao diện người dùng và dữ liệu thực tế. Nó đảm bảo rằng dữ liệu được hiển thị và cập nhật một cách đúng đắn và nhất quán trên giao diện người dùng. Data Binding trong IFML hoạt động bằng cách định nghĩa các quy tắc ánh xạ giữa các thành phần giao diện và các phần tử trong Data Model. Khi dữ liệu thay đổi hoặc sự kiện xảy ra, Data Binding sẽ đảm bảo cập nhật giao diện và dữ liệu thực tế theo quy tắc đã được xác định. IFML hỗ trợ cả hai hướng One-Way và Two-Way Binding. Trong One-Way Binding, dữ liệu được ánh xạ từ nguồn dữ liệu tới giao diện. Trong Two-Way Binding, sự thay đổi trên giao diện cũng có thể được ánh xạ trở lại nguồn dữ liệu.



Hình 2.11: Ví dụ về 3 thành phần List, Details, Forms trong IFML.

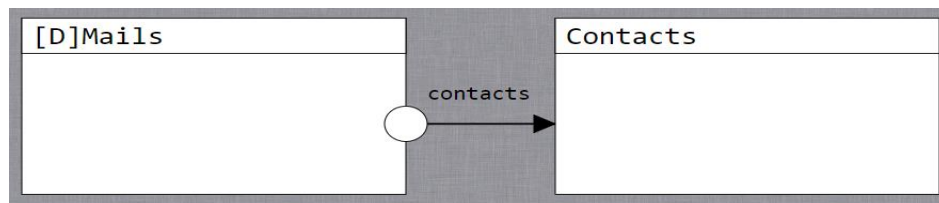
Detail: Một "Detail ViewComponent" (Thành phần hiển thị chi tiết) là một thành phần giao diện được sử dụng để hiển thị các giá trị thuộc tính của một đối tượng cụ thể, thường được truy xuất thông qua một "ContentBinding" (Liên kết nội dung). Khi một "DetailsViewComponent" được kích hoạt thông qua một sự kiện (Event), điều này có nghĩa rằng các thông tin chi tiết về đối tượng đang được hiển thị có thể được sử dụng để kích hoạt một hành động khác hoặc luồng tương tác.

Mục đích của "Detail ViewComponent" là cung cấp cho người dùng thông tin chi tiết về một đối tượng hoặc một phần của giao diện, giúp họ hiểu rõ hơn về các thuộc tính và chi tiết cụ thể liên quan.

List: là một thành phần quan trọng trong giao diện người dùng được định nghĩa trong IFML. Nó thường được sử dụng để hiển thị một danh sách các mục dữ liệu, ví dụ như danh sách sản phẩm, danh sách liên hệ, hoặc danh sách bài viết. Các mục trong danh sách thường có cùng cấu trúc hoặc các thuộc tính tương tự. Điều này giúp tạo ra sự nhất quán và dễ dàng trong việc hiển thị và tương tác với dữ liệu. List thường được liên kết với các tương tác như Selection (Lựa chọn) và Navigation (Điều hướng).

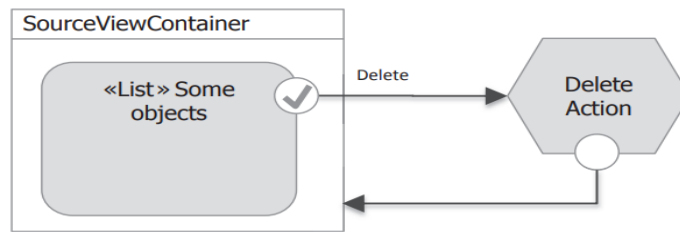
Người dùng có thể chọn một hoặc nhiều mục từ danh sách, và các tương tác này có thể kích hoạt các hành động như xem chi tiết, chỉnh sửa, xóa, hoặc điều hướng tới trang liên quan. List giúp người dùng dễ dàng quản lý và tương tác với dữ liệu một cách hiệu quả, đặc biệt khi cần hiển thị nhiều mục dữ liệu cùng lúc.

Form: là một loại View Component được sử dụng để hiển thị và quản lý dữ liệu thông qua các trường nhập liệu và các tùy chọn khác trên giao diện người dùng. Form cho phép người dùng nhập thông tin và thực hiện tương tác với dữ liệu một cách dễ dàng. Form được sử dụng để tạo ra các biểu mẫu nhập liệu cho người dùng, cho phép họ cung cấp dữ liệu cho ứng dụng. Các trường nhập liệu trong Form có thể liên quan đến thuộc tính của đối tượng trong Data Model. Một Form trong IFML bao gồm các thành phần sau: Các trường nhập liệu (Input Fields) là các thành phần để người dùng nhập dữ liệu, chẳng hạn như ô văn bản, ô chọn thời gian, ô chọn ngày,...; Các nút (Buttons) có thể được thêm vào Form để thực hiện các hành động như lưu, hủy, xác nhận...; Các tùy chọn (Options) bổ sung có thể được thêm vào Form, chẳng hạn như ô chọn đa lựa chọn, ô chọn một lựa chọn. Form trong IFML có thể liên kết với dữ liệu thực tế thông qua Data Binding. Các trường nhập liệu trong Form có thể ánh xạ với các thuộc tính của đối tượng trong Data Model, đảm bảo tính nhất quán giữa giao diện và dữ liệu. Đồng thời nó cung cấp khả năng kiểm tra dữ liệu đầu vào từ người dùng. Người dùng có thể thấy thông báo lỗi nếu họ nhập sai hoặc bỏ trống các trường.



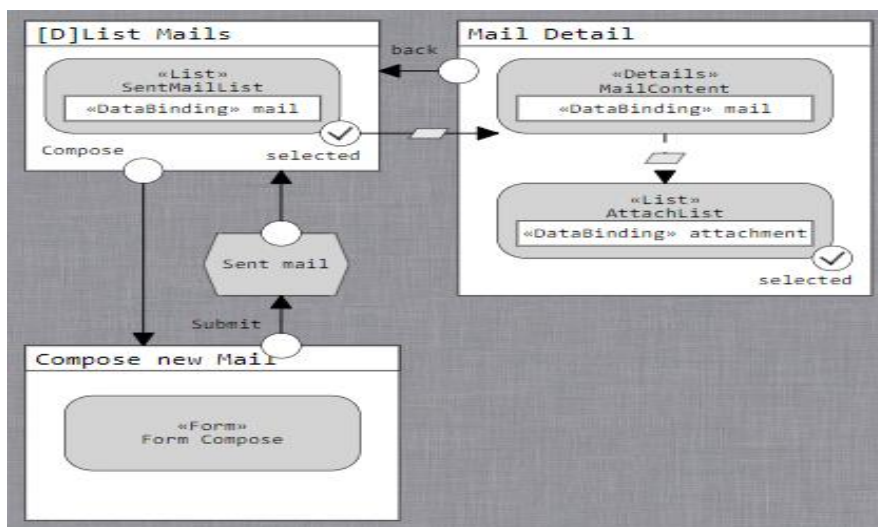
Hình 2.12: Mô tả một ví dụ về event trong IFML.

Event (Sự kiện): là một thành phần quan trọng để mô hình hóa các sự kiện hoặc tương tác giữa người dùng và ứng dụng. Event đại diện cho hành động hoặc tình huống có thể xảy ra trong giao diện người dùng và có thể gây ra các thay đổi hoặc tương tác. IFML hỗ trợ nhiều loại Event khác nhau như "OnLoad" (khi giao diện được tải), "OnSubmit" (khi người dùng gửi một biểu mẫu), "OnSelect" (khi người dùng chọn một mục trong danh sách), "OnSwipe" (khi người dùng vuốt màn hình). Event có thể được kích hoạt bởi người dùng hoặc bởi hệ thống. Một Event có thể có các phần tử mục tiêu (target elements) liên quan, như event "OnSelect" có thể có mục tiêu là một danh sách để xác định mục nào đã được chọn.



Hình 2.13: Mô tả ví dụ thành phần Action trong IFML.

Action (Hành động): là một thành phần quan trọng để mô hình hóa các hành động và tương tác mà ứng dụng thực hiện trong phản hồi với các sự kiện hoặc tương tác từ người dùng. Action đại diện cho các hoạt động cụ thể mà ứng dụng thực hiện để đáp ứng cho các sự kiện hoặc yêu cầu. Ví dụ, action "SubmitForm" để gửi dữ liệu biểu mẫu, "LoadData" để tải dữ liệu từ máy chủ. IFML hỗ trợ nhiều loại action khác nhau như "Create" (tạo mới đối tượng), "Update" (cập nhật đối tượng), "Delete" (xóa đối tượng), "Navigate" (điều hướng đến một trang khác),... Một sự kiện hoặc tương tác có thể liên quan đến một hoặc nhiều action. Các action thường có một đối tượng hoặc tài nguyên cụ thể mà nó thực hiện lên.



Hình 2.14: Mô hình IFML mô tả chương trình Quản lý mail đã gửi.

Hình 2.14 mô tả chương trình quản lý danh sách mail đã gửi theo mô hình IFML. Chương trình Quản lý mail đã gửi được mô tả gồm ba thành phần View Container List Mails, Compose new Mail và Mail Detail. List Mails chứa một View Component dạng List, hiển thị danh sách các mail đã gửi, Compose new Mail chứa một thành phần Form để soạn thư mới và Mail Detail chứa hai thành phần View Element là Detail Mail Content và List Attach List. Có bốn Event trong mô hình trên bao gồm compose, submit, select và back và một Action Sent Mail. Khi người dùng muốn soạn một thư mới thì Event compose sẽ thực hiện chuyển từ View Container List Mails sang trang Compose new Mail, ở đây có một Form bao gồm các thông tin liên quan đến mail để người dùng biên soạn, sau đó Event submit sẽ thực hiện việc xác nhận chuyển đến Action Sent mail. Khi đó thông tin mail mới sẽ được cập nhật ở View Container List Mails. Khi người dùng muốn xem chi tiết nội dung mail bao gồm nội dung mail và danh sách các file đính kèm thì sẽ thực hiện Event select để chuyển từ View Container List Mails sang Mail Detail và ngược lại từ Mail Detail quay lại List Mails sẽ sử dụng Event back.

2.5 Tổng kết chương

Các cơ sở lý thuyết và kiến thức nền tảng cần thiết cho luận văn đã được giới thiệu trong chương vừa rồi. Các khái niệm, vai trò và các cấp độ mô hình hóa của kỹ nghệ hướng mô hình MDE đã được trình bày ở đầu chương cung cấp hướng tiếp cận mới so với các phương pháp phát triển truyền thống.

Tiếp theo luận văn trình bày các phương pháp chuyển đổi mô hình bao gồm phương pháp chuyển đổi mô hình sang mô hình (M2M) và phương pháp chuyển đổi mô hình sang văn bản (M2T). Ngoài ra, luận văn cũng cung cấp một cái nhìn tổng quan về yêu cầu, giải thích cách thực hiện việc đặc tả yêu cầu thông qua ca sử dụng và ngôn ngữ chuyên biệt miền FRSL. Luận văn đề cập tới các tầm quan trọng của kỹ thuật tạo bản mẫu giao diện trong quá trình phát triển phần mềm và một số công cụ và phương pháp truyền thống. Cuối cùng, luận văn giới thiệu mô hình hóa luồng tương tác IFML làm cơ sở cho việc sinh tự động giao diện bản mẫu người dùng.

CHƯƠNG 3: SINH GIAO DIỆN BẢN MẪU DỰA VÀO CHUYỂN ĐỔI MÔ HÌNH

Trong ngữ cảnh phát triển hướng mô hình, chế tác GUI (Graphical User Interface) có thể được tích hợp như một mô hình nguồn hoặc đích của các bộ chuyển đổi mô hình. Các bộ chuyển đổi mô hình là công cụ hỗ trợ quan trọng trong quá trình tích hợp chế tác GUI vào ngữ cảnh phát triển hướng mô hình. Công cụ hỗ trợ chuyển đổi giữa các mô hình với nhau và hỗ trợ việc sinh, biên dịch và tương tác với GUI. Các bộ chuyển đổi mô hình có thể cung cấp các khả năng như tự động sinh mã, kiểm tra và xác thực giao diện người dùng, tạo ra tài liệu hướng dẫn sử dụng, và nhiều tính năng khác.

Tích hợp chế tác GUI vào ngữ cảnh phát triển hướng mô hình giúp tạo ra sự liên kết mạnh mẽ giữa các mô hình và giao diện người dùng, đồng thời tăng cường quy trình phát triển phần mềm bằng việc tự động hóa công việc tạo giao diện người dùng.

3.1 Giới thiệu

Một số dự án nghiên cứu đã trình bày khái niệm về việc tự động sinh ra các ứng dụng tương tác dựa trên mô hình. Tuy nhiên, chỉ một số ít trong số này đã đề xuất cách chuyển đổi từ mô hình sang mô hình của giao diện người dùng. Các giải pháp hiện tại thường chỉ tạo ra một phần của ứng dụng tương tác và thường yêu cầu thông số kỹ thuật đầy đủ và chi tiết về mô hình giao diện người dùng [10].

Hiện tại, có rất nhiều ngôn ngữ lập trình và framework được sử dụng bởi các nhà phát triển để phát triển ứng dụng phần mềm. Tuy nhiên, việc sử dụng những công cụ và công nghệ phức tạp này đòi hỏi các nhà phát triển phải có kiến thức chi tiết về chúng, và điều này đôi khi mất thời gian học tập lâu dài. Ngoài ra, cũng có những thách thức như tạo giao diện người dùng hấp dẫn và đa nền tảng, cũng như xử lý các giao diện đa nền tảng. Các yếu tố khác như khả năng mở rộng, hiệu suất, bảo mật và các vấn đề khác cũng phải được xem xét.

Trong bối cảnh này, kỹ thuật yêu cầu (RE) đóng một vai trò rất quan trọng trong việc phát triển và quản lý phần mềm, giúp giảm thiểu lỗi phần mềm từ giai đoạn đầu của quá trình phát triển. RE đã đóng vai trò quan trọng trong các giai đoạn khác nhau của kỹ thuật phần mềm và cung cấp nhiều phương pháp tiếp cận khác nhau. Thực hiện RE là cần thiết để có kiến thức rộng về miền vấn đề trước khi bắt đầu thiết kế, phát triển và triển khai một giải pháp cụ thể, cũng như để tránh chi phí làm lại.

ASL [11]: Mục tiêu của ASL là hỗ trợ việc phát triển các ứng dụng phần mềm một cách có hệ thống và chặt chẽ, đặc biệt là trong lĩnh vực ứng dụng kinh doanh. ASL hướng tới việc hỗ trợ các đặc điểm kỹ thuật của các ứng dụng phần mềm bằng cách cung cấp một cú pháp và cấu trúc rõ ràng để mô tả yêu cầu, luồng tương tác và các khía cạnh khác của ứng dụng. Bằng cách sử dụng ASL, nhà phát triển có thể xác định rõ ràng các yêu cầu, quy trình và quy tắc kinh doanh, từ đó giúp cải thiện tính khả thi, tính nhất quán và tính chính xác của ứng dụng.

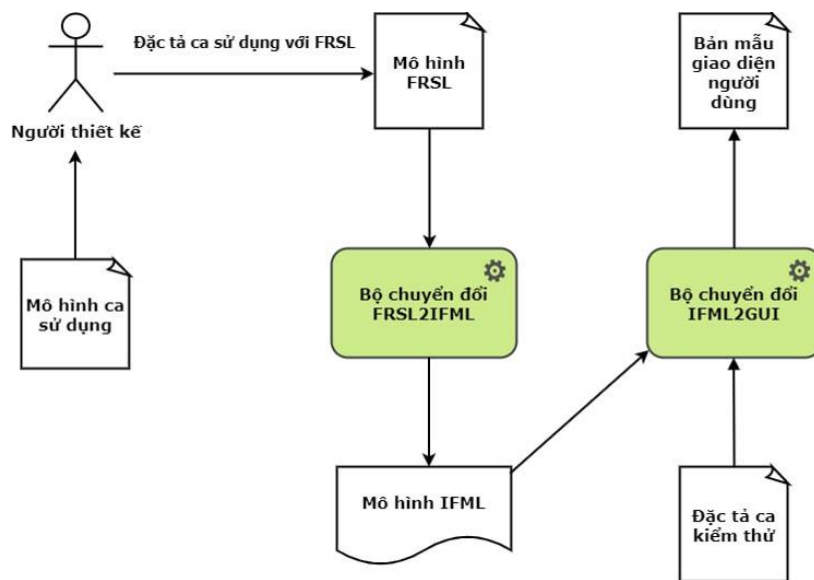
RSL [15]: là một ngôn ngữ đặc tả được tạo ra để giảm thiểu các vấn đề phát sinh khi viết yêu cầu trong lĩnh vực phát triển phần mềm. RSL được thiết kế nhằm hỗ trợ việc viết yêu cầu và kiểm tra các thông số kỹ thuật một cách có hệ thống, chặt chẽ và nhất quán. RSL sử dụng một ngôn ngữ tự nhiên được kiểm soát, có nghĩa là nó cung cấp các từ và cú pháp cụ thể để mô tả các yêu cầu và thông số kỹ thuật. Ngôn ngữ này bao gồm một tập hợp phong phú các cấu trúc được sắp xếp một cách hợp lý, phù hợp với các chế độ xem từ các mối quan tâm khác nhau tồn tại ở các mức độ trừu tượng khác nhau. Cấu trúc trong RSL có thể được sử dụng để xác định nghiệp vụ, ứng dụng, phần mềm, hoặc thậm chí cả hệ thống phần cứng, tùy thuộc vào mức độ trừu tượng. Ngoài ra, cấu trúc trong RSL được phân loại theo các khía cạnh cụ thể của việc xác định yêu cầu.

3.2 Tổng quan phương pháp

Tự động sinh ứng dụng tương tác hướng mô hình cũng đã được trình bày trong một vài dự án nghiên cứu nhưng chỉ một số ít đề xuất việc chuyển đổi mô hình sang mô hình của giao diện người dùng. Các giải pháp hiện tại chỉ tạo ra một phần của ứng dụng tương tác và hầu hết chúng đều yêu cầu đầu vào là thông số kỹ thuật đầy đủ của một mô hình giao diện người dùng.

Trong quá trình phân tích, tìm hiểu các yêu cầu chức năng và việc đặc tả chi tiết ca sử dụng thông thường sẽ gặp những khó khăn và cản trở nhất định. Yêu cầu chức năng cần xác định rõ thông tin đầu vào (input), hành vi và đầu ra (output) của hệ thống. Các hành vi của hệ thống lại được biểu diễn qua các ca sử dụng. Người dùng thường sử dụng mẫu mô tả riêng biệt cho các trường hợp sử dụng, dẫn đến sự đa dạng về cú pháp và cấu trúc trong việc mô tả. Kết quả là, sự không nhất quán và sự khác biệt xuất hiện trong việc mô tả.

Người lập mô hình thường ưa thích mô tả các trường hợp sử dụng bằng ngôn ngữ tự nhiên dưới dạng văn bản, tuy nhiên, việc này có thể hình thành các yếu tố mập mờ và không rõ ràng về mặt ngữ nghĩa. Điều này thường làm cho quá trình phân tích và thiết kế ở các giai đoạn sau trở nên khó khăn hơn. Để giải quyết vấn đề khó khăn này, ta có thể sử dụng một cách tiếp cận bằng cách đầu vào là một chế tác của FRSL, hỗ trợ người dùng trong việc mô tả chính xác các yêu cầu chức năng và mô hình hóa chúng cho mục đích sử dụng trong kiến trúc hướng mô hình. Hình 3.1 mô tả tổng quan cách tiếp cận để xây dựng ngôn ngữ đặc tả giao diện người dùng từ chế tác FRSL, thông tin mô tả các ca sử dụng sẽ là nguồn dữ liệu đầu vào. Người mô hình hóa sẽ sử dụng FRSL để đặc tả các ca sử dụng. Đặc tả của các ca sử dụng dưới định dạng FRSL này sau đó sẽ được chuyển đổi thành mô hình FRSL, tuân theo cú pháp trừu tượng. Mô hình FRSL này sẽ là mô hình nguồn cho các quy trình chuyển đổi, và kết quả của quá trình này sẽ là mô hình luồng tương tác IFML để tự động sinh ra giao diện người dùng.



Hình 3.1: Mô tả tổng quan cách tiếp cận để xây dựng ngôn ngữ đặc tả giao diện người dùng IFML từ chế tác FRSL.

Bằng cách sử dụng mô hình FRSL với đầu vào là các đặc tả ca sử dụng với FRSL, quá trình chuyển đổi tự động sẽ diễn ra để tạo ra mô hình IFML. Khi đó, một biểu đồ trực quan cho phép người dùng chỉnh sửa các thành phần của mô hình thông qua một đồ thị. Sau khi người dùng hoàn tất việc điều chỉnh mô hình IFML, ta có thể áp dụng các bộ luật chuyển đổi như một bộ công cụ cùng với đặc tả ca kiểm thử được sinh tự động từ mô hình FRSL giúp sinh tự động giao diện người dùng tương ứng. Đầu ra được chọn là mã nguồn HTML, kèm theo các tệp *css* và *javascript* cố định.

3.3 Chuyển đổi từ mô hình yêu cầu sang mô hình bản mẫu

Quá trình chuyển đổi từ đặc tả yêu cầu FRSL sang IFML là quá trình chuyển đổi mô hình sang mô hình. Thông thường, tất cả hoạt động trên mô hình được thực hiện qua quá trình chuyển đổi mô hình hoặc dưới dạng chuyển đổi mô hình sang mô hình (M2M) hoặc chuyển đổi mô hình sang văn bản (M2T) [13]. Các phép biến đổi M2M là những chương trình nhận một hoặc nhiều mô hình làm đầu vào và tạo ra một hoặc nhiều mô hình làm đầu ra. Thông thường, việc chuyển đổi 1-1 từ một mô hình đầu vào sang một mô hình đầu ra là đủ.

Luận văn sẽ tập trung vào các thành phần quan trọng đặc trưng cho cú pháp trừu tượng của ngôn ngữ đặc tả giao diện người dùng IFML để tạo ra các luật đổi mô hình. Dưới đây, sẽ chỉ trình bày các luật chuyển đổi chính liên quan đến các thành phần quan trọng của mô hình luồng tương tác IFML:

Luật chuyển FI1: Ánh xạ thông tin từ mô hình FRSL sang mô hình IFML

Hình 3.2 mô tả tóm lược nội dung luật FI1. Luật này cho phép tạo mô hình IFML (IFMLModel) tương ứng mô hình FRSL (FrslModel). Hai mô hình này có tên trùng nhau. Phần tử mô hình luồng tương tác (interactionFlowModel) tương ứng với ca sử dụng của mô hình FRSL (tức `frslModel.usecase->at(1)`). Mô hình miền (domainModel) của mô hình IFML được xác định tương ứng với gói mô hình miền của mô hình FRSL (tức `frslModel.ownedPackages.at(1)`).

```
rule FrslModel2IfmlModel {
  from
    frslModel: FRSL!FrslModel,
    t : UML!Model in UmlFrsl
  to
    ifmlModel: IFML!IFMLModel (
      name <- frslModel.name,
      interactionFlowModel <- frslModel.usecase->at(1),
      domainModel <- thisModule.FrslModel2DomainModel( frslModel )
    )
  do {
    thisModule.umlFrslModel <- t;
  }
}
```

Hình 3.2: Mô tả tóm lược luật FI1 - FrslModel2IfmlModel.

Luật chuyển FI2: Ánh xạ FRSL!Usecase sang IFML!InteractionFlowModel

Như minh họa trong Hình 3.3, luật này cho phép tạo phần tử `InteractionFlowModel` tương ứng với phần tử `Usecase`. Các phần tử mô hình luồng tương tác `interactionFlowModelElements` bao gồm các phần tử `ViewContainer` tương ứng với các `Step` của `Usecase`. Tham số `ucPara` cũng được xác định để lưu trạng thái thực hiện của ca sử dụng.

```
rule Usecase2InteractionFlowModel {
  from
    uc: FRSL!Usecase --(uc.name.toString() = 'HandleCreditPayment')
  to
    fm: IFML!InteractionFlowModel(
      name <- uc.name,
      interactionFlowModelElements <- Sequence{
        ucPara,
        firstViewContainer
      }.union(
        allSteps->excluding(uc.firstStep)
        -> collect( st | thisModule.Step2ViewContainer(st) )
      )
    ),
    ucPara: IFML!IFMLParameter(
      direction <- #inout,
      name <- 'uc' + uc.name
      --, type <- uc.getUcClass()
    ),
}
```

Hình 3.3: Mô tả tóm lược luật FI2 - Usecase2InteractionFlowModel.

```
unique lazy rule Step2ViewContainer {
  from
    step: FRSL!Step
  to
    viewContainer: IFML!ViewContainer (
      name <- 'page:' + step.getDescName(),
      annotations <- Sequence{
        stepInforAnnotation
      },
      viewElements <- Sequence{
        stepViewComponent
      }
    ),
    stepInforAnnotation: IFML!Annotation(
      text <- if( step.ocIsKindOf(FRSL!UCStep) ) then
        '<<ucStep>> including ' + step.includedUC.name
      else if( step.ocIsKindOf(FRSL!RejoinStep) ) then
        '<<rejoinStep>> ' + step.description.toString().regexReplaceAll('\s+', ' ')
      else if( step.isActorStep ) then
        '<<actorAction>> ' + step.description.toString().regexReplaceAll('\s+', ' ')
      else
        '<<sysAction>> ' + step.description.toString().regexReplaceAll('\s+', ' ')
      endif endif endif
    ),
    stepViewComponent: IFML!ViewComponent(
      name <- step.getDescName(),
      constraints <- Sequence{
        stepPrecond
      }
    ),
    stepPrecond: IFML!Constraint (
      language <- 'OCL',
      body <- 'getStepPrecond'
    )
}
```

Hình 3.4: Mô tả tóm lược luật FI2.1 - Step2ViewContainer.

Luật chuyển FI2.1 - Step2ViewContainer: Luật này được mô tả như Hình 3.4 cho phép ánh xạ đầy đủ các thành phần của `FRSL!Step` chuyển thành `ViewContainer` trong IFML với IFML!Annotations cung cấp các thông tin của từng Step bao gồm mô tả hành động của các tác nhân `actorAction`, hành động của hệ thống

systemAction; IFML!ViewComponent mô tả các phần tử trong Step và định nghĩa các IFML! Constraint trong từng Step của ca sử dụng.

```
unique lazy rule InputActStep2Form {
  from
    step: FRSL!ActStep (
      step.isInputActStep()
    )
  using {
    viewContainer: IFML!ViewContainer = thisModule.Step2ViewContainer(step);
  }
  to
    stepForm: IFML!Form(
      name <- step.getDescName(),
      viewComponentParts <- step.action->select(act | act.type.toString() = 'inputAct')
        ->iterate(act; ret: Sequence(IFML!SimpleField) = Sequence{} |
          ret->union( act.objVars->collect( var | thisModule.ObjVar2SimpleField(var) ) )
        )
    )
  do {
    viewContainer.viewElements <- viewContainer.viewElements->append(stepForm);
  }
}

-----
lazy rule ObjVar2SimpleField {
  from
    objVar: FRSL!ObjVar
  using{
    prop: UML!Property = objVar.type;
  }
  to
    field: IFML!SimpleField(
      --direction <- #in,
      name <- objVar.name,
      type <- thisModule.umlType(prop.type)
    )
}
}
```

Hình 3.5: Mô tả luật FI2.2 - InputActStep2Form.

Luật chuyển FI2.2 - InputActStep2Form: Với các FRSL!ActStep mang thông tin Input sẽ được áp dụng bộ chuyển đổi trở thành thành phần IFML!Form tương ứng với các thông tin được ánh xạ. Khi đó các trường thông tin trong Form sẽ được ánh xạ từ FRSL!ObjVar sang IFML!SimpleField với tên tương ứng theo FRSL!ObjVar và loại là 1 trong 4 loại dữ liệu bao gồm Integer, String, Real và Boolean như được mô tả trong Hình 3.5.

Luật chuyển FI2.3: Ánh xạ các thông tin chuyển trang phụ thuộc vào FRSL!Step và FRSL!RejoinStep để xác định các tham số sourceInteractionFlowElement và targetInteractionFlowElement qua đó kết nối các thành phần IFML!ViewContainer có tác dụng giúp điều hướng trang tương ứng. Hình 3.6 minh họa cụ thể về 2 luật NextStep2Flow và RejoinStep2Flow kết nối các thành phần IFML!ViewContainer.

```

unique lazy rule NextStep2Flow {
  from
    step: FRSL!Step
  using{
    viewContainer: IFML!ViewContainer = thisModule.Step2ViewContainer(step);
    nextViewContainer: IFML!ViewContainer = thisModule.Step2ViewContainer(step.nextStep);
  }
  to
    stepExit: IFML!ViewElementEvent(
      name <- 'exit_' + step.getDescName(),
      outInteractionFlows <- Sequence {
        viewFlow
      }
    ),
    viewFlow: IFML!NavigationFlow (
      sourceInteractionFlowElement <- stepExit,
      targetInteractionFlowElement <- nextViewContainer
    )
  do {
    viewContainer.viewElements->at(1).viewElementEvents <- Sequence{ stepExit };
  }
}

-----
unique lazy rule RejoinStep2Flow {
  from
    step: FRSL!RejoinStep
  using{
    viewContainer: IFML!ViewContainer = thisModule.Step2ViewContainer(step);
    viewComponent: IFML!ViewComponent = viewContainer.viewElements->at(1);
  }
  to
    rejoinStepEvent: IFML!ViewElementEvent(
      name <- 'rejoin_' + step.rejoinTo.getDescName(),
      outInteractionFlows <- Sequence {
        rejoinFlow
      }
    ),
    rejoinFlow: IFML!NavigationFlow (
      sourceInteractionFlowElement <- rejoinStepEvent,
      targetInteractionFlowElement <- IFML!ViewContainer.allInstances()
        ->select( v | v.name.toString().startsWith('page:' + step.rejoinTo.getDescName() ) )->last()
    )
  do{
    viewComponent.viewElementEvents <- viewComponent.viewElementEvents->append(rejoinStepEvent);
  }
}

```

Hình 3.6: Mô tả 2 luật NextStep2Flow và RejoinStep2Flow.

```

unique lazy rule FrslModel2DomainModel {
  from
    frslModel: FRSL!Model
  to
    domainModel: IFML!DomainModel(
      domainElements <- frslModel.ownedPackages->at(1).ownedClasses
        ->union(frslModel.ownedPackages->at(2).ownedClasses)
    )
}

rule Class2DomainConcept {
  from
    cls: FRSL!Class
  to
    domConcept: IFML!DomainConcept(
      name <- cls.name
    )
}

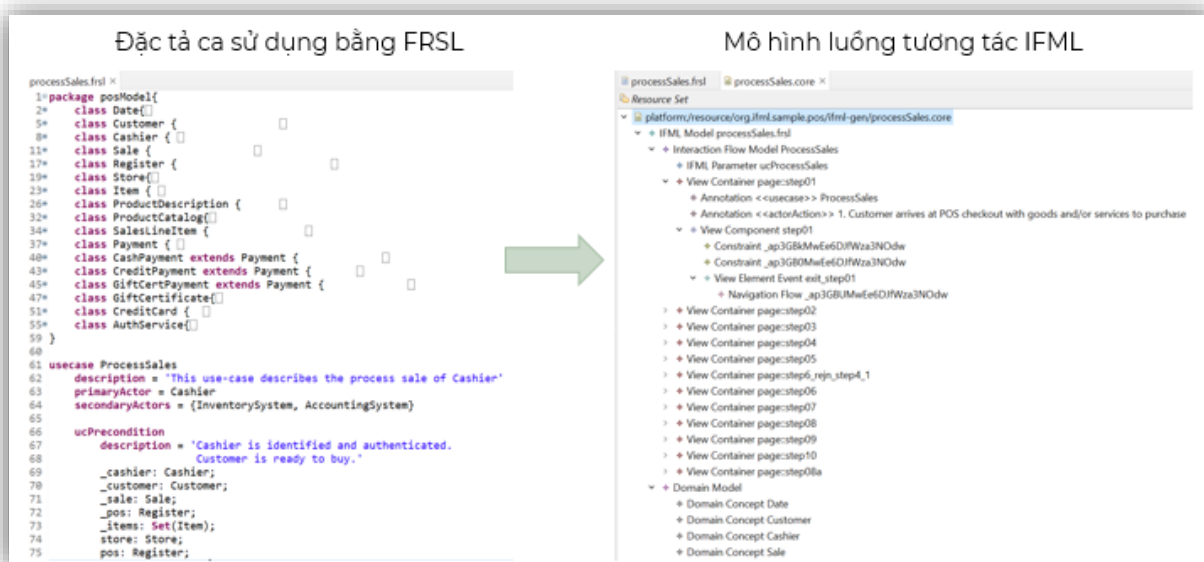
```

Hình 3.7: Mô tả luật chuyển đổi tạo ra mô hình miền IFML.

Luật chuyển FI3: Ánh xạ thông tin của FRSL!Model thành mô hình miền IFML!DomainModel

Trong hình 3.7 mô tả luật chuyển đổi tạo ra mô hình miền IFML với các thông tin ánh xạ từ FRSL!Model và FRSL!Class. Ở đây IFML!DomainModel cung cấp các thông tin từ FRSLModel.ownedPackages để xác định các lớp trong ca sử dụng.

Sau quá trình chuyển đổi, Hình 3.8 mô tả đặc tả ca sử dụng bằng FRSL được chuyển đổi thành mô hình luồng tương tác IFML dưới dạng cú pháp trừu tượng với 2 thành phần chính là InteractionFlowModel chứa thông tin của Usecase và mô hình miền DomainModel chứa thông tin các lớp.



Hình 3.8: Mô tả đặc tả ca sử dụng bằng FRSL được chuyển đổi thành mô hình luồng tương tác IFML.

3.4 Sinh giao diện bản mẫu từ mô hình IFML

Chuyển đổi từ mô hình IFML sang giao diện người dùng sử dụng phương pháp chính là chuyển đổi mô hình model-to-text với kỹ thuật sinh tự động mã nguồn (programming language) từ mô hình IFML. Acceleo được chọn là ngôn ngữ cho chuyển đổi mô hình sang văn bản M2T vì tính phù hợp thực tế và khả năng cung cấp đầy đủ sự hỗ trợ trong quá trình phát triển.

Trong phạm vi của luận văn, đầu ra được chọn là mã nguồn HTML, kèm theo các tệp css, javascript cố định và các đặc tả ca kiểm thử được tạo sẵn. Mô hình luồng tương tác sau khi trải qua quá trình chuyển đổi M2M sẽ được sử dụng làm đầu vào. Dưới đây là một số luật chuyển chính được sử dụng để tạo ra giao diện người dùng (chế tác cuối cùng là mã nguồn HTML). Luận văn đã sử dụng Acceleo để xây dựng một số bộ luật chuyển chính, ánh xạ từ các thành phần của mô hình luồng tương tác sang các thành phần giao diện. Các luật chính bao gồm:

Luật chuyển IG1: Ánh xạ thông tin IFML Model thành các thông tin của tệp.

Tại Hình 3.9, luật chuyển này để tạo ra tệp đầu ra có tên là 'MyProject.html', chứa các templates khác nhau được cung cấp sẵn các file .javascript và .css định nghĩa sẵn.

```
[template public generateElement(ifmlModel : IFMLModel)]
[comment @main/]

[Let arrInteractionElement: Sequence(InteractionFlowModelElement) = ifmlModel.interactionFlowModel->asSequence()]

[file ('MyProject.html', false, 'UTF-8')]
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <link rel="stylesheet" href="style.css"></link>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <title>GUI_GENERATE</title>
  </head>
  <body>

    </body>
    <script src="data.js"></script>
    <script src="myscript.js"></script>
  </html>

[/file]
[/Let]
[/template]
```

Hình 3.9: Mô tả luật chuyển tạo ra tệp đầu ra .html.

Luật chuyển IG2: Ánh xạ thông tin ViewContainer thành thông tin của từng trang hiện tại.

- Hình 3.10 mô tả hai luật chuyển áp dụng cho các ViewContainer, một luật cho các ViewContainer có thuộc tính isDefault = true (trang hiển thị mặc định) và một luật cho các ViewContainer không có thuộc tính này.
- Mỗi luật tương ứng với một template riêng, chúng nhận cùng một đầu vào và chứa các template ánh xạ thông tin thành phần tương tự nhau.
- Các thành phần trong ViewContainer được biến đổi theo ca sử dụng tương ứng

```
[template public generatePageDefault(con : ViewContainer,
arrInteractionElement: Sequence(InteractionFlowModelElement) )]

<div class="v-card defaultPageContainer currentPage" id="Container_[arrInteractionElement->indexOf(con)]">
| href="[con.name.trim()]">
  <div class="v-card-text">
    <nav aria-label="breadcrumb">
      <ol class="breadcrumb">
        <li class="breadcrumb-item active" aria-current="page">
          [con.name/]
        </li>
      </ol>
    </nav>
  </div>
</div>
[/template]

[template public generatePageNext(con : ViewContainer,
arrInteractionElement: Sequence(InteractionFlowModelElement))]
<div class="v-card hiddenPage" id="Container_[arrInteractionElement->indexOf(con)]">
  href="[con.name.trim()]">
    <div class="v-card-text">
      <nav aria-label="breadcrumb">
        <ol class="breadcrumb">
          <li class="breadcrumb-item active" aria-current="page">
            [con.name/]
          </li>
        </ol>
      </nav>
    </div>
  </div>
</div>
[/template]
```

Hình 3.10: Mô tả luật chuyển thành phần của từng trang.

Luật chuyển IG3: Ánh xạ thông tin ViewComponent thành thông tin của các thành phần hiển thị cơ bản

```
[template public generateForm(form : Form, arrInteractionElement: Sequence(InteractionFlowModelElement),
arrViewElement: Sequence(ViewElement))]
[/template]

[template public generateList(list : List, arrInteractionElement: Sequence(InteractionFlowModelElement),
arrViewElement: Sequence(ViewElement))]
[/template]

[template public generateDetails(details : Details, arrInteractionElement: Sequence(InteractionFlowModelElement),
arrViewElement: Sequence(ViewElement))]
[/template]
```

Hình 3.11: Mô tả tóm lược luật chuyển các thành phần trong ViewComponent.

- ViewComponent, được xây dựng trong mô hình đặc tả giao diện người dùng IFML, bao gồm: Formt, List, Details như mô tả tóm lược trong hình 3.11. Các loại này cũng được ánh xạ thành các thành phần khác nhau trên giao diện.
- Quá trình ánh xạ thực hiện việc biến đổi tên của ViewComponent thành tiêu đề của thẻ tương ứng, và sử dụng thông tin liên quan từ các thuộc tính fields, columns, msg, options như thông tin đầu vào input hoặc để hiển thị dưới dạng bảng.

Luật chuyển IG4: Ánh xạ thông tin NavigationFlow thành luồng tương tác chính giữa các trang.

NavigationFlow cũng được tạo trong mô hình đặc tả giao diện người dùng IFML. Các thành phần này thường thể hiện dưới dạng các nút để biểu thị tương tác giữa các trang hỗ trợ trong quá trình điều hướng, chuyển trang (Hình 3.12).

```
[template public generateNavigationFlow(btn : NavigationFlow,
  arrInteractionElement: Sequence(InteractionFlowModelElement), arrViewElement: Sequence(ViewElement),
  arrModule: Sequence(ModuleDefinition))]
  <button type="button" class="btn btn-success" onclick="navigationPage(
    [if (arrInteractionElement->includes(btn.targetInteractionFlowElement))]
    Container_[arrInteractionElement->indexOf(btn.targetInteractionFlowElement)]
    [elseif (arrViewElement->includes(btn.targetInteractionFlowElement))]
    Container_[arrInteractionElement->indexOf(btn.targetInteractionFlowElement.eContainer())]
    [else]Module_[arrModule->includes(btn.targetInteractionFlowElement.eContainer())]
    [/if])">[btn/]
  </button>
[/template]
```

Hình 3.12: Mô tả luật chuyển các thành phần chuyển trang NavigationFlow.

3.5 Tổng kết chương

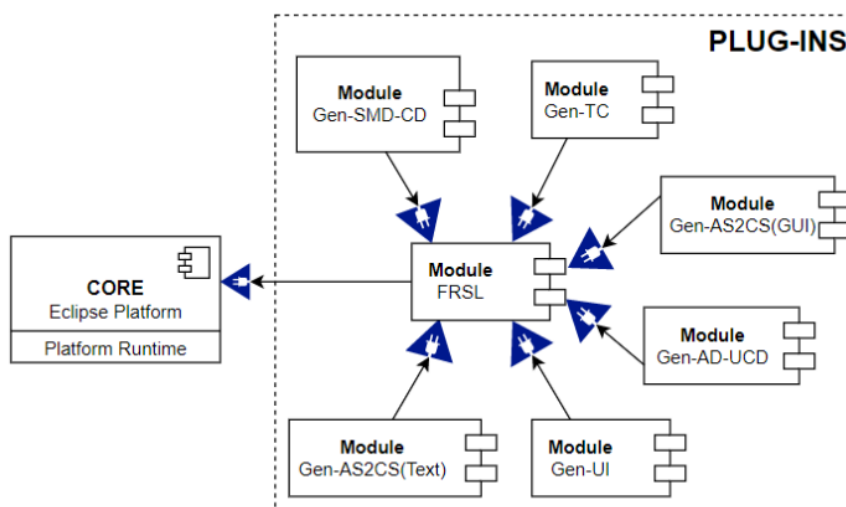
Chương 3 đã được thực hiện xây dựng các luật chuyển đổi dựa trên đầu vào là chế tác FRSL để tạo ra giao diện người dùng, thông qua hai giai đoạn chuyển đổi mô hình M2M và M2T. Qua chương này, ta đã có thể có một cái nhìn sơ bộ về cách giải quyết vấn đề tự động tạo giao diện người dùng và quá trình thực hiện nó. Trong Chương tiếp theo, ta sẽ đi sâu vào bài toán thực tế thông qua việc triển khai và thực nghiệm để đánh giá kết quả đạt được.

CHƯƠNG 4: CÀI ĐẶT VÀ THỰC NGHIỆM

Từ những kiến thức nền tảng trong Chương 2 và tổng quan giới thiệu phương pháp trong Chương 3, luận văn trình bày chức năng của các công cụ hỗ trợ. Tiếp theo, Mục 4.3 luận văn sẽ trình bày tình huống nghiên cứu cụ thể với ca sử dụng Process Sales để minh họa cho phương pháp và cuối cùng đưa ra các kết quả đánh giá phương pháp và thảo luận tại Mục 4.4.

4.1 Công cụ hỗ trợ


Công cụ hỗ trợ cho phương pháp sinh tự động giao diện người dùng được phát triển ở dạng plugins của công cụ VNU-FRSL. Hình 4.1 mô tả kiến trúc tổng thể của công cụ VNU-FRSL. Công cụ này được phát triển dựa trên nền tảng Eclipse với kiến trúc plugin cho phép mở rộng. Thiết kế này cho phép thêm và cấm chức năng mô-đun vào mô-đun lõi, mang lại khả năng mở rộng, tính linh hoạt và tách biệt các tính năng ứng dụng. Mô-đun lõi cho phép chỉ định cú pháp trừu tượng, cú pháp này sẽ được lấy làm đầu vào cho các mô-đun chức năng khác để tạo ra các tạo phẩm khác.



Hình 4.1: Kiến trúc tổng thể của VNU-FRSL [8].

Công cụ hỗ trợ phương pháp được đề xuất trong luận văn bao gồm các thành phần chính sau: (1) Thành phần FRSL để đặc tả yêu cầu chức năng; (2) Thành phần IFML để biểu diễn mô hình tương tác. (3) Thành phần FRSL2IFML để chuyển đổi đặc tả yêu cầu FRSL sang mô hình luồng tương tác IFML; (4) Thành phần IFML2GUI để sinh giao diện bản mẫu từ mô hình luồng tương tác IFML.

Thành phần FRSL được tích hợp sẵn trong bộ công cụ VNU-FRSL. Thành phần IFML được tích hợp từ nguồn có sẵn. Hai thành phần còn lại được chúng tôi phát triển như mô tả trong luận văn. Đầu tiên, thành phần **FRSL2IFML** được xây dựng dựa vào công nghệ chuyển đổi mô hình ATL, như đặc tả trong Hình 4.2. Hình 4.3 thể hiện các package cần thiết để xây dựng công cụ, trong đó tệp frsl2ifml.atl để cài đặt luật chuyển mô hình.

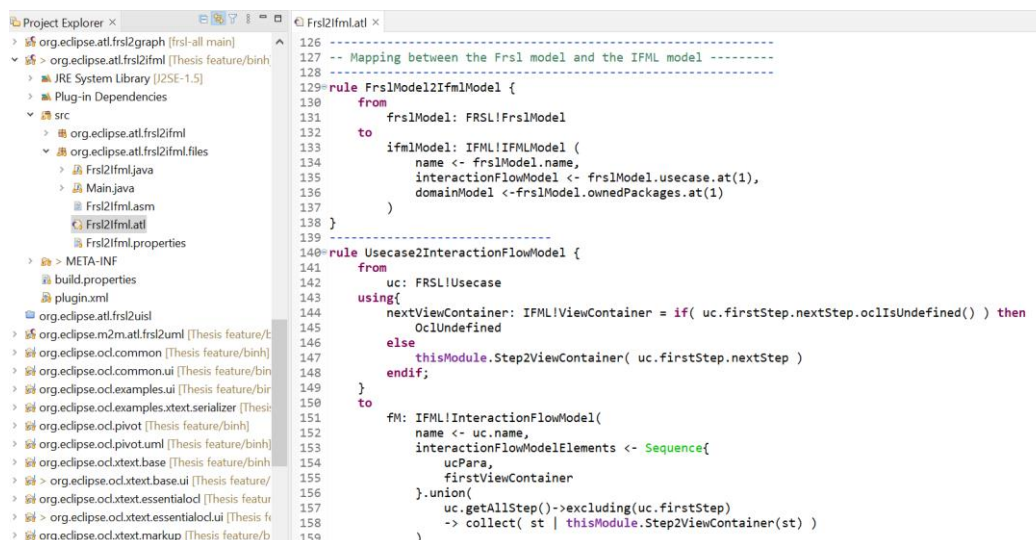


```

1<?xml version="1.0" encoding="UTF-8"?>
2<?eclipse version="3.4"?>
3<plugin>
4  <extension
5    point="org.eclipse.ui.popupMenus">
6    <objectContribution
7      id="org.eclipse.sme.frslas2ifml.ui.popupMenus.contribution.IFile"
8      nameFilter="*.frslas"
9      objectClass="org.eclipse.core.resources.IFile">
10      <menu
11        id="org.eclipse.sme.frslas2ifml.module.menu"
12        label="Frslas2Ifml" path="frslas">
13        <groupMarker name="frslas"/>
14      </menu>
15      <action
16        class="org.eclipse.atl.frsl2ifml.files.Main"
17        id="atlplugin.action"
18        label="Generate IFML model"
19        menubarPath="org.eclipse.sme.frslas2ifml.module.menu/group">
20      </action>
21    </objectContribution>
22  </extension>
23</plugin>
24
25

```

Hình 4.2: Công cụ FRSL2IFML được tích hợp vào Eclipse IDE.



```

126 -----
127 -- Mapping between the Frsl model and the IFML model -----
128 -----
129 rule FrslModel2IfmlModel {
130   from
131     frslModel: FRSL!FrslModel
132   to
133     ifmlModel: IFML!IFMLModel (
134       name <- frslModel.name,
135       interactionFlowModel <- frslModel.usecase.at(1),
136       domainModel <- frslModel.ownedPackages.at(1)
137     )
138 }
139 -----
140 rule Usecase2InteractionFlowModel {
141   from
142     uc: FRSL!Usecase
143   using{
144     nextViewContainer: IFML!ViewContainer = if( uc.firstStep.nextStep.ocIsUndefined() ) then
145       OclUndefined
146     else
147       thisModule.Step2ViewContainer( uc.firstStep.nextStep )
148     endif;
149   }
150   to
151     fm: IFML!InteractionFlowModel(
152       name <- uc.name,
153       interactionFlowModelElements <- Sequence{
154         ucPara,
155         firstViewContainer
156       }.union(
157         uc.getAllStep()->excluding(uc.firstStep)
158         -> collect( st | thisModule.Step2ViewContainer(st) )
159       )
160     )
161 }

```

Hình 4.3: Các package của công cụ FRSL2IFML tool trên Eclipse IDE.

Tiếp đó, thành phần **IFML2GUI** được phát triển bằng công nghệ chuyển đổi mô hình sang văn bản Acceleo. Hình 4.4 mô tả các package cần thiết phát triển công cụ IFML2GUI



Hình 4.4: Các package của công cụ IFML2GUI tool trên Eclipse.

4.2 Minh họa phương pháp

Luận văn tập trung vào hệ thống bán hàng siêu thị POS như được giới thiệu trong [14] để minh họa cho phương pháp đề xuất. Dưới đây là đặc tả ca sử dụng ProcessSale của hệ thống này:

Mô hình: Máy bán hàng tự động POS

Ca sử dụng: Thanh toán hóa đơn (Process Sale)

Tác nhân chính: Nhân viên thu ngân

Các bộ phận liên quan:

- Nhân viên thu ngân: Muốn nhập dữ liệu thanh toán chính xác, nhanh chóng, không mắc lỗi thanh toán.
- Người mua hàng: Muốn mua hàng và sử dụng dịch vụ nhanh chóng. Muốn có bằng chứng mua hàng để hỗ trợ trả lại.
- Công ty: Mong muốn ghi chép chính xác các giao dịch và thỏa mãn lợi ích của khách hàng.

Tiền điều kiện: Nhân viên thu ngân được xác định và xác thực.

Hậu điều kiện: Hóa đơn được lưu. Thuế được tính toán chính xác.

Kịch bản ca sử dụng:

1. Khách hàng đến quầy thanh toán POS với hàng hóa và/hoặc dịch vụ cần mua.

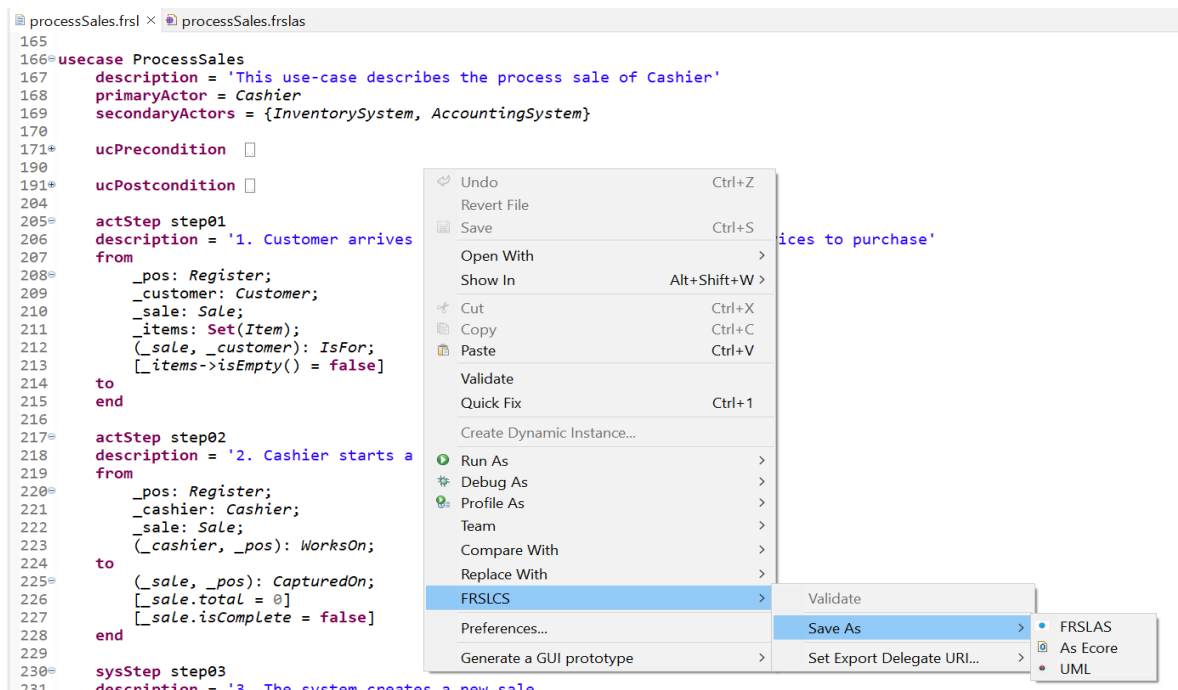
2. Nhân viên thu ngân bắt đầu thiết lập hóa đơn mới.
 3. Nhân viên thu ngân nhập mã định danh mặt hàng.
 4. Hệ thống ghi lại chi tiết đơn hàng bán hàng và trình bày mô tả mặt hàng, giá cả và tổng số tiền đang chạy. Giá được tính từ danh mục các sản phẩm.
- Nhân viên thu ngân lặp lại các bước 3-4 cho đến khi báo xong.
5. Hệ thống hiển thị tổng số thuế đã tính.
 6. Nhân viên thu ngân báo cho Khách hàng số tiền và yêu cầu thanh toán.
 7. Khách hàng thanh toán và Hệ thống xử lý thanh toán.
 8. Hệ thống ghi lại việc bán hàng đã hoàn tất và gửi thông tin bán hàng, thanh toán ra bên ngoài.
 9. Hệ thống xuất hóa đơn.
 10. Khách hàng ra về mang theo biên lai và hàng hóa.

```

processSales.frs1 x processSales.frs1as
165
166 usecase ProcessSales
167   description = 'This use-case describes the process sale of Cashier'
168   primaryActor = Cashier
169   secondaryActors = {InventorySystem, AccountingSystem}
170
171*  ucPrecondition
190
191*  ucPostcondition
204
205*  actStep step01
206    description = '1. Customer arrives at POS checkout with goods and/or services to purchase'
207    from
208      _pos: Register;
209      _customer: Customer;
210      _sale: Sale;
211      _items: Set(Item);
212      (_sale, _customer): IsFor;
213      [_items->isEmpty() = false]
214    to
215    end
216
217*  actStep step02
218    description = '2. Cashier starts a new sale'
219    from
220      _pos: Register;
221      _cashier: Cashier;
222      _sale: Sale;
223      (_cashier, _pos): WorksOn;
224    to
225      (_sale, _pos): CapturedOn;
226      [_sale.total = 0]
227      [_sale.isComplete = false]
228    end
229

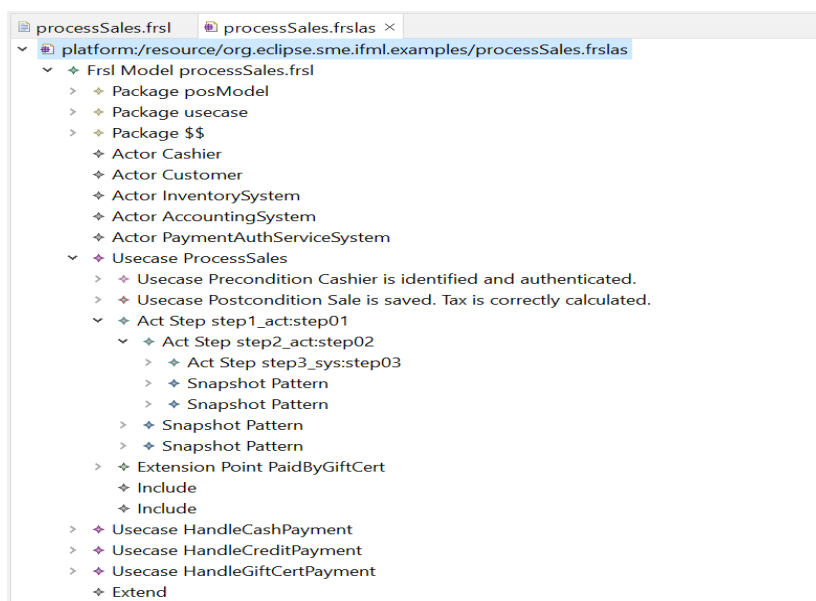
```

Hình 4.5: Đặc tả ca sử dụng dưới dạng FRSL.

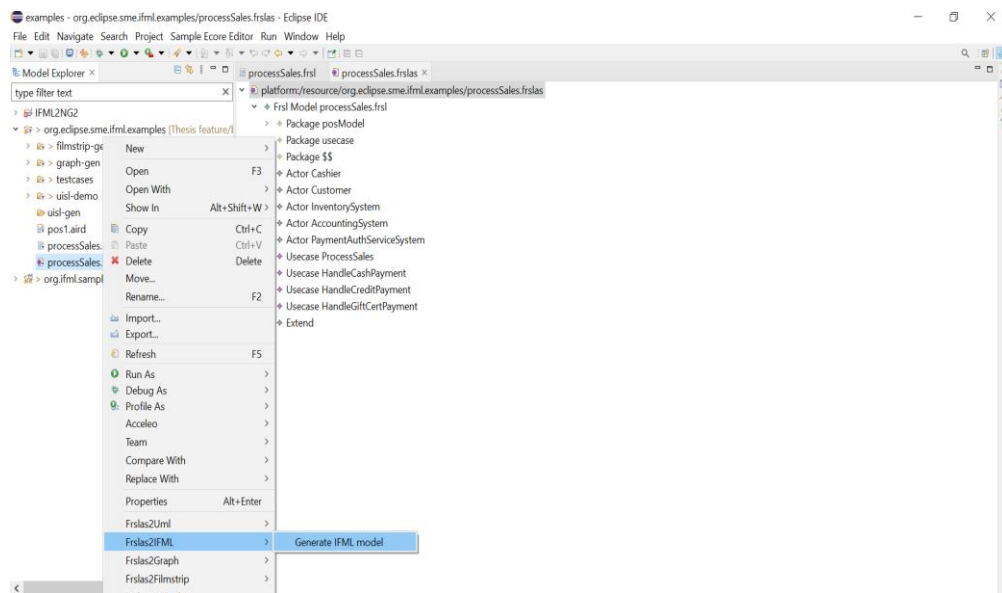


Hình 4.6: Chuyển đổi từ FRSL sang FRSLAS.

Hình 4.5 biểu diễn đặc tả FRSL cho ca sử dụng ProcessSale. Từ bản đặc tả ở dạng cú pháp văn bản này, công cụ VNU-FRSL cho phép chuyển đổi sang dạng cú pháp trừu tượng, như minh họa trong Hình 4.6. Bản đặc tả ca sử dụng này được lưu ở dạng tệp .frslas như minh họa trong Hình 4.7.

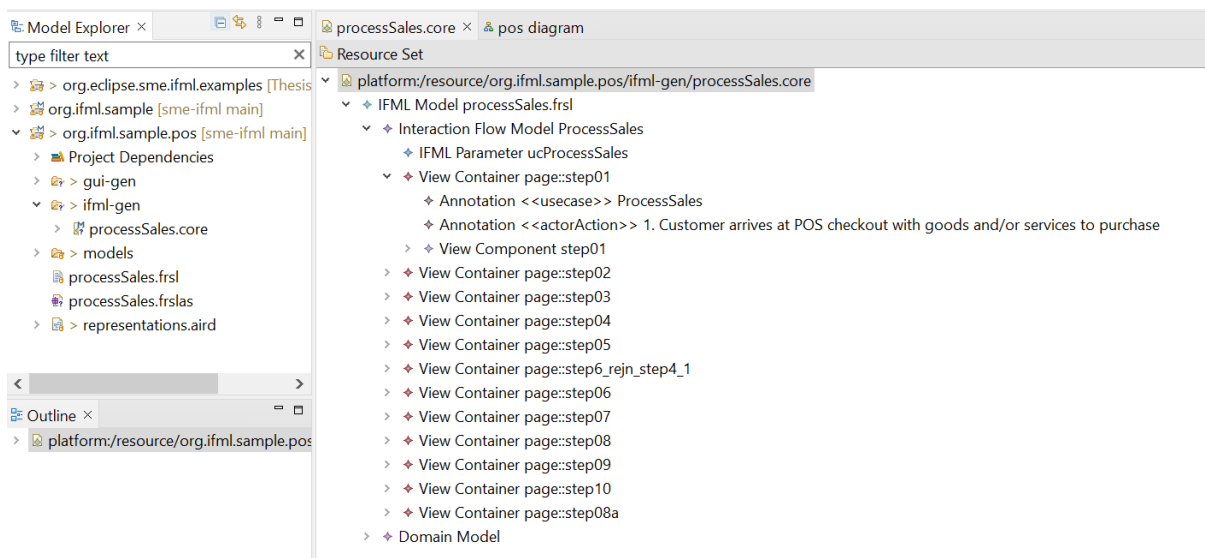


Hình 4.7: Đặc tả ca sử dụng dưới dạng FRSLAS.

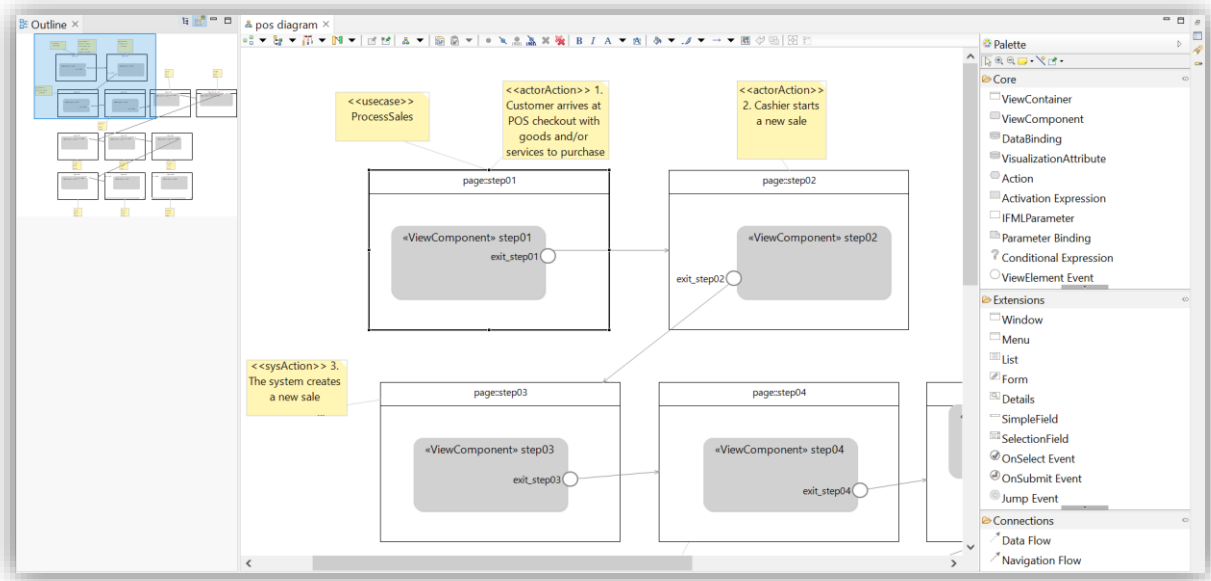


Hình 4.8: Sử dụng công cụ chuyển đổi mô hình FRSL2IFML.

Từ tệp đặc tả đầu vào frslas, như minh họa trong Hình 4.8, công cụ chuyển đổi FRSL2IFML cho phép sinh tự động mô hình luồng tương tác IFML. Mô hình tương tác IFML này có thể được biểu diễn ở dạng cú pháp trừu tượng (Hình 4.9) và dạng đồ họa (Hình 4.10).

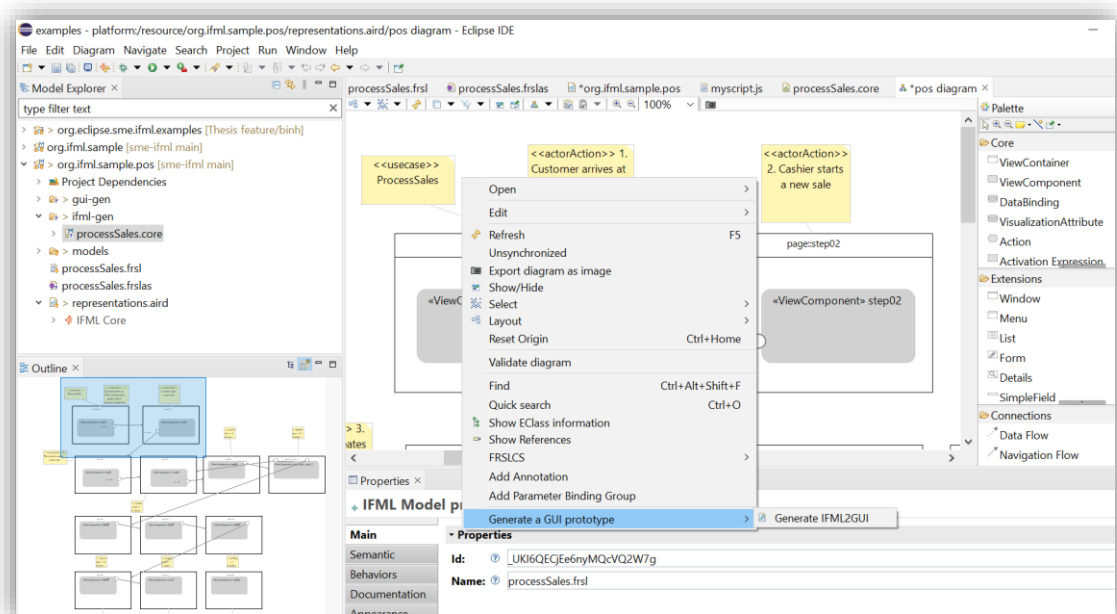


Hình 4.9: Mô hình IFML dạng cú pháp trừu tượng được sinh từ đặc tả đầu vào FRSL.



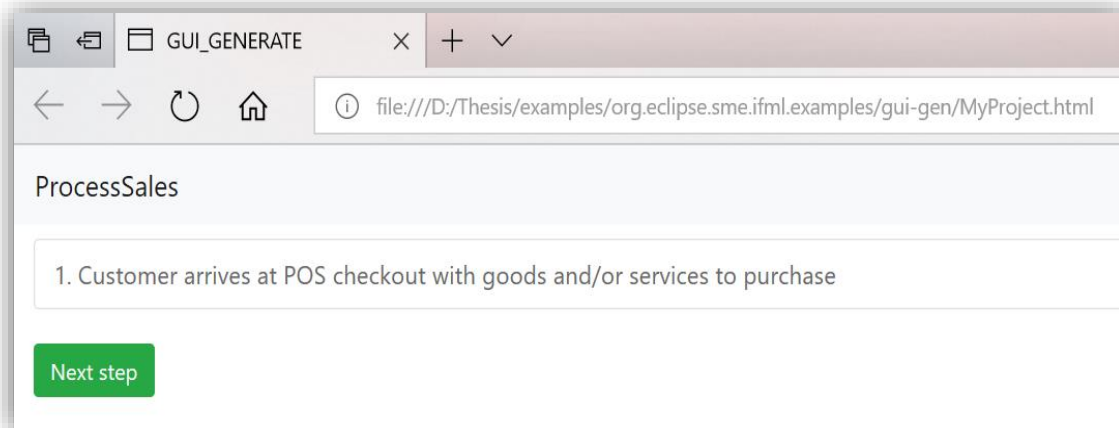
Hình 4.10: Mô hình IFML được sinh ở dạng đồ họa.

Lúc này người thiết kế có thể chỉnh sửa mô hình luồng tương tác bằng cách thêm và cập nhật các thành phần IFML, hoặc thay đổi thông tin cho từng thành phần ở khung nhìn Properties, cho đến khi đạt đến mức độ hoàn chỉnh nhất định.

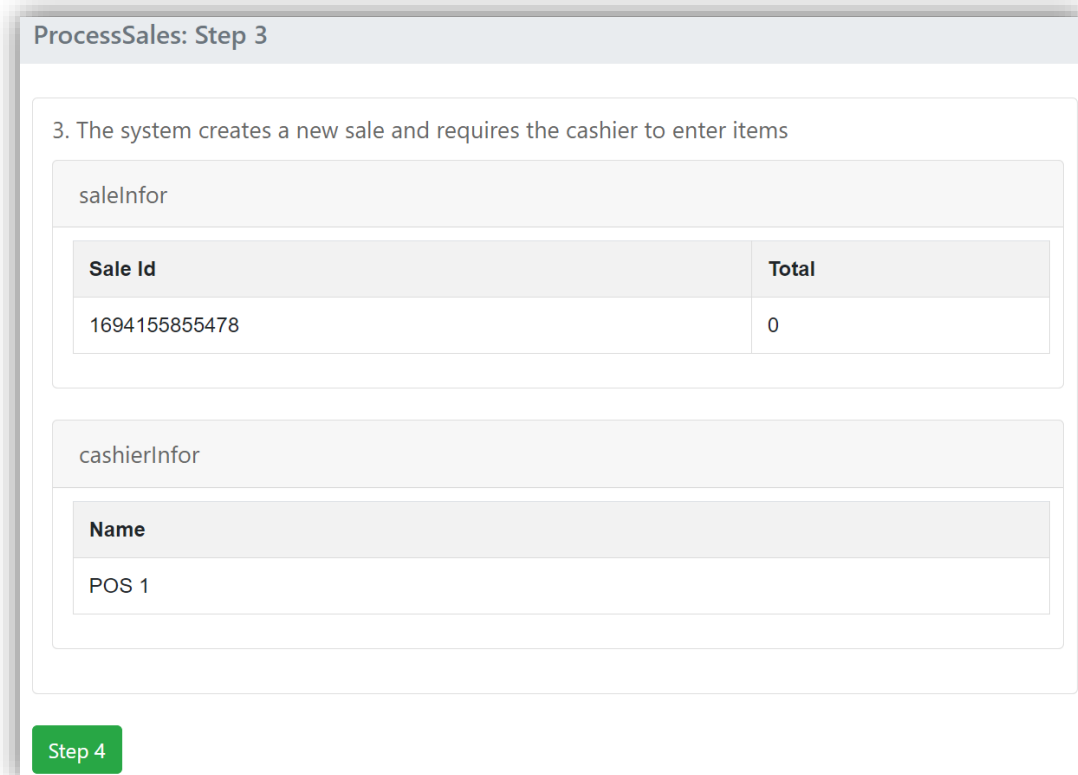


Hình 4.11: Sử dụng công cụ chuyển đổi IFML2GUI.

Từ tệp đặc tả mô hình IFML (có đuôi .core) được sinh trong thư mục */ifml-gen* bởi thành phần FRSL2IFML, người thiết kế tiếp tục sử dụng sử dụng bộ chuyển đổi IFML2GUI để sinh tự động giao diện người dùng ở dạng trang Web như minh họa trong Hình 4.12, Hình 4.13 và Hình 4.14.



Hình 4.12: Giao diện tự động bản mẫu được sinh tự động từ mô hình IFML.



Hình 4.13: Giao diện bản mẫu minh họa cho Step 3 trong Usecase ProcessSale.

ProcessSales: Step 4

4. Cashier enters item identifier

itemId 10

Step 5

Hình 4.14: Giao diện bản mẫu minh họa cho Step 4 có Form Component

4.3 Đánh giá phương pháp và thảo luận

Đặc tả FRSL cho phép chỉ định rõ ràng các ca sử dụng trong quá trình thu thập yêu cầu và nắm bắt các yêu cầu chức năng. Bộ công cụ FRSL cần soạn thảo để tạo ra các mô hình FRSL, và chuyển đổi sang FRSLAS. Đặc tả với FRSL có mức trừu tượng cao, điều này giúp cho việc nắm bắt yêu cầu chức năng ở giai đoạn thu thập, đặc tả yêu cầu rõ ràng nhưng vẫn đủ đơn giản và nhanh chóng.

Mô hình luồng tương tác IFML được sinh ra sau khi sử dụng bộ công cụ chuyển đổi hỗ trợ đảm bảo mô tả đầy đủ ca sử dụng. Mô hình được biểu diễn bằng dạng cú pháp trừu tượng và đồ họa giúp người sử dụng có thể dễ dàng tinh chỉnh trước khi sinh ra mã nguồn biểu diễn giao diện người dùng.

Áp dụng hai bộ công cụ chuyển đổi mô hình từ đặc tả yêu cầu FRSL, thực nghiệm trên ca sử dụng ProcessSale, đã sinh ra được bản mẫu giao diện đáp ứng với các yêu cầu của người dùng được mô tả trong phần đặc tả yêu cầu chức năng. Tuy nhiên, trong bối cảnh này, có thể nói rằng quy mô của nghiên cứu tình huống là hơi nhỏ để thực hiện đánh giá thực tế. Ở đây, mục tiêu bước đầu để chứng minh tính khả thi của phương pháp đề xuất thông qua một nghiên cứu tình huống nhất định.

Bộ công cụ giúp sinh tự động giao diện bản mẫu người dùng đạt được một nguyên mẫu thực thi đơn giản, ánh xạ các cấu trúc dữ liệu và cung cấp các hoạt động CRUD trên các cấu trúc đó. Tuy nhiên việc sử dụng database và kết nối server hay thực thi API tích hợp với công cụ hiện tại luận văn chưa giải quyết được. Đây có thể là hướng tiếp cận phát triển tiếp theo trong tương lai để hoàn thiện bộ công cụ sinh tự động giao diện người dùng từ mô hình đặc tả yêu cầu FRSL.

CHƯƠNG 5: KẾT LUẬN

Việc sinh tự động giao diện người dùng theo hướng mô hình mang lại nhiều lợi ích và có tầm quan trọng đáng kể trong quá trình phát triển phần mềm. Sinh tự động giao diện từ mô hình giúp giảm thiểu công việc lặp lại và công sức cần thiết để phát triển giao diện người dùng. Thay vì xây dựng giao diện thủ công từ đầu, việc tạo ra giao diện tự động từ mô hình giúp tiết kiệm thời gian và tăng năng suất công việc. Khi sử dụng phương pháp sinh tự động giao diện theo hướng mô hình, giao diện được tạo ra dựa trên cấu trúc và thông tin từ mô hình. Điều này đảm bảo tính nhất quán và đúng đắn của giao diện, đồng thời giảm thiểu sai sót và lỗi trong quá trình phát triển. Khi có thay đổi trong mô hình, việc cập nhật giao diện cũng được thực hiện tự động. Việc sinh tự động giao diện giúp giảm thiểu công việc bảo trì và cập nhật giao diện thủ công, đồng thời đảm bảo tính nhất quán của giao diện trong suốt quá trình phát triển và sửa lỗi. Sinh tự động giao diện từ mô hình giúp tạo ra giao diện có thể tương thích với nhiều nền tảng và thiết bị khác nhau. Việc phát triển giao diện đa nền tảng trở nên dễ dàng hơn, giúp mở rộng phạm vi sử dụng ứng dụng và đáp ứng nhu cầu của người dùng trên các nền tảng khác nhau. Việc sinh tự động giao diện từ mô hình giúp giảm khả năng mắc lỗi do con người gây ra trong quá trình phát triển giao diện. Các công cụ và quy trình tự động hóa giúp đảm bảo tính chính xác và nhất quán của giao diện, giảm thiểu rủi ro và các lỗi phát sinh.

Chính vì lẽ đó, luận văn này giới thiệu một phương pháp sinh tự động giao diện người dùng từ đặc tả yêu cầu thông qua mô hình hóa luồng tương tác IFML. Quy trình này giúp giảm thiểu sai sót và tăng tính nhất quán trong quá trình phát triển cũng như tăng hiệu suất phát triển nhờ việc tái sử dụng thiết kế hệ thống.

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:

- Luận văn tập trung vào ứng dụng phương pháp MDD trong việc phát triển giao diện người dùng dựa trên mô hình trừu tượng và tự động sinh mã nguồn từ các mô hình đặc tả ca sử dụng, trình bày kỹ thuật đặc tả yêu cầu chức năng với ngôn ngữ FRSL và kỹ thuật xây dựng mô hình tương tác với ngôn ngữ IFML, kết hợp với diễn giải các kỹ thuật chuyển đổi mô hình, bao gồm chuyển đổi sang mô hình (M2M) và sang văn bản (M2T).

- Luận văn đã xây dựng các quy tắc chuyển đổi mô hình: Nghiên cứu xây dựng các quy tắc chuyển đổi mô hình (M2M, M2T) để chuyển đổi mô hình từ đặc tả ca sử dụng FRSLAS sang các mô hình và mã nguồn tương ứng.
- Cuối cùng, luận văn đã xây dựng bộ công cụ hỗ trợ dưới dạng cho phép sinh tự động giao diện người dùng theo hướng mô hình và áp dụng vào bài toán thực tế Process Sale để đánh giá tính khả thi của bộ công cụ.

5.2 Hướng phát triển

Tuy đã đạt được các kết quả sơ bộ về việc sinh tự động bản mẫu giao diện người dùng từ đặc tả yêu cầu chức năng, phương pháp trình bày trong luận văn vẫn còn các vấn đề hạn chế. Do đó luận văn đề xuất các hướng phát triển tiếp theo trong tương lai:

- Công cụ hiện tại vẫn còn khá đơn giản, hoàn thiện công cụ, chỉnh sửa giao diện thân thiện với người dùng. Khả năng sinh giao diện, hiện tại chỉ dừng ở mức bản mẫu, tập trung chuyển đổi cho các thành phần giao diện hay dùng đến như biểu mẫu, bảng, nút,... chứ chưa đặc tả chi tiết được hết tất cả các thành phần mức cao cần dùng đến. Thiết kế cho giao diện cũng hơi đơn giản. Trong tương lai, ta cần đặc tả mức chi tiết mô hình hơn, bao phủ được nhiều thành phần hơn để ứng dụng được vào nhiều hệ thống và nâng cấp giao diện cho phù hợp.
- Luận văn mới chỉ áp dụng phương pháp trong các bài toán nhỏ đơn giản, chưa thử nghiệm với các chương trình đa dạng và phức tạp nên chưa đánh giá thực tế được hết phạm vi các bài toán mà phương pháp có thể áp dụng. Trong tương lai, luận văn kỳ vọng có thể xây dựng một bộ đầu vào các đặc tả cho nhiều bài toán đa dạng khác nhau để áp dụng quy trình sinh tự động bản mẫu giao diện nhằm đưa ra các số liệu thống kê, đánh giá được phạm vi áp dụng và độ chính xác của phương pháp.

TÀI LIỆU THAM KHẢO

- [1] S. Viswanathan and J. C. Peters, “Automating UI guidelines verification by leveraging pattern based UI and model based development,” in *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, in CHI EA '10. New York, NY, USA: Association for Computing Machinery, Apr. 2010, pp. 4733–4742. doi: 10.1145/1753846.1754222.
- [2] M. D. Lozano, P. Gonzalez, and I. Ramos, “User interface specification and modeling in an object oriented environment for automatic software development,” in *Proceedings. 34th International Conference on Technology of Object-Oriented Languages and Systems - TOOLS 34*, Aug. 2000, pp. 373–381. doi: 10.1109/TOOLS.2000.868987.
- [3] A. M. Rosado da Cruz and J. Faria, “Automatic Generation of user Interface Models and Prototypes from Domain and Use Case Models,” Jan. 2009.
- [4] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice: Second Edition*, 2nd edition. San Rafael, Calif.: Morgan & Claypool Publishers, 2017.
- [5] B. Combemale, R. France, J.-M. Jézéquel, B. Rumpe, J. Steel, and D. Vojtisek, *Engineering Modeling Languages: Turning Domain Knowledge into Tools*. CRC Press, 2016.
- [6] A. G. Kleppe, J. B. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture : Practice and Promise*. Addison-Wesley Professional, 2003.
- [7] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, “ATL: A model transformation tool,” *Sci. Comput. Program.*, vol. 72, no. 1, pp. 31–39, Jun. 2008, doi: 10.1016/j.scico.2007.08.002.
- [8] Duc-Hanh Dang, “FRSL: A Domain Specific Language to Specify Functional Requirements,” *VNU J. Sci. Comput. Sci. Commun. Eng.*, vol. 39, no. 1, pp. 87–106, 2023.
- [9] R. R. Young, *The Requirements Engineering Handbook*. Artech House, 2004.
- [10] M. Kamalrudin, nor aiza Mocketar, J. Grundy, J. Hosking, and M. Robinson, “Automatic acceptance test case generation from essential use cases,” 2014.
- [11] I. Gamito and A. R. da Silva, “From Rigorous Requirements and User Interfaces Specifications into Software Business Applications,” in *Quality of Information and Communications Technology*, M. Shepperd, F. Brito e Abreu, A. Rodrigues da Silva, and R. Pérez-Castillo, Eds., in Communications in Computer and Information Science, vol. 1266. Cham: Springer International Publishing, 2020, pp. 459–473. doi: 10.1007/978-3-030-58793-2_37..

- [12] M. Brambilla and P. Fraternali, “Interaction Flow Modeling Language Model-Driven UI Engineering of Web and Mobile Apps with IFML,” in *Interaction Flow Modeling Language*, Elsevier, 2015, pp. 1–8. doi: 10.1016/B978-0-12-800108-0.00001-1.
- [13] S. Sendall and W. Kozaczynski, “Model transformation: the heart and soul of model-driven software development,” *IEEE Softw.*, vol. 20, no. 5, pp. 42–45, Sep. 2003, doi: 10.1109/MS.2003.1231150.
- [14] C. Larman, *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development*. Prentice Hall PTR, 2005.
- [15] A. C. R. Paiva, D. Maciel, and A. R. da Silva, “From Requirements to Automated Acceptance Tests with the RSL Language,” in *Evaluation of Novel Approaches to Software Engineering*, E. Damiani, G. Spanoudakis, and L. A. Maciaszek, Eds., in Communications in Computer and Information Science, vol. 1172. Cham: Springer International Publishing, 2020, pp. 39–57. doi: 10.1007/978-3-030-40223-5_3.