ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

LÊ VĂN VINH

NGHIÊN CỨU CÁC KỸ THUẬT BIỂU DIỄN VÀ CHUYỂN ĐỔI MÔ HÌNH CHO THIẾT KẾ HƯỚNG MIỀN

LUẬN ÁN TIẾN SĨ KỸ THUẬT PHẦN MỀM

Hà Nội - 2026

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

LÊ VĂN VINH

NGHIÊN CỨU CÁC KỸ THUẬT BIỂU DIỄN VÀ CHUYỂN ĐỔI MÔ HÌNH CHO THIẾT KẾ HƯỚNG MIỀN

Chuyên ngành: Kỹ thuật phần mềm
Mã số: 9480103

LUẬN ÁN TIẾN SĨ KỸ THUẬT PHẦN MỀM

NGHIÊN CỨU SINH	CÁN BỘ HƯỚNG DẪN
LÊ VĂN VINH	PGS. TS. ĐẶNG ĐỨC HẠNH

XÁC NHẬN CỦA ĐƠN VỊ ĐÀO TẠO

T/L HIỆU TRƯỞNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
KT. TRƯỞNG PHÒNG CÔNG TÁC SINH VIÊN
PHÓ TRƯỞNG PHÒNG
Nguyễn Huy Tiệp

HÀ NỘI - 2026

LỜI CAM ĐOAN

Tôi xin cam đoan luận án **"Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền"** là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả được trình bày trong luận án là hoàn toàn trung thực và chưa từng được công bố trong bất kỳ một công trình nào khác.

- Tôi đã trích dẫn đầy đủ các tài liệu tham khảo, công trình nghiên cứu liên quan ở trong nước và quốc tế. Ngoại trừ các tài liệu tham khảo này, luận án hoàn toàn là công việc của riêng tôi.
- Trong các công trình khoa học được công bố trong luận án, tôi đã thể hiện rõ ràng và chính xác đóng góp của các đồng tác giả và những gì do tôi đã đóng góp.
- Luận án được hoàn thành trong thời gian tôi làm nghiên cứu sinh tại Bộ môn Công nghệ phần mềm, Khoa Công nghệ thông tin, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội.

Tác giả: _____

Hà Nội: _____

LỜI CẢM ƠN

Trước hết, tôi muốn bày tỏ sự biết ơn đến cán bộ hướng dẫn PGS.TS. Đặng Đức Hạnh, người đã hướng dẫn, khuyến khích, truyền cảm hứng, chỉ bảo và tạo cho tôi những điều kiện tốt nhất từ khi bắt đầu làm nghiên cứu sinh đến khi hoàn thành luận án này.

Tôi xin chân thành cảm ơn các thầy cô giáo Khoa Công nghệ thông tin, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội, đặc biệt là các thầy cô trong Bộ môn Công nghệ phần mềm đã tận tình đào tạo, cung cấp cho tôi những kiến thức vô cùng quý giá, đã tạo điều kiện tốt nhất cho tôi về môi trường làm việc trong suốt quá trình học tập và nghiên cứu.

Tôi xin trân trọng cảm ơn Phòng Đào tạo, Phòng Công tác sinh viên và Ban giám hiệu Trường Đại học Công nghệ đã tạo điều kiện thuận lợi cho tôi trong suốt quá trình thực hiện luận án.

Tôi cũng bày tỏ sự biết ơn đến Trường Đại học Sư phạm Kỹ thuật Vinh đã tạo điều kiện về thời gian và tài chính cho tôi thực hiện luận án này. Tôi muốn cảm ơn đến tập thể, các cán bộ giảng viên Trường Đại học Sư phạm Kỹ thuật Vinh đã cổ vũ, động viên và tạo điều kiện về thời gian cho tôi trong suốt quá trình nghiên cứu.

Tôi muốn cảm ơn đến tất cả những người bạn, các anh chị em nghiên cứu sinh và những người đồng nghiệp của tôi. Những người đã luôn chia sẻ với tôi những khó khăn và động viên giúp đỡ tôi bất cứ khi nào tôi cần và tôi luôn ghi nhớ điều đó.

Cuối cùng, tôi xin bày tỏ lòng biết ơn vô hạn tới gia đình, đặc biệt là vợ và các con, những người luôn ủng hộ và yêu thương tôi vô điều kiện. Nếu không có sự động viên và hậu thuẫn đó, tôi không thể hoàn thành luận án này.

NCS. Lê Văn Vinh

TÓM TẮT

Thiết kế hướng miền (*Domain-Driven Design – DDD*) đã nổi lên như một phương pháp nổi bật nhằm giải quyết các thách thức liên quan đến sự gia tăng về độ phức tạp, quy mô và tính không đồng nhất của các hệ thống phần mềm hiện đại. Phương pháp DDD thực hiện phát triển lặp lại dựa trên một mô hình miền giàu ngữ nghĩa, trong đó mô hình hóa lô-gic cốt lõi và các quy tắc của miền vấn đề. DDD sử dụng nhất quán ngôn ngữ chung, qua đó tăng cường giao tiếp hiệu quả, sự hiểu biết chung giữa chuyên gia miền và lập trình viên trong suốt vòng đời phần mềm. Đã có nhiều nghiên cứu về các kỹ thuật biểu diễn mô hình miền bằng ngôn ngữ chuyên biệt miền (*Domain-Specific Language – DSL*). Các nghiên cứu này sử dụng DSL để liên kết chặt chẽ mô hình thiết kế với quá trình triển khai, qua đó nâng cao tính chính xác, khả năng bảo trì và mở rộng của phần mềm. Các nghiên cứu gần đây theo nguyên lý của DDD tập trung vào việc sử dụng các ngôn ngữ chuyên biệt miền dựa trên chú thích, kết hợp với các kỹ thuật chuyển đổi mô hình nhằm sinh tự động phần mềm. Tuy nhiên, các nghiên cứu trước đây vẫn chưa mô tả một cách rõ ràng và thống nhất các khía cạnh hành vi trong mô hình miền, đồng thời gặp hạn chế trong việc đặc tả và tích hợp các ràng buộc phức tạp. Bên cạnh đó, các mô hình miền thực thi hiện nay còn thiếu một cơ sở ngữ nghĩa hình thức để kiểm chứng tính đúng đắn của chúng, và chưa có một phương pháp hoàn chỉnh cho bộ chuyển đổi mô hình từ đặc tả yêu cầu sang mã nguồn có thể thực thi.

Nhằm giải quyết các thách thức về kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền, qua đó thu hẹp khoảng cách giữa mô hình miền và phần mềm thực thi, luận án này đề xuất các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền. Các đóng góp chính của luận án như sau.

Thứ nhất, luận án đề xuất các kỹ thuật biểu diễn mô hình miền, hướng tới việc tích hợp đặc tả cấu trúc, các ràng buộc phức tạp, hành vi và bảo mật trong cùng một khuôn khổ biểu diễn, làm cơ sở cho việc đặc tả đầy đủ các thông tin cần thiết phục vụ sinh tự động phần mềm trong bối cảnh thiết kế hướng miền.

Thứ hai, luận án đề xuất một kỹ thuật tích hợp các DSL theo các mối quan tâm không đồng nhất vào một mô hình miền hợp nhất, được gọi là *Unified Domain Model Language (UDML)*. Trong đó, UDML được đặc tả

đầy đủ ở cả cú pháp và ngữ nghĩa thực thi hình thức, đồng thời hỗ trợ kiểm chứng hình thức thông qua cơ chế ánh xạ ngữ nghĩa sang các đặc tả hình thức tương ứng.

Thứ ba, luận án đề xuất các kỹ thuật chuyển đổi mô hình, cải tiến một hoặc một số đặc trưng được lựa chọn, cho phép sinh tự động các bản mẫu phần mềm từ mô hình miền dựa trên các kỹ thuật biểu diễn trên miền đã xây dựng, hướng tới các chuyển đổi mô hình có chất lượng và phù hợp với các kịch bản sử dụng cụ thể trong các miền ứng dụng chuyên biệt.

Cuối cùng, luận án phát triển công cụ thực nghiệm và tiến hành đánh giá tính khả thi của việc áp dụng phương pháp đề xuất trong thực tế.

Các đóng góp này có ý nghĩa trong việc hỗ trợ đặc tả chính xác các mô hình miền, đồng thời hiện thực hóa các thao tác tự động trên mô hình miền, bao gồm sinh tự động mã nguồn và kiểm tra tính tuân thủ giữa thiết kế và cài đặt. Qua đó, luận án góp phần gia tăng mức độ tự động hóa trong phát triển phần mềm.

Từ khóa: Thiết kế hướng miền, kỹ thuật biểu diễn, chuyển đổi mô hình, UDML, DSL, aDSL.

Mục lục

DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	Tiếng Anh	Tiếng Việt
AI	Artificial Intelligence	Trí tuệ nhân tạo
AGL	Activity Graph Language	Ngôn ngữ đồ thị hoạt động
ASM	Abstract Syntax Meta-Model	Siêu mô hình cú pháp trừu tượng
AST	Abstract Syntax Tree	Cây cú pháp trừu tượng
aDSL	annotation-based Domain-Specific Language	Ngôn ngữ chuyên biệt miền dựa vào chú thích
CAP	Constraint Annotation Pattern	Mẫu chú thích ràng buộc
CSM	Concrete Syntax Meta-Model	Siêu mô hình cú pháp cụ thể dạng văn bản
DDD	Domain-Driven Design	Thiết kế hướng miền
DM	Domain Model	Mô hình miền
DCSL	Domain Class Specification Language	Ngôn ngữ đặc tả lớp miền
DSML	Domain-Specific Modeling	Ngôn ngữ mô hình hóa chuyên biệt miền
DSM	Domain-Specific Modeling	Mô hình hóa chuyên biệt miền
DSL	Domain-Specific Language	Ngôn ngữ chuyên biệt miền
EMF	Eclipse Modeling Framework	Khung mô hình hóa Eclipse
GUI	Graphical User Interface	Giao diện người dùng đồ họa

IDE	Integrated Development Environment	Môi trường phát triển tích hợp
IFML	Interaction Flow Modeling Language	Ngôn ngữ mô hình hóa luồng tương tác
LLM	Large Language Model	Mô hình ngôn ngữ lớn
MDE	Model-Driven Engineering	Kỹ nghệ hướng mô hình
MOF	Meta-Object Facility	Phương tiện siêu mô hình
M2M	Model to Model	Mô hình sang mô hình
M2T	Model to Text	Mô hình sang văn bản
MOSA	Module-based Software Architecture	Kiến trúc phần mềm dựa vào mô-đun
MCC	Module Configuration Class	Lớp cấu hình mô-đun phần mềm
MDA	Model-Driven Architecture	Kiến trúc hướng mô hình
MDSE	Model-Driven Software Engineering	Kỹ nghệ phát triển phần mềm hướng mô hình
MVC	Model View Controller	Mô hình thiết kế phần mềm
OMG	Object Management Group	Nhóm quản lý đối tượng
OCL	Object Constraint Language	Ngôn ngữ ràng buộc đối tượng
OOPL	Object-Oriented Programming Language	Ngôn ngữ lập trình hướng đối tượng
RM	Requirement Model	Mô hình yêu cầu
UL	Ubiquitous Language	Ngôn ngữ chung
UDML	Unified Domain Model Language	Ngôn ngữ mô hình miền hợp nhất
UML	Unified Modeling Language	Ngôn ngữ mô hình hóa hợp nhất
USE	UML-based Specification Environment	Môi trường đặc tả dựa trên UML
XML	eXtensible Markup Language	Ngôn ngữ đánh dấu mở rộng

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ

Chương 1

GIỚI THIỆU

1.1 Đặt vấn đề

Trong nghiên cứu kỹ nghệ phần mềm hiện đại, thiết kế phần mềm được xem là một quá trình mang tính sáng tạo và lặp lại, trong đó các nguyên tắc, kỹ thuật và công cụ được khám phá và áp dụng xuyên suốt vòng đời phát triển phần mềm [32, 36]. Chất lượng của sản phẩm phần mềm vì vậy phụ thuộc đáng kể vào kinh nghiệm, kỹ năng và cách tiếp cận của các nhà phát triển phần mềm. Trong thực tiễn phát triển phần mềm, việc xây dựng các hệ thống có quy mô lớn và độ phức tạp cao bằng các phương pháp mang tính thủ công vẫn gặp nhiều thách thức, bao gồm khó khăn trong quản lý sự phụ thuộc, gia tăng chi phí và kéo dài thời gian triển khai. Do đó, nhu cầu rút ngắn chu kỳ phát triển, giảm thiểu chi phí, đồng thời đảm bảo khả năng thích ứng với các yêu cầu ngày càng đa dạng của phần mềm trong nhiều lĩnh vực ứng dụng trở nên cấp thiết.

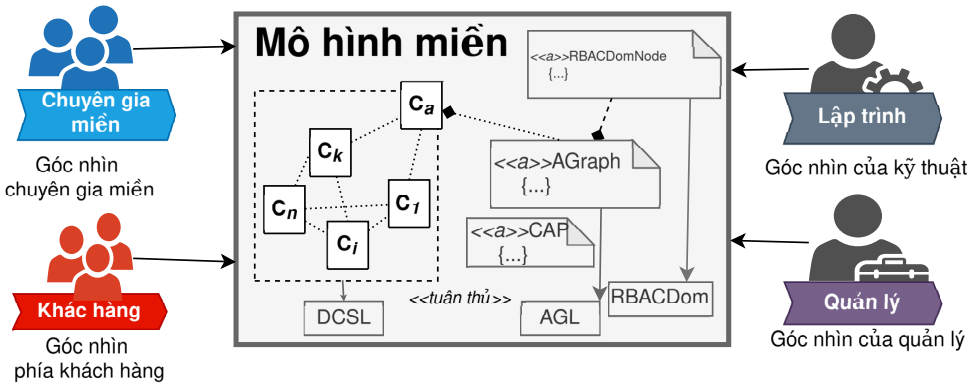
Trong bối cảnh đó, phát triển phần mềm dựa trên mô hình đã nổi lên như một hướng tiếp cận có hệ thống nhằm giảm thiểu độ phức tạp của phát triển phần mềm bằng việc sử dụng các mô hình trừu tượng làm trung tâm [13, 36]. Đặc biệt, kỹ nghệ phần mềm hướng mô hình (*Model-Driven Software Engineering – MDSE*) [13, 18, 36] được xem là một trong những hướng phát triển rõ nét nhất của phát triển phần mềm dựa trên mô hình, tập trung vào việc khai thác các mô hình ở mức trừu tượng cao để hỗ trợ và tự động hóa quá trình phát triển phần mềm, từ thiết kế đến hiện thực hóa [32, 36]. Song song với MDSE, một phương pháp tiếp cận khác, khiêm

tốn hơn nhưng trực tiếp và thực tiễn hơn, là phương pháp thiết kế hướng miền (*Domain-Driven Design – DDD*) do Evans đề xuất [41, 124, 125].

DDD nhấn mạnh vai trò trung tâm của mô hình miền (*Domain Model - DM*) trong toàn bộ quá trình phát triển phần mềm. Theo Evans [41], DM không chỉ là một biểu diễn trừu tượng của miền nghiệp vụ mà còn đóng vai trò như một ngôn ngữ chung (*Ubiquitous Language – UL*) được chia sẻ giữa các bên liên quan. Trên cơ sở đó, DM trở thành nền tảng để định hình cấu trúc, hành vi và ngữ nghĩa nghiệp vụ của hệ thống phần mềm, đồng thời định hướng việc xây dựng và tiến hóa hệ thống một cách nhất quán. DDD có mối liên hệ chặt chẽ với các ngôn ngữ lập trình hướng đối tượng (*Object-Oriented Programming Languages – OOPL*). Theo Evans [41], các khái niệm cốt lõi của DDD như thực thể, đối tượng giá trị, tập hợp và dịch vụ miền có thể được ánh xạ một cách tự nhiên sang các cấu trúc của lập trình hướng đối tượng. Nhờ đó, DM vừa có khả năng biểu đạt cao, vừa khả thi về mặt kỹ thuật khi được triển khai trong các hệ thống phần mềm hướng đối tượng. Sự phù hợp này đã được chỉ ra trong nghiên cứu [17] cho rằng đối tượng là phương tiện tự nhiên để biểu diễn các thực thể của miền thế giới thực, đồng thời cấu trúc đối tượng cũng là nền tảng của các ngôn ngữ mô hình hóa phân tích và thiết kế ở mức cao. Quan điểm này tiếp tục được kế thừa và mở rộng trong các nghiên cứu gần đây, trong đó lập trình hướng đối tượng được xem như một ngôn ngữ khái niệm, có thể kết hợp với các yếu tố của lập trình hàm nhằm tăng cường khả năng biểu đạt trong việc mô tả các khái niệm miền và DM [113]. Tuy nhiên, mặc dù OOPL tạo điều kiện thuận lợi cho việc biểu diễn và triển khai DM, quá trình chuyển từ DM trừu tượng sang phần mềm có khả năng thực thi vẫn chủ yếu dựa vào thao tác thủ công của nhà phát triển, từ đó làm gia tăng nguy cơ sai lệch ngữ nghĩa giữa DM và hệ thống được xây dựng.

Trong bối cảnh đó, các OOPL không chỉ đóng vai trò là môi trường triển khai mà còn có thể được xem như ngôn ngữ chủ để biểu diễn và mở rộng DM. Thông qua các cơ chế như chú thích, kiểu dữ liệu trừu tượng và cấu trúc ngôn ngữ, OOPL cho phép nhúng trực tiếp các đặc tả miền mở rộng bao gồm hành vi, ràng buộc và các chính sách bảo mật vào DM. Tuy nhiên, các cơ chế này thường được sử dụng rời rạc và thiếu một nền tảng ngữ nghĩa thống nhất, dẫn đến hạn chế trong việc tích hợp các mối quan tâm và hỗ trợ thực thi một cách có hệ thống theo DDD.

Trong thực tiễn phát triển phần mềm, các bên liên quan tiếp cận hệ thống từ những góc nhìn khác nhau; chẳng hạn, người dùng chủ yếu tương tác bằng ngôn ngữ tự nhiên và giao diện, trong khi nhà phát triển làm việc với các cấu trúc kỹ thuật và ngôn ngữ lập trình được trình bày trong Hình 1.1. Theo triết lý của DDD, sự khác biệt về góc nhìn này cần được quy chiếu về một ngôn ngữ chung thống nhất, trong đó DM đóng vai trò là lõi ngữ nghĩa, liên kết tri thức nghiệp vụ với thiết kế và triển khai hệ thống. Tuy nhiên, trong thực tiễn, vẫn tồn tại một khoảng cách đáng kể giữa mô hình miền và khả năng thực thi. DM thường được đặc tả bằng các biểu đồ lớp UML (*Unified Modeling Language*) kết hợp với các ràng buộc OCL (*Object Constraint Language*) [94] nhằm mô tả cấu trúc và các ràng buộc nghiệp vụ trên DM, trong khi việc đưa các đặc tả này đến trạng thái có khả năng thực thi phần lớn vẫn được thực hiện thủ công. Cách tiếp cận này khiến DM chủ yếu đóng vai trò như một đặc tả ở mức thiết kế, khó đạt được ngữ nghĩa của một DM *có khả năng thực thi*. Mặc dù các tiếp cận dựa trên ngôn ngữ chuyên biệt miền (*Domain-Specific Language – DSL*) [46, 106, 135] và MDSE [18] đã được đề xuất nhằm thu hẹp khoảng cách này, việc áp dụng và tích hợp chúng một cách chặt chẽ trong bối cảnh DDD vẫn còn nhiều hạn chế. Khoảng cách này làm suy giảm tính nhất quán ngữ nghĩa, đồng thời gây khó khăn cho việc bảo trì, mở rộng và tiến hóa phần mềm theo đúng tinh thần của DDD.



Hình 1.1: Các góc nhìn khác nhau trên cùng một mô hình miền.

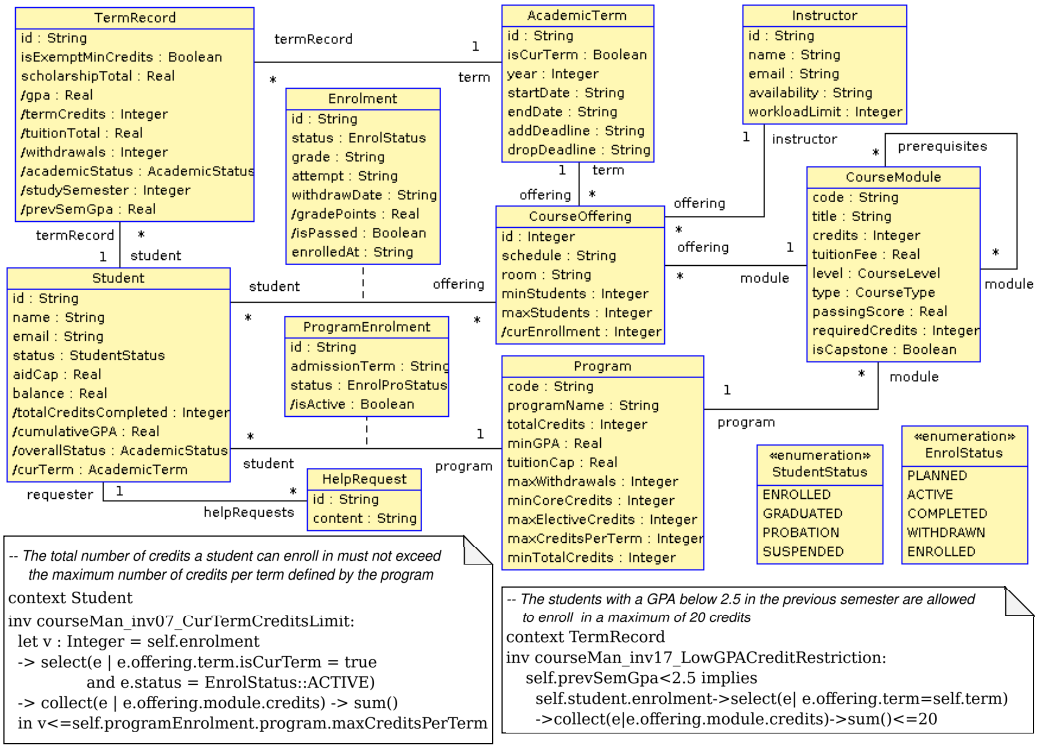
Mặc dù DDD đặt DM làm trung tâm của quá trình phát triển phần mềm, các cách tiếp cận hiện hữu chưa cung cấp được một cơ chế biểu diễn cho phép đặc tả một cách đầy đủ và nhất quán các khía cạnh cốt lõi của miền bao gồm: cấu trúc, hành vi và các ràng buộc nghiệp vụ cũng như chưa hỗ trợ hiệu quả việc tích hợp các đặc tả này thành một mô hình miền hợp

nhất. Hơn nữa, việc chuyển đổi mô hình để thu được mô hình có khả năng thực thi, làm cơ sở cho sinh tự động phần mềm đúng yêu cầu và tuân thủ các nguyên lý của DDD, vẫn còn thiếu các cơ chế mang tính hệ thống. Các vấn đề này được làm rõ hơn bằng ví dụ thúc đẩy ở phần tiếp theo.

1.1.1 Ví dụ thúc đẩy nghiên cứu: Hệ thống `CourseMan`

Luận án sử dụng Hệ thống quản lý khóa học (CourseMan) như một ví dụ thúc đẩy nghiên cứu, nhằm làm rõ các vấn đề cốt lõi trong biểu diễn và chuyển đổi mô hình miền. CourseMan là một hệ thống thông tin điển hình trong miền quản lý đào tạo, bao quát nhiều khía cạnh nghiệp vụ như quản lý sinh viên, học phần, tổ chức giảng dạy, đăng ký học và theo dõi kết quả học tập.

CourseMan được lựa chọn do tính đại diện và mức độ phức tạp vừa phải của miền ứng dụng: hệ thống không chỉ bao gồm các cấu trúc dữ liệu phong phú mà còn liên quan đến nhiều quy tắc nghiệp vụ và quy trình xử lý đặc trưng. Nhờ đó, ví dụ này cho phép làm rõ các vấn đề về biểu diễn mô hình miền một cách cụ thể, đồng thời tạo cơ sở để dẫn dắt và minh họa các thách thức nghiên cứu được xem xét trong luận án.



Hình 1.2: Mô hình miền của Hệ thống quản lý khóa học.

Hình 1.2 trình bày mô hình miền của hệ thống CourseMan, trong đó các khái niệm miền chính được biểu diễn sử dụng các lớp như `Student`, `CourseModule`, `CourseOffering`, `Instructor`, `AcademicTerm` và `Program`. Quan hệ đăng ký học phần của sinh viên được mô hình hóa bằng lớp `Enrolment`, trong khi kết quả học tập theo từng học kỳ được thể hiện bởi `TermRecord`. Mô hình này phản ánh các thực thể và quan hệ cốt lõi của miền ứng dụng. Bên cạnh cấu trúc, hệ thống còn bao gồm các ràng buộc nghiệp vụ và các hành vi miền cơ bản. Các ràng buộc thể hiện các quy tắc toàn vẹn, chẳng hạn điều kiện đăng ký học phần hoặc giới hạn số lượng sinh viên của lớp học phần. Các hành vi miền liên quan đến các quy trình như mở lớp học phần, đăng ký học và cập nhật kết quả học tập. Các yếu tố này cho thấy mô hình miền cần bao quát đồng thời cấu trúc, ràng buộc và hành vi ở mức trừu tượng phù hợp.

Tuy nhiên, từ ví dụ trên có thể nhận thấy rằng các khía cạnh của mô hình miền, bao gồm cấu trúc, ràng buộc và hành vi, thường được biểu diễn bằng các ngôn ngữ và ở các mức trừu tượng khác nhau, và được phát triển tương đối độc lập. Điều này dẫn đến sự không đồng nhất trong biểu diễn và gây khó khăn trong việc duy trì một cách hiểu nhất quán về miền. Hệ quả là việc thiết lập và duy trì các liên kết ngữ nghĩa giữa các mô hình trở nên khó khăn, đặc biệt trong bối cảnh hệ thống tiến hóa và các thay đổi cần được phản ánh nhất quán trên nhiều khía cạnh. Bên cạnh đó, việc thiếu các cơ chế tích hợp và chuyển đổi mô hình một cách có hệ thống làm hạn chế khả năng bảo toàn ngữ nghĩa, đồng thời gây khó khăn cho việc kiểm chứng và tự động hóa các hoạt động phát triển phần mềm. Do đó, bài toán đặt ra không chỉ là xây dựng các mô hình riêng lẻ cho từng khía cạnh, mà là làm thế nào để biểu diễn, liên kết và chuyển đổi các mô hình này một cách nhất quán, nhằm hướng tới một biểu diễn miền hợp nhất có khả năng bảo toàn ngữ nghĩa.

Mặc dù CourseMan được sử dụng như một ví dụ thúc đẩy để làm rõ bài toán nghiên cứu, luận án không giới hạn việc đánh giá các kỹ thuật đề xuất trong một miền ứng dụng duy nhất. Nhằm đánh giá tính tổng quát và khả năng áp dụng của phương pháp, luận án còn xem xét các ca nghiên cứu bổ sung bao gồm: Hệ thống quản lý đặt hàng (OrderMan), Hệ thống quản lý quy trình nghiệp vụ (ProcessMan) và Hệ thống quản lý tạp chí mở (OJS). Cụ thể, OrderMan được sử dụng để khảo sát các đặc trưng liên quan đến

quy trình nghiệp vụ và sự phối hợp giữa các hành vi miền; ProcessMan cho phép đánh giá khả năng biểu diễn và xử lý các mô hình quy trình trong các bối cảnh ứng dụng khác nhau; trong khi đó, OJS là một hệ thống thực tế, dùng để minh họa các yêu cầu về kiểm soát truy cập dựa trên vai trò và khả năng tích hợp các chính sách bảo mật ở mức mô hình. Việc xem xét đồng thời các hệ thống này cho phép đánh giá phương pháp đề xuất trên nhiều miền ứng dụng với các đặc trưng khác nhau, từ đó làm rõ khả năng mở rộng và tính tổng quát của cách tiếp cận.

1.1.2 Các thách thức nghiên cứu trong DDD

Thứ nhất, các nghiên cứu hiện tại về mô hình hóa trong DDD cho thấy vẫn còn hạn chế trong việc biểu diễn đồng thời và nhất quán các khía cạnh cấu trúc, hành vi và ràng buộc của miền trong một khuôn khổ hợp nhất. Phần lớn các tiếp cận sử dụng các biểu đồ UML và các ràng buộc OCL ở các không gian biểu diễn tách biệt, hoặc chỉ khai thác từng lát cắt riêng lẻ của mô hình miền (DM) thông qua các DSL [46, 106]. Một số nghiên cứu đề xuất các DSL nội sinh [135], đặc biệt là các tiếp cận dựa trên chú thích (*annotation-based Domain-Specific Language – aDSL*), trong đó DM được nhúng trực tiếp vào ngôn ngữ chủ [14, 33, 73, 98, 122]. Mặc dù các tiếp cận này cho phép xây dựng các DM có khả năng thực thi ở một mức độ nhất định, mối quan hệ ngữ nghĩa giữa cấu trúc, hành vi và ràng buộc vẫn chưa được đặc tả đầy đủ và nhất quán trong một khuôn khổ DM hợp nhất.

Thứ hai, trong các nghiên cứu hiện có, việc mô hình hóa và tích hợp nhiều mối quan tâm trong bối cảnh DDD hiện nay vẫn thiếu một nền tảng hợp nhất có ngữ nghĩa rõ ràng và khả năng thực thi. Các mối quan tâm như hành vi nghiệp vụ, bảo mật và các yêu cầu phi chức năng thường được đặc tả bằng các DSL riêng biệt theo từng góc nhìn [28, 39]. Mặc dù cách tiếp cận này làm tăng tính mô-đun và khả năng biểu đạt cho từng mối quan tâm, việc thiếu một DM hợp nhất phản ánh đầy đủ ngữ nghĩa tổng thể của hệ thống khiến DM khó đóng vai trò là trung tâm ngữ nghĩa như yêu cầu của DDD.

Thứ ba, trong các nghiên cứu về chuyển đổi mô hình trong kỹ nghệ phần mềm hướng mô hình (*Model-Driven Engineering – MDE*) [18, 31] chủ yếu tập trung vào các phép chuyển đổi riêng lẻ và thường được thực hiện ở mức

cú pháp. Các bộ chuyển đổi từ mô hình yêu cầu, gồm các biểu đồ UML và các ràng buộc OCL, sang DM và tiếp tục tới các hệ thống có khả năng thực thi hiện chưa được tổ chức một cách có hệ thống, đặc biệt về cơ chế bảo toàn ngữ nghĩa của hành vi và các ràng buộc miền [18, 72]. Điều này hạn chế khả năng kiểm soát chất lượng và tính đúng đắn trong quá trình sinh phần mềm dựa trên DM.

Từ những hạn chế đã được chỉ ra trong các nghiên cứu hiện có, luận án tập trung nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình nhằm xây dựng một DM làm trung tâm, vừa đóng vai trò là ngôn ngữ chung cho các bên liên quan, vừa cho phép tích hợp các mối quan tâm như hành vi, ràng buộc và bảo mật. Trên cơ sở đó, luận án hướng tới xây dựng các mô hình miền có khả năng thực thi và hỗ trợ sinh tự động phần mềm thông qua các cơ chế chuyển đổi mô hình, phù hợp với triết lý của DDD.

1.1.3 Phát biểu bài toán nghiên cứu

Luận án xác định các bài toán nghiên cứu cốt lõi cần được giải quyết, được cụ thể hóa thành các vấn đề nghiên cứu sau.

Vấn đề 1: Kỹ thuật biểu diễn mô hình miền tích hợp, giàu ngữ nghĩa và có khả năng thực thi. Trong DDD, mô hình miền được xem là trung tâm ngữ nghĩa của hệ thống phần mềm [41]. Tuy nhiên, các tiếp cận mô hình hóa hiện nay chủ yếu tập trung vào khía cạnh cấu trúc, trong khi các khía cạnh quan trọng khác của miền như: hành vi và các ràng buộc nghiệp vụ thường được đặc tả tách rời bằng các mô hình hoặc ngôn ngữ khác nhau [18, 46].

Mặc dù một số tiếp cận dựa trên DSL nội sinh và chú thích cho phép gắn kết mô hình miền với triển khai [135], chúng vẫn chưa cung cấp được một cơ chế biểu diễn hợp nhất cho phép đặc tả đầy đủ và nhất quán các khía cạnh cốt lõi của miền trong một mô hình miền có khả năng thực thi. Do đó, việc nghiên cứu các kỹ thuật biểu diễn mô hình miền tích hợp, giàu ngữ nghĩa và hướng thực thi đặt ra như một vấn đề nghiên cứu quan trọng trong bối cảnh DDD.

Vấn đề 2: Cơ chế hợp nhất các mối quan tâm trong một mô hình miền thống nhất. Trong thực tiễn phát triển phần mềm, các mối

quan tâm khác nhau của miền chẳng hạn như: cấu trúc, hành vi và bảo mật thường được đặc tả thông qua các mô hình hoặc DSL riêng biệt [18]. Cách tiếp cận này giúp tăng tính mô-đun, nhưng đồng thời làm suy giảm vai trò trung tâm ngữ nghĩa của mô hình miền theo tinh thần của DDD [41].

Các nghiên cứu hiện nay chủ yếu dừng ở việc tích hợp các mối quan tâm ở mức cú pháp hoặc cấu trúc mô hình, trong khi thiếu một cơ chế hợp nhất dựa trên ngữ nghĩa cho phép diễn giải thống nhất các mối quan tâm này trong một mô hình miền duy nhất [123]. Do đó, việc xây dựng một cơ chế hợp nhất các mối quan tâm dựa trên một nền tảng ngữ nghĩa chung, cho phép mô hình miền được xem như một thực thể thống nhất có khả năng thực thi và kiểm chứng, vẫn là một thách thức nghiên cứu mở.

Vấn đề 3: Bộ chuyển đổi mô hình. Mặc dù nhiều nghiên cứu trong kỹ nghệ phần mềm hướng mô hình đã đề xuất các kỹ thuật chuyển đổi mô hình nhằm tự động hóa quá trình phát triển phần mềm [18], phần lớn các tiếp cận này mới chỉ tập trung vào các phép chuyển đổi riêng lẻ và chưa hình thành được một khung chuyển đổi mô hình hoàn chỉnh đặt mô hình miền làm trung tâm. Bên cạnh đó, vấn đề duy trì tính nhất quán giữa các góc nhìn khác nhau trên cùng một mô hình miền vẫn chưa được xem xét một cách đầy đủ và có hệ thống.

Trong bối cảnh DDD, mô hình miền không chỉ đóng vai trò là công cụ thiết kế, mà còn là nơi hội tụ và chia sẻ tri thức miền giữa nhiều nhóm liên quan, bao gồm nhà phân tích nghiệp vụ, kiến trúc sư, nhà phát triển và các bên liên quan khác. Tuy nhiên, việc chuyển đổi từ mô hình miền sang các hiện thực phần mềm có khả năng thực thi hiện nay vẫn thiếu các cơ chế mang tính hệ thống nhằm bảo toàn ngữ nghĩa nghiệp vụ và đảm bảo tính nhất quán giữa các biểu diễn mô hình ở các mức trừu tượng khác nhau, từ đặc tả yêu cầu, mô hình thiết kế cho đến hiện thực phần mềm [124].

Để giải quyết vấn đề này, luận án đặt ra yêu cầu cần có một tập các bộ chuyển đổi mô hình có vai trò bổ trợ lẫn nhau, nhằm duy trì tính nhất quán giữa các góc nhìn của các bên liên quan trên cùng một mô hình miền. Cụ thể, các bộ chuyển đổi này bao gồm: (i) bộ chuyển đổi sinh tự động các bản mẫu phần mềm từ mô hình miền nhằm cung cấp góc nhìn thực thi; (ii) bộ chuyển đổi từ các đặc tả yêu cầu mức cao (UML/OCL) sang mô hình miền nhằm gắn kết góc nhìn phân tích với mô hình nghiệp vụ; (iii) bộ chuyển

điổi giữa cú pháp trừu tượng và cú pháp cụ thể nhằm đảm bảo tính nhất quán giữa các biểu diễn của ngôn ngữ miền; và (iv) bộ chuyển đổi từ mô hình miền sang không gian ngữ nghĩa hình thức, chẳng hạn Event-B, nhằm hỗ trợ kiểm chứng và xác nhận các thuộc tính ngữ nghĩa quan trọng.

Do đó, nghiên cứu các bộ chuyển đổi mô hình có tính hệ thống, đặt mô hình miền làm trung tâm, bảo toàn ngữ nghĩa và hỗ trợ đồng thời nhiều góc nhìn khác nhau không chỉ là một yêu cầu kỹ thuật, mà còn là một vấn đề nghiên cứu quan trọng nhằm thu hẹp khoảng cách giữa đặc tả nghiệp vụ, mô hình và hiện thực phần mềm trong phát triển phần mềm hướng miền.

Tổng hợp các phân tích trên cho thấy rằng, vẫn còn tồn tại những vấn đề quan trọng chưa được giải quyết một cách hệ thống, cụ thể như sau.

i. Thiếu các kỹ thuật biểu diễn DM tích hợp, giàu ngữ nghĩa và có khả năng thực thi, trong đó các khía cạnh hành vi, bảo mật và ràng buộc được gắn trực tiếp vào DM, đồng thời vẫn bảo đảm vai trò của DM như một ngôn ngữ chung giữa các bên liên quan.

ii. Thiếu một cơ chế hợp nhất các mối quan tâm vào DM hợp nhất, cũng như cơ chế kiểm chứng một cách tổng thể và có hệ thống, nhằm đảm bảo tính mô-đun, tính nhất quán ngữ nghĩa và khả năng sinh mã tự động.

iii. Thiếu các bộ chuyển đổi mô hình hỗ trợ thao tác tự động trên các mô hình miền hợp nhất, chẳng hạn như sinh các bản mẫu phần mềm từ mô hình yêu cầu theo DDD.

1.2 Mục tiêu và phạm vi nghiên cứu

Luận án nghiên cứu và đề xuất các kỹ thuật biểu diễn và chuyển đổi mô hình trong thiết kế hướng miền (DDD), nhằm thu hẹp khoảng cách giữa mô hình miền và hiện thực phần mềm. Luận án hướng tới việc biểu diễn đầy đủ các khía cạnh của miền nghiệp vụ, được cấu thành bởi cấu trúc, hành vi và các chính sách bảo mật, đồng thời nghiên cứu các kỹ thuật sinh tự động phần mềm từ mô hình miền, qua đó đảm bảo ngữ nghĩa của các cấu trúc, hành vi và ràng buộc nghiệp vụ được duy trì nhất quán trong quá trình chuyển đổi từ mô hình miền đến hiện thực phần mềm. Để đạt được mục

tiêu tổng quát này, luận án tập trung vào các mục tiêu nghiên cứu cụ thể sau.

Mục tiêu 1. Nghiên cứu và đề xuất các kỹ thuật biểu diễn mở rộng cho DM, gồm: đặc tả hành vi bằng ngôn ngữ đồ thị hoạt động (*Activity Graph Language - AGL*); đặc tả các chính sách bảo mật dựa trên vai trò bằng DSL là RBACDom (*Domain RBAC Specific Language*); và đặc tả các ràng buộc OCL bằng các mẫu chú thích ràng buộc CAP (*Constraint Annotation Patterns*). Qua đó, hình thành một DM biểu diễn đầy đủ và nhất quán các khía cạnh cấu trúc, hành vi và ràng buộc nghiệp vụ của miền, làm cơ sở cho việc phát triển phần mềm đáp ứng các yêu cầu nghiệp vụ.

Mục tiêu 2. Đề xuất ngôn ngữ mô hình miền hợp nhất UDML (*Unified Domain Model Language*) trong bối cảnh DDD, nhằm tích hợp các mối quan tâm của DM: cấu trúc, hành vi và bảo mật ở mức ngữ nghĩa miền và hành vi. Trên cơ sở đó, luận án đề xuất cơ chế hợp nhất dựa trên chú thích để kết hợp các DSL chuyên biệt vào một mô hình miền thống nhất, đồng thời xây dựng nền tảng ngữ nghĩa phục vụ kiểm chứng có hệ thống và hỗ trợ sinh tự động các mô hình miền có khả năng thực thi, góp phần thu hẹp khoảng cách giữa đặc tả miền và hiện thực triển khai theo tinh thần của DDD.

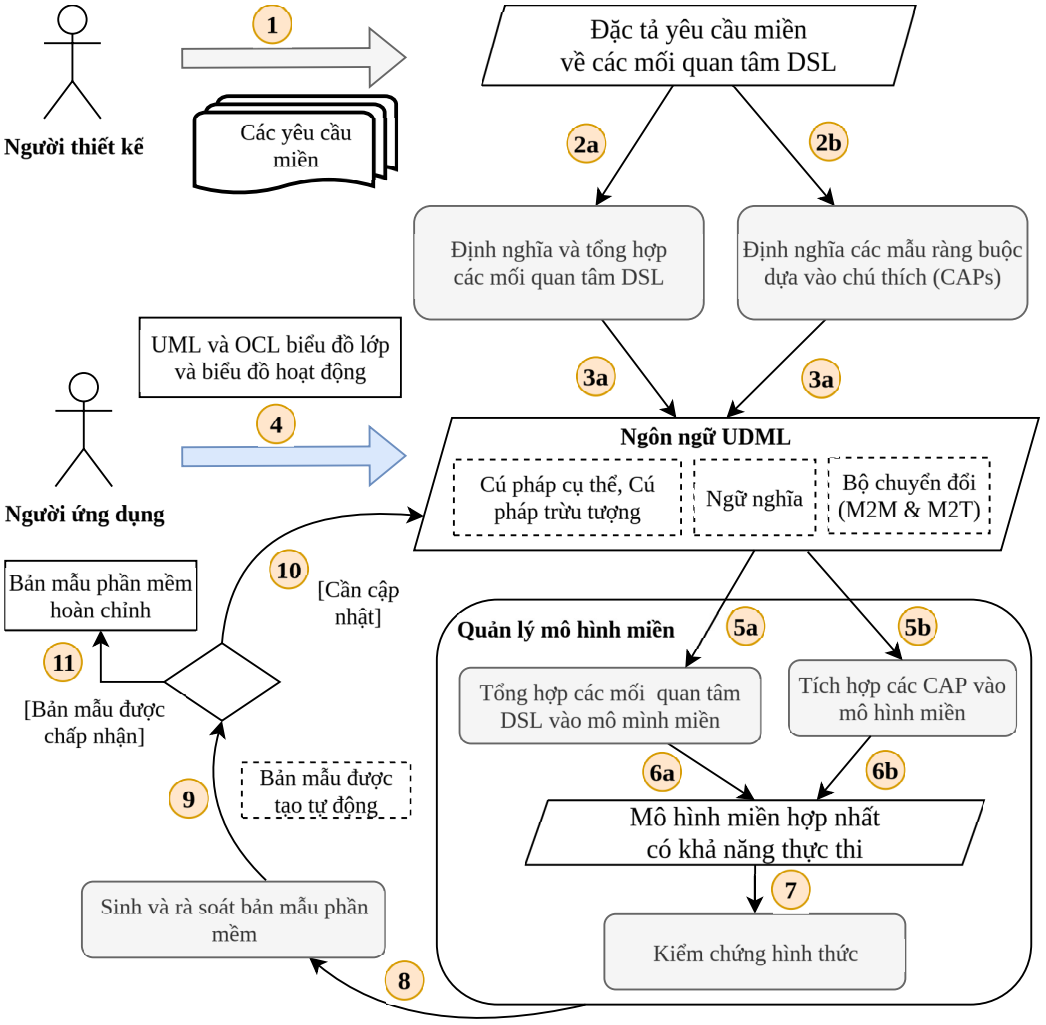
Mục tiêu 3. Nghiên cứu và đề xuất bộ chuyển đổi mô hình nhằm gia tăng mức độ tự động hóa trong phát triển phần mềm, hướng tới việc sinh tự động các bản mẫu phần mềm có khả năng thực thi.

Phạm vi nghiên cứu của luận án tập trung vào các kỹ thuật biểu diễn, hợp nhất và chuyển đổi mô hình dựa trên DSL trong bối cảnh DDD. Luận án kết hợp mô hình hóa bằng DSL với các kỹ thuật thiết kế và kiểm chứng hình thức, đồng thời phát triển các ngôn ngữ và bộ chuyển đổi mô hình theo quy trình lặp lấy mô hình miền làm trung tâm.

Để kiểm chứng tính khả thi và khả năng áp dụng của phương pháp, các đề xuất được đánh giá bằng thực nghiệm trên một tập các ca nghiên cứu điển hình (CourseMan, OrderMan, ProcessMan và OJS). Các ca nghiên cứu này được trình bày và phân tích trong Chương 6 của luận án.

1.3 Nội dung và phương pháp nghiên cứu

Để đạt được các mục tiêu đã nêu, luận án tập trung nghiên cứu các kỹ thuật biểu diễn, hợp nhất và chuyển đổi DM trong bối cảnh DDD. Cụ thể, luận án xem xét việc đặc tả các mối quan tâm khác nhau của miền gồm: cấu trúc, hành vi và bảo mật thông qua các DSL, đồng thời nghiên cứu cơ chế hợp nhất các đặc tả này vào một ngôn ngữ mô hình miền hợp nhất, gọi là UDML. Trên cơ sở đó, luận án đề xuất các cơ chế chuyển đổi mô hình và nền tảng ngữ nghĩa phục vụ kiểm chứng hình thức, hướng tới việc xây dựng các mô hình miền hợp nhất có khả năng thực thi và hỗ trợ sinh tự động các bản mẫu phần mềm. Các thành phần chính của mô hình miền hợp nhất được minh họa trong Hình 1.3 được cấu thành bởi các tác vụ chính, được gán nhãn lần lượt từ *1* đến *11*.



Hình 1.3: Tổng quan phương pháp đề xuất cho kỹ thuật biểu diễn và chuyển đổi mô hình.

Trước hết, người thiết kế ngôn ngữ tiến hành đặc tả các yêu cầu miền xuất phát từ nhiều miền ứng dụng khác nhau, bao gồm các mối quan tâm được đặc tả bằng DSL và các nhóm ràng buộc OCL, như minh họa trong *nhãn 1*. Bước này nhằm xác định và khái quát hóa các mối quan tâm miền liên quan đến cấu trúc, hành vi, bảo mật, cũng như các nhóm ràng buộc OCL tương ứng. Trên cơ sở đó, các mối quan tâm DSL được định nghĩa và hợp nhất, như thể hiện trong *nhãn 2a*, bao gồm các thành phần cú pháp trừu tượng và các dạng cú pháp cụ thể (văn bản hoặc đồ họa) để hỗ trợ xây dựng các mô hình chi tiết cho từng mối quan tâm cụ thể. Song song với đó, một danh mục các mẫu chú thích ràng buộc CAP được xây dựng bằng cách khai thác và khái quát hóa các ràng buộc từ yêu cầu miền, như minh họa trong *nhãn 2b*. Mỗi CAP được định nghĩa bao gồm: (i) một biểu đồ lớp UML mô tả các khái niệm miền và mối quan hệ giữa chúng; (ii) một đặc tả OCL biểu diễn các luật nghiệp vụ và các bất biến; và (iii) tập các chú thích dùng để gắn các ràng buộc OCL vào DM.

Tiếp đó, với ngôn ngữ UDML đã được thiết kế đầy đủ về cú pháp, ngữ nghĩa, cùng với các bộ chuyển đổi M2M và M2T, người dùng tập trung áp dụng ngôn ngữ này để biểu diễn mô hình miền cho hệ thống cụ thể. Các đặc tả DSL theo mối quan tâm chuyên biệt (*nhãn 3a*) và các mẫu CAP (*nhãn 3b*), được xây dựng trong giai đoạn thiết kế bởi chuyên gia miền, được sử dụng kết hợp với các DM cụ thể dưới dạng biểu đồ lớp và biểu đồ hoạt động UML/OCL, như minh họa trong *nhãn 4*. Trên cơ sở đó, các mối quan tâm DSL được tổng hợp và các CAP được tích hợp vào DM, như thể hiện trong *nhãn 5a* và *nhãn 5b*, nhằm suy dẫn ra một mô hình miền hợp nhất làm nền tảng cho việc sinh bản mẫu phần mềm. UDML được thiết kế với một mô hình lõi cho phép tích hợp một cách thống nhất các mối quan tâm DSL vào DM, như minh họa trong *nhãn 6a* và *nhãn 6b*. Kết quả thu được là một mô hình miền hợp nhất có khả năng thực thi, làm cơ sở cho bước kiểm chứng hình thức mô hình miền hợp nhất trước khi sinh bản mẫu phần mềm, như thể hiện trong *nhãn 7*.

Cuối cùng, mô hình UDML đã được tích hợp đầy đủ đóng vai trò làm nền tảng để sinh tự động bản mẫu phần mềm, như thể hiện trong *nhãn 8*. Bản mẫu phần mềm này được tạo dựa trên khung JDA [74] và sau đó được các chuyên gia miền đánh giá để thu thập phản hồi, như minh họa trong *nhãn 9*. Trên cơ sở phản hồi thu được, cả mô hình UDML và các đặc tả

mối quan tâm liên quan sẽ được cập nhật tương ứng, như thể hiện trong *nhãn 10*. Quy trình này được lặp lại một cách có hệ thống, hỗ trợ chu trình cải tiến liên tục cho đến khi hệ thống phần mềm đáp ứng đầy đủ các yêu cầu đã đặt ra, như minh họa trong *nhãn 11*.

1.4 Các đóng góp chính của luận án

Luận án đề xuất các kỹ thuật biểu diễn và chuyển đổi mô hình cho DDD, tập trung vào việc biểu diễn và tích hợp các mối quan tâm vào mô hình miền hợp nhất, hỗ trợ kiểm chứng và hướng thực thi, đồng thời sinh tự động các bản mẫu phần mềm sử dụng chuyển đổi mô hình, cụ thể như sau.

i. Đề xuất các kỹ thuật biểu diễn mô hình miền (DM) trong bối cảnh DDD, cho phép đặc tả một cách tường minh và nhất quán các khía cạnh cấu trúc, hành vi và ràng buộc nghiệp vụ của các miền nghiệp vụ. Các kỹ thuật này mở rộng khả năng biểu đạt của DM, làm cơ sở cho các bước hợp nhất, kiểm chứng và chuyển đổi mô hình tiếp theo.

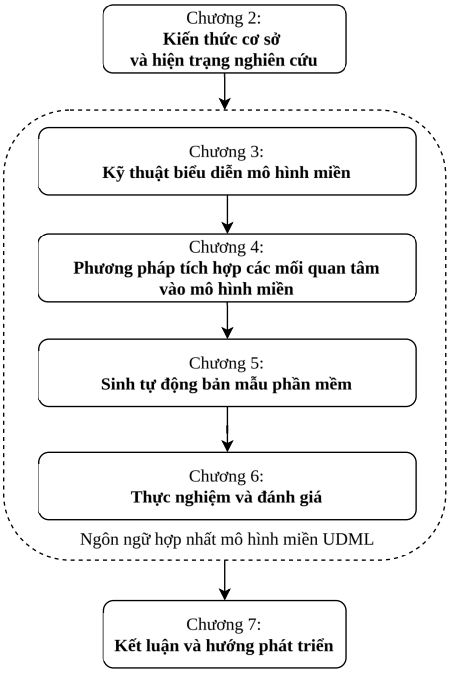
ii. Đề xuất một kỹ thuật hợp nhất các DSL theo các mối quan tâm chuyên biệt vào một mô hình miền hợp nhất có khả năng thực thi. Trên cơ sở đó, luận án xây dựng nền tảng ngữ nghĩa phục vụ kiểm chứng hình thức, nhằm đảm bảo tính nhất quán ngữ nghĩa của mô hình miền hợp nhất ở giai đoạn thiết kế.

iii. Đề xuất các kỹ thuật chuyển đổi mô hình dựa trên DM, cho phép sinh tự động các bản mẫu phần mềm có khả năng thực thi, đồng thời bảo toàn ngữ nghĩa miền và chất lượng chuyển đổi, phù hợp với các kịch bản sử dụng cụ thể trong các miền ứng dụng chuyên biệt.

Để kiểm chứng và đánh giá các kỹ thuật đề xuất, luận án đã hiện thực hóa một bộ công cụ hỗ trợ thực nghiệm, cho phép áp dụng các kỹ thuật biểu diễn, tích hợp và chuyển đổi mô hình trên các ca nghiên cứu thực tế. Bộ công cụ đóng vai trò nền tảng đánh giá, cung cấp bằng chứng thực nghiệm cho tính khả thi và hiệu quả của các đóng góp khoa học trong luận án.

1.5 Cấu trúc luận án

Luận án *"Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền"* được tổ chức thành bảy chương. Trong đó, *Chương 1 – Giới thiệu* trình bày lý do chọn đề tài, mục tiêu, nội dung, đối tượng nghiên cứu và các đóng góp chính của luận án. Cấu trúc tổng thể của luận án được minh họa trong Hình 1.4, với nội dung các chương được tổ chức như sau.



Hình 1.4: Cấu trúc luận án.

Chương 2 trình bày cơ sở lý thuyết và tổng quan nghiên cứu liên quan đến đề tài. Chương giới thiệu các kỹ thuật biểu diễn mô hình miền trong DDD dựa trên DSL (nội sinh và ngoại sinh), đồng thời phân tích vai trò của các mối quan tâm như hành vi, ràng buộc và bảo mật trong mô hình miền. Bên cạnh đó, chương trình bày các kỹ thuật chuyển đổi mô hình và sinh tự động phần mềm trong kỹ nghệ phần mềm hướng mô hình. Trên cơ sở đó, chương đánh giá các hạn chế của các tiếp cận hiện tại, đặc biệt trong việc tích hợp các mối quan tâm và bảo toàn ngữ nghĩa, qua đó xác định khoảng trống nghiên cứu cho luận án.

Chương 3 trình bày nghiên cứu về kỹ thuật tích hợp các khía cạnh hành vi và ràng buộc vào mô hình miền (DM) nhằm hỗ trợ sinh tự động phần mềm theo DDD. Cụ thể, chương đề xuất mẫu *Constraint Annotation Pattern* (CAP) để tích hợp các ràng buộc phức tạp vào DM và DSL dựa trên chú thích *Activity Graph Language* (AGL) để biểu diễn các hành vi miền có khả năng thực thi.

Chương 4 trình bày phương pháp tích hợp các mối quan tâm vào mô hình miền hợp nhất có khả năng thực thi trong DDD. Chương đề xuất cơ chế hợp nhất các DSL theo mối quan tâm sử dụng tiếp cận dựa trên chú thích ở mức cây cú pháp trừu tượng, cho phép tích hợp các đặc tả hành vi, ràng buộc và bảo mật vào mô hình miền một cách nhất quán. Trên cơ

sở đó, chương giới thiệu DSL bảo mật RBACDom như một mối quan tâm cụ thể và trình bày cách tích hợp với mô hình hành vi. Đồng thời, chương đề xuất nền tảng ngữ nghĩa và phương pháp kiểm chứng hình thức cho mô hình miền hợp nhất, nhằm đảm bảo tính nhất quán ngữ nghĩa giữa các mối quan tâm.

Chương 5 trình bày kỹ thuật sinh tự động bản mẫu phần mềm thông qua hai bộ chuyển đổi mô hình: RM2UDM, chuyển đổi mô hình yêu cầu (*Requirement Model* – RM) sang mô hình miền hợp nhất (*Unified Domain Model* – UDM), và UDM2AGL nhằm sinh đặc tả miền thực thi. Hai bộ chuyển đổi này hỗ trợ tự động hóa chuỗi phát triển từ đặc tả yêu cầu đến phần mềm có khả năng thực thi.

Chương 6 trình bày công cụ hỗ trợ và đánh giá các phương pháp đề xuất bằng các ca nghiên cứu điển hình, gồm CourseMan, OrderMan, ProcessMan và OJS. Các kết quả thực nghiệm và thảo luận trong chương này được sử dụng để đánh giá tính khả thi và hiệu quả của phương pháp đề xuất trong các bối cảnh ứng dụng thực tế.

Cuối cùng, *Chương 7* tổng kết các đóng góp chính của luận án và thảo luận các hướng nghiên cứu tiếp theo dựa trên các kết quả đã đạt được.

Chương 2

CƠ SỞ LÝ THUYẾT VÀ TỔNG QUAN TÌNH HÌNH NGHIÊN CỨU

Trong chương này, luận án trình bày các kiến thức nền tảng liên quan đến biểu diễn mô hình miền, tích hợp các mối quan tâm và chuyển đổi mô hình trong thiết kế hướng miền. Nội dung chương nhằm làm rõ hiện trạng nghiên cứu của các tiếp cận hiện có trong việc đặc tả đầy đủ các khía cạnh cấu trúc, hành vi và ràng buộc miền, cũng như trong việc hợp nhất các mối quan tâm miền và bảo toàn ngữ nghĩa khi sinh phần mềm. Trên cơ sở đó, chương thiết lập nền tảng khái niệm và ngữ nghĩa cho việc đề xuất các kỹ thuật biểu diễn mô hình miền giàu ngữ nghĩa, hợp nhất các mối quan tâm miền vào một mô hình miền hợp nhất có khả năng thực thi, và hỗ trợ chuyển đổi mô hình để sinh tự động các bản mẫu phần mềm trong các chương tiếp theo.

2.1 Tổng quan về thiết kế hướng miền

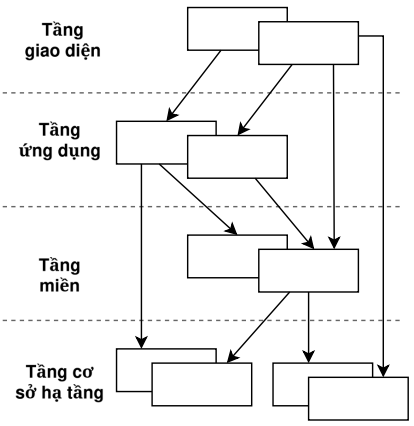
Trong mục này, luận án trình bày các khái niệm nền tảng của thiết kế hướng miền làm cơ sở lý thuyết cho các kỹ thuật biểu diễn được đề xuất trong các chương sau, bao gồm AGL, CAP, RBACDom và UDML.

2.1.1 Nền tảng thiết kế hướng miền

Thiết kế hướng miền (*Domain-Driven Design - DDD*) [41] hướng tới việc phát triển phần mềm theo cách lặp và tiến hóa, dựa trên mô hình miền

phản ánh trung thực miền ứng dụng; Mô hình miền (*Domain Model - DM*) này vừa phải nắm bắt đầy đủ các yêu cầu nghiệp vụ, vừa phải khả thi về mặt kỹ thuật để hiện thực hóa trong triển khai. Theo Evans [41], các ngôn ngữ lập trình hướng đối tượng (*Object-Oriented Programming Language - OOPL*) có sự phù hợp tự nhiên khi áp dụng DDD. Trước đó, các nghiên cứu gần đây về DM đã chỉ ra rằng DM cần đảm bảo cả tính biểu đạt và khả thi thực thi [16, 36]. Thứ nhất, các đối tượng trong ngôn ngữ hướng đối tượng phản ánh trực tiếp các thực thể tồn tại trong miền thực tế. Thứ hai, cấu trúc đối tượng cũng là cơ sở cho các ngôn ngữ mô hình hóa trừu tượng được sử dụng trong phân tích và thiết kế, qua đó hỗ trợ khái niệm hóa và hiện thực hóa miền ứng dụng [34]. Trong bối cảnh này, DDD được xem như một phương pháp cụ thể hóa việc phát triển phần mềm hướng đối tượng dựa trên DM.

Trong phát triển phần mềm hiện đại, đặc biệt với các hệ thống nghiệp vụ phức tạp, người ta ngày càng nhận ra rằng vấn đề chính không nằm ở công nghệ, mà nằm ở chỗ hiểu và nắm bắt đúng miền nghiệp vụ. Trong phương pháp DDD, DM đóng vai trò trung tâm của phần mềm nơi chứa đựng phần lớn độ phức tạp. Nơi các khái niệm nghiệp vụ, hành vi và quy tắc được mô hình hóa thống nhất, và dùng một ngôn ngữ chung để tất cả các bên tham gia cùng trao đổi. Evans nhấn mạnh rằng DM không chỉ là tài liệu phân tích mà phải gắn chặt với cài đặt, chi phối việc phát triển phần mềm trong suốt vòng đời. Điều này bao hàm hai yêu cầu: (i) DM phải giàu ngữ nghĩa để phản ánh đúng bản chất nghiệp vụ, và (ii) DM phải đủ cấu trúc để định hướng mã nguồn triển khai. Mô hình miền (DM) là mô hình mô tả cấu trúc các khái niệm, mối quan hệ giữa chúng trong một miền vấn đề nào đó. Mỗi miền nghiệp vụ đều có thể được biểu diễn bởi một DM; ví dụ DM cho quản lý đào tạo, giao hàng, ngân hàng, v.v.



Hình 2.1: Kiến trúc phân tầng chung cho DDD (Nguồn [41]).

Hình 2.1 mô tả kiến trúc DDD phân tầng, được trích từ Evans, trong đó DM nằm ở tầng miền và giữ vai trò trung tâm. Kiến trúc gồm bốn tầng.

Tầng trên cùng là tầng giao diện (*User Interface - UI*), cung cấp các thành phần tương tác với người dùng. Tiếp theo là tầng ứng dụng, nơi chứa các dịch vụ của hệ thống và điều phối hoạt động của các thành phần miền. Tầng miền chứa DM, biểu diễn lô-gic nghiệp vụ cốt lõi của hệ thống và đòi hỏi sự hiểu biết sâu sắc về miền vấn đề. Tầng dưới cùng là tầng cơ sở hạ tầng, chịu trách nhiệm tương tác với nền tảng công nghệ, giúp các tầng trên hoạt động độc lập với công nghệ cụ thể và tăng khả năng tái sử dụng phần mềm.

Trong DDD, hai đặc trưng quan trọng được nhấn mạnh gồm: (i) tính khả thi, tức DM phải có khả năng trở thành mã nguồn và ngược lại; và (ii) tính thỏa mãn, tức DM phải phản ánh đúng các yêu cầu nghiệp vụ bằng một ngôn ngữ chung được phát triển và tinh chỉnh trong quá trình làm việc với các bên liên quan [41]. Ngôn ngữ này cho phép các yêu cầu miền được diễn đạt chính xác và nhất quán, đồng thời đảm bảo mô hình luôn bám sát nhu cầu nghiệp vụ. Việc đạt được hai đặc trưng cốt lõi này hiện đang là một trong những trọng tâm của các nghiên cứu về DDD.

Bên cạnh đó, DDD còn dựa trên hai nguyên lý nền tảng giúp duy trì sự tương thích giữa mô hình và hiện thực là ngữ cảnh giới hạn và ngôn ngữ chung (UL). Ngữ cảnh giới hạn xác định phạm vi mà các khái niệm miền được diễn giải, ngăn ngừa sự pha tạp ngữ nghĩa giữa các vùng nghiệp vụ, đồng thời bảo đảm mọi kỹ thuật mô hình hóa được triển khai trong một không gian ngữ nghĩa thống nhất. Mặt khác, UL đóng vai trò là phương tiện ngôn ngữ xuyên suốt giữa DM, ngôn ngữ chuyên biệt miền (*Domain-Specific Language - DSL*) [46] và mã nguồn, giúp loại bỏ sự chênh lệch giữa tài liệu phân tích và mã chương trình, vốn là nguyên nhân sâu xa của khoảng cách giữa mô hình và cài đặt. DSL giúp liên kết chặt chẽ giữa mô hình thiết kế và hiện thực, từ đó cải thiện tính đúng đắn, khả năng bảo trì, và tính mở rộng của phần mềm [23, 45, 97, 120].

Evans nhấn mạnh rằng DM cần được hiện thực hóa trực tiếp trong mã nguồn thay vì chỉ tồn tại dưới dạng tài liệu. Tuy nhiên, trong thực tiễn, DM thường được mô tả bằng UML/OCL [94], tài liệu văn bản hoặc trao đổi bằng lời nói, dẫn đến khoảng cách giữa mô hình và hiện thực triển khai. Mặc dù DDD yêu cầu sử dụng UL xuyên suốt từ phân tích đến cài đặt, phương pháp này chưa cung cấp cơ chế kỹ thuật để hình thức hóa UL và chuyển hóa một cách hệ thống vào mã nguồn. Do đó, UL dễ bị sai lệch giữa các giai đoạn phân tích, thiết kế và cài đặt [62].

2.1.2 Mô hình miền hướng thực thi

Mục này trình bày các tiếp cận biểu diễn phổ biến của DM, nhấn mạnh ưu điểm về khả năng biểu đạt và chuẩn hóa, đồng thời chỉ ra hạn chế về tính hợp nhất và khả năng hướng thực thi của các biểu diễn hiện hành trong bối cảnh DDD.

Trong nhiều hệ thống phần mềm hiện nay, DM chủ yếu đóng vai trò mô tả, chưa có khả năng thực thi trực tiếp. Tuy nhiên, nghiên cứu [38] đã nhấn mạnh rằng DM cần được **kích hoạt** nhằm hỗ trợ sinh tự động bản mẫu phần mềm hoặc thậm chí có thể được thực thi trực tiếp.

Các hướng tiếp cận nghiên cứu liên quan đến biểu diễn và thực thi hành vi miền hiện nay tập trung giải quyết một số thách thức chính sau:

- Các nghiên cứu về DDD chủ yếu tập trung vào khía cạnh cấu trúc của DM bao gồm: thực thể, tổng hợp, kho lưu trữ và các khối xây dựng tương tự–trong khi hành vi miền thường bị tách ra khỏi mô hình. Trong nhiều công trình, hành vi được hiện thực trong mã nguồn thông qua các dịch vụ miền hoặc các cơ chế sự kiện, hoặc được mô tả bằng các biểu đồ UML tách rời [62, 125, 137, 138].

- Một số hướng nghiên cứu trong kỹ nghệ hướng mô hình (*Model-Driven Engineering – MDE*) cho phép gắn các mô hình hành vi như máy trạng thái hoặc biểu đồ hoạt động vào mô hình, tuy nhiên các tiếp cận này thường nằm ngoài ngữ cảnh của DDD và chưa đồng nhất hóa hành vi với cấu trúc trong một DM duy nhất [15, 18, 80, 123, 134].

- Nhiều công trình trong nhóm này tập trung vào việc định nghĩa các phép ánh xạ từ biểu đồ hoạt động, trạng thái hoặc tuần tự của UML sang mã thực thi, chủ yếu dựa trên các quy tắc kỹ thuật ở mức điều khiển luồng [78, 127].

- Bài toán *thu hẹp khoảng cách giữa yêu cầu nghiệp vụ và mô hình hóa hướng đối tượng* luôn được xem là trọng tâm. Tuy nhiên, phần lớn các tiếp cận hiện nay mới dừng lại ở mức khái niệm nhằm hỗ trợ mô hình hóa đúng tư duy miền, trong khi việc ánh xạ có hệ thống hành vi miền sang mã nguồn thường được giao cho các khung làm việc hoặc các quy ước lập trình, thiếu một ngữ nghĩa hình thức làm cơ sở [138].

- Song song đó, một số nghiên cứu trong MDE và lĩnh vực sinh mã nguồn từ mô hình hành vi đã đề xuất các bộ chuyển đổi từ mô hình hành vi sang mã nguồn [111]. Tuy nhiên, các tiếp cận này thường vận hành độc lập với kiến trúc hướng miền và chưa được tích hợp với một DM cụ thể hay một ngôn ngữ miền thống nhất.

- Nhiều công trình khác tập trung vào sinh mã tự động từ các mô hình hành vi UML với mục tiêu đảm bảo tính nhất quán giữa thiết kế hành vi và mã nguồn thông qua các phép ánh xạ mang tính cơ học. Mã sinh ra chủ yếu là mã điều khiển tổng quát, chưa được cố kết trong một khung làm việc hướng miền rõ ràng [127].

- Ngoài ra, một số nghiên cứu ưu tiên các tiêu chí như an toàn, độ tin cậy hoặc hiệu năng. Mặc dù có giá trị trong lĩnh vực sinh mã nguồn, các tiếp cận này không đặt trọng tâm vào các nguyên lý cốt lõi của DDD, chẳng hạn như UL hay khả năng để chuyên gia miền trực tiếp hiểu và xác nhận mô hình hành vi.

Trong bối cảnh phát triển phần mềm theo DDD, một số công trình như Apache Isis [122] và OpenXava [98] đã khai thác các ngôn ngữ chuyên biệt miền dựa trên chú thích (*annotation-based Domain-Specific Languages – aDSL*). Đây là một dạng DSL nội sinh được nhúng trong các OOPL như Java, tận dụng cú pháp và ngữ nghĩa của ngôn ngữ chủ để biểu đạt và hiện thực hóa DM.

Từ cách biểu diễn này, DM thực thi có thể được xây dựng theo hai hướng. (i) Hướng trực tiếp nhúng các mối quan tâm và hiện thực chương trình vào OOPL như Java [49] hoặc C# [57], giúp tạo ra phần mềm chạy được nhưng dễ làm DM bị pha tạp; (ii) Hướng gián tiếp dựa trên phát triển hướng mô hình, trong đó DM được tích hợp với các mối quan tâm khác thông qua UML/OCL hoặc các DSL bổ trợ và được chuyển đổi thành chương trình bằng các kỹ thuật M2M và M2T. Tiêu biểu cho hướng này là cách biểu diễn DM trực tiếp trên các lớp miền, trong đó các ràng buộc thiết yếu được đặc tả ngay trong ngữ cảnh của DM, giúp duy trì tính biểu đạt của UL đồng thời gắn kết với cấu trúc và ngữ nghĩa thực thi [73, 74].

Do đó, việc kế thừa và mở rộng hướng tiếp cận này bằng một phiên bản tinh chỉnh của phương pháp phát triển phần mềm dựa trên aDSL của

DDD [73], theo hướng mở rộng biểu diễn mô hình miền cho hành vi nghiệp vụ, bảo mật và các ràng buộc nghiệp vụ ngoài những nghiệp vụ thiết yếu 2.1, nhằm tăng cường khả năng mô hình hóa và hỗ trợ sinh mã nhất quán, vẫn còn là một khoảng trống trong nghiên cứu.

2.1.3 Ngôn ngữ chuyên biệt miền DSL

Ngôn ngữ chuyên biệt miền (*Domain-Specific Language - DSL*) [46] là một ngôn ngữ được thiết kế để mô tả và thao tác trong một miền bài toán cụ thể, DSL được xây dựng với mục tiêu làm cho các khái niệm, quy tắc và thao tác của miền trở nên *tự nhiên* và *gần gũi* nhất với cách mà chuyên gia nghiệp vụ suy nghĩ và trao đổi, từ đó tăng khả năng biểu đạt, giảm khoảng cách giữa đặc tả và hiện thực. Trong DDD, DSL đóng vai trò như một ngôn ngữ hình thức, hỗ trợ mô hình hóa miền, sinh mã tự động và kiểm soát sự tiến hóa của hệ thống.

DSL ngoại sinh là các ngôn ngữ chuyên biệt miền tồn tại độc lập với ngôn ngữ lập trình chủ, có cú pháp và ngữ nghĩa được xác định tường minh cùng hệ sinh thái công cụ riêng. Nhờ đó, DSL ngoại sinh phù hợp cho phân tích mô hình ở mức trừu tượng cao, chuyển đổi mô hình giữa nhiều DSL và kiểm chứng ngữ nghĩa một cách có hệ thống [18].

DSL nội sinh được nhúng trong ngôn ngữ lập trình chủ và khai thác trực tiếp cú pháp, ngữ nghĩa của ngôn ngữ này để biểu đạt các khái niệm miền. Cách tiếp cận này giúp DSL nội sinh dễ tích hợp với mã nguồn và không yêu cầu xây dựng một ngữ nghĩa hình thức độc lập [14, 33, 135]. Các DSL nội sinh, đặc biệt là các tiếp cận dựa trên chú thích (aDSL), cho phép nhúng trực tiếp các đặc tả miền vào ngôn ngữ lập trình chủ như Java hoặc C# [122, 135]. Trong bối cảnh DDD, aDSL có vai trò quan trọng nhờ các đặc điểm sau:

- Gắn ngữ nghĩa miền trực tiếp vào lớp miền, trong đó mỗi chú thích vừa thể hiện ý nghĩa nghiệp vụ, vừa cung cấp thông tin cho các công cụ chuyển đổi mô hình và sinh mã.
- Thu hẹp khoảng cách giữa mô hình và mã nguồn, khi DM được thể hiện trực tiếp trong mã thay vì tồn tại tách rời dưới dạng UML/OCL, phù hợp với nguyên lý gắn kết mô hình và hiện thực của Evans.

- Hỗ trợ sinh tự động bản mẫu phần mềm, cho phép các công cụ như JDA (*Java Domain Application framework*) [74] khai thác trực tiếp chú thích để tạo các thành phần phần mềm mà không cần lớp mô hình trung gian.

Các tiếp cận này giúp rút ngắn khoảng cách giữa mô hình và triển khai, đồng thời hỗ trợ xây dựng các DM có khả năng thực thi ở một mức độ nhất định. Tuy nhiên, việc tích hợp đồng thời hành vi, ràng buộc và các chính sách miền trong một mô hình thống nhất vẫn còn nhiều hạn chế [73].

Trong DDD, UL không chỉ là ngôn ngữ giao tiếp mà còn phải được nhúng trực tiếp vào mô hình và mã nguồn, và DSL chính là phương tiện hiện thực hóa yêu cầu này: từ vựng của DSL như tên chú thích, thuộc tính, hành vi được thiết kế trùng khớp với thuật ngữ nghiệp vụ, trong khi siêu mô hình và các quy tắc DSL mô tả chính xác cấu trúc và lô-gic của miền. Khi các bộ chuyển đổi và sinh mã nguồn được áp dụng, mọi tạo tác phần mềm sinh ra đều tuân thủ cùng một UL, giúp chuyên gia nghiệp vụ có thể đọc và xác nhận DM ngay trong DSL mà không cần phụ thuộc vào UML/OCL.

Ví dụ. Đặc tả 2.1 thể hiện một aDSL của DCSL được sử dụng cho lớp `Student`. Chú thích `@DAttr(id = true, auto = true)` xác định `id` là khóa định danh của thực thể, với giá trị được tự động sinh. Chú thích `@DAttr(optional = false, length = 50)` quy định `name` là thuộc tính bắt buộc và áp đặt ràng buộc độ dài tối đa của giá trị.

```
1    public class Student {
2        @DAttr(id = true, auto = true)
3        private int id;
4        @DAttr(optional = false, length = 50)
5        private String name;
6        ...
7    }
```

Đặc tả 2.1: aDSL biểu diễn DCSL cho lớp `Student`

2.1.4 DCSL và kiến trúc JDA

DCSL (*Domain-Class Specification Language - DCSL*) là một ngôn ngữ miền chuyên biệt miền dựa trên chú thích, được giới thiệu trong [73], dùng để đặc tả DM. Ngôn ngữ DCSL này cho phép mô tả ngắn gọn và dễ đọc các khái niệm miền, ràng buộc và quy tắc nghiệp vụ, với cú pháp gần giống cú

pháp của OOPL. DCSL hỗ trợ biểu diễn các ràng buộc OCL thiết yếu [93]. Cụ thể, các chú thích được sử dụng để ghi nhận và mô tả các ràng buộc cốt lõi trong DM, như minh họa ở Bảng 2.1.

Bảng 2.1: Các ràng buộc cấu trúc thiết yếu cho DM

Ràng buộc	Loại	Mô tả
Tính bất biến của đối tượng	Boolean	Đối tượng của một lớp có thể thay đổi hay không [77].
Tính bất biến của trường	Boolean	Một trường có thể thay đổi hay không (tức là giá trị của nó có thể bị thay đổi) [59].
Tính tùy chọn của trường	Boolean	Một trường có tùy chọn hay không (tức là giá trị của nó không cần được khởi tạo khi một đối tượng được tạo) [59].
Tính duy nhất của trường	Boolean	Các giá trị của một trường có duy nhất hay không [59].
Trường Id	Boolean	Một trường có phải là trường định danh của đối tượng hay không [59].
Trường tự động	Boolean	Giá trị của một trường có được tạo tự động bởi hệ thống hay không [59].
Độ dài của trường	Non-Boolean	Độ dài tối đa (nếu có) của một trường (tức là các giá trị của trường không được vượt quá độ dài này) [59].
Giá trị tối thiểu của trường	Non-Boolean	Giá trị tối thiểu (nếu có) của một trường (tức là các giá trị của trường không được thấp hơn giá trị này) [59].
Giá trị tối đa của trường	Non-Boolean	Giá trị tối đa (nếu có) của một trường (tức là các giá trị của trường không được cao hơn giá trị này) [59].
Số lượng đối tượng liên kết tối thiểu	Non-Boolean	Số lượng đối tượng tối thiểu mà mỗi đối tượng của một lớp có thể được liên kết [94].
Số lượng đối tượng liên kết tối đa	Non-Boolean	Số lượng đối tượng tối đa mà mỗi đối tượng của một lớp có thể được liên kết [94].

Trong ngôn ngữ DCSL được định nghĩa bởi các siêu khái niệm của nó bao gồm các cấu trúc cốt lõi của OOPL cùng các thành phần liên quan đến ràng buộc. Các siêu khái niệm chính bao gồm: lớp miền (`DClass`), thuộc

tính miền (`DAttr`), thuộc tính liên kết (`DAssoc`), và phương thức miền (`DOpt`). Các siêu khái niệm còn lại, cùng với các thuộc tính của tất cả các khái niệm, được định nghĩa nhằm hỗ trợ biểu diễn các ràng buộc OCL thiết yếu được tóm tắt trong Bảng 2.1.

Việc áp dụng kiến trúc mô hình (*Model–View–Controller - MVC*) trong phát triển phần mềm thực tiễn là điều phổ biến, đặc biệt đối với các hệ thống cần giao diện người dùng đồ họa để hỗ trợ nhóm phát triển. Nguyên nhân là bởi người ta tin rằng quá trình xây dựng phần mềm không thể được tự động hóa hoàn toàn do sự tham gia của yếu tố con người trong vòng đời phát triển [47]. Kiến trúc MVC bao gồm ba thành phần: *model*, *view* và *controller*, trong đó mỗi thành phần có thiết kế nội tại độc lập và ảnh hưởng tối thiểu đến hai thành phần còn lại. Tính mô-đun có thể được tăng cường hơn nữa bằng cách áp dụng kiến trúc này ở cấp mô-đun, hình thành nên một kiến trúc thiết kế phân cấp trong đó phần mềm được cấu thành từ nhiều mô-đun theo cấu trúc cây.

Để xây dựng phần mềm theo DDD từ DM, cần một mô hình kiến trúc tuân theo kiến trúc phân lớp tổng quát [41, 124], trong đó DM được đặt ở lớp lõi và tách biệt với giao diện người dùng cũng như các lớp còn lại, kiến trúc JDA [74] được đề xuất để giải quyết vấn đề này. Ngoài ra, các khung làm việc DDD hiện có [98, 122] cũng sử dụng kiến trúc MVC. Giao diện người dùng đóng vai trò quan trọng trong việc trình bày DM cho các bên liên quan và hỗ trợ họ xây dựng mô hình hiệu quả. Do đó, các công cụ DDD tuân theo kiến trúc phân lớp thường lựa chọn kiến trúc MVC như một khuôn mẫu triển khai phù hợp, nhằm hỗ trợ trình bày và tương tác hiệu quả với DM.

2.1.5 Phương pháp biểu diễn bằng siêu mô hình hóa

Trong mô hình hóa hướng miền, DSL đóng vai trò là phương tiện cốt lõi để biểu diễn DM một cách chính xác, nhất quán và có thể thực thi. Mỗi DSL được xây dựng dựa trên một siêu mô hình nhằm mô tả rõ ràng các khía cạnh cấu trúc, ràng buộc và ngữ nghĩa của ngôn ngữ. Việc lựa chọn MOF/Ecore [31, 82] làm chuẩn khung siêu mô hình hóa bảo đảm rằng DSL có nền tảng hình thức đủ mạnh để kiểm chứng, phân tích và tham gia vào chuyển đổi mô hình theo chuẩn MDE [18]. Siêu mô hình xác định các phần

tử ngôn ngữ, quan hệ giữa chúng và các ràng buộc cần tuân thủ, từ đó cung cấp một cấu trúc thống nhất cho việc định nghĩa, mở rộng và tạo nền tảng thống nhất để định nghĩa, mở rộng và tích hợp các DSL trong kiến trúc mô hình.

Từ góc nhìn của DDD, MOF/Ecore cung cấp cơ chế cho phép các khái niệm miền bao gồm đối tượng, quan hệ, hành vi, ràng buộc được định nghĩa một cách chặt chẽ và ánh xạ trực tiếp vào DSL tương ứng, mang lại ba lợi ích sau:

- Thứ nhất, hỗ trợ bảo toàn ngữ nghĩa trong quá trình chuyển đổi mô hình, vì mọi chuyển đổi mô hình đều vận hành trên cấu trúc được đặc tả rõ ràng, tránh sai lệch khi ánh xạ từ mô hình này sang mô hình khác.
- Thứ hai, siêu mô hình cho phép kiểm tra tính đúng đắn và tính nhất quán của DM ở mức cú pháp và ngữ nghĩa
- Cuối cùng, siêu mô hình chính là cầu nối giúp DM chuyển hóa thành mô hình thực thi thông qua các bộ chuyển đổi mô hình và trở thành một phần của hệ thống chạy được.

Vì vậy, DSL và siêu mô hình hóa không chỉ là công nghệ hỗ trợ mô hình hóa mà còn là nền tảng phương pháp luận để bảo đảm DM giữ vai trò trung tâm, có thể diễn đạt chính xác kiến thức miền và được chuyển hóa một cách tin cậy thành phần mềm theo đúng tinh thần của DDD.

2.2 Các hướng tiếp cận biểu diễn mô hình miền

Trong thực tiễn, mô hình miền (DM) có thể được biểu diễn bằng các ngôn ngữ mô hình hóa tổng quát như UML/OCL, hoặc bằng các DSL theo các mức độ gắn kết khác nhau với mã nguồn. Mục này trình bày các tiếp cận biểu diễn phổ biến, nhấn mạnh ưu điểm về khả năng biểu đạt và chuẩn hóa, đồng thời chỉ ra hạn chế về tính hợp nhất và khả năng hướng thực thi của các biểu diễn hiện hành trong bối cảnh DDD.

2.2.1 Biểu diễn các ràng buộc trên mô hình miền

UML và OCL là các công cụ phổ biến để biểu diễn cấu trúc và ràng buộc của DM trong DDD. UML cho phép mô tả các khái niệm miền và mối quan hệ giữa chúng, trong khi OCL cung cấp cơ chế đặc tả chính xác các bất biến và điều kiện nghiệp vụ [94]. Tuy nhiên, các đặc tả này thường tồn tại tách rời khỏi mã nguồn và thiếu khả năng hỗ trợ trực tiếp cho việc thực thi DM.

Trong bối cảnh DDD, biểu đồ lớp UML thường được dùng để phác thảo DM ở mức khái niệm, giúp các bên liên quan thảo luận và thống nhất về các thực thể miền, thuộc tính, liên kết và các cấu trúc tổng hợp. Tuy nhiên, UML chủ yếu mạnh ở việc mô tả cấu trúc; các quy tắc nghiệp vụ và các điều kiện hợp lệ của miền thường không thể hiện đầy đủ chỉ bằng quan hệ và bội số.

Để bổ sung khả năng đặc tả chính xác các quy tắc và ràng buộc nghiệp vụ, OCL thường được sử dụng kèm theo UML [93]. OCL cho phép mô tả một cách tường minh các bất biến, điều kiện trước/sau của trạng thái, và các ràng buộc dẫn xuất dựa trên ngữ nghĩa của mô hình UML [94]. Nhờ đặc tính khai báo và có cú pháp hình thức, OCL giúp giảm tính mơ hồ so với mô tả bằng ngôn ngữ tự nhiên, đồng thời hỗ trợ phân tích tĩnh, kiểm tra tính nhất quán và phát hiện vi phạm ràng buộc ở giai đoạn thiết kế.

Mặc dù UML/OCL cung cấp một nền tảng chuẩn hóa và giàu khả năng mô tả cho DM, cách tiếp cận này bộc lộ một số hạn chế khi đặt trong yêu cầu "mô hình miền hướng thực thi" của DDD. Thứ nhất, UML phân tách cấu trúc, hành vi và ràng buộc thành các không gian biểu diễn khác nhau (biểu đồ lớp, biểu đồ hành vi và đặc tả OCL), khiến tri thức miền bị phân mảnh và khó duy trì tính nhất quán khi mô hình tiến hóa. Thứ hai, các đặc tả UML/OCL thường tồn tại tách rời khỏi mã nguồn; do đó, việc hiện thực hóa DM từ UML/OCL vẫn chủ yếu phụ thuộc vào thao tác thủ công của nhà phát triển, làm gia tăng nguy cơ sai lệch ngữ nghĩa giữa đặc tả và hệ thống triển khai. Thứ ba, mặc dù OCL có thể được dùng để kiểm tra ràng buộc ở mức mô hình, nhưng bản thân UML/OCL không cung cấp một cơ chế thống nhất để đưa DM đến trạng thái có khả năng thực thi hoặc phục vụ sinh mã một cách trực tiếp trong quy trình DDD. Các công cụ MDE có thể hỗ trợ sinh mã từ UML và dịch một phần ràng buộc OCL sang các đoạn kiểm tra, tuy nhiên việc chuyển hóa này thường phụ thuộc vào công

cụ, quy ước triển khai và phạm vi ràng buộc được hỗ trợ; đồng thời khó đảm bảo rằng ngữ nghĩa nghiệp vụ được bảo toàn xuyên suốt từ DM đến hệ thống chạy được. Vì vậy, UML/OCL là tiếp cận nền tảng để biểu diễn cấu trúc và ràng buộc của DM nhờ tính chuẩn hóa và khả năng đặc tả hình thức. Tuy nhiên, trong bối cảnh DDD, việc biểu diễn phân tách và tách rời khỏi mã nguồn khiến UML/OCL chưa đáp ứng đầy đủ yêu cầu về một DM hợp nhất, hướng thực thi và có khả năng làm trung tâm cho tự động hóa phát triển phần mềm.

Nhiều nghiên cứu đã tập trung vào việc chuyển đổi các biểu đồ lớp và các ràng buộc nghiệp vụ được đặc tả bằng OCL thành mã thực thi hoặc các cấu trúc tương ứng trong OOPL. Công trình của Cabot và Teniente [25] đề xuất sinh mã Java từ các ràng buộc OCL bằng cách dịch chúng thành các truy vấn SQL; cách tiếp cận này chủ yếu hướng đến việc kiểm tra tính hợp lệ ở mức cơ sở dữ liệu, thay vì tích hợp các ràng buộc trực tiếp vào DM trong bộ nhớ. Tương tự, Rackov và cộng sự [99] khảo sát việc sinh mã Java dựa trên các quy tắc OCL, trong đó các ràng buộc được hiện thực hóa thành các phương thức kiểm tra độc lập, dẫn đến sự tách rời giữa lô-gic ràng buộc và cấu trúc DM.

Một số công trình khác [29, 54] đề xuất các khung làm việc dịch OCL sang các chú thích Java nhằm hỗ trợ tích hợp ràng buộc vào mã nguồn. Tuy nhiên, các giải pháp này thường yêu cầu sự can thiệp thủ công đáng kể khi xử lý các ràng buộc phức tạp. Công cụ `OCL2Java`, mặc dù hỗ trợ bán tự động, chủ yếu chỉ xử lý được các mẫu OCL đơn giản và thiếu hỗ trợ cho các cấu trúc như `exists`, `forAll` và các phép toán trên tập hợp. Gần đây hơn, một số nghiên cứu [8, 19, 33, 90] đã kết hợp OCL với cơ chế chú thích để biểu diễn DM theo tinh thần DDD. Ngoài ra, phương pháp `OCL2MSFOL` [35] chuyển các ràng buộc OCL sang lô-gic vị từ bậc nhất nhằm phục vụ kiểm chứng hình thức, nhưng không xem xét việc hiện thực và tích hợp các ràng buộc này trong DM thực thi.

Tổng hợp các công trình trên cho thấy các giải pháp chuyển đổi và hiện thực hóa ràng buộc OCL vẫn còn nhiều thách thức. Nhiều phương pháp mới chỉ đạt mức bán tự động và thường phải can thiệp thủ công khi gặp các ràng buộc phức tạp. Bên cạnh đó, phạm vi hỗ trợ OCL trong thực tế còn hạn chế, đặc biệt với các ràng buộc nghiệp vụ giàu ngữ nghĩa trên lớp, thuộc tính, quan hệ và phương thức [29]. Ngoài ra, nhiều giải pháp chủ yếu

tập trung vào cơ chế kiểm tra ràng buộc, trong khi ít xem xét dẫn xuất giá trị hay cơ chế lan truyền thay đổi theo sự tiến hóa trạng thái của mô hình, làm suy giảm mức độ gắn kết giữa lô-gic ràng buộc và thực thi miền. Cuối cùng, việc phụ thuộc vào các công cụ và bước sinh mã chuyên biệt làm tăng độ phức tạp của quy trình, đồng thời gây khó khăn cho bảo trì khi mô hình tiếp tục tiến hóa.

Trên cơ sở các hạn chế đó, kỹ thuật biểu diễn ràng buộc dựa vào mẫu chú thích (gọi là CAP) trong luận án được đề xuất như một cách tiếp cận dựa trên chú thích nhằm đặc tả cấu trúc và ngữ nghĩa của các ràng buộc OCL trực tiếp trong ngữ cảnh của DM thực thi được trình bày trong Mục 3.2. Cách tiếp cận này hướng đến việc tăng cường tính rõ ràng và khả năng hiểu theo ngữ cảnh của các ràng buộc, đồng thời cải thiện khả năng cộng tác, truy vết và mức độ gắn kết giữa các ràng buộc nghiệp vụ và DM trong các hệ thống phần mềm phức tạp.

2.2.2 Biểu diễn khía cạnh hành vi miền

Hành vi miền là thành phần cốt lõi phản ánh cách các khái niệm miền tương tác, tiến hóa trạng thái và hiện thực hóa các quy trình nghiệp vụ [16]. Trong DDD, hành vi không chỉ là các thao tác kỹ thuật mà còn mang ý nghĩa nghiệp vụ rõ ràng, gắn chặt với ngôn ngữ chung và tri thức miền [41, 124]. Do đó, việc biểu diễn và tích hợp hành vi vào DM là điều kiện cần để DM có thể đóng vai trò trung tâm ngữ nghĩa và hướng thực thi.

(i) Biểu diễn hành vi bằng UML tách rời cấu trúc. Cách tiếp cận phổ biến nhất hiện nay là sử dụng các biểu đồ hành vi của UML, chẳng hạn biểu đồ hoạt động, biểu đồ trạng thái hoặc biểu đồ tuần tự, để mô tả hành vi nghiệp vụ [94]. Các biểu đồ này cho phép thể hiện luồng điều khiển, sự thay đổi trạng thái và tương tác giữa các đối tượng ở mức trực quan [105]. Tuy nhiên, trong thực tiễn, các biểu đồ hành vi thường tồn tại như các tạo tác tách rời khỏi biểu đồ lớp mô tả cấu trúc miền. Hành vi vì vậy không được gắn kết trực tiếp với các thực thể miền cụ thể, mà chủ yếu đóng vai trò tài liệu thiết kế hoặc hỗ trợ trao đổi. Sự tách rời này dẫn đến khó khăn trong việc duy trì tính nhất quán giữa cấu trúc và hành vi, đặc biệt khi mô hình tiến hóa, đồng thời hạn chế khả năng sử dụng trực tiếp các biểu đồ hành vi như một phần của DM có khả năng thực thi.

(ii) Biểu diễn hành vi bằng DSL. Một hướng tiếp cận khác là định nghĩa các DSL cho hành vi miền, cho phép mô tả các quy trình nghiệp vụ, luồng hoạt động hoặc máy trạng thái bằng cú pháp và thuật ngữ gần với ngôn ngữ nghiệp vụ. Các DSL này giúp tăng khả năng biểu đạt và tạo điều kiện cho chuyên gia miền tham gia trực tiếp vào việc đặc tả hành vi [86]. Trong bối cảnh MDE, hành vi được đặc tả bằng DSL có thể được chuyển đổi sang mã thực thi bằng các bộ chuyển đổi M2M hoặc M2T. Tuy nhiên, các DSL hành vi thường được thiết kế độc lập với DSL cấu trúc hoặc mô hình lớp UML, dẫn đến việc hành vi và cấu trúc được biểu diễn trong các không gian ngữ nghĩa khác nhau. Việc hợp nhất các DSL này đòi hỏi các cơ chế ánh xạ và phối hợp phức tạp, và trong nhiều trường hợp chưa có một nền tảng ngữ nghĩa thống nhất để đảm bảo rằng hành vi được diễn giải đúng trên cấu trúc miền tương ứng.

(iii) Tích hợp hành vi thông qua DSL nội sinh. Gần hơn với tinh thần của DDD, một số tiếp cận dựa trên DSL nội sinh, đặc biệt là các DSL dựa trên chú thích (aDSL), cho phép gắn hành vi trực tiếp vào các lớp miền trong ngôn ngữ lập trình chủ [46, 92]. Theo hướng này, hành vi được biểu diễn ngay trong ngữ cảnh của thực thể miền, giúp rút ngắn khoảng cách giữa mô hình và hiện thực, đồng thời tạo ra các DM có khả năng thực thi ở một mức độ nhất định. Tuy nhiên, phần lớn các tiếp cận aDSL hiện nay mới chỉ hỗ trợ hành vi ở mức thao tác cục bộ hoặc các quy ước lập trình, chưa cung cấp một cơ chế hình thức để mô tả hành vi nghiệp vụ phức hợp, có ngữ nghĩa rõ ràng và có thể kiểm chứng, cũng như chưa giải quyết triệt để việc hợp nhất hành vi với các mối quan tâm khác như ràng buộc hay bảo mật trong một DM hợp nhất.

Vấn đề ngữ nghĩa chung khi hợp nhất hành vi và cấu trúc. Từ các tiếp cận trên có thể thấy rằng thách thức cốt lõi không chỉ nằm ở việc biểu diễn hành vi, mà ở chỗ thiếu một nền tảng ngữ nghĩa chung để hợp nhất hành vi với cấu trúc miền. Khi hành vi được mô tả tách rời hoặc bằng các ngôn ngữ khác nhau, việc diễn giải hành vi trên trạng thái của DM trở nên không rõ ràng và khó kiểm chứng. Điều này khiến DM khó được xem như một hệ thống chuyển trạng thái thống nhất, trong đó cấu trúc xác định không gian trạng thái và hành vi xác định các phép chuyển trạng thái hợp lệ.

Do đó, trong bối cảnh DDD, cần có các kỹ thuật biểu diễn và hợp nhất hành vi cho phép: (i) gắn hành vi trực tiếp với các khái niệm miền; (ii) diễn

giải hành vi trên cùng một không gian ngữ nghĩa với cấu trúc và các mối quan tâm khác; và (iii) tạo nền tảng cho việc kiểm chứng và chuyển đổi mô hình hướng tới các DM có khả năng thực thi.

Trong lĩnh vực kỹ nghệ DSL, nhiều nghiên cứu đã đề xuất các cách phân loại DSL theo phạm vi áp dụng và mối quan hệ với ngôn ngữ chủ [9, 46, 129]. Trên cơ sở đó, kỹ thuật tích hợp hành vi vào mô hình miền (gọi là AGL) mà luận án đề xuất, được trình bày trong Mục 3.3 có định vị như một DSL nội sinh và ngang, tập trung vào một miền con cụ thể là các đồ thị hoạt động. Điểm khác biệt của AGL không nằm ở việc mở rộng hay tinh chỉnh các tiêu chí phân loại DSL nói chung, mà ở cách khai thác trực tiếp các đặc trưng hình thành nên miền đồ thị hoạt động để phục vụ cho việc mô hình hóa hành vi miền trong bối cảnh DDD.

Ý tưởng kết hợp DDD với DSL nhằm nâng cao mức trừu tượng của mô hình phần mềm đã được gợi mở trong [46]. Tuy nhiên, công trình này không đề xuất một cơ chế cụ thể để tích hợp hành vi miền như một phần của DM thực thi. Trong luận án này, AGL được đề xuất nhằm lấp đầy khoảng trống đó bằng cách kết hợp mô hình lớp hợp nhất với mô hình đồ thị hoạt động trong cùng một DM thống nhất, trong đó cấu trúc và hành vi được biểu diễn bằng hai aDSL bổ trợ cho nhau là DCSL [73] và AGL.

Mặc dù Evans [41] không đề cập tường minh việc mô hình hóa hành vi như một thành phần độc lập của phương pháp DDD, tác giả nhấn mạnh vai trò thiết yếu của hành vi đối tượng trong DM và sử dụng các biểu đồ tương tác UML làm ví dụ minh họa. Trong các hiện thực cụ thể của DDD như Apache Isis [122], một ngôn ngữ hành động đơn giản được sử dụng để đặc tả hành vi của đối tượng bằng các chú thích trong ngôn ngữ lập trình hướng đối tượng. Cách tiếp cận này có thể được xem là một hiện thực cụ thể của ngôn ngữ con hành động trong biểu đồ hoạt động UML [94], nhưng không cung cấp một phương pháp mô hình hóa hành vi có cấu trúc ở mức mô hình. Trái lại, AGL cung cấp một biểu diễn hành vi miền ở mức mô hình, cho phép tích hợp trực tiếp các đặc tả hành vi vào mô hình miền hợp nhất.

Trong thiết kế hướng đối tượng truyền thống, hành vi thường được xử lý thông qua các mẫu thiết kế hành vi [17], chủ yếu ở mức hiện thực và tổ chức tương tác giữa các đối tượng. Vì vậy, hành vi miền thường bị phân

tán trong các lớp và chưa được biểu diễn một cách tường minh trong mô hình miền. Một số nghiên cứu gần đây đã đề xuất các trừu tượng hành vi có khả năng thực thi và tích hợp với DM, chẳng hạn `ReMoDeL` [113]; tuy nhiên, các tiếp cận này thường tách rời hành vi khỏi cấu trúc miền hướng đối tượng. Ngược lại, AGL biểu diễn hành vi như một khía cạnh của mô hình miền, được mô hình hóa bằng các đồ thị hoạt động có khả năng thực thi và tích hợp trực tiếp với các đặc tả cấu trúc trong DCSL.

Về mặt ngữ nghĩa, AGL có cùng mục tiêu với các nghiên cứu kết hợp các mô hình cấu trúc và hành vi của UML [7, 88], nhưng sử dụng đồ thị hoạt động thay vì máy trạng thái để nhấn mạnh hành vi thực thi, trong đó trạng thái được thể hiện thông qua quan hệ trước - sau giữa các mô-đun hoạt động [94]. Bên cạnh đó, luận án khai thác cách tiếp cận MVC ở mức *vi mô* để tổ chức các mô-đun phần mềm và kết hợp chúng bằng các đồ thị hoạt động nhằm hỗ trợ tích hợp hành vi vào DM hợp nhất.

Cuối cùng, trong khi nhiều công trình trong lĩnh vực kỹ nghệ phần mềm hướng mô hình sử dụng nhiều DSL khác nhau để xây dựng các mô hình phần mềm hoàn chỉnh [18, 69], các DSL này thường không phân định rõ vai trò của DM so với các mô hình khác trong hệ thống. Khác với các tiếp cận đó, AGL trong luận án này được thiết kế như một aDSL nội sinh, gắn chặt với ranh giới của DM theo tinh thần DDD và chỉ được sử dụng cùng với DCSL để xây dựng DM hợp nhất, thay vì mở rộng sang các DSL phục vụ các mối quan tâm ngoài miền.

2.2.3 Biểu diễn khía cạnh bảo mật với RBAC

Kiểm soát truy cập dựa trên vai trò (*Role-Based Access Control - RBAC*) [10, 61] gắn các quyền với các vai trò phản ánh trách nhiệm trong tổ chức, và người dùng được gán vào các vai trò này. Cách phân tách này đơn giản hóa quản trị, hỗ trợ nguyên lý đặc quyền tối thiểu và cải thiện khả năng bảo trì. Trong bối cảnh các hệ thống có yêu cầu phân quyền phức tạp, đặc biệt là các hệ thống với vai trò người dùng động, kiểm soát truy cập trở thành một vấn đề trung tâm [85]. Việc tách rời các chính sách bảo mật khỏi lô-gic nghiệp vụ lõi dẫn đến sự không nhất quán, khiến các chính sách này khó bảo trì và khó kiểm chứng [94, 96]. Trong bối cảnh DDD, RBAC không chỉ được xem là một mối quan tâm bảo mật độc lập, mà còn ảnh hưởng trực

tiếp đến hành vi của các thực thể miền; do đó, RBAC cần được đặc tả tường minh trong DM [48, 76]. Theo đó, việc tích hợp RBAC vào DM trở thành một yêu cầu thiết yếu đối với các hệ thống DDD.

Theo tiêu chuẩn RBAC [10], mô hình RBAC lõi được định nghĩa dựa trên các khái niệm cơ bản sau:

- **Người dùng.** Biểu diễn một thực thể hoạt động (con người hoặc hệ thống) tương tác với hệ thống và yêu cầu truy cập tới các tài nguyên được bảo vệ.
- **Vai trò.** Biểu diễn một chức năng công việc hoặc trách nhiệm trong tổ chức (ví dụ: *Author*, *Reviewer*, *Editor*, *Subscriber*). **Vai trò** là một lớp trừu tượng trung gian giữa người dùng và quyền.
- **Quyền.** Biểu diễn một sự cho phép thực hiện một thao tác cụ thể trên một tài nguyên được bảo vệ. Phù hợp với mô hình RBAC, **Quyền** được định nghĩa dưới dạng các cặp $\langle$*Hành động, Tài nguyên được bảo vệ*$\rangle$.
- **Phiên.** Biểu diễn một ngữ cảnh thời gian chạy trong đó một người dùng kích hoạt một tập con các vai trò đã được gán cho người dùng đó. Một người dùng có thể có nhiều phiên đồng thời, mỗi phiên có thể có một tập vai trò đang hoạt động khác nhau.
- **Ràng buộc phân tách nhiệm vụ.** Các ràng buộc phân tách nhiệm vụ (*Separation of Duty - SoD*) này được sử dụng để ngăn ngừa xung đột lợi ích và cưỡng chế các chính sách được tổ chức:
  - *Phân tách nhiệm vụ tĩnh (Static Separation of Duty - SSD).* Hạn chế các gán vai trò xung đột bằng cách ngăn người dùng được gán vào các vai trò loại trừ lẫn nhau.
  - *Phân tách nhiệm vụ động (Dynamic Separation of Duty - DSD).* Hạn chế việc kích hoạt các vai trò xung đột bằng cách ngăn các vai trò loại trừ lẫn nhau được kích hoạt trong cùng một phiên.
- **Quy tắc truy cập.** Xác định điều kiện chấp nhận một yêu cầu truy cập, theo đó yêu cầu được chấp nhận nếu tồn tại một vai trò đang hoạt động trong phiên hiện tại cung cấp quyền cần thiết, đồng thời tất cả các điều kiện ngữ cảnh hoặc chính sách liên quan đều được thỏa mãn.

Một vấn đề thường gặp là RBAC thường được tích hợp vào các DM theo cách không chính thức hoặc thủ công. Li et al. [76] chỉ ra sự thiếu hụt hỗ trợ hình thức cho việc tích hợp RBAC trong các hệ thống đa miền, đồng thời cho rằng các cách tiếp cận thủ công thường dẫn đến sự không nhất quán. Tương tự, Oruc et al. [96] nghiên cứu việc tích hợp RBAC thủ công trong các hệ thống phức tạp như hệ đa tác tử kết hợp với chuỗi khối, và đề xuất một giải pháp dựa trên DSL.

Việc biểu diễn các chính sách như SoD [5] hoặc các trạng thái truy cập động thường chỉ giới hạn trong mã nguồn hoặc các ghi chú tài liệu, thiếu cơ chế kiểm chứng hình thức. Điều này có thể dẫn đến các vi phạm chính sách không được phát hiện trong các giai đoạn thiết kế hoặc kiểm thử.

Do đó, việc tăng cường khả năng biểu đạt của DM nhằm tự động tích hợp RBAC cùng với một cơ chế kiểm chứng đầy đủ và có thể chứng minh được có thể được xem là một trọng tâm nghiên cứu quan trọng hiện nay.

2.2.4 Mô tả ngữ nghĩa thực thi của DM với Event-B

Event-B là một phương pháp mô hình hóa hình thức dựa trên trạng thái, được phát triển từ phương pháp B [2, 83], dựa trên lô-gic bậc nhất và lý thuyết tập có kiểu để đặc tả và kiểm chứng các hệ thống rời rạc. Phương pháp này tách biệt rõ ràng các khía cạnh tĩnh trong ngữ cảnh và các khía cạnh động trong máy trạng thái, nơi hành vi hệ thống được mô hình hóa bằng các sự kiện có điều kiện bảo vệ và được ràng buộc bởi các bất biến. Tính đúng đắn của mô hình được đảm bảo chặt chẽ thông qua cơ chế tinh chỉnh và tập các nghĩa vụ chứng minh (*Proof Obligations - POs*) [126].

Trên cơ sở Event-B, UML-B kết hợp UML với Event-B nhằm hỗ trợ kiểm chứng hình thức các mô hình phần mềm bằng cách ánh xạ các mô hình UML sang Event-B [114]. Tuy nhiên, UML-B chủ yếu tập trung vào tính đúng đắn ở mức hệ thống và chưa đặt DM làm trung tâm ngữ nghĩa theo tinh thần của DDD, cũng như chưa hỗ trợ tích hợp các mối quan tâm miền như hành vi và bảo mật trong một DM hợp nhất.

Trong bối cảnh đó, các chính sách kiểm soát truy cập như SSD, DSD và các ràng buộc động có thể được biểu diễn một cách tự nhiên trong Event-B dưới dạng các bất biến hoặc các điều kiện bảo vệ trên các quan hệ phân

quyền. Ví dụ, quan hệ gán vai trò cho người dùng (UA) được ràng buộc để chỉ cho phép các cặp hợp lệ giữa người dùng (USR) và vai trò (R): $inv1 : UA \subseteq USR \times R$; Giả sử $r_1, r_2 \in R$ là hai vai trò xung đột tĩnh. $inv2 : \forall u \in USR \cdot \neg\big((u, r_1) \in UA \wedge (u, r_2) \in UA\big)$.

Khả năng tự động sinh và kiểm chứng các PO bằng công cụ Rodin [3] giúp phát hiện các lỗi lô-gic hoặc xung đột trong DM ngay từ giai đoạn thiết kế, mà không cần chờ tới hiện thực. Hơn nữa, Event-B có thể chứng minh rằng các chính sách phân quyền động được duy trì trong suốt quá trình tinh chỉnh, từ đặc tả trừu tượng đến hiện thực.

Trong bối cảnh tích hợp với DDD và các DSL, Event-B đảm nhiệm vai trò kiểm chứng hình thức ở tầng xử lý phía sau bằng cách chuyển các khái niệm chính sách từ DSL thành các biến và các điều kiện bất biến tương ứng [6, 68, 81, 101, 126]. Liên kết này cho phép phân tích và kiểm tra tự động các mâu thuẫn trong chính sách RBAC đã được định nghĩa. Quan trọng hơn, nó đảm bảo rằng mã nguồn được sinh ra từ DSL luôn tuân thủ các quy tắc bảo mật đã được chứng minh, đồng thời duy trì sự đồng bộ chặt chẽ giữa DM và các bản mẫu phần mềm được kiểm chứng hình thức.

2.3 Các hướng tiếp cận tích hợp mối quan tâm trong mô hình miền

Trong luận án này, một *mối quan tâm* được hiểu là một khía cạnh của miền nghiệp vụ cần được mô hình hóa một cách chuyên biệt nhằm biểu diễn đầy đủ các khái niệm, quan hệ, quy tắc và ngữ nghĩa liên quan. Theo cách tiếp cận này, mỗi mối quan tâm được đặc tả bằng một ngôn ngữ chuyên biệt miền (DSL) phù hợp với đặc điểm biểu diễn của nó. Các DSL này cho phép mô hình hóa một khía cạnh cụ thể của miền với cú pháp và ngữ nghĩa chuyên biệt, thay vì sử dụng một ngôn ngữ tổng quát để biểu diễn đồng thời mọi khía cạnh của hệ thống. Tuy nhiên, do các mối quan tâm thường có quan hệ phụ thuộc và tương tác lẫn nhau, các DSL tương ứng cần được tích hợp một cách có hệ thống nhằm hình thành một mô hình miền thống nhất. Trong phạm vi luận án, các mối quan tâm chính bao gồm cấu trúc, ràng buộc, hành vi và bảo mật. Các mối quan tâm này được đặc tả bằng các DSL tương ứng và sẽ được trình bày chi tiết trong các chương tiếp theo.

Trên cơ sở đó, các DSL theo mối quan tâm được hợp nhất trong mô hình miền hợp nhất (UDML) nhằm tạo ra một biểu diễn miền thống nhất, có khả năng thực thi, hỗ trợ kiểm chứng hình thức và làm cơ sở cho các phép chuyển đổi mô hình cũng như sinh tự động phần mềm.

Trong kỹ nghệ phần mềm, việc đặc tả và tích hợp các mối quan tâm nhằm quản lý các khía cạnh xuyên suốt của hệ thống như cấu trúc, hành vi, bảo mật hay hiệu năng. Mặc dù phân tách mối quan tâm là một nguyên tắc thiết kế quan trọng, thách thức cốt lõi nằm ở việc hợp nhất các mối quan tâm trở lại thành một DM thống nhất, có khả năng thực thi và bảo toàn ngữ nghĩa nghiệp vụ. Nhiều nghiên cứu đã đề xuất các kỹ thuật hợp nhất DSL và siêu mô hình [92, 102, 128]; tuy nhiên, phần lớn các tiếp cận này tập trung vào cú pháp hoặc cấu trúc ngôn ngữ, chưa giải quyết triệt để bài toán hợp nhất mối quan tâm trong phạm vi một DM có khả năng thực thi.

Việc hiện thực hóa các DM hợp nhất thành các tạo tác phần mềm có khả năng thực thi trong thực tiễn có thể được thực hiện theo nhiều cách tiếp cận kiến trúc khác nhau, phụ thuộc vào mục tiêu phát triển cũng như góc nhìn của các bên liên quan. Trong luận án này, các DM được hiện thực hóa dựa trên kiến trúc khung phần mềm dựa trên mô-đun (Module-Based Software Architecture – MOSA) [75], một kiến trúc tuân theo phong cách MVC. MOSA cung cấp một khung kiến trúc thực tiễn cho việc tổ chức, hiện thực và thực thi các DM trong các hệ thống phần mềm phát triển theo phương pháp DDD, qua đó thu hẹp khoảng cách giữa DM và hiện thực triển khai. Tuy nhiên, MOSA chủ yếu tập trung vào khía cạnh hiện thực và thực thi DM, mà chưa cung cấp một cơ chế hình thức để kiểm chứng việc các mối quan tâm bảo mật, chẳng hạn như RBAC, có thỏa mãn đầy đủ các ràng buộc an toàn đã được đặc tả hay không.

2.3.1 Tích hợp các mối quan tâm trong mô hình miền

Trong các hệ thống phần mềm thực tế, DM không chỉ mô tả lô-gic nghiệp vụ mà còn phải bao quát nhiều mối quan tâm khác như hành vi, bảo mật, ghi nhật ký hay hiệu năng. DM vì vậy thường được làm giàu và chính xác hóa thông qua các DSL [46, 66], bao gồm DSL ngoại sinh với cú pháp trừu tượng [9, 18] và DSL nội sinh với cú pháp cụ thể dạng đồ họa hoặc văn bản [14, 33, 135].

Việc định nghĩa và tích hợp nhiều DSL chuyên biệt theo mối quan tâm–như DSL cho lô-gic nghiệp vụ, bảo mật, giao diện người dùng hoặc hiệu năng–là thiết yếu để nắm bắt đầy đủ các yêu cầu đa dạng của các hệ thống phần mềm phức tạp [18, 53, 84]. Tuy nhiên, các nghiên cứu hiện có thường tập trung vào: (i) định nghĩa DSL để mô tả chính xác DM nhưng vẫn phải dịch ngữ nghĩa hoạt động sang ngôn ngữ lập trình; (ii) sử dụng DSL nội sinh, đặc biệt là các cơ chế dựa trên chú thích, để biểu diễn DM có khả năng thực thi; hoặc (iii) nghiên cứu việc phối hợp nhiều DSL theo mối quan tâm. Các tiếp cận này hoặc thiên về khả năng thực thi, hoặc tập trung vào tách/ghép mối quan tâm, nhưng chưa giải quyết triệt để bài toán hợp nhất các quan điểm mối quan tâm vào một DM hợp nhất thực thi theo đúng nguyên lý DDD [123].

Các mối quan tâm xuyên suốt như bảo mật và hiệu năng thường lan tỏa qua nhiều thành phần của DM và không thể được biểu diễn cục bộ trong một cấu trúc đơn lẻ. Đây là các mối quan tâm phi chức năng hoặc bổ trợ, không thuộc lô-gic cốt lõi của miền nhưng có ảnh hưởng đáng kể đến nhiều phần của mô hình. Do đó, chúng cần được mô hình hóa và tích hợp nhất quán thông qua các cơ chế hợp thành phù hợp nhằm đảm bảo tính khả thi, tính nhất quán ngữ nghĩa và khả năng thực thi của DM hợp nhất.

Các nỗ lực gần đây dựa trên phương pháp siêu mô hình để tổng hợp các DSL chuyên biệt theo mối quan tâm đã giúp hình thành các siêu mô hình cố kết và ánh xạ một–một sang các aDSL nội sinh nhúng trong OOPL [7]. Tuy nhiên, các tiếp cận này vẫn gặp nhiều hạn chế như quá trình tích hợp mang tính bán tự động, dễ phát sinh lỗi, phụ thuộc mạnh vào chất lượng ánh xạ và chưa hỗ trợ hiệu quả các mối quan tâm xuyên suốt trong các hệ thống phức tạp.

Một hướng tiếp cận khác là tổng hợp các DSL chuyên biệt theo mối quan tâm dựa trên cây cú pháp trừu tượng (*Abstract Syntax Tree - AST*) [20], khai thác tính linh hoạt và khả năng thao tác trực tiếp trên cấu trúc ngôn ngữ để hợp nhất nhiều góc nhìn miền. Tuy nhiên, việc định nghĩa đầy đủ cú pháp, ngữ nghĩa cho từng DSL và xây dựng cơ chế hợp thành dựa trên chú thích ở mức AST nhằm tạo ra một DM hợp nhất có khả năng thực thi vẫn là một thách thức nghiên cứu mở.

2.3.2 Tích hợp hành vi miền

Trong DDD, hành vi nghiệp vụ là một thành phần cốt lõi phản ánh động lực và sự tiến hóa trạng thái của DM [41, 130]. Các hành vi này thường gắn liền với các quy trình nghiệp vụ, các quy tắc miền và các điều kiện ngữ cảnh, đồng thời chi phối cách các thực thể miền tương tác và thay đổi theo thời gian. Do đó, việc biểu diễn và tích hợp hành vi miền một cách nhất quán vào DM là yêu cầu thiết yếu để đảm bảo rằng mô hình không chỉ mô tả cấu trúc tĩnh mà còn phản ánh đầy đủ lô-gic nghiệp vụ của hệ thống.

Tuy nhiên, trong nhiều tiếp cận DDD hiện nay [51, 62, 110, 125, 138], DM chủ yếu tập trung vào khía cạnh cấu trúc, trong khi hành vi miền thường được xử lý theo các cách tách rời. Một hướng tiếp cận phổ biến là mô tả hành vi bằng các biểu đồ UML như biểu đồ hoạt động, trạng thái hoặc tuần tự [9, 119, 127]. Các biểu đồ này cung cấp cái nhìn trực quan về luồng xử lý, nhưng thường tồn tại như các tạo tác độc lập, không được tích hợp chặt chẽ với cấu trúc miền và thiếu một ngữ nghĩa thống nhất để hỗ trợ khả năng thực thi hoặc kiểm chứng.

Một hướng khác là sử dụng các DSL cho hành vi nhằm mô tả các quy trình hoặc luồng nghiệp vụ ở mức trừu tượng cao [109]. Mặc dù các DSL này giúp tăng khả năng biểu đạt hành vi và hỗ trợ tự động hóa ở một mức độ nhất định, chúng thường được thiết kế và vận hành như các ngôn ngữ độc lập. Việc hợp nhất ngữ nghĩa hành vi với cấu trúc miền vì vậy phải dựa vào các cơ chế ánh xạ hoặc chuyển đổi bổ sung, làm gia tăng độ phức tạp và nguy cơ sai lệch ngữ nghĩa.

Trong các tiếp cận dựa trên DSL nội sinh là aDSL, hành vi miền có thể được gắn trực tiếp vào các lớp miền thông qua các cấu trúc ngôn ngữ của ngôn ngữ lập trình chủ. Cách tiếp cận này giúp rút ngắn khoảng cách giữa mô hình và hiện thực, đồng thời hỗ trợ khả năng thực thi [98, 122]. Tuy nhiên, hành vi miền trong trường hợp này thường bị pha trộn với chi tiết kỹ thuật hoặc bị phân tán trong mã nguồn, khiến DM khó giữ vai trò là một đặc tả nghiệp vụ rõ ràng và nhất quán.

Nhìn chung, các tiếp cận hiện có cho thấy một vấn đề chung: hành vi miền chưa được tích hợp một cách hệ thống vào DM với một nền tảng ngữ nghĩa thống nhất. Sự thiếu vắng một cơ chế hợp nhất hành vi với cấu trúc

miền khiến mô hình khó được xem như một thực thể ngữ nghĩa hoàn chỉnh, vừa có khả năng diễn đạt nghiệp vụ, vừa có khả năng thực thi và kiểm chứng. Khoảng trống này đặt ra nhu cầu nghiên cứu các kỹ thuật biểu diễn hành vi miền gắn chặt với DM, đồng thời hỗ trợ hợp nhất ngữ nghĩa hành vi và cấu trúc trong một DM hướng thực thi theo tinh thần của DDD.

2.3.3 Tích hợp ràng buộc và chính sách bảo mật

UML/OCL [93, 94] mạnh về mô tả, nhưng chúng chưa đáp ứng yêu cầu của DM có thể thực thi. Các DM thường được biểu diễn bằng biểu đồ lớp UML/OCL [94] nhằm mô tả các khái niệm miền và mối quan hệ giữa chúng. Nhiều nghiên cứu về khả năng sử dụng OCL cho thấy cú pháp và cách diễn đạt của ngôn ngữ này gây khó khăn đáng kể đối với những người không chuyên mô hình hóa, ngay cả khi họ am hiểu nghiệp vụ. Nhiều công trình nghiên cứu đã đề xuất các cải tiến cho OCL, các nỗ lực này chủ yếu tập trung vào việc nâng cấp môi trường thao tác bao gồm: trình soạn thảo chuyên dụng, làm nổi bật cú pháp, hướng dẫn tái cấu trúc, hay các tiện ích hỗ trợ khác. Tuy nhiên, những cải tiến đó vẫn duy trì quan điểm xem OCL như một ngôn ngữ hình thức độc lập, được gắn kèm vào mô hình UML/Ecore [22, 82, 132] và tách rời khỏi ngôn ngữ nghiệp vụ mà chuyên gia miền sử dụng. Do đó, khoảng cách ngữ nghĩa giữa mô hình kỹ thuật và cách biểu đạt tri thức miền vẫn còn tồn tại, thay vì nhúng OCL vào một ngôn ngữ miền nhằm tăng khả năng diễn đạt trực tiếp các khái niệm nghiệp vụ.

Một số thư viện hiện đại như `OCL.js` hoặc các hướng tiếp cận như `JjOM` đưa OCL vào hệ sinh thái `JavaScript` và ứng dụng web, song cách dùng vẫn xoay quanh việc viết và quản lý các biểu thức OCL rời rạc, buộc chuyên gia miền phải học cú pháp OCL thay vì tương tác với một cơ chế ràng buộc được thiết kế theo ngôn ngữ miền [21]. Nhiều nỗ lực cải thiện công cụ và trải nghiệm thao tác gắn kết trực tiếp với mã nguồn [21, 26, 30, 37, 118]. Các nghiên cứu trong MDE cho phép gắn OCL vào siêu mô hình (như Ecore hoặc UML biểu đồ lớp) nhằm tạo ra các ràng buộc tích hợp, tuy nhiên cách tích hợp này vẫn để ràng buộc tồn tại dưới dạng OCL độc lập, không đồng nhất với cú pháp của lớp miền và không thân thiện với chuyên gia miền. Tương tự, một số khung làm việc dựa trên chú thích cho phép mô tả các

ràng buộc mức cơ bản như phạm vi giá trị, tính tùy chọn hoặc quan hệ, nhưng hoàn toàn không hỗ trợ ràng buộc phức tạp tương đương OCL. Vì vậy, dù có khả năng gắn siêu dữ liệu lên lớp miền, các khung làm việc này chưa thể hiện thực hóa việc gắn các ràng buộc nghiệp vụ như `forAll`, `exists` một cách tự nhiên, dễ đọc, và tích hợp liền mạch vào DM như một phần của DSL đặc tả miền. Các nghiên cứu liên quan đến OCL và DSL cho cấu hình sản phẩm, cũng như các nghiên cứu về sinh kiểm thử từ OCL hoặc suy luận dựa trên ràng buộc cũng coi OCL như một tập ràng buộc tổng quát mà không tổ chức các ràng buộc này thành các nhóm miền có ngữ nghĩa ổn định và tên gọi dễ nhớ. Từ đó, có thể thấy rằng các công trình hiện tại mới chỉ chạm tới vấn đề phân loại ràng buộc ở mức lô-gic, chưa phát triển thành một danh mục ràng buộc theo miền nghiệp vụ với cú pháp và ngữ nghĩa được chuẩn hóa để chuyên gia miền có thể nhận diện và sử dụng trực tiếp.

Trong các công trình sinh mã từ OCL, việc truy vết thường được hiểu như một hệ quả tự nhiên của luật chuyển đổi, chứ không phải một cơ chế được thiết kế có chủ đích. Các khung như OCL2Java [133], USE Tool [58] hay những bộ sinh mã nguồn xác thực trong EMF cung cấp khả năng dịch ràng buộc OCL thành mã kiểm tra. Các môi trường chạy như Eclipse OCL có thể lưu trữ ràng buộc như một phần của mô hình, nhưng cũng không đưa ra hệ thống truy vết hai chiều mang tính hệ thống, nơi mỗi ràng buộc có thể được theo dõi xuyên suốt qua mô hình, chuyển đổi và mã thực thi. Trong khi đó, các nỗ lực sử dụng aDSL như DCSL [73] đã cho phép biểu diễn và chạy mô hình miền hiệu quả, nhưng lại chỉ hỗ trợ một tập ràng buộc cơ bản, không đủ khả năng mô tả và tích hợp các ràng buộc OCL phức tạp, đa lớp và phụ thuộc ngữ cảnh. Sự tách rời giữa năng lực biểu đạt mạnh của OCL và khả năng thực thi của aDSL đã trở thành nút thắt ngăn cản việc phát triển một DM thực thi thống nhất.

Trong phân tích và thiết kế hướng đối tượng, DM thường được biểu diễn bằng các biểu đồ lớp UML kết hợp với OCL [71, 94] nhằm đặc tả cấu trúc và các ràng buộc nghiệp vụ. Trong DDD, Evans [41] mở rộng vai trò của DM, coi đây vừa là trung tâm của tri thức và luật lệ miền, vừa là phương tiện giao tiếp giữa chuyên gia miền và lập trình viên.

Các nghiên cứu trong MDE [9, 18] tiếp tục hướng tiếp cận này bằng cách phát triển các DSL để mô tả DM một cách chính xác hơn, dựa trên siêu

mô hình và cú pháp chuyên biệt. Tuy nhiên, các mô hình theo hướng mô tả này chủ yếu tập trung vào tính biểu đạt cú pháp, trong khi chưa trực tiếp đáp ứng yêu cầu về khả năng thực thi và ngữ nghĩa hành vi trong bối cảnh DDD. Phương pháp tích hợp các mối quan tâm vào mô hình miền hợp nhất (gọi là, UDML) được đề xuất nhằm khắc phục hạn chế này bằng cách đặc tả một DM hợp nhất có khả năng thực thi.

Trong các hệ thống có yêu cầu phân quyền phức tạp, kiểm soát truy cập trở thành một mối quan tâm trung tâm thay vì thứ yếu [115]. Nhiều nghiên cứu cho thấy việc tách rời chính sách bảo mật khỏi lô-gic nghiệp vụ lõi có thể dẫn đến sự không nhất quán giữa hành vi miền và các quy tắc phân quyền [12, 94, 96]. Dưới góc nhìn DDD, các chính sách RBAC vì thế cần được biểu diễn tường minh như một phần của hành vi miền [48, 76]. Tuy nhiên, phần lớn các cách tiếp cận hiện nay tích hợp RBAC theo cách thủ công hoặc phi hình thức, thiếu một nền tảng ngữ nghĩa hợp nhất cho toàn bộ DM.

Các nghiên cứu về DSL nội sinh [46, 135] cho thấy đây là một hướng tiếp cận hiệu quả để xây dựng các DM có khả năng thực thi, nhờ tận dụng cú pháp và ngữ nghĩa của ngôn ngữ chủ. Nhiều công trình đã sử dụng chú thích để mô hình hóa ràng buộc miền và xây dựng các DSL nhúng trong OOPL [33, 98, 122], hoặc duy trì ánh xạ giữa DM ở mức mã nguồn và UML [14]. Tiêu biểu trong số đó là DCSL [73], cho phép đặc tả cấu trúc miền có khả năng thực thi.

Tuy nhiên, các tiếp cận này chủ yếu nhấn mạnh tính thực thi ở mức hiện thực hóa, trong khi chưa giải quyết thỏa đáng bài toán hợp nhất nhiều mối quan tâm và kiểm chứng hình thức các ràng buộc động, đặc biệt là các ràng buộc phân quyền. OCL hoặc mã dựa trên chú thích chủ yếu hỗ trợ phân tích tĩnh, chưa cung cấp các bảo đảm hình thức về tính an toàn và không xung đột. Phương pháp của luận án hướng đến xây dựng một DM hợp nhất vừa có thể thực thi vừa được kiểm chứng hình thức, trong đó các chính sách RBAC được đảm bảo an toàn và nhất quán trong toàn bộ vòng đời phần mềm.

Tách biệt và hợp nhất mối quan tâm là nguyên tắc cốt lõi trong quản lý độ phức tạp hệ thống [123, 131]. Nhiều nghiên cứu đã sử dụng DSL để mô tả các mối quan tâm riêng biệt và chỉ ra khó khăn trong việc kết hợp chúng

thành một mô hình thống nhất [92, 102, 129]. Các cơ chế hợp nhất phổ biến bao gồm tham chiếu, mở rộng, nhúng và tái sử dụng, hoặc điều phối ở mức siêu mô hình [108].

Khác với các cách tiếp cận tập trung vào hợp nhất cú pháp hoặc cấu trúc, luận án đề xuất cơ chế hợp nhất mối quan tâm trong một DM hợp nhất có khả năng thực thi, dựa trên chú thích và hợp nhất ở mức cây cú pháp trừu tượng (AST). Cách tiếp cận này cho phép hợp nhất đồng thời cú pháp, ngữ nghĩa và khả năng thực thi của các mối quan tâm, đồng thời duy trì sự gắn kết với DM theo tinh thần DDD.

Các phương pháp hình thức, đặc biệt là Event-B, đã được sử dụng rộng rãi để kiểm chứng các hệ thống an toàn và đảm bảo về bảo mật [68, 79]. Nhiều nghiên cứu đã mã hóa RBAC dưới dạng bất biến và điều kiện bảo vệ để kiểm chứng các thuộc tính bảo mật [6, 81]. Một số công trình cũng khảo sát việc kết nối DSL với Event-B nhằm cung cấp kiểm tra tính đúng đắn dữ liệu và lô-gic nghiệp vụ ở phía máy chủ [101, 126].

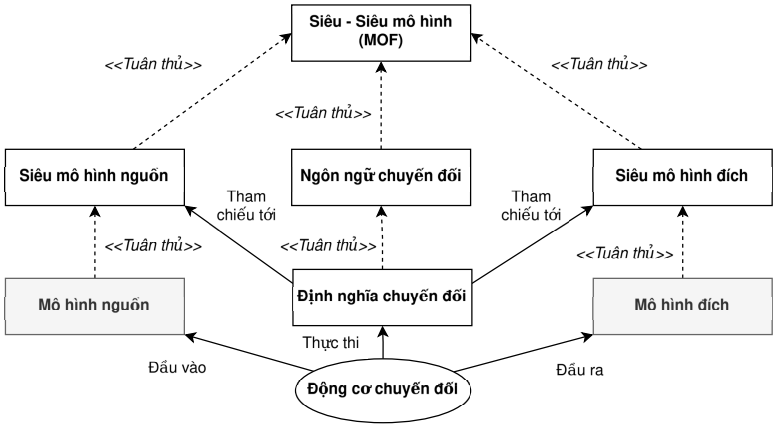
Tuy nhiên, các cách tiếp cận này thường xem phân quyền như một bài toán chính sách độc lập, tách rời khỏi hành vi miền. Ngược lại, phương pháp được đề xuất trong luận án sử dụng Event-B như một nền tảng ngữ nghĩa hình thức cho DM hợp nhất được đặc tả bằng UDML được trình bày chi tiết trong Chương 4 của luận án, trong đó các ràng buộc RBAC trực tiếp giới hạn các chuyển tiếp hành vi. Cách tiếp cận này cho phép kiểm chứng đồng thời tính đúng đắn miền và các thuộc tính bảo mật trong một khuôn khổ hình thức thống nhất.

2.4 Các thao tác chuyển đổi mô hình miền

Phần này trình bày các thao tác chuyển đổi mô hình miền trong khuôn khổ kỹ nghệ phần mềm hướng mô hình, với mục tiêu làm rõ vai trò của chuyển đổi mô hình trong việc liên kết các mức trừu tượng khác nhau của mô hình miền, đồng thời tạo cơ sở cho quá trình sinh các chế tác phần mềm từ mô hình miền trong bối cảnh thiết kế hướng miền. Trong cách tiếp cận này, mô hình miền không chỉ được xem như một biểu diễn khái niệm của tri thức nghiệp vụ, mà còn là đầu vào trung tâm cho các bộ chuyển đổi phục vụ sinh các chế tác phần mềm.

2.4.1 Kỹ thuật chuyển đổi mô hình

Trong kỹ nghệ phần mềm hướng mô hình [18], chuyển đổi mô hình là cơ chế trung tâm để tự động hóa quá trình phát triển phần mềm. Một bộ chuyển đổi điển hình tham gia vào các giai đoạn phát triển từ mô hình yêu cầu, mô hình phân tích, mô hình thiết kế đến mã nguồn, trong đó mỗi giai đoạn đều cần duy trì tính nhất quán giữa mô hình nguồn và mô hình đích. Xét về bản chất, một chuyển đổi mô hình không chỉ được thực hiện giữa hai mô hình cụ thể, mà được xác lập ở cấp độ siêu mô hình, với điều kiện các mô hình nguồn và mô hình đích phải tuân thủ các siêu mô hình tương ứng. Đây cũng là quan điểm chung của kỹ nghệ hướng mô hình (MDE) và kiến trúc hướng mô hình (MDA), trong đó các phép chuyển đổi được sử dụng để liên kết các mô hình ở các mức trừu tượng khác nhau, từ mô hình độc lập tính toán (CIM), mô hình độc lập nền tảng (PIM) đến mô hình chuyên biệt nền tảng (PSM).



Hình 2.2: Kiến trúc chung của bộ chuyển đổi mô hình.

Hình 2.2 mô tả khung công việc chung của một bộ chuyển đổi mô hình điển hình [18]. Theo khung này, bộ chuyển đổi được xác định bởi cặp siêu mô hình nguồn và siêu mô hình đích, cùng với định nghĩa chuyển đổi và môi trường thực thi tương ứng. Việc phát biểu chuyển đổi ở mức siêu mô hình cho phép tách biệt giữa đặc tả chuyển đổi và các mô hình thể hiện cụ thể, đồng thời tạo điều kiện cho việc tái sử dụng, kiểm chứng và hiện thực hóa các phép chuyển đổi trên các công cụ khác nhau.

Xét theo cách tiếp cận cài đặt, các bộ chuyển đổi mô hình thường được phân thành bốn nhóm chính, bao gồm: cách tiếp cận khai báo/quan hệ, cách tiếp cận mệnh lệnh/thực thi, cách tiếp cận dựa trên chuyển đổi đồ thị,

và cách tiếp cận lai. Cách tiếp cận khai báo tập trung mô tả quan hệ giữa các phần tử của mô hình nguồn và mô hình đích; cách tiếp cận mệnh lệnh tập trung mô tả trình tự thực hiện các bước chuyển đổi; cách tiếp cận dựa trên chuyển đổi đồ thị biểu diễn mô hình bằng đồ thị định kiểu và đặc tả chuyển đổi bằng các luật viết lại đồ thị; trong khi cách tiếp cận lai cố gắng kết hợp ưu điểm của các hướng trên. Trong thực tiễn, các ngôn ngữ và công cụ như ATL [52], QVT [91] và Acceleo [18] thường được sử dụng để hiện thực các bộ chuyển đổi mô hình và sinh mã nguồn.

Trong bối cảnh mô hình miền giàu thông tin, đặc biệt khi mô hình miền đồng thời chứa cấu trúc, hành vi và các ràng buộc nghiệp vụ, thách thức cốt lõi của chuyển đổi mô hình không chỉ nằm ở việc ánh xạ cấu trúc, mà còn ở việc bảo toàn ngữ nghĩa của mô hình trong toàn bộ quá trình chuyển đổi. Theo quan điểm chất lượng của các bộ chuyển đổi mô hình, các thuộc tính cần được quan tâm bao gồm: tính đúng đắn về ngữ pháp của mô hình đích, tính đúng đắn về ngữ nghĩa, tính đầy đủ, và các thuộc tính hành vi chức năng như kết thúc hay hội tụ. Đối với mô hình miền trong DDD, yêu cầu này càng trở nên quan trọng vì mô hình đích không chỉ cần hợp lệ về cú pháp, mà còn phải phản ánh nhất quán tri thức miền và lô-gic nghiệp vụ của mô hình nguồn. Do đó, các phép chuyển đổi cần dựa trên một biểu diễn trung gian có ngữ nghĩa rõ ràng và nhất quán, chẳng hạn như trong khung JDA [74].

Trên phương diện thao tác, chuyển đổi mô hình thường được thực hiện dưới hai dạng chính. Thứ nhất là chuyển đổi mô hình sang mô hình (M2M), trong đó mô hình nguồn được ánh xạ sang một mô hình trung gian hoặc mô hình đích ở cùng hay khác mức trừu tượng. Thứ hai là chuyển đổi mô hình sang văn bản (M2T), trong đó mô hình được chuyển thành các biểu diễn văn bản như mã nguồn, tệp cấu hình hoặc các dạng đặc tả có thể thực thi. Trong nhiều trường hợp, đặc biệt đối với sinh phần mềm dựa trên mô hình, M2M và M2T không tồn tại tách biệt mà được tổ chức thành một bộ chuyển đổi: trước hết mô hình nguồn được chuyển thành một biểu diễn trung gian có ngữ nghĩa rõ ràng, sau đó mới tiếp tục sinh ra các tạo tác văn bản hoặc mã nguồn tương ứng. Cách tổ chức này giúp giảm độ phụ thuộc trực tiếp giữa mô hình yêu cầu và mã nguồn, đồng thời tạo điều kiện thuận lợi cho việc kiểm soát, kiểm chứng và tiến hóa mô hình.

Một hướng tiếp cận phù hợp cho mục tiêu đó là kỹ thuật sinh mã nguồn

dựa trên siêu mô hình và khuôn mẫu văn bản [130]. Thay vì chuyển trực tiếp từ mô hình sang mã nguồn, mô hình đầu vào trước hết được ánh xạ sang một mô hình trung gian tuân thủ một siêu mô hình xác định, từ đó các thông tin cần thiết được trích xuất và kết hợp với các khuôn mẫu để sinh ra mã nguồn hoặc các biểu diễn thực thi tương ứng. Trong bối cảnh luận án này, cách tiếp cận như vậy đặc biệt phù hợp với yêu cầu chuyển đổi mô hình miền giàu hành vi và ràng buộc, vì nó cho phép xác lập một biểu diễn trung gian có ngữ nghĩa rõ ràng trước khi sinh các chế tác phần mềm.

2.4.2 Sinh chế tác phần mềm từ mô hình miền

Trên cơ sở các kỹ thuật chuyển đổi mô hình nêu trên, một hướng ứng dụng quan trọng là sinh các chế tác phần mềm từ mô hình miền. Trong kỹ nghệ phần mềm hướng mô hình, các chế tác sinh ra có thể bao gồm mô hình trung gian, mô hình thực thi, mã nguồn, giao diện người dùng, các mô-đun cấu hình, hoặc bản mẫu phần mềm. Đối với DDD, mục tiêu của các thao tác này không chỉ là tự động hóa phát triển phần mềm, mà còn là duy trì mối liên kết chặt chẽ giữa tri thức miền, mô hình miền và hiện thực phần mềm.

Một lớp thao tác quan trọng là sinh bản mẫu phần mềm từ mô hình miền. Nhiều nghiên cứu đã đề xuất các phương pháp sinh tự động các tạo tác phần mềm từ đặc tả yêu cầu, chủ yếu tập trung vào các thành phần giao diện và các chức năng cơ bản. Chẳng hạn, IFML được sử dụng để mô hình hóa luồng tương tác và sinh tự động giao diện [104]; các nghiên cứu khác khai thác mô hình ca sử dụng hoặc các biểu đồ UML như biểu đồ lớp, biểu đồ hoạt động và biểu đồ tuần tự để sinh giao diện phục vụ phát triển nhanh ứng dụng [11, 15, 70, 87, 134]. Tuy nhiên, phần lớn các tiếp cận này mới dừng ở việc sinh giao diện hoặc một phần cấu trúc chức năng, trong khi chưa thiết lập được một chuỗi chuyển đổi đủ chặt chẽ từ đặc tả yêu cầu đến mô hình miền có khả năng thực thi và bản mẫu phần mềm hoàn chỉnh.

Sinh giao diện người dùng là một trường hợp tiêu biểu của sinh chế tác phần mềm từ mô hình. Đây là hướng được quan tâm nhiều vì giao diện là tạo tác trực quan, dễ đánh giá và có giá trị cao trong việc xác minh yêu cầu với người dùng cuối. Tuy vậy, nếu chỉ dừng ở sinh giao diện thì khoảng cách giữa mô hình yêu cầu và lô-gic nghiệp vụ bên trong hệ thống vẫn chưa được giải quyết triệt để. Đối với các hệ thống phát triển theo DDD, giao diện người dùng chỉ là một phần của bản mẫu phần mềm; điều quan trọng

hơn là phải bảo đảm rằng giao diện, hành vi và cấu trúc dữ liệu đều được dẫn xuất nhất quán từ cùng một mô hình miền hoặc từ cùng một đặc tả yêu cầu hướng miền.

Việc đặc tả yêu cầu chức năng phần mềm nhằm xác định rõ các chức năng cần triển khai và các yêu cầu mà hệ thống phải đáp ứng, bao gồm mô tả chức năng, giao diện người dùng, các yêu cầu kỹ thuật và các ràng buộc hoạt động [43]. Trong thực tiễn, nhiều phương pháp và công cụ đã được đề xuất để đặc tả yêu cầu chức năng, sử dụng các hình thức như ngôn ngữ tự nhiên, UML hoặc các DSL [50]. Trong đó, UML/OCL là các công cụ được chuẩn hóa và sử dụng rộng rãi, trong khi DSL mang lại lợi thế về tính chuyên biệt, khả năng đơn giản hóa và hỗ trợ tự động hóa [18, 46, 63].

Một lớp thao tác khác có ý nghĩa nền tảng là sinh mô hình miền từ mô hình yêu cầu. Nhiều nghiên cứu đã tập trung vào các phương pháp chuyển đổi mô hình hành vi sang mã triển khai [119], tiêu biểu là các kỹ thuật sinh mã từ biểu đồ hoạt động và biểu đồ tuần tự UML [127]. Một hướng tiếp cận khác đề xuất tự động hóa chuyển đổi từ mô hình độc lập tính toán dựa trên nghiệp vụ sang mô hình độc lập nền tảng cho các ứng dụng web [136]. Trong nghiên cứu [75], tác giả đề xuất phương pháp phát triển phần mềm sinh theo DDD dựa trên kiến trúc phần mềm hướng mô-đun và kỹ thuật sinh cấu hình mô-đun.

Từ khảo sát trên có thể thấy nhu cầu cấp thiết về các bộ chuyển đổi giúp sinh nhanh các tạo tác phần mềm từ mô hình miền, đồng thời cung cấp nhiều góc nhìn phù hợp cho các bên liên quan. Các bộ chuyển đổi này đóng vai trò then chốt trong việc thu hẹp khoảng trống giữa các mô hình miền ở mức trừu tượng cao và các hiện thực phần mềm có khả năng thực thi, qua đó duy trì tính nhất quán ngữ nghĩa giữa các đặc tả, mô hình và các tạo tác phần mềm sinh ra.

Phương pháp thiết kế hướng miền mang lại nhiều lợi ích trong phát triển các hệ thống phần mềm phức tạp [62]. Tuy nhiên, khi yêu cầu nghiệp vụ thay đổi, mô hình miền cần được cập nhật tương ứng, đòi hỏi tính linh hoạt cao trong thiết kế và hiện thực, điều này có thể gây khó khăn cho cả nhà thiết kế và nhà phát triển. Trong nghiên cứu về DSL [46], mô hình được xem như một đặc tả trong ngôn ngữ chuyên biệt miền, và quá trình sinh mã nguồn thường trải qua nhiều bước chuyển đổi mô hình–sang–mô hình trước khi sinh mã và thực thi. Các nghiên cứu gần đây đã đề xuất DCSL [73] và

MCCL [75] nhằm hỗ trợ phát triển phần mềm theo DDD sử dụng các ngôn ngữ chuyên biệt miền dựa trên chú thích (aDSL) được nhúng trong OOPL.

Bên cạnh đó, nhiều công trình đã đề xuất các kỹ thuật sinh tự động giao diện người dùng và bản mẫu phần mềm từ các biểu đồ UML và các đặc tả ca sử dụng [70, 87, 104], tập trung vào phát triển công cụ chuyển đổi và đảm bảo tính nhất quán của mã nguồn sinh ra. Tuy nhiên, vẫn chưa có một phương pháp hoàn chỉnh nào thu hẹp hiệu quả khoảng cách ngữ nghĩa giữa đặc tả yêu cầu ở dạng ngôn ngữ tự nhiên hoặc bán hình thức như UML/OCL [45, 94] và mã nguồn cài đặt bản mẫu phần mềm.

Từ khảo sát trên có thể thấy rằng bài toán sinh chế tác phần mềm từ mô hình miền thực chất bao gồm một chuỗi các thao tác chuyển đổi liên tiếp: từ đặc tả yêu cầu sang mô hình miền, từ mô hình miền sang mô hình thực thi, và từ đó sang mã nguồn hoặc bản mẫu phần mềm. Trong chuỗi này, vai trò của biểu diễn trung gian là đặc biệt quan trọng, vì nó giúp thu hẹp khoảng cách giữa mô hình khái niệm và hiện thực phần mềm, đồng thời tạo điều kiện bảo toàn ngữ nghĩa của miền nghiệp vụ trong suốt quá trình chuyển đổi. Trên cơ sở đó, luận án tập trung nghiên cứu các thao tác chuyển đổi mô hình miền theo hướng tích hợp đồng thời khía cạnh cấu trúc và hành vi, từ đó tạo nền tảng cho việc sinh tự động bản mẫu phần mềm được trình bày trong Chương 5.

2.5 Hướng tiếp cận sử dụng AI/LLM

Sự phát triển của các mô hình ngôn ngữ lớn (*Large Language Models – LLMs*) như GPT-4 đã thúc đẩy mạnh mẽ việc ứng dụng trí tuệ nhân tạo (*Artificial Intelligence – AI*) trong kỹ nghệ phần mềm, đặc biệt trong các hoạt động phân tích yêu cầu, mô hình hóa phần mềm, sinh mã nguồn và hỗ trợ kiểm thử. Các khảo sát hệ thống gần đây về LLMs trong kỹ nghệ phần mềm, tiêu biểu là công trình của Hou và cộng sự [60] phân tích 395 nghiên cứu trong giai đoạn 2017–2024, cũng như khảo sát của Fan và cộng sự [42], cho thấy LLMs có khả năng xử lý ngôn ngữ tự nhiên, sinh văn bản có cấu trúc, sinh mã nguồn và hỗ trợ nhiều tác vụ trong vòng đời phát triển phần mềm, đồng thời chỉ ra các vấn đề mở liên quan đến độ tin cậy và khả năng kiểm soát kết quả sinh. Trong bối cảnh phân tích yêu cầu và mô hình hóa, LLMs được quan tâm do khả năng trích xuất tri thức miền

từ các đặc tả không hình thức và chuyển hóa chúng thành các biểu diễn có cấu trúc hơn, chẳng hạn biểu đồ UML, mô hình miền hoặc các đặc tả bán hình thức [64, 89, 95].

Một hướng nghiên cứu nổi bật là sử dụng LLMs để sinh mô hình từ đặc tả yêu cầu bằng ngôn ngữ tự nhiên. Ferrari và cộng sự [44] đã tiến hành nghiên cứu định tính trên các tài liệu yêu cầu thuộc nhiều miền khác nhau và cho thấy các biểu đồ tuần tự UML sinh bởi ChatGPT nhìn chung tuân thủ chuẩn và có mức độ dễ hiểu hợp lý, song tính đầy đủ và tính đúng đắn suy giảm đáng kể khi yêu cầu đầu vào chứa các yếu tố mơ hồ. Trong bối cảnh thiết kế hướng miền, Eisenreich và cộng sự [40] bước đầu khảo sát khả năng sử dụng LLMs để hỗ trợ các hoạt động của DDD như khai phá ngôn ngữ chung, xác định khái niệm miền và gợi ý các thành phần thiết kế. Các kết quả này cho thấy LLMs có tiềm năng hỗ trợ giai đoạn khởi tạo mô hình miền, đặc biệt khi yêu cầu ban đầu được mô tả bằng ngôn ngữ tự nhiên và còn thiếu cấu trúc hình thức.các

Tuy nhiên, các mô hình được sinh bởi LLMs thường mang tính xác suất và phụ thuộc mạnh vào cách diễn đạt yêu cầu, chiến lược nhắc lệnh và ngữ cảnh đầu vào. Nghiên cứu thực nghiệm trong nghiên cứu của Camera và cộng sự [27] khi đánh giá ChatGPT trong các tác vụ mô hình hóa UML cho thấy mặc dù công cụ hỗ trợ tốt việc biểu diễn các biểu thức OCL, nó bộc lộ nhiều hạn chế nghiêm trọng về độ chính xác ngữ nghĩa, đặc biệt là các lỗi đáng kể đối với các mô hình có quy mô lớn, cũng như khả năng hỗ trợ yếu đối với lớp kết hợp và thuộc tính trừu tượng. Điều này cho thấy, mặc dù LLMs có thể hỗ trợ nhận diện các thực thể, thuộc tính, quan hệ hoặc hành vi ở mức ban đầu, các kết quả sinh ra chưa đảm bảo tính đầy đủ, tính nhất quán và tính đúng đắn ngữ nghĩa của mô hình. Vì vậy, các mô hình do LLMs sinh ra thường chỉ phù hợp với vai trò bản nháp ban đầu hoặc gợi ý hỗ trợ người dùng, thay vì được sử dụng trực tiếp như mô hình miền chính thức.

Một hướng tiếp cận khác là sử dụng AI/LLM trong sinh mã nguồn. Các mô hình sinh mã như `CodeGen`, `Codex` hoặc GPT-4 có khả năng sinh mã từ mô tả tự nhiên, từ ví dụ hoặc từ các đặc tả bán hình thức [89, 95]. Trong thực tế, cách tiếp cận này có thể rút ngắn thời gian hiện thực và hỗ trợ lập trình viên trong các tác vụ lặp lại. Tuy nhiên, sinh mã trực tiếp từ mô tả yêu cầu thường bỏ qua tầng mô hình miền trung gian, làm suy giảm khả

năng kiểm soát mối quan hệ giữa yêu cầu, thiết kế và cài đặt. Mã nguồn được sinh ra có thể chạy được nhưng không nhất thiết phản ánh chính xác mô hình miền, các quy tắc nghiệp vụ hoặc các ràng buộc cần được bảo toàn. Điều này đặc biệt quan trọng đối với DDD, nơi mô hình miền và ngôn ngữ chung cần được duy trì nhất quán trong suốt quá trình phát triển phần mềm.

Trong bối cảnh kỹ nghệ phần mềm hướng mô hình, các nghiên cứu gần đây cũng bắt đầu xem xét việc kết hợp LLMs với MDE. Các hướng nghiên cứu bao gồm sử dụng LLMs để sinh mô hình từ yêu cầu, hỗ trợ hiểu và truy vấn kho mô hình, gợi ý luật chuyển đổi mô hình, sinh mã từ mô hình và hỗ trợ phát triển DSL. Một số công trình định vị LLMs như một công nghệ bổ trợ cho MDE, có khả năng giảm rào cản sử dụng công cụ mô hình hóa và hỗ trợ người dùng trong các tác vụ đòi hỏi nhiều tri thức ngôn ngữ tự nhiên. Tuy nhiên, các nghiên cứu này cũng chỉ ra rằng LLMs chưa thay thế được các cơ chế cốt lõi của MDE như siêu mô hình, luật hợp lệ cấu trúc, chuyển đổi mô hình có đặc tả tường minh và sinh mã có kiểm soát.

Đối với bài toán siêu mô hình và DSL, LLMs có thể hỗ trợ đề xuất các khái niệm, thuộc tính, quan hệ hoặc cú pháp ban đầu của một DSL từ mô tả miền. Tuy nhiên, việc sinh siêu mô hình đòi hỏi tính chặt chẽ cao về kiểu, quan hệ chứa, bội số, ràng buộc hợp lệ và ngữ nghĩa thao tác. Đây là những yếu tố mà LLMs khó đảm bảo nếu không có cơ chế kiểm tra hình thức đi kèm. Vì vậy, trong phát triển DSL, LLMs phù hợp hơn với vai trò hỗ trợ khai phá khái niệm, gợi ý cấu trúc ban đầu hoặc sinh tài liệu, trong khi việc xác định siêu mô hình, ngữ nghĩa và các ràng buộc hợp lệ vẫn cần dựa trên các kỹ thuật MDE truyền thống.

Một vấn đề quan trọng khác là kiểm chứng mô hình. Đối với việc sinh ràng buộc OCL là một thành phần cốt lõi trong đặc tả chính xác mô hình miền Abukhalaf và cộng sự [4] đã tiến hành nghiên cứu thực nghiệm trên tập dữ liệu gồm 15 mô hình UML và 168 đặc tả, đo lường tính hợp lệ cú pháp và độ chính xác thực thi của các ràng buộc OCL được sinh tự động. Kết quả cho thấy độ tin cậy của ràng buộc sinh ra chỉ tăng lên khi lời nhắc được làm giàu bằng thông tin UML và áp dụng học theo mẫu, cho thấy chất lượng đầu ra phụ thuộc chặt chẽ vào ngữ cảnh và chiến lược nhắc lệnh. Quan trọng hơn, các kết quả do LLMs đưa ra không phải là bằng chứng hình thức cho tính đúng đắn của mô hình. Các thuộc tính như tính nhất quán,

tính thỏa mãn ràng buộc, tính không xung đột của chính sách bảo mật hoặc tính bảo toàn ngữ nghĩa giữa mô hình nguồn và mô hình đích vẫn cần được kiểm tra bằng các cơ chế hình thức hoặc bán hình thức. Do đó, LLMs có thể hỗ trợ quá trình kiểm chứng ở mức tương tác và giải thích, nhưng không thể thay thế các công cụ kiểm chứng hình thức như OCL checker, Event-B, Rodin hoặc ProB.

Từ các phân tích trên có thể thấy, AI/LLM tạo ra một hướng tiếp cận mới và có tiềm năng lớn cho mô hình hóa miền và sinh phần mềm. Tuy nhiên, hướng tiếp cận này hiện vẫn chủ yếu hỗ trợ các tác vụ sinh nháp, gợi ý, diễn giải và tự động hóa một phần ở giai đoạn đầu. Các hạn chế chính bao gồm thiếu cơ sở ngữ nghĩa hình thức, khó kiểm soát quá trình sinh kết quả, thiếu bảo đảm về tính nhất quán, và chưa có cơ chế rõ ràng để tích hợp nhiều mối quan tâm trong một mô hình miền thống nhất.

Trong bối cảnh đó, hướng nghiên cứu của luận án có tính bổ sung cho các tiếp cận AI/LLM. Thay vì sinh trực tiếp mô hình hoặc mã nguồn bằng LLMs, luận án tập trung xây dựng các kỹ thuật biểu diễn và chuyển đổi mô hình có ngữ nghĩa rõ ràng, trong đó các khía cạnh cấu trúc, ràng buộc, hành vi và bảo mật được đặc tả bằng các DSL theo mối quan tâm và được hợp nhất trong UDML. Cách tiếp cận này cho phép mô hình miền đóng vai trò là biểu diễn trung tâm, có khả năng thực thi, hỗ trợ kiểm chứng hình thức và làm cơ sở cho các phép chuyển đổi mô hình cũng như sinh tự động phần mềm.

Vì vậy, AI/LLM có thể được tích hợp vào phương pháp của luận án như một thành phần hỗ trợ, thay vì thay thế các kỹ thuật mô hình hóa và kiểm chứng. Cụ thể, LLMs có thể hỗ trợ khởi tạo mô hình miền từ tài liệu yêu cầu, gợi ý các thực thể, thuộc tính, quan hệ, hành động nghiệp vụ, vai trò người dùng và tài nguyên cần bảo vệ. LLMs cũng có thể hỗ trợ sinh nháp các đặc tả DSL như CAP, AGL hoặc RBACDom từ mô tả nghiệp vụ. Các kết quả này sau đó cần được chuẩn hóa theo cú pháp và ngữ nghĩa của các DSL tương ứng, kiểm tra bằng các quy tắc hợp lệ cấu trúc, tích hợp vào UDML và kiểm chứng bằng các cơ chế hình thức trước khi sử dụng cho chuyển đổi mô hình và sinh phần mềm.

Như vậy, sự kết hợp giữa AI/LLM và phương pháp của luận án có thể được xem theo mô hình hỗ trợ có kiểm soát: LLMs hỗ trợ khai phá tri thức miền, sinh nháp mô hình và tương tác với người dùng; trong khi UDML, các

DSL theo mối quan tâm, các luật chuyển đổi mô hình và cơ chế kiểm chứng hình thức đảm nhiệm vai trò bảo đảm tính đúng đắn, tính nhất quán và khả năng thực thi của mô hình miền hợp nhất. Đây là hướng kết hợp tiềm năng giữa khả năng xử lý ngôn ngữ tự nhiên của AI và tính chặt chẽ của kỹ nghệ phần mềm hướng mô hình trong phát triển phần mềm theo thiết kế hướng miền.

2.6 Tổng kết chương

Chương này đã trình bày cơ sở lý thuyết và tổng quan tình hình nghiên cứu làm cơ sở lý luận cho luận án, tập trung vào DDD và vai trò trung tâm của DM trong toàn bộ quá trình phát triển phần mềm. Các khái niệm cốt lõi của DDD, bao gồm DM, UL và yêu cầu gắn kết chặt chẽ giữa mô hình và hiện thực triển khai, đã được phân tích nhằm làm rõ bối cảnh phương pháp luận của nghiên cứu.

Trên cơ sở đó, chương đã khảo sát và phân tích các tiếp cận hiện có trong việc biểu diễn DM, tích hợp cấu trúc, hành vi, ràng buộc và bảo mật thông qua UML/OCL và các DSL, cũng như các kỹ thuật hợp nhất mối quan tâm và chuyển đổi mô hình trong kỹ nghệ phần mềm hướng mô hình. Phân tích này cho thấy rằng mặc dù đã tồn tại nhiều tiếp cận nhằm hỗ trợ mô hình hóa, thực thi và kiểm chứng DM, các giải pháp hiện nay vẫn còn phân tán, thiếu một khuôn khổ thống nhất đặt DM làm trung tâm ngữ nghĩa, cho phép tích hợp đồng thời các mối quan tâm cấu trúc, hành vi và bảo mật, đồng thời hỗ trợ khả năng thực thi và chuyển đổi mô hình theo đúng tinh thần của DDD.

Khoảng trống nghiên cứu này cho thấy nhu cầu cần có các kỹ thuật biểu diễn và hợp nhất DM giàu ngữ nghĩa hơn, cùng với các cơ chế kiểm chứng và chuyển đổi mô hình bảo toàn ngữ nghĩa, nhằm thu hẹp khoảng cách giữa đặc tả miền và hiện thực phần mềm. Đây chính là động lực và nền tảng cho các kỹ thuật và phương pháp được đề xuất trong các chương tiếp theo của luận án.

Chương 3

KỸ THUẬT BIỂU DIỄN MÔ HÌNH MIỀN

Trong chương này, luận án đề xuất các kỹ thuật mở rộng và tích hợp mô hình miền (DM) nhằm đưa các khía cạnh hành vi và ràng buộc nghiệp vụ vào một mô hình miền hợp nhất có khả năng thực thi. Cụ thể, chương giới thiệu ngôn ngữ đồ thị hoạt động *Activity Graph Language* (AGL) như một DSL dựa trên chú thích để biểu diễn và gắn kết trực tiếp hành vi nghiệp vụ với các khái niệm miền, đồng thời trình bày mẫu chú thích ràng buộc *Constraint Annotation Pattern* (CAP) nhằm biểu diễn và tích hợp các ràng buộc OCL vào DM thực thi. Thông qua đó, các kỹ thuật đề xuất góp phần thu hẹp khoảng cách giữa miền nghiệp vụ và không gian kỹ thuật, đồng thời nâng cao khả năng biểu đạt, thực thi và kiểm chứng của DM trong DDD.

3.1 Giới thiệu

Trong các tiếp cận DDD hiện nay, các DSL dựa trên chú thích (aDSL) được nhúng trong OOPL đã được đề xuất nhằm hỗ trợ xây dựng DM thực thi [98, 122]. Các tiếp cận này cho phép mã hóa trực tiếp các khái niệm miền trong mã nguồn của hệ thống, hoặc kết hợp DM với các đặc tả ở mức trừu tượng cao hơn bằng UML và các DSL, làm cơ sở cho các phép biến đổi mô hình và sinh tạo tác phần mềm.

Tuy nhiên, các aDSL hiện có [9, 33, 46, 129] chủ yếu tập trung vào biểu diễn cấu trúc của DM. Trong khi đó, các khía cạnh hành vi, thường được mô tả bằng biểu đồ hoạt động hoặc biểu đồ máy trạng thái UML, cùng với các

ràng buộc OCL nghiệp vụ phức tạp, như các ràng buộc sử dụng `forAll`, `exists` hoặc các phép toán trên tập hợp, vẫn chưa được tích hợp một cách tường minh, có cấu trúc và hợp nhất trong DM thực thi. Hạn chế này làm suy giảm khả năng biểu đạt đầy đủ các quy tắc nghiệp vụ, đồng thời gây khó khăn cho việc hợp nhất các khía cạnh miền trong một mô hình hợp nhất nhất.

Luận án hướng đến việc mở rộng DM theo hướng tích hợp các khía cạnh hành vi, từ đó hình thành một DM hợp nhất có khả năng biểu đạt và hỗ trợ tốt hơn cho quá trình xây dựng phần mềm.

Trong bối cảnh đó, luận án đề xuất một ngôn ngữ hỗ trợ chuyên biệt cho việc tích hợp hành vi miền nhằm thu hẹp khoảng cách giữa DM và hiện thực của nó, đồng thời tạo điều kiện thuận lợi cho việc xây dựng phần mềm thông qua các phép biến đổi mô hình từ DM có tích hợp các đặc tả hành vi được biểu đạt bằng UML và DSL. Ngôn ngữ này được gọi là AGL (*Activity Graph Language*), được thiết kế như một aDSL tập trung vào việc biểu diễn các khía cạnh hành vi miền dưới dạng đồ thị hoạt động, cũng như tích hợp trực tiếp các đặc tả hành vi này vào DM hợp nhất. AGL được giới hạn trong một miền con của biểu đồ hoạt động của UML, dựa trên các mẫu mô hình hóa hoạt động cốt lõi [94]. Cách tiếp cận hướng mô hình cho DSL [69] được áp dụng, với UML/OCL [93, 94] dùng để đặc tả mô hình cú pháp trừu tượng và cú pháp cụ thể của AGL.

Để tích hợp AGL vào DM hợp nhất, aDSL đã được phát triển trước đó là DCSL [73] được sử dụng nhằm biểu diễn mô hình lớp hợp nhất. Mô hình lớp hợp nhất này được xem như một DM mở rộng kiến trúc phần mềm dạng mô-đun (MOSA) [75], trong đó các lớp miền, bao gồm các lớp hoạt động gắn với đồ thị AGL, được ánh xạ trực tiếp sang hiện thực kỹ thuật thực thi bằng JDA [75].

Mặc dù DCSL cung cấp một cơ chế biểu diễn ngắn gọn và gắn chặt với hiện thực triển khai cho các khái niệm cấu trúc của miền, các aDSL dạng này vẫn gặp hạn chế trong việc biểu diễn và tích hợp các ràng buộc OCL phức tạp. Các ràng buộc này đóng vai trò thiết yếu trong việc đặc tả chính xác mô hình miền bằng các quy tắc nghiệp vụ theo các nguyên lý của DDD [41, 93], như minh họa trong Bảng 2.1.

Các DSL gần đây như B-OCL [55] hỗ trợ mô hình hóa cấu trúc và một phần các ràng buộc nghiệp vụ; tuy nhiên, các tiếp cận này vẫn gặp nhiều thách thức trong việc chuyển đổi UML/OCL sang mã thực thi, tích hợp cơ chế kiểm tra OCL tại giai đoạn chương trình thực thi, cũng như bảo toàn tính đúng đắn của mô hình xuyên suốt vòng đời phát triển. Song song đó, các DSL nội sinh [8, 19, 62, 90, 138] nhúng trực tiếp lô-gic miền vào ngôn ngữ chủ như Java [117], qua đó hỗ trợ sinh kiểm thử tự động và xây dựng các hiện thực có thể kiểm chứng. Tuy nhiên, các tiếp cận này thường chỉ xử lý được một tập con nhỏ của OCL, chưa giải quyết đầy đủ việc biểu diễn các ràng buộc OCL nghiệp vụ phức tạp, cũng như chưa hỗ trợ sinh mã tự động ở mức toàn diện. Hạn chế cốt lõi nằm ở tính thỏa mãn của cách tiếp cận, do cú pháp của ngôn ngữ lập trình chủ thường là OOPL có giới hạn trong việc biểu diễn tự nhiên các khái niệm miền phức tạp.

Trước những hạn chế đó, luận án đề xuất một mở rộng aDSL nhằm biểu diễn và tích hợp các ràng buộc OCL nghiệp vụ phức tạp trực tiếp trong DM thực thi. Theo đó, mỗi ràng buộc OCL được ánh xạ sang một mẫu chú thích, gọi là CAP (*Constraint Annotation Pattern*), cho phép biểu diễn ngữ nghĩa ràng buộc bằng các cấu trúc chú thích gắn với các phần tử miền tương ứng.

CAP cung cấp một cơ chế tổ chức ràng buộc có tính hệ thống, giúp tích hợp chặt chẽ các ràng buộc vào DM thống nhất thực thi, qua đó tạo nền tảng cho việc hợp thành các khía cạnh cấu trúc, hành vi và ràng buộc trong cùng một DM. Cách tiếp cận này hỗ trợ cho các bước kiểm chứng và sinh tạo tác phần mềm.

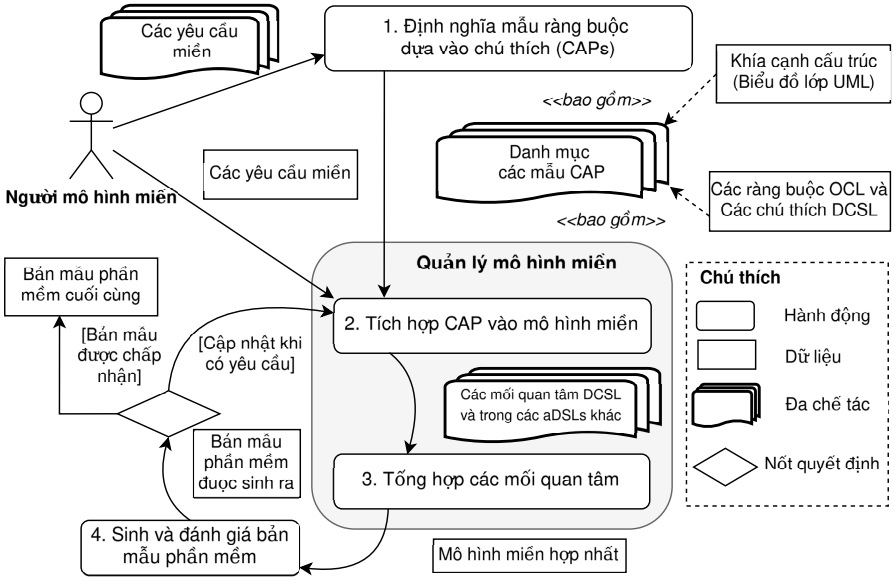
Các mục còn lại của chương này được cấu trúc như sau. Mục 3.2 trình bày kỹ thuật biểu diễn và tích hợp các ràng buộc OCL dựa vào mẫu chú thích (CAP) trình bày trong . Mục 3.3 trình bày kỹ thuật biểu diễn, tích hợp hành vi vào mô hình miền (AGL). Cuối cùng, tổng kết các kết quả đạt được trình bày trong Mục 3.4 của luận án.

3.2 Kỹ thuật tích hợp ràng buộc OCL vào mô hình miền

Phần này trình bày một kỹ thuật tích hợp các ràng buộc OCL nghiệp vụ phức tạp vào DM, gọi là CAP, nhằm thu hẹp khoảng cách giữa không gian bài toán và không gian kỹ thuật, đồng thời vẫn bảo toàn khả năng biểu đạt của DDD. Kỹ thuật này hỗ trợ tự động hóa việc kiểm soát ràng buộc và sinh mã trong quá trình phát triển phần mềm.

3.2.1 Tổng quan về phương pháp đề xuất

Đề xuất kỹ thuật tích hợp ràng buộc phức tạp vào DM theo DDD được mô tả trong Hình 3.1, gồm hai giai đoạn chính:



Hình 3.1: Kỹ thuật tích hợp ràng buộc phức tạp vào DM theo DDD.

Giai đoạn 1 tập trung vào việc xác định một tập hợp các mẫu dựa trên chú thích ràng buộc (*Constraint Annotation Patterns* – viết tắt là CAPs) bằng việc trích xuất và khái quát hóa chúng từ các yêu cầu miền hiện có (Bước 1 trong Hình 3.1). *Giai đoạn 2*, bao gồm ba bước còn lại trong Hình 3.1, nhằm mục tiêu tạo ra một bản mẫu phần mềm cuối cùng thỏa mãn các yêu cầu đầu vào của hệ thống đích. Các bước chính của phương pháp được tóm lược như sau.

Bước 1 – Xác định CAP: Dựa trên tập hợp các yêu cầu miền thu thập từ nhiều lĩnh vực khác nhau, bước này nhằm nhận diện các nhóm ràng buộc OCL, khái quát hóa từng nhóm và xây dựng một mẫu chung – gọi là CAP – để biểu diễn các ràng buộc OCL thuộc nhóm đó. Mỗi CAP bao gồm: (i) một biểu đồ lớp UML để đặc tả các khái niệm miền và quan hệ giữa chúng, (ii) một đặc tả OCL mô tả các quy tắc nghiệp vụ và bất biến, và (iii) các chú thích biểu diễn các ràng buộc OCL.

Bước 2 – Tích hợp CAPs vào DM: Đầu vào của bước này gồm tập các ràng buộc miền OCL và danh mục CAPs đã có. Mục tiêu là biểu diễn các ràng buộc OCL dưới dạng các CAP, qua đó ràng buộc DM. Mỗi ràng buộc dựa trên CAP được thể hiện thông qua phần mở rộng của DCSL.

Bước 3 – Hợp thành các mối quan tâm để tạo DM hợp nhất: Tất cả các mối quan tâm liên quan đến cấu trúc và hành vi của DM – được biểu diễn bởi các aDSL như DCSL, AGL – sẽ được hợp thành để tạo nên một DM hợp nhất.

Bước 4 – Sinh và đánh giá bản mẫu phần mềm: DM hợp nhất đóng vai trò làm bản thiết kế cho quá trình sinh tự động các bản mẫu phần mềm. Khung phần mềm JDA có thể được sử dụng để thực hiện việc sinh mã này. Bản mẫu thu được sẽ được bàn giao cho các bên liên quan để đánh giá. Nếu có phản hồi, DM được quản lý sẽ được cập nhật tương ứng. Quy trình sau đó được lặp lại, hỗ trợ chu trình cải tiến liên tục cho đến khi bản mẫu phần mềm đáp ứng đầy đủ các yêu cầu đã chỉ định.

3.2.2 Tích hợp mẫu CAP vào mô hình miền

Phần này trình bày phương pháp cho việc đặc tả và quản lý các CAP. Là phương pháp mở rộng DCSL nhằm hỗ trợ biểu diễn các ràng buộc dựa trên CAP. Ngoài ra, thực hiện xây dựng một danh mục CAP ban đầu, cho phép biểu diễn và tích hợp các ràng buộc OCL vào DM, từ đó tạo ra DM hợp nhất và khả thi để thực thi.

3.2.2.1 Khía cạnh cú pháp

Ý tưởng cốt lõi của một CAP là sử dụng một mẫu để biểu diễn các ràng buộc OCL trong DM. Về bản chất, một CAP là một mẫu được *"tham số*

hóa". Mỗi lần áp dụng CAP nhằm biểu diễn một ràng buộc OCL cụ thể tương ứng với việc gán giá trị cụ thể cho các tham số của mẫu.

Các tham số đầu vào của mỗi mẫu được định nghĩa thông qua một cơ chế chú thích. Thực hiện mở rộng DCSL để biểu diễn CAP, cho phép đặc tả và quản lý chính quy các CAP trong cùng một khung mô hình hóa.

Xét trên phương diện cú pháp, mỗi mẫu CAP được mô tả với các thành phần chính sau:

Tên mẫu: Mỗi CAP được gán một định danh duy nhất để hỗ trợ quản lý và truy xuất trong danh mục CAP hiện có.

Mô tả: Cung cấp giải thích ngắn gọn về mục đích và ngữ nghĩa của nhóm ràng buộc OCL mà CAP biểu diễn.

Mẫu tham số hóa: Mô tả biểu thức OCL mang tính tham số, tuân thủ cú pháp OCL và tương ứng với một nhóm ràng buộc OCL có cùng cấu trúc. Cụ thể, mẫu bao gồm:

- *Cấu trúc biểu đồ lớp* — mô tả phần liên quan của biểu đồ lớp dùng để xác lập ngữ cảnh OCL. Cấu trúc này có các tham số biểu diễn tên lớp, thuộc tính và quan hệ.
- *Mẫu OCL* — biểu diễn dạng tham số hóa của biểu thức OCL.
- *Đặc tả chú thích* — định nghĩa các tham số thông qua cơ chế chú thích có cấu trúc, được sử dụng trong biểu thức OCL của mẫu.

Ví dụ: Cung cấp minh họa cụ thể về cách một CAP được áp dụng trong thực tế.

Tóm lại, CAP cung cấp một cơ chế tái sử dụng và có tham số hóa để biểu diễn các ràng buộc OCL lặp lại trong các mô hình miền. Theo đó, Định nghĩa 3.1 tóm tắt khái niệm CAP.

Định nghĩa 3.1 (Mẫu chú thích ràng buộc)**.** *Mẫu chú thích ràng buộc – CAP được định nghĩa là một khuôn mẫu có tham số, dùng để nắm bắt cấu trúc cú pháp và ngữ nghĩa chung của một lớp các ràng buộc OCL. Một cách hình thức, CAP được biểu diễn như một bộ: CAP* $= (N, D, \mathcal{T})$ *trong đó:*

- N *là tên của mẫu,*

- *D là mô tả bằng văn bản nhằm giải thích mục đích và ngữ nghĩa của các ràng buộc OCL được biểu diễn bởi mẫu, và*

- *$\mathcal{T}$ là một tập các bộ $\langle P, T \rangle$, trong đó: P là tập các tham số biểu diễn các thành phần biến đổi của khuôn mẫu (ví dụ: thuộc tính, liên kết hoặc giá trị), được đặc tả thông qua các chú thích; T là một biểu thức OCL có tham số mô tả cấu trúc của ràng buộc OCL.*

Một thể hiện của CAP được tạo ra bằng cách thay thế các tham số P bằng các phần tử cụ thể của mô hình miền, từ đó sinh ra một ràng buộc OCL hợp lệ trong ngữ cảnh tương ứng.

3.2.2.2 Ví dụ và danh mục CAP đầu tiên

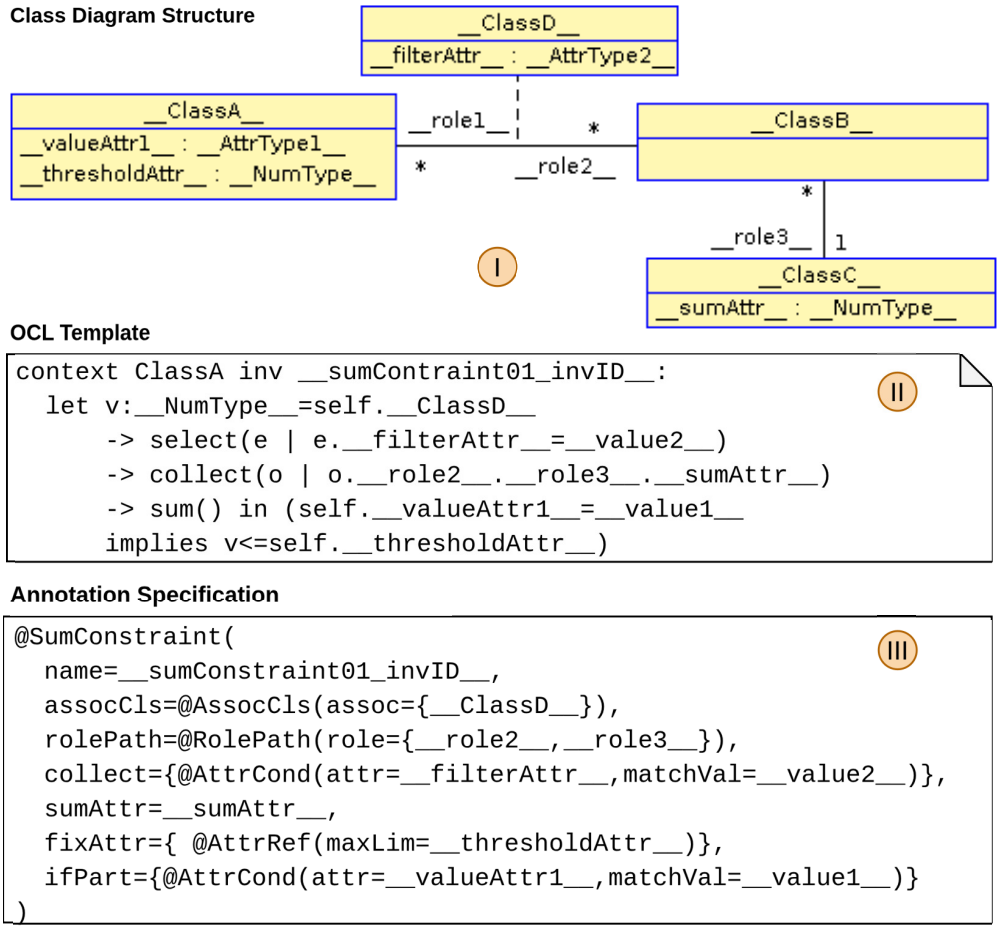
Trong phần này, luận án giới thiệu một mẫu CAP có tên SumConstraint nhằm minh họa cho phương pháp đề xuất. Sau đó, mô tả danh mục CAP đầu tiên được thu thập và đặc tả từ nhiều nguồn yêu cầu miền khác nhau.

Tên mẫu: SumConstraint

Mô tả: Mẫu này định nghĩa các ràng buộc OCL nhằm kiểm soát và quản lý số lượng, giá trị hoặc tài nguyên áp dụng các phép tổng hợp. Cụ thể, mẫu này đảm bảo rằng giá trị tổng hợp được lưu trữ trong một thực thể luôn bằng tổng các thành phần liên quan của nó. Mẫu được sử dụng để ngăn chặn sự sai lệch giữa dữ liệu dẫn xuất và dữ liệu nguồn, từ đó hỗ trợ đối soát và kiểm toán.

Mẫu tham số hóa: Biểu thức OCL tham số hóa của mẫu này được định nghĩa dựa trên các thành phần chính sau.

- *Cấu trúc biểu đồ lớp:* Hình 3.2 (nhãn I) minh họa biểu đồ lớp chứa ba lớp, các thuộc tính và các liên kết giữa chúng. Biểu đồ này xác lập ngữ cảnh cho mẫu ràng buộc OCL tương ứng.
- *Mẫu OCL:* Hình 3.2 (nhãn II) mô tả mẫu OCL dùng để sinh các ràng buộc thể hiện hạn chế sau: Tổng giá trị của thuộc tính `__sumAttr__` trên tất cả các đối tượng `o` trong một tập con được lọc từ tập các đối tượng `__ClassD__` liên kết với thể hiện `__ClassA__` hiện tại (`self`) được tính toán dựa trên tiêu chí



Hình 3.2: Đặc tả cho mẫu CAP SumConstraint.

lọc được xác định bởi biến `__filterAttr__`. Giá trị tổng được tính này sau đó được kiểm tra đối với `__thresholdAttr__` để đảm bảo điều kiện ngưỡng được thỏa mãn, và để bảo đảm rằng các đối tượng không vượt quá giới hạn cho phép. Lưu ý rằng tên các biến trong mẫu bắt đầu và kết thúc bằng "__". Tên biến `__sumConstraint01_invID__` cho biết đây là mẫu OCL loại 01 thuộc mẫu SumConstraint.

- *Đặc tả chú thích:* Mở rộng DCSL với các chú thích mới như minh họa trong Hình 3.2 (nhãn III), nhằm biểu diễn các tham số của mẫu OCL. Đặc tả chú thích này cho phép sinh ra ràng buộc OCL khi các tham số được gán giá trị cụ thể.

Ví dụ: Trên mô hình CourseMan, như thể hiện trong Hình 1.2, để minh họa mẫu này. Ràng buộc liên quan đảm bảo rằng số tín chỉ tối đa một sinh viên được phép đăng ký là một ngưỡng có giá trị xác định (12 tín

chỉ) tín chỉ mỗi học kỳ khi sinh viên đang trong diện cảnh báo học vụ. Điều này giúp sinh viên tập trung cải thiện điểm trung bình (GPA). Ràng buộc được biểu diễn bằng OCL như sau:

```
context Student
    inv courseMan_inv06_ProbationRules:
    let v:Integer=self.enrolment
    ->select(e|e.status=EnrolStatus::ACTIVE)
    ->collect(e|e.offering.module.credits)->sum()
    in self.overallStatus=AcademicStatus::PROBATION
    implies v<=12
```

Thực hiện định nghĩa đặc tả trong DCSL để sinh ra ràng buộc như sau:

```
@SumConstraint(
    name='courseMan_inv06_ProbationRules',
    assocCls=@AssocCls(assoc={'Enrolment'}),
    rolePath=@RolePath(role={'offering','module'}),
    collect={@AttrCond(attr='status',
      matchVal='EnrolStatus::ACTIVE')},
    sumAttr='credits',
    fixAttr={@AttrRef(maxLim = 12)},
    ifPart={@AttrCond(attr='overallStatus',
    matchVal='AcademicStatus::PROBATION')}
)
class Student {...}
```

Một mẫu CAP biểu diễn một họ ngữ nghĩa của các ràng buộc OCL có cấu trúc liên quan với nhau. Mỗi CAP có thể bao gồm nhiều biến thể khuôn mẫu OCL có tham số, tương ứng với các dạng cú pháp khác nhau của các ràng buộc trong cùng một lớp ngữ nghĩa.

Ví dụ, mẫu SumConstraint bao gồm một số biến thể khuôn mẫu OCL tương ứng với các nhóm ràng buộc dựa trên phép tổng hợp khác nhau. Khuôn mẫu OCL được minh họa trong Hình 3.2 là một trong các biến thể như vậy. Các biến thể khác có thể biểu diễn các ràng buộc tương tự nhưng có bổ sung các điều kiện lọc hoặc các vị từ ngữ cảnh.

Mặc dù các biến thể này cùng chia sẻ một tên mẫu, chúng khác nhau về cấu trúc biểu diễn. Trong quá trình mô hình hóa, biến thể khuôn mẫu phù hợp sẽ được lựa chọn và các tham số của nó được khởi tạo thông qua chú thích DCSL. Người dùng xây dựng các ràng buộc bằng cách khởi tạo chú thích CAP tương ứng với các giá trị tham số phù hợp. Từ đó, ràng buộc

OCL cụ thể sẽ được sinh tự động từ biến thể khuôn mẫu đã chọn. Mẫu SumConstraint bao gồm nhiều biến thể mẫu OCL tương ứng với các nhóm ràng buộc OCL khác nhau. Mẫu OCL minh họa trong Hình 3.2 là một trong những biến thể đó. Chẳng hạn, hãy xét một ràng buộc OCL khác trong DM được thể hiện ở Hình 1.2, như mô tả dưới đây. Ràng buộc này đảm bảo rằng sinh viên có GPA dưới 2.5 trong học kỳ trước chỉ được phép đăng ký tối đa 20 tín chỉ được minh họa bởi Đặc tả 3.1.

```
1     context TermRecord
2         inv courseMan_inv17_LowGPACreditRestriction:
3         self.prevSemGpa<2.5 implies
4         self.student.enrolment
5         ->select(e| e.offering.term=self.term)
6         ->collect(e|e.offering.module.credits)->sum()<=20
```

Đặc tả 3.1: Một ràng buộc OCL cho ngữ cảnh `Student`

Mẫu chú thích CAP tương ứng của SumConstraint để sinh ra ràng buộc này được định nghĩa theo Đặc tả 3.2 như sau:

```
1  @SumConstraint(
2      name='courseMan_inv17_LowGPACreditRestriction',
3      assocCls=@AssocCls(assoc={'Enrollments'}),
4      rolePath=@RolePath(role={'student','offering','module'}),
5      rolePath2=@RolePath(role={'term','termRecord'}),
6      collect={@AttrCond(attr='curTerm',matchVal='true')},
7      sumAttr='credits',
8      fixAttr={@AttrRef(maxLim=20)},
9      ifPart={@AttrCond(attr='prevSemGpa',maxLim=2.5)}
10 )
11 class TermRecord { ... }
```

Đặc tả 3.2: Mẫu chú thích CAP tương ứng với OCL

Tiếp cận này đã xác định một danh mục CAP gồm mười mẫu, mỗi mẫu được đặc tả chính quy và được quản lý trong một khung mô hình hóa thống nhất. Mỗi CAP biểu diễn một họ ràng buộc, trong đó các ràng buộc có cùng mục đích ngữ nghĩa được gom nhóm và khái quát hóa thành một cấu trúc mẫu chung. Mỗi CAP bao gồm một tập hợp các kiểu nhằm phân biệt giữa các biến thể mẫu OCL khác nhau của cùng một mẫu CAP. Hai bất biến OCL được trình bày trong phần này được sinh bởi hai mẫu OCL tương ứng của mẫu SumConstraint.

3.2.2.3 Khía cạnh ngữ nghĩa

Ngữ nghĩa của CAP tương ứng trực tiếp với ngữ nghĩa của OCL, bởi mỗi thể hiện CAP có quan hệ một-một với một bất biến OCL trong DM. Sự tương ứng trực tiếp này bảo đảm rằng việc đánh giá bất kỳ ràng buộc CAP nào cũng cho kết quả lô-gic giống hệt với việc đánh giá bất biến OCL tương ứng, qua đó duy trì ngữ nghĩa OCL trong DCSL.

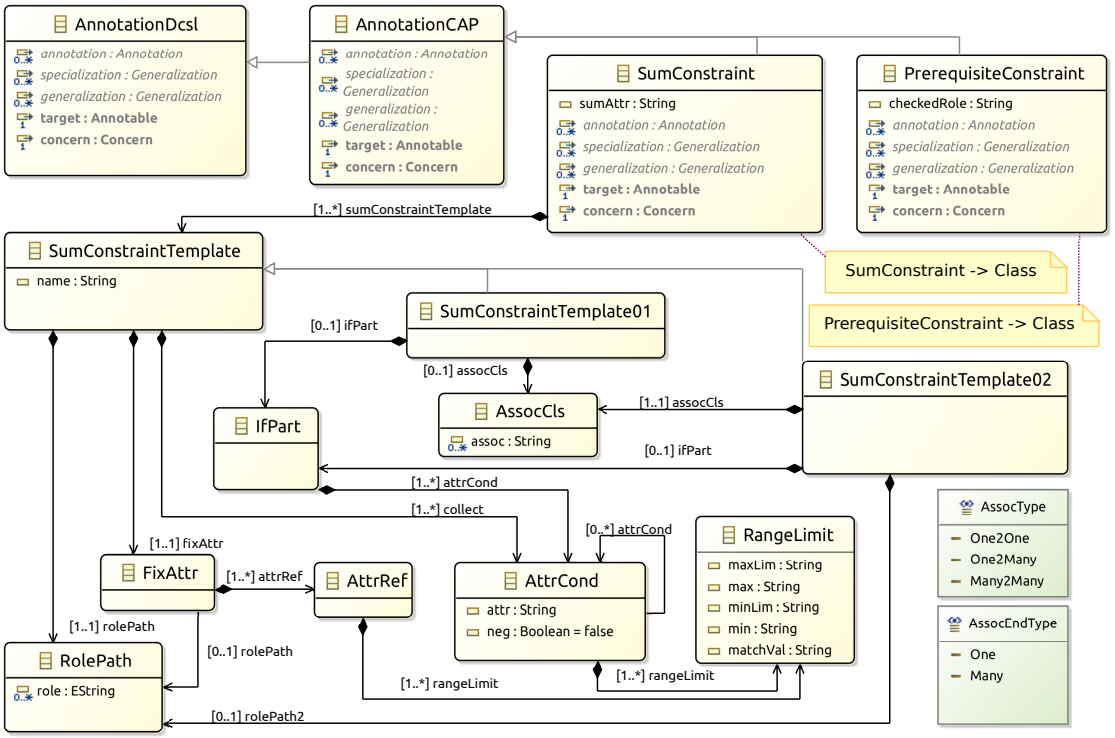
Định nghĩa 3.2 (Ứng dụng CAP). *Cho CD là một biểu đồ lớp biểu diễn mô hình miền; Cho OCL_{CD} là tập tất cả các bất biến OCL hợp lệ có thể biểu diễn trên CD; Cho CAP_i là một mẫu CAP được định nghĩa bởi mẫu OCL tham số hóa T_i với không gian tham số $\mathcal{D}_i$. Một ứng dụng CAP trên CD là quá trình gán một bộ tham số $d_i \in \mathcal{D}_i$ cho T_i sao cho mẫu OCL của CAP trở thành một bất biến OCL hợp lệ trên CD. Một cách hình thức, quá trình này được biểu diễn bằng ánh xạ sinh: $\mathcal{G} : (CAP_i, d_i) \longmapsto oclInv \in OCL_{CD}$, trong đó $\mathcal{G}(CAP_i, d_i)$ biểu diễn bất biến OCL cụ thể được sinh bằng cách thể hiện mẫu CAP là T_i với tập tham số d_i.*

Ví dụ: Xét ràng buộc `courseMan_inv06_ProbationRules`. Ràng buộc này được sinh bằng cách áp dụng mẫu SumConstraint, như đã giải thích trong Mục 3.2.2.2. Ánh xạ sinh tương ứng được xác định như sau:

- CD là mô hình miền CourseMan, được mô tả trong Hình 1.2.
- CAP_i là mẫu `SumConstraint`, trong đó:
  - + T_i là mẫu OCL của CAP được minh họa trong Hình 3.2,
  - + $\mathcal{D}_i$ là không gian tham số được xác định trên CD, và
  - + $d_i \in \mathcal{D}_i$ là bộ tham số tương ứng với đặc tả chú thích của bất biến `courseMan_inv06_ProbationRules`, như mô tả trong Mục 3.2.2.2.

Định nghĩa 3.3 (Tính đầy đủ của danh mục CAP). *Cho $\mathcal{C}_{CAP}$ là tập các CAP. Danh mục $\mathcal{C}_{CAP}$ được gọi là đầy đủ đối với mô hình miền CD nếu, với mọi bất biến $oclInv \in OCL_{CD}$, tồn tại một $CAP_i \in \mathcal{C}_{CAP}$ và một bộ tham số $d_i \in \mathcal{D}_i$ sao cho: $\mathcal{G}(CAP_i, d_i) = oclInv$.*

Hình 3.3 minh họa phần mở rộng của siêu mô hình DCSL, như được trình bày trong Hình 4.4, nhằm biểu diễn CAPs. Mỗi CAP được mô hình hóa bởi



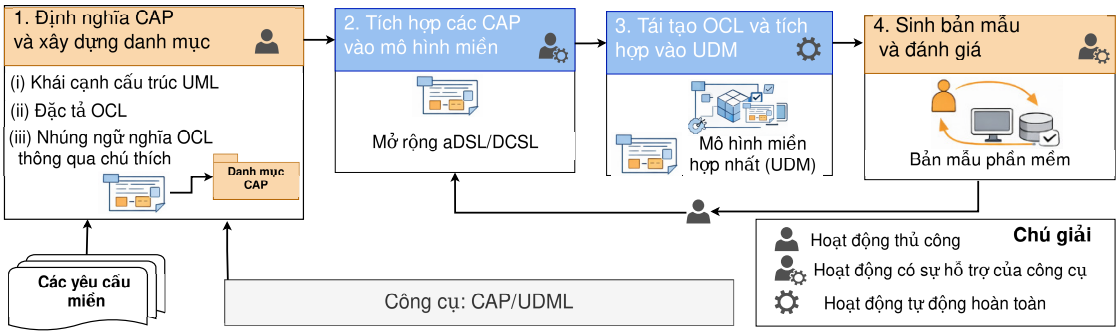
Hình 3.3: Mở rộng siêu mô hình DCSL để biểu diễn các CAP.

một siêu lớp (*Meta-class*) tương ứng, chẳng hạn như `SumConstraint` và `PrerequisiteConstraint`. Siêu lớp này kế thừa từ `AnnotationDcsl`, đóng vai trò là điểm mở rộng của DCSL dành cho CAPs. Mỗi siêu lớp CAP bao gồm một tập các siêu lớp biểu diễn các mẫu CAP tương ứng. Mỗi mẫu CAP gồm các tham số được đặc tả bằng các siêu lớp liên quan. Ví dụ, CAP `SumConstraint` bao gồm hai mẫu: `SumConstraintTemplate01` và `SumConstraintTemplate02`, cả hai cùng kế thừa một lõi chung được định nghĩa bởi `SumConstraintTemplate`. Các tham số của mỗi mẫu ánh xạ một-một với các thuộc tính của chú thích `SumConstraint` trong Hình 3.2.

3.2.3 Áp dụng mẫu CAP và sinh bản mẫu phần mềm

Mục này trình bày quy trình phương pháp áp dụng các mẫu chú thích ràng buộc (CAP) trong quá trình mô hình hóa hướng miền, cũng như việc sinh các bản mẫu phần mềm thực thi từ mô hình miền hợp nhất (UDM). Hình 3.4 minh họa tổng quan quy trình dựa trên CAP từ định nghĩa mẫu ràng buộc đến sinh bản mẫu phần mềm, kết hợp các giai đoạn thủ công, bán tự động và tự động.

Quy trình gồm bốn bước có liên kết lô-gic, trong đó các ràng buộc miền được chuyển đổi dần thành các bản mẫu phần mềm có thể thực thi. Tùy theo từng bước, các hoạt động có thể bao gồm các thao tác mô hình hóa thủ công, cấu hình bán tự động hoặc chuyển đổi hoàn toàn tự động, như được tóm tắt trong Hình 3.4.



Hình 3.4: Quy trình sinh bản mẫu phần mềm dựa trên CAP.

Bước 1 – Định nghĩa CAP và xây dựng danh mục. Giai đoạn đầu tập trung vào việc xác định các cấu trúc ràng buộc có thể tái sử dụng và tổ chức chúng thành một danh mục CAP. Đây chủ yếu là hoạt động mô hình hóa thủ công do các chuyên gia miền hoặc người thiết kế ngôn ngữ thực hiện. Các CAP được xây dựng bằng cách phân tích các ràng buộc OCL lặp lại trong nhiều mô hình miền khác nhau và khái quát hóa chúng thành các họ ràng buộc có thể tái sử dụng. Mỗi CAP được đặc tả hình thức bằng ba thành phần bổ trợ:

- Khía cạnh cấu trúc UML mô tả ngữ cảnh mô hình liên quan;
- Khuôn mẫu OCL có tham số đặc tả cấu trúc ngữ nghĩa của ràng buộc;
- Định nghĩa chú thích xác định không gian tham số cho việc khởi tạo mẫu.

Sau khi được định nghĩa, các khuôn mẫu CAP có thể được kiểm tra dựa trên ngữ cảnh cấu trúc để đảm bảo tính nhất quán ngữ nghĩa. Danh mục CAP thu được đóng vai trò như một kho mẫu ràng buộc có thể tái sử dụng cho nhiều mô hình miền khác nhau.

Bước 2 – Khởi tạo CAP trong mô hình miền. Ở giai đoạn thứ hai, các CAP được khởi tạo trong mô hình miền được biểu diễn bằng ngôn ngữ đặc tả miền dựa trên chú thích. Đây là một hoạt động bán tự động, kết hợp giữa việc đặc tả thủ công của người mô hình hóa và việc diễn giải tự động của môi trường mô hình hóa. Thay vì đặc tả trực tiếp các ràng buộc OCL, người mô hình hóa biểu diễn các ràng buộc miền dưới dạng các chú thích CAP (ví dụ: `@SumConstraint`). Mỗi chú thích tương ứng với một phép gán tham số cụ thể $d_i \in \mathcal{D}_i$ cho khuôn mẫu CAP T_i, theo ánh xạ áp dụng CAP được định nghĩa trong Định nghĩa 3.2.

Các khía cạnh cấu trúc của miền được biểu diễn bằng các lớp UML, trong khi ngữ nghĩa ràng buộc được đặc tả thông qua các chú thích CAP, liên kết các phần tử miền với các tham số của khuôn mẫu tương ứng. Cách tiếp cận này cho phép biểu diễn ràng buộc ở mức khai báo mà không cần thao tác trực tiếp với cú pháp OCL.

Bước 3 – Tái tạo OCL và tích hợp vào UDM. Sau khi các chú thích CAP được đặc tả, các ràng buộc OCL tương ứng được tái tạo tự động thông qua ánh xạ sinh (được trình bày trong Định nghĩa 3.3). Đây là một bước hoàn toàn tự động. Đối với mỗi khởi tạo CAP (CAP_i, d_i), khuôn mẫu OCL có tham số T_i được khởi tạo để tạo ra một ràng buộc cụ thể: $G(CAP_i, d_i) \longmapsto oclInv$.

Các ràng buộc sinh ra được tích hợp vào mô hình miền cùng với các đặc tả cấu trúc, tạo thành UDM. Do đó, UDM kết hợp hai thành phần bổ sung:

- Đặc tả cấu trúc của các thực thể và quan hệ miền;
- Các ràng buộc OCL hình thức được tái tạo từ các khuôn mẫu CAP.

Bước tái tạo tự động này bảo toàn ngữ nghĩa hình thức của OCL, đồng thời cho phép đặc tả ràng buộc bằng các trừu tượng mức cao hơn. Mô hình kết quả có thể được kiểm chứng bằng các cơ chế kiểm tra OCL chuẩn nhằm đảm bảo tính đúng đắn cú pháp và sự nhất quán với cấu trúc miền.

Bước 4 – Sinh bản mẫu và đánh giá. UDM sau khi được kiểm chứng đóng vai trò đầu vào cho quá trình sinh bản mẫu phần mềm theo hướng mô

hình. Từ mô hình này, một khung sinh mã có thể tự động tạo ra các bản mẫu phần mềm thực thi, bao gồm lớp miền, lô-gic kiểm tra ràng buộc, cấu hình lưu trữ và các thành phần giao diện người dùng.

Bản mẫu sinh ra cung cấp một biểu diễn vận hành của mô hình miền, cho phép đánh giá sớm các quy tắc miền và hành vi hệ thống. Việc đánh giá bản mẫu thường được thực hiện ở hai mức bổ trợ. Thứ nhất, tính đúng đắn ở mức cấu trúc và ràng buộc được kiểm tra thông qua các cơ chế kiểm chứng tự động. Thứ hai, các chuyên gia miền và các bên liên quan đánh giá bản mẫu để xác nhận rằng hành vi hệ thống phù hợp với các quy tắc nghiệp vụ mong muốn. Phản hồi từ quá trình đánh giá có thể dẫn đến việc điều chỉnh mô hình miền hoặc các khởi tạo CAP. Quy trình lặp này hỗ trợ việc đồng bộ liên tục giữa tri thức miền, đặc tả ràng buộc và các bản mẫu phần mềm thực thi.

3.3 Kỹ thuật tích hợp hành vi vào mô hình miền

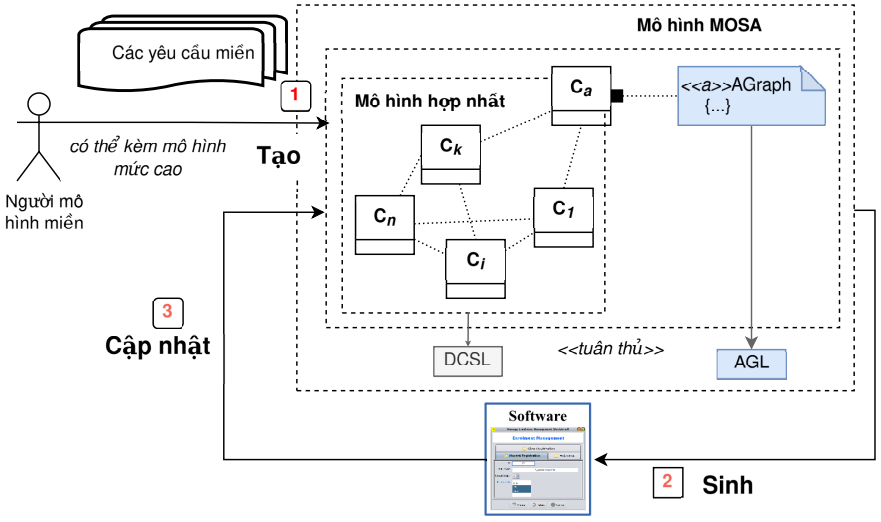
Trong mục này, trình bày nghiên cứu về kỹ thuật tích hợp khía cạnh hành vi vào DM nhằm hỗ trợ sinh tự động các bản mẫu phần mềm theo tiếp cận DDD. Cụ thể, nội dung tập trung vào ba vấn đề chính: (i) đề xuất một cơ chế tích hợp hành vi miền vào DM hợp nhất thông qua một aDSL, được định nghĩa để biểu diễn hành vi miền phục vụ quá trình tích hợp; (ii) trình bày phương pháp mô hình hóa hợp nhất cho phát triển phần mềm theo DDD, trong đó hành vi được gắn kết trực tiếp với DM; và (ii) làm rõ vai trò của kỹ thuật tích hợp hành vi trong việc nâng cao khả năng biểu đạt của mô hình miền.

3.3.1 Tổng quan về phương pháp đề xuất

Đề xuất của luận án về kỹ thuật biểu diễn tích hợp hành vi vào DM theo DDD được mô tả trong Hình 3.5, bao gồm ba bước lặp lại:

Trước hết, quá trình bắt đầu từ các yêu cầu miền, được biểu diễn bởi một DM thiết yếu, mô tả góc nhìn cấu trúc bằng các khái niệm miền, và các biểu đồ hoạt động UML, mô tả hành vi miền. Mục tiêu là kết hợp các yêu cầu miền đầu vào này thành một DM hợp nhất, được cấu thành từ một mô

hình DCSL và một mô hình AGL. Mô hình DCSL mở rộng DM thiết yếu nhằm liên kết góc nhìn cấu trúc với góc nhìn hành vi. Mô hình AGL biểu diễn các hành vi miền. Định nghĩa chi tiết hơn của mô hình DCSL được trình bày ở Mục 3.3.1.2. Các Mục 3.3.1.1 và 3.3.2 giải thích chi tiết cách định nghĩa mô hình AGL.



Hình 3.5: Kỹ thuật biểu diễn tích hợp hành vi vào DM theo DDD.

Thứ hai, DM hợp nhất được sử dụng làm đầu vào để tự động sinh phần mềm dựa trên giao diện đồ họa và mô-đun. Phần mềm sinh ra sẽ được trình bày cho chuyên gia miền để thu thập phản hồi.

Thứ ba, nếu có phản hồi, mô hình đầu vào sẽ được cập nhật và chu trình được lặp lại. Nếu chuyên gia miền hài lòng với các mô hình, chu trình kết thúc.

3.3.1.1 Tích hợp hành vi miền

Cơ chế được đề xuất để tích hợp hành vi miền vào DM dựa trên cấu trúc và ngữ nghĩa hành vi tại hai điểm. Thứ nhất, xác định một tập hành động thiết yếu cho mỗi lớp mô-đun sở hữu một lớp miền tương ứng. Các hành động thiết yếu này là hành động nguyên tử (trình bày chi tiết hơn trong Mục 3.3.2), cho phép thao tác trên các thể hiện của lớp miền. Thứ hai, hành vi miền được xem như sự hợp tác giữa các mô-đun trong MOSA [75]. Mỗi sự hợp tác mô-đun được điều phối bởi một mô-đun hợp thành và được mô tả bởi một mô hình hoạt động tương ứng. Mỗi mô hình hoạt động được ánh

xạ đến một lớp miền mới, gọi là lớp hoạt động, lớp này trực thuộc mô-đun hoạt động tương ứng.

Cơ chế trên cho phép sử dụng biểu đồ hoạt động UML để biểu diễn hành vi miền, nhưng có những giới hạn nhằm đảm bảo rằng các biểu đồ này phù hợp với ngữ nghĩa hành vi của mô-đun hợp thành điều phối sự hợp tác giữa các mô-đun. Để đạt được điều này, áp dụng phương pháp dựa trên mẫu, trong đó hành vi miền được đặc tả bằng biểu đồ hoạt động UML sử dụng các cấu trúc cơ bản tương ứng với năm mẫu mô hình hoạt động thiết yếu được trình bày trong [73]. Các mẫu này được đặt tên theo năm luồng hoạt động cơ bản: tuần tự, rẽ nhánh quyết định, phân nhánh song song, hợp nhất song song và gộp nhánh. Mô tả chi tiết hơn được trình bày trong Mục 3.3.3.

3.3.1.2 Mô hình lớp hợp nhất

Mô hình lớp hợp nhất là phần mở rộng của DM nhằm cho phép tích hợp các hành vi miền. Trong cách tiếp cận này, hành vi miền được mô tả bằng các mô hình hoạt động sử dụng các biểu đồ hoạt động UML.

Các lớp hoạt động chẳng hạn như lớp $\texttt{C}_\texttt{a}$ trong Hình 3.5 được thêm vào để biểu diễn từng hoạt động trong hành vi miền. Các lớp hoạt động này liên kết với mô hình hoạt động tương ứng, đóng vai trò như đồ thị hoạt động và đồng bộ hóa lô-gic hành vi của hoạt động với trạng thái hiện tại của DM. Mô hình lớp hợp nhất thu được có thể hiện thực bằng DCSL, và được gọi là mô hình hợp nhất DCSL.

Định nghĩa 3.4. (Mô hình lớp hợp nhất) *Cho trước mô hình hoạt động được đặc tả bằng biểu đồ hoạt động UML để biểu diễn hành vi miền.* ***Mô hình lớp hợp nhất*** *liên quan đến mô hình hoạt động là DM được mở rộng với các thành phần sau:*

- ***Lớp hoạt động:*** *lớp miền đại diện cho hoạt động,*
- ***Lớp thành phần dữ liệu (hay gọi tắt là lớp dữ liệu):*** *lớp miền đại diện cho mỗi kho dữ liệu,*
- ***Lớp thành phần điều khiển (hoặc lớp điều khiển):*** *nắm bắt trạng thái miền cụ thể của nút điều khiển. Lớp điều khiển đại diện (không đại diện) nút điều khiển được đặt tên theo (phủ định) ví dụ, lớp quyết định (không quyết định), lớp tham gia (không tham gia), v.v, và*

- ***Liên kết dành riêng cho hoạt động:*** *liên kết giữa mỗi cặp lớp sau:*
  + *Lớp hoạt động và lớp hợp nhất.*
  + *Lớp hoạt động và lớp rẽ nhánh.*
  + *Lớp hợp nhất (rẽ nhánh) và lớp dữ liệu đại diện cho kho lưu trữ dữ liệu của một nút hành động được kết nối với nút hợp nhất (rẽ nhánh).*
  + *Lớp hoạt động và lớp dữ liệu không đại diện cho kho dữ liệu của nút hành động được kết nối với nút hợp nhất hoặc nút rẽ nhánh.*

Gọi chung các lớp dữ liệu và điều khiển của mô hình lớp hoạt động là các lớp thành phần.v

Lưu ý rằng biểu đồ biểu diễn trong định nghĩa trên không bao gồm tất cả các mối liên kết có thể có giữa các lớp thành phần. Nó chỉ tập trung vào các hoạt động cụ thể, tức là nói chung, trong phạm vi của luận án chỉ tập trung vào miền ngữ nghĩa hạn chế của biểu đồ hoạt động UML cho AGL.

Các liên kết đóng hai vai trò quan trọng: Đầu tiên, mô hình hóa rõ ràng các liên kết giữa các trạng thái theo miền cụ thể của các nút hành động. Thứ hai, chúng được sử dụng để kết hợp các mô-đun của các lớp dữ liệu và điều khiển vào cây chứa mô-đun hoạt động, do đó thúc đẩy mô-đun này trở thành mô-đun chính để quản lý toàn bộ hoạt động.

Điều kiện áp đặt cho cặp liên kết dành riêng cho hoạt động của lớp thứ tư xuất phát từ thực tế là không cần xác định rõ ràng mối liên kết giữa một lớp hoạt động và một lớp dữ liệu đại diện cho kho lưu trữ dữ liệu của một nút hành động được kết nối với hợp nhất hoặc nút rẽ nhánh. Một lớp dữ liệu như vậy được liên kết "gián tiếp" với lớp hoạt động, sử dụng hai liên kết: một là giữa nó và lớp hợp nhất hoặc rẽ nhánh (cặp lớp thứ ba), và lớp khác là giữa lớp hoạt động và lớp điều khiển này (cặp lớp đầu tiên hoặc cặp lớp thứ hai).

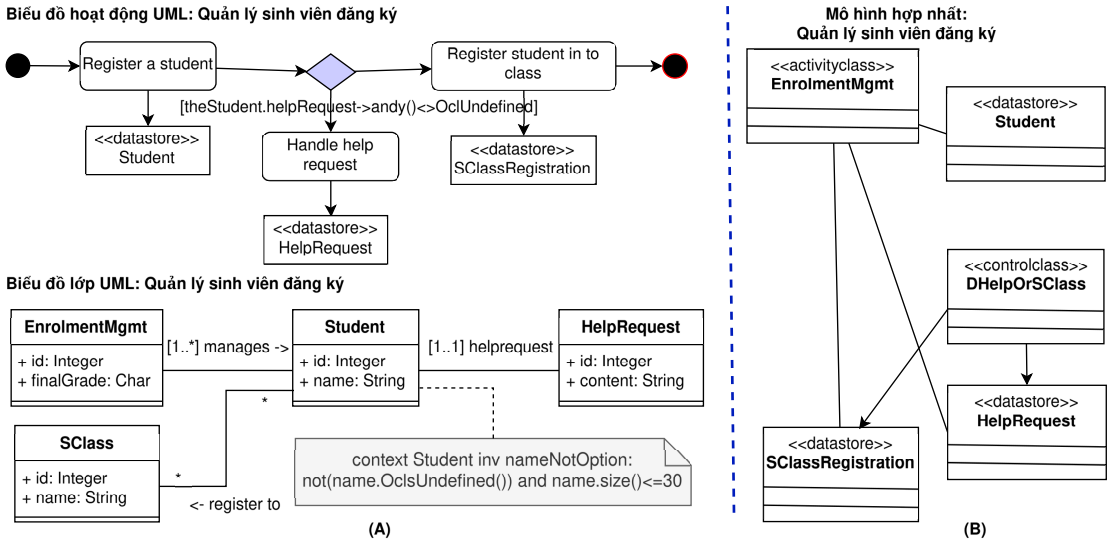
Định nghĩa 3.5. (Mô hình hợp nhất DCSL) *Mô hình hợp nhất DCSL là mô hình DCSL hiện thực hóa mô hình lớp hợp nhất như sau:*

- *Lớp miền C_a (được gọi là lớp miền hoạt động) để hiện thực hóa lớp hoạt động,*
- *Các lớp miền $C_1, ..., C_n$ để hiện thực hóa các lớp thành phần, và*

- *Để $C_{i1}, ..., C_{ik} \in \{C_1, ..., C_n\}$ hiện thực hóa các lớp thành phần không quyết định và không tham gia, sau đó $C_a, C_{i1}, ..., C_{ik}$ chứa các trường kết hợp nhận ra các đầu liên kết tương ứng của các liên kết dành riêng cho hoạt động có liên quan.*

Để đơn giản ký pháp, lớp hoạt động được dùng để chỉ lớp miền C_a và lớp thành phần để chỉ các lớp $C_1, ..., C_n$.

Ví dụ: Hình 3.6 (A) mô tả biểu đồ hoạt động và biểu đồ lớp UML của quản lý việc đăng ký (tên là CourseMan), trong khi Hình 3.6(B) minh họa mô hình lớp hợp nhất kết quả của hoạt động. Biểu đồ lớp hợp nhất bao gồm năm lớp miền và việc hiện thực hóa năm mối liên kết dành riêng cho hoạt động. Lớp `EnrolmentMgmt` đóng vai trò lớp hoạt động; `DHelpOrSClass` là lớp quyết định ghi nhận lô-gic quyết định miền chuyên biệt. Ba lớp còn lại là các lớp dữ liệu tương ứng với ba kho dữ liệu, đồng thời cũng là các lớp miền trong biểu đồ lớp UML. Ba lớp còn lại là các lớp dữ liệu tương ứng với ba kho dữ liệu và chúng cũng tương ứng với ba lớp miền trong biểu đồ lớp UML. Mô hình lớp hợp nhất bỏ qua các liên kết theo miền cụ thể.



Hình 3.6: (A: Bên trái) biểu đồ hoạt động và biểu đồ lớp UML của CourseMan cho hoạt động đăng ký; (B: Phải) Kết quả là mô hình lớp hợp nhất.

Ba trong số các liên kết này liên kết lớp hoạt động `EnrolmentMgmt` với các lớp dữ liệu, liên kết các mô-đun của chúng với cây chứa mô-đun hoạt động `ModuleEnrolmentMgmt`. Hai liên kết còn lại liên kết lớp quyết định

`DHelpOrSClass` với hai lớp dữ liệu `SClassRegistration` và `HelpRequest`, tương ứng với các kho dữ liệu được kết nối với hai nút hành động phân nhánh từ nút quyết định. Các liên kết này được coi là liên kết phụ thuộc yếu và được đưa vào trường hợp này để cho phép lô-gic quyết định được gói gọn bởi `DHelpOrSClass` để tham chiếu hai lớp dữ liệu.

3.3.2 Ngữ nghĩa hành động mô-đun

Trong mục này trình bày định nghĩa hình thức của *hành động mô-đun* dựa trên biểu đồ hoạt động của UML [94]. Định nghĩa này tập trung mô tả cấu trúc của hành động mô-đun cùng các tiền và hậu trạng thái của nó.

Hành động mô-đun được định nghĩa đệ quy, bắt đầu từ kiểu hành động nguyên thủy nhất gọi là hành động nguyên tử. Các hành động nguyên tử này sau đó được kết hợp để tạo thành chuỗi hành động nguyên tử và rộng hơn là hành động nguyên tử có cấu trúc, dẫn đến một đặc tả chính xác về ngữ nghĩa hành vi của các mô-đun trong MOSA.

3.3.2.1 Hành động nguyên tử

Mặc dù mỗi mô-đun có thể khác nhau, có thể quan sát rằng tồn tại một tập các hành vi nguyên thủy làm nền tảng cho mọi mô-đun. Các hành vi nguyên thủy này được khái quát hóa thành hành động nguyên tử.

Định nghĩa 3.6. (Hành động nguyên tử) ***Hành động nguyên tử*** *là hành vi mô-đun nhỏ nhất nhưng có ý nghĩa, được cung cấp cho tác nhân (con người hoặc mô-đun/hệ thống khác) thông qua khung nhìn nhằm thao tác trên các đối tượng miền của lớp miền tương ứng. Hành động nguyên tử được đặc trưng bởi:*

- `name`*: tên của hành động,*
- `preStates` *(tương ứng* `localPrecondition` *[94]): tập các trạng thái mà mô-đun hiện tại phải đang ở để hành động này có thể được thực hiện,*
- `postStates` *(tương ứng* `localPostcondition` *[94]): tập các trạng thái mà hành động đạt tới khi hoàn tất quá trình thực thi trên mô-đun hiện tại,*
- `fieldValSet` *(tương ứng* `input` *[94]): biểu diễn dữ liệu đầu vào của hành động. Đây là một tập các cặp* (f, v)*, trong đó* f *là tên của một trường*

miền (domain field) của lớp miền, và v là giá trị được hành động gán cho trường đó, và

- `output`*: lớp miền được sử dụng cho các hành động thao tác trên đối tượng, và rỗng đối với tất cả các hành động còn lại.*
  Mặc dù thuộc tính `name` *là duy nhất, để thuận tiện trình bày, thường liệt kê thêm* `postStates` *và* `fieldValSet` *cùng với* `name`*. Vì vậy, một hành động nguyên tử a được ký hiệu* $a = (o, s, i)$ *có thể được ký hiệu, trong đó o là* `name`*, s là* `postStates`*, và i là* `fieldValSet`*. Dấu chấm được dùng để truy cập thành phần, ví dụ:* $a.$`postStates` $= s$*.*

Các lưu ý về định nghĩa trên: Thứ nhất, các trạng thái mô-đun được dùng để trừu tượng hóa các tiền điều kiện-trạng thái và hậu-trạng thái của mỗi hành động, giúp tạo ra sự linh hoạt trong việc kết hợp các hành động dựa trên trạng thái để hình thành hành vi phức tạp hơn. Một trạng thái mô-đun biểu diễn trạng thái của MVC của mô-đun khi xử lý một hành động mô-đun. Một số trạng thái có thể diễn ra đồng thời, được gọi là trạng thái đồng thời và được biểu diễn bằng toán tử "+". Hành động nguyên tử có đúng một hậu trạng thái `postStates`; các hành động phức tạp hơn có thể có nhiều hậu trạng thái `postStates`. Thứ hai, một khía cạnh quan trọng của hành động là giá trị đầu vào. Do các hành động thao tác trên các trường miền, đầu vào được định nghĩa như một tập các cặp trường–giá trị (có thể rỗng), trong đó giá trị có thể được cung cấp bởi người dùng hoặc được sinh ra từ các hành động trước đó trong hành vi hợp thành. Thứ ba, mỗi hành động có tối đa một kiểu đầu ra, tương ứng với lớp miền của mô-đun hiện tại. Chỉ các hành động thao tác đối tượng mới sinh ra đầu ra này; các hành động khác không tạo ra giá trị kết quả.

Bảng 3.1 liệt kê các hành động nguyên tử cốt lõi. Nghiên cứu này chia các hành động thành hai nhóm. Nhóm thứ nhất bao gồm các hành động liên quan đến ngữ cảnh hoạt động tổng thể của mô-đun. Các hành động trong nhóm này gồm: `open`, `newObject`, `setDataFieldValues`, `reset`, và `cancel`. Các `postStates` của các hành động này bao gồm các trạng thái: `Opened`, `NewObject`, `Editing`, `Reset`, và `Cancelled` (tương ứng). Nhóm thứ hai bao gồm ba hành động thao tác đối tượng miền thiết yếu: `createObject`, `updateObject`, và `deleteObject`. Các `postStates` của những hành động này bao gồm các trạng thái: `Created`, `Updated`, và `Deleted` (tương ứng).

Bảng 3.1: Các hành động nguyên tử cốt lõi

name	preStates	postStates	Mô tả
`open`	{`Init`}	{`Opened`}	Mở giao diện của mô-đun để hiển thị lớp miền.
`newObject`	{`Opened`, `Created`, `Updated`, `Reset`, `Cancelled`}	{`NewObject`}	Xóa khỏi giao diện bất kỳ đối tượng nào đang được hiển thị và chuẩn bị giao diện để tạo một đối tượng mới.
`setDataField−Values`	{`NewObject`, `Editing`, `Created`, `Updated`, `Reset`, `Cancelled`}	{`Editing`}	Gán giá trị cho một tập con các trường dữ liệu của giao diện.
`createObject`	{`NewObject`, `Editing` + `ObjIsNotPresent`}	{`Created`}	Tạo một đối tượng mới từ dữ liệu được nhập trên giao diện. Đối tượng vừa tạo sẽ được hiển thị trên giao diện.
`updateObject`	{`Editing` + `ObjIsPresent`}	{`Updated`}	Cập nhật đối tượng hiện tại từ dữ liệu được nhập trên giao diện. Đối tượng đã được cập nhật vẫn được hiển thị trên giao diện.
`deleteObject`	{`Created`, `Updated`, `Reset` + `ObjIsPresent`, `Cancelled` + `ObjIsPresent`}	{`Deleted`}	Xóa đối tượng hiện tại. Đối tượng bị xóa sẽ được loại khỏi giao diện.
`reset`	{`Editing`}	{`Reset`}	Khởi tạo lại giao diện để hiển thị lại đối tượng hiện tại (hủy bỏ toàn bộ dữ liệu người dùng đã nhập).
`cancel`	{`NewObject`, `Editing` + `ObjIsNotPresent`}	{`Cancelled`}	Hủy thao tác tạo đối tượng mới (hủy bỏ toàn bộ dữ liệu người dùng đã nhập, nếu có).

Lưu ý từ Bảng 3.1 rằng chỉ có hành động `setDataFieldValues` yêu cầu `fieldValSet` được chỉ định làm đầu vào. Các hành động khác không yêu cầu đầu vào, do đó tập này là rỗng đối với chúng. Cũng lưu ý rằng hai trạng thái của mô-đun `ObjIsPresent` và `ObjIsNotPresent` có thể xảy ra đồng thời với bất kỳ một trong các trạng thái sau: `Editing`, `Reset`, và `Cancelled`. Ví dụ, trạng thái đồng thời `Editing` + `ObjIsPresent` có nghĩa là mô-đun hiện đang hiển thị một đối tượng trên giao diện và đối tượng này đang được người dùng chỉnh sửa. Ngược lại, `Editing` + `ObjIsNotPresent` có nghĩa là mô-đun hiện đang yêu cầu người dùng nhập dữ liệu cho một đối tượng mới; đối tượng này vẫn chưa được tạo ra.

3.3.2.2 Chuỗi hành động nguyên tử

Trong thực tế, các hành động nguyên tử cốt lõi được kết hợp theo trình tự để tạo thành những hành vi hữu ích hơn. Hành vi này, được gọi là chuỗi

hành động nguyên tử (*Atomic Action Sequence - ASE*), tương ứng với một kịch bản tương tác. Mô hình hóa chuỗi này bằng các hành động có cấu trúc trong biểu đồ hoạt động UML [94]. Ký hiệu `first` và `last` là hai hàm trả về phần tử đầu tiên và phần tử cuối cùng (tương ứng) trong một chuỗi.

Định nghĩa 3.7. (Chuỗi hành động nguyên tử và trạng thái có thể đạt tới) ***Chuỗi hành động nguyên tử*** $S = (a_1, \ldots, a_n)$ *được gọi là Chành động của mô-đun khi và chỉ khi* $a_i.\texttt{postStates} \subseteq a_{i+1}.\texttt{preStates}$ $(\forall\, a_i, a_{i+1} \in S)$. *Chuỗi* S *có các thuộc tính sau:*

- $S.\texttt{preStates} = \texttt{first}(S).\texttt{preStates}$
- $S.\texttt{postStates} = \texttt{last}(S).\texttt{postStates}$
- $S.\texttt{fieldValSet} = \texttt{first}(S).\texttt{fieldValSet}$
- $S.\texttt{output} = \texttt{last}(S).\texttt{output}$

Trạng thái của mô-đun s' *được gọi là* ***có thể đạt tới*** *từ hành động nguyên tử* a *nếu tồn tại chuỗi hành động nguyên tử* S *sao cho* $a = \texttt{first}(S)$ *và* $S.\texttt{postStates} = s'$. *Khi đó,* a *được gọi là hành động nguồn của* s'.

Rõ ràng, một hành động nguyên tử luôn có thể đạt đến hậu trạng thái của chính nó. Việc xác định các trạng thái có thể đạt tới cho các hành động nguyên tử được trình bày trong Bảng 3.1. Trước hết, các trạng thái có thể đạt tới của hành động `open` bao gồm `Opened`, `NewObject`, `Editing`, `Created`, `Updated`, `Deleted`, `Reset`, và `Cancelled`. Thứ hai, các trạng thái có thể đạt tới của `newObject` bao gồm `NewObject`, `Editing`, `Created`, `Reset`, và `Cancelled`. Ngoài ra, `newObject` không thể đạt tới `Updated` và `Deleted` vì hành động này được dành riêng cho việc tạo một đối tượng mới và không thể dẫn đến việc cập nhật hoặc xóa một đối tượng đã tồn tại. Thứ ba, các trạng thái có thể đạt tới của `setDataFieldValues` bao gồm `Editing`, `Created`, `Updated`, và `Reset`. Hành động này không thể đạt tới `NewObject`, `Deleted` và `Cancelled` vì nó chỉ liên quan đến dữ liệu nhập vào và không thể khởi tạo hoặc hủy việc tạo đối tượng, cũng như không thể dẫn đến việc xóa đối tượng. Cuối cùng, năm hành động còn lại mỗi hành động chỉ có một trạng thái có thể đạt tới, đó chính là trạng thái của riêng hành động đó. Các hành động này là "`stubs`" theo nghĩa chúng kết thúc mọi chuỗi ASE dẫn đến chúng.

3.3.2.3 Hành động nguyên tử có cấu trúc

Ở mức tổng quát hơn, trong phần này trình bày xác định một tập các ASE liên quan có thể hình thành một hành động nguyên tử có cấu trúc. Về bản chất, hành động này định nghĩa một hành vi tổng quát bao gồm nhiều kịch bản tương tác thay thế nhau (mỗi kịch bản được đặc tả bởi một ASE trong tập), và những kịch bản này thường được người dùng thực hiện (có thể đồng thời).

Định nghĩa 3.8. (Hành động nguyên tử có cấu trúc) ***Hành động nguyên tử có cấu trúc - SAA****, đối với hành động nguyên tử nguồn a và tập các hậu trạng thái $E = \{s_1, \ldots, s_n\}$ có thể đạt tới từ a, là tập $A = \{S : ASE \mid \texttt{first}(S) = a,\ S.\texttt{postStates} \subseteq E\}$, trong đó:*

- $A.\texttt{preStates} = a.\texttt{preStates}$
- $A.\texttt{postStates} = E$
- $A.\texttt{fieldValSet} = a.\texttt{fieldValSet}$
- $A.\texttt{output} = \bigcup_{S \in A}(S.\texttt{output})$

Ở dạng trừu tượng, viết $A = (a, \{s_1, \ldots, s_n\}, i)$ với $i = a.\texttt{fieldValSet}$. Nếu $i = \varnothing$ thì được lược bỏ thành phần này và chỉ viết $A = (a, \{s_1, \ldots, s_n\})$.

Rõ ràng, SAA khái quát hóa cả hành động nguyên tử lẫn ASE: một ASE là một SAA chỉ có một phần tử, trong khi một hành động nguyên tử a chính là SAA $(a, \{a.\texttt{postState}\})$. Hơn nữa, SAA ngắn gọn hơn đáng kể so với việc xây dựng một tập ASE: tất cả những gì cần làm là chỉ định hành động nguyên tử khởi đầu và các hậu trạng thái mong muốn.

3.3.3 Các mẫu hành vi miền

Phần này trình bày cách tiếp cận dựa trên mẫu nhằm tích hợp các hành vi miền vào DM. Mỗi hành vi miền, được mô tả ở mức trừu tượng cao bằng biểu đồ hoạt động UML và các câu lệnh dựa trên DM, được chuyển đổi thành một đặc tả gồm hai thành phần: (i) một phần của mô hình lớp hợp nhất, trong đó bổ sung các lớp hoạt động mới; và (ii) lô-gic đồ thị hoạt động tương ứng với hành vi đầu vào, cùng với các ánh xạ kết nối đồ thị này với mô hình lớp hợp nhất. Quá trình chuyển đổi được thực hiện thông qua việc

áp dụng các mẫu hành vi miền, được định nghĩa tương ứng với năm mẫu mô hình hóa hoạt động UML thiết yếu [73], đóng vai trò như các ràng buộc nhằm tinh chỉnh miền ngữ nghĩa của AGL.

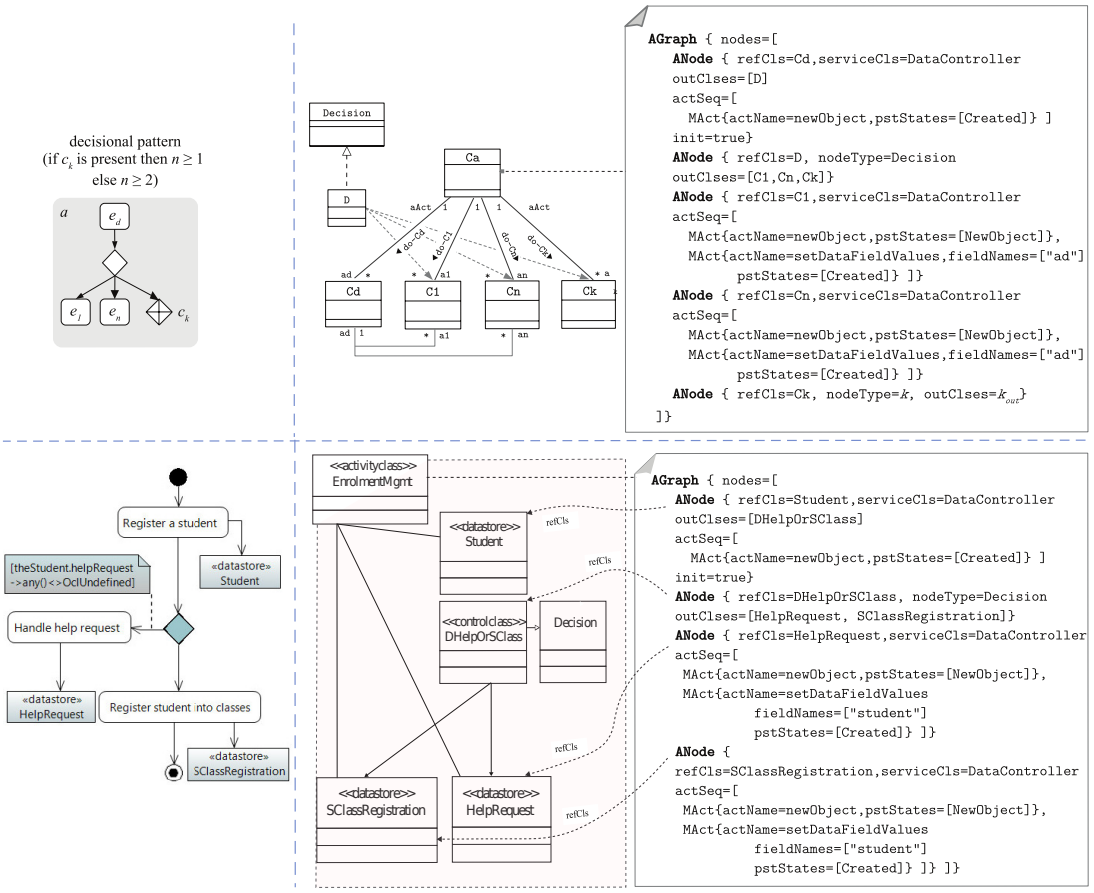
3.3.3.1 Đặc tả các mẫu hành vi miền

Nghiên cứu này đặc biệt quan tâm đến thiết kế các dạng mẫu [56, 67]. Để bảo đảm tính tổng quát của các mẫu, với mỗi dạng mẫu, luận án trình bày một biểu đồ hoạt động UML và một mô hình lớp hợp nhất cấu hình, hiện thực hóa mẫu tương ứng. Mô hình mẫu này là một mô hình DCSL hợp nhất đã được *tham số hóa*, trong đó các phần tử của các siêu khái niệm (không phải chú thích) được đặt tên theo vai trò tổng quát mà chúng đảm nhiệm trong mẫu. Để đơn giản hóa việc trình bày, các trường kết hợp và các phương thức miền cơ bản được lược bỏ khỏi biểu đồ của mô hình mẫu. Gắn với mỗi mô hình mẫu là một đặc tả đồ thị hoạt động trong AGL (được trình bày chi tiết trong Mục 3.3.4), nhằm mô tả lô-gic của đồ thị hoạt động tương ứng với hành vi đầu vào, đồng thời duy trì sự đồng bộ giữa trạng thái thực thi của hoạt động và trạng thái hiện tại của miền trong mô hình lớp hợp nhất.

Phương pháp đặc tả các mẫu hành vi miền được xây dựng dựa trên năm mẫu mô hình hóa hoạt động UML thiết yếu, bao gồm: *Sequential*, *Decisional*, *Forked*, *Joined* và *Merged*. Mỗi mẫu tương ứng với một biến thể của DM hợp nhất dành cho các hoạt động trong hệ thống CourseMan. Đối với mỗi mẫu, luận án cung cấp một ví dụ minh họa bao gồm DM hợp nhất đã được cấu hình tương ứng và một hoặc nhiều giao diện người dùng, được trình bày trong Mục 6.2.2. Tuy nhiên, để làm rõ cơ chế đặc tả và tích hợp hành vi miền, luận án tập trung phân tích chi tiết phương pháp đặc tả mẫu hành vi miền cho mẫu quyết định như một trường hợp đại diện.

Mẫu quyết định

Phần trên bên trái của Hình 3.7 minh họa biểu đồ hoạt động UML, trong khi phần trên bên phải thể hiện mô hình lớp hợp nhất được cấu hình mẫu. Ngoài lớp hoạt động $\texttt{C}_\texttt{a}$, mô hình này bao gồm năm lớp miền khác: $\texttt{C}_\texttt{d}$, $\texttt{D}$, $\texttt{C}_1$, $\texttt{C}_\texttt{n}$, và $\texttt{C}_\texttt{k}$, tương ứng với năm nút hành động. Đặc biệt, lớp $\texttt{C}_\texttt{k}$ là một lớp điều khiển được tham chiếu bởi nút điều khiển c_k của biểu đồ hoạt động. Lớp $\texttt{D}$ là lớp quyết định triển khai giao diện `Decision`. Vì lô-gic quyết định



Hình 3.7: Đặc tả mẫu hành vi miền cho mẫu quyết định.

có thể cần biết về các lớp miền liên quan (cụ thể là `C1`, `Cn`, và `Ck`), nên tồn tại các liên kết phụ thuộc yếu tùy chọn giữa `D` và các lớp này. Tùy thuộc vào yêu cầu miền, một số liên kết này có thể cần hoặc không cần được sử dụng. Lớp `Ca` được liên kết với bốn lớp miền còn lại theo mối quan hệ một – nhiều. Cũng cần lưu ý rằng liên kết với lớp `Ck` có thể đóng vai trò như một cầu nối đến các khối luồng hoạt động khác trong một biểu đồ hoạt động lớn hơn. Tuy nhiên, nếu c_k là một nút quyết định, thì `Ck` không có liên kết nào, và liên kết đến `Ck` sẽ được thay thế hoặc "mở rộng" thành một tập các liên kết nối trực tiếp `Ca` với các lớp miền của mô hình có chứa `Ck`. Trong mô hình mẫu, hai liên kết giữa `Cd` và `C1`, `Cn` cho thấy rằng cả `C1` và `Cn` đều biết đến `Cd` do việc truyền các thẻ đối tượng từ e_d sang e_1 và e_n thông qua nút quyết định.

Đặc tả AGL cho mẫu này bao gồm năm `ANode`. `ANode` đầu tiên tạo một đối tượng `Cd` mới, trong khi `ANode` thứ hai thực thi lô-gic quyết định. `ANode` thứ ba và thứ tư biểu diễn hai trường hợp quyết định: trường hợp thứ nhất tạo một đối tượng `C1` mới cho đối tượng `Cd` đã cho, và trường hợp thứ hai

(được lặp lại cho tất cả n) tạo một đối tượng `C_n` mới cho cùng một `C_d`. `ANode` thứ năm được sử dụng khi `C_k` được chỉ định và bao gồm hai biến, k và k_{out}, cả hai đều phụ thuộc vào `C_k`. Biến k đặc tả kiểu của nút điều khiển, trong khi biến k_{out} đặc tả mảng các lớp miền đầu ra của `C_k`.

3.3.3.2 Áp dụng các mẫu hành vi miền

Để thu được một đặc tả AGL cụ thể khi áp dụng một mẫu hành vi miền, về cơ bản thực hiện theo ba bước chính:

i. Biểu diễn biểu đồ hoạt động UML đầu vào và mô hình lớp hợp nhất được cấu hình mẫu dưới dạng các đồ thị hoạt động;

ii. Xác định việc ghép nối giữa mô hình mẫu và hoạt động nhằm xác định các `ANode`, thứ tự của chúng, và các lớp miền được tham chiếu bởi chúng. Việc tham chiếu từ các `ANode` đến các lớp miền (được thể hiện bằng các từ khóa `refCls` và `outCls`) cung cấp một ánh xạ giữa activity và mô hình lớp hợp nhất;

iii. Biểu diễn hành vi tổng quát của mô-đun miền đối với lớp miền được tham chiếu bởi mỗi `ANode` thông qua một SAA bằng từ khóa `actSeq`. Điều này đảm bảo rằng trạng thái thực thi của hoạt động được đồng bộ với trạng thái hiện tại của mô hình lớp hợp nhất.

Định nghĩa 3.9. (DM hợp nhất dựa trên AGL) *Cho một mô hình lớp hợp nhất chứa một tập không rỗng các lớp hoạt động, mỗi lớp trong số đó được gắn với một đặc tả AGL mô tả lô-gic đồ thị hoạt động của một hoạt động miền. Một DM hợp nhất D là sự kết hợp giữa mô hình lớp hợp nhất và các đặc tả AGL. Một phần mềm được sinh ra trong MOSA đối với D bao gồm một tập các mô-đun, mỗi mô-đun sở hữu một lớp miền trong D, và hành vi của hành động* `newObject` *của mỗi mô-đun sở hữu một lớp hoạt động sẽ bao gồm lô-gic được mô tả bởi đồ thị hoạt động được cấu hình bởi đặc tả AGL gắn với lớp đó.*

3.3.4 Ngôn ngữ hành vi miền dựa trên mô-đun

Phần này cung cấp một tổng quan ngắn gọn về AGL như một aDSL dùng để tích hợp các hành vi miền vào DM. Ngôn ngữ này cho phép tạo các đồ

thị hoạt động bằng cách cấu hình chúng trực tiếp trên DM thông qua các chú thích. Theo cách tiếp cận siêu mô hình hóa cho DSLs [69], một siêu mô hình cú pháp trừu tượng (*Abstract Syntax Meta-Model - ASM*) và một siêu mô hình cú pháp cụ thể dạng văn bản dựa trên chú thích (*Concrete Syntax Meta-Model - CSM*) cho AGL được định nghĩa. Chỉ các khía cạnh cú pháp của AGL được trình bày ở đây, vì ngữ nghĩa hành vi của ngôn ngữ được mô tả trong các Mục 3.3.2 và 3.3.3.

3.3.4.1 Cú pháp trừu tượng

Thực hiện định nghĩa các yêu cầu miền của AGL bằng cách áp dụng các điều khoản bao gồm (I), loại trừ (X), và ràng buộc (R) lên các yêu cầu của biểu đồ hoạt động trong UML như được mô tả chi tiết trong [94, p. 373]. Cụ thể, các điều khoản sau được áp dụng:

- (I1) một hành động của mô-đun, như đã thảo luận trong Mục 3.3.2, là một dạng đặc biệt của hành động [94, p. 441];
- (R1) mỗi nút thực thi [94, p. 403] thực hiện một chuỗi các hành động của mô-đun;
- (X1) không sử dụng biến trong hoạt động [94, p. 377];
- (X2) không sử dụng các hành động thay đổi [94, p. 469].

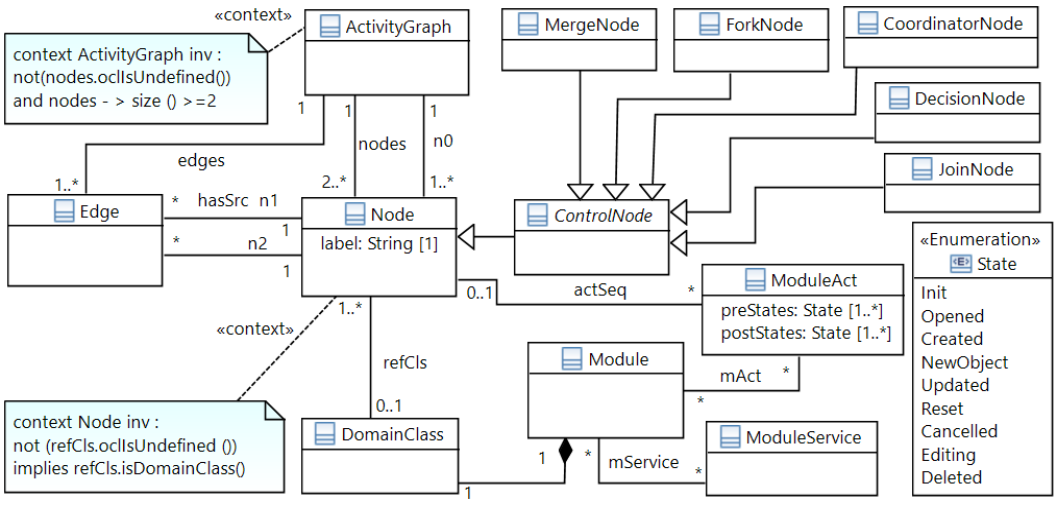
I1 và R1 là cần thiết để tích hợp đồ thị hoạt động vào MOSA. X1 và X2 loại bỏ việc sử dụng biến, vốn là một lựa chọn thay thế cho luồng đối tượng, phương tiện chính để truyền dữ liệu trong các hoạt động của UML [94, p. 377]. Mô hình hoạt động của AGL nắm bắt trạng thái hiện tại của hệ thống bằng cách tham chiếu trực tiếp đến mô hình lớp hợp nhất, thay vì sử dụng luồng đối tượng.

Hình 3.8 mô tả một mô hình siêu đơn giản hóa cho cú pháp trừu tượng của AGL. Cụ thể, siêu mô hình của AGL bao gồm các siêu khái niệm được điều chỉnh từ siêu mô hình UML dành cho mô hình đồ thị hoạt động: `ActivityGraph`, `Node`, `ControlNode`, và `Edge`. Một số ràng buộc được định nghĩa trên `Node` để hình thành miền đồ thị hoạt động của AGL, vốn hẹp hơn (như một tập con của) miền đồ thị hoạt động của UML.

Siêu khái niệm `Node` biểu diễn các nút hành động và có bốn thuộc tính. Thuộc tính thứ nhất là `label`, biểu diễn nhãn của nút. Thuộc tính thứ hai là `out`, được dẫn xuất từ liên kết `hasSrc(Edge, Node)` và ghi nhận các cạnh đi ra từ một `Node`. Hai thuộc tính tiếp theo đặc tả mô-đun được tham chiếu và gắn với nút này. Thuộc tính `refCls` tham chiếu đến lớp miền của mô-đun được tham chiếu, trong khi thuộc tính `serviceCls` đặc tả lớp `ModuleService` thực tế của mô-đun. Thuộc tính `serviceCls` tương ứng với vai trò đích của liên kết từ `Node` đến `ModuleMAct` và cho phép `Node` hiện tại thực thi các `ModuleMAct` được chỉ định bởi thuộc tính `actSeq`.

Siêu khái niệm `CoordinatorNode` được dùng để biểu diễn một nhóm tác vụ và không truyền dữ liệu của nó ra các cạnh đi ra. Thay vào đó, nó truyền dữ liệu đầu vào ban đầu ra ngoài, và các tác vụ thành viên tương tác với nhau để thực hiện lô-gic của nhóm. Giao diện người dùng của điều phối đóng vai trò là vùng chứa giao diện người dùng của các tác vụ thành viên, cho phép người dùng quan sát tổng thể nhóm.

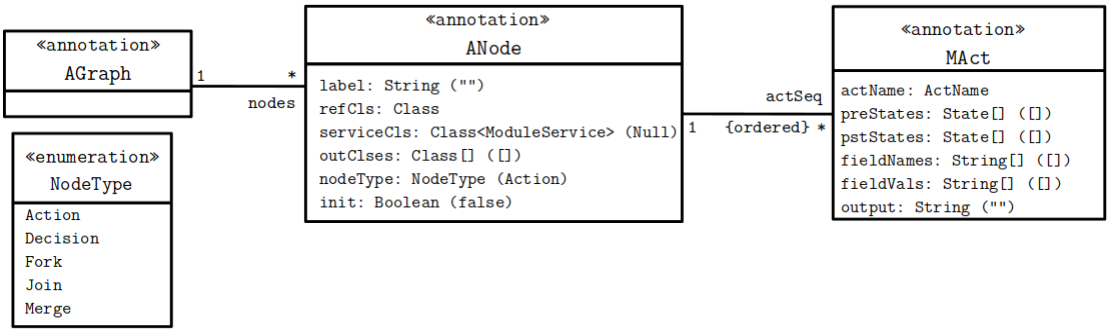
Siêu khái niệm `ModuleMAct` biểu diễn các hành động mô-đun kiểu SAA. Lưu ý rằng việc sử dụng một kiểu liệt kê gọi là `ActName` và một kiểu liệt kê gọi là `State` để biểu diễn tên hành động và hợp của các tiền và hậu trạng thái (tương ứng). Đặc biệt, `State` biểu diễn cả các trạng thái bình thường và trạng thái đồng thời (xem Mục 3.3.2.1).



Hình 3.8: Siêu mô hình giản lược cho cú pháp trừu tượng của AGL.

3.3.4.2 Cú pháp cụ thể dạng văn bản dựa trên chú thích

Cú pháp cụ thể của AGL được đề xuất theo cách tiếp cận xây dựng một siêu mô hình cú pháp cụ thể thông qua việc biến đổi từ siêu mô hình cú pháp trừu tượng. Hình 3.9 minh họa siêu mô hình CSM dựa trên chú thích, phù hợp để nhúng trực tiếp vào OOPL.



Hình 3.9: Cú pháp văn bản dựa trên chú thích của AGL, được hiện thực bằng Java.

Ngoài ra, nghiên cứu hướng đến việc biểu diễn một CSM súc tích, sử dụng một tập nhỏ các chú thích. Từ góc độ thực tiễn, một mô hình như vậy là mong muốn vì nó tạo ra cú pháp cụ thể gọn nhẹ, đòi hỏi ít nỗ lực hơn từ ngôn ngữ lập trình được sử dụng để xây dựng DM hợp nhất.

Để thực hiện việc chuyển đổi từ siêu mô hình ASM sang một siêu mô hình phù hợp cho biểu diễn dựa trên chú thích, quá trình được thực hiện qua hai bước. Ở bước thứ nhất, ASM được chuyển đổi thành siêu mô hình trung gian CSM_T, là một mô hình gọn nhẹ và thích hợp cho biểu diễn dựa trên chú thích. CSM_T bao gồm ba siêu khái niệm chính: đồ thị hoạt động (`AGraph`), nút hành động (`ANode`) và hành động mô-đun (`MAct`).

Ở bước thứ hai, CSM_T được chuyển đổi thành siêu mô hình cú pháp cụ thể dựa trên chú thích, được nhúng trực tiếp vào OOPL. Siêu mô hình này được xây dựng dựa trên ba siêu khái niệm cốt lõi của OOPL: `class`, `annotation` và `property`. Do được nhúng trực tiếp vào OOPL, cấu trúc của CSM xác định cấu trúc lõi của cú pháp văn bản AGL.

Ví dụ. Phần phía dưới bên phải của Hình 3.7 minh họa mô hình lớp hợp nhất của hoạt động quản lý đăng ký học trong hệ thống CourseMan. Đặc tả AGL cho hoạt động này được biểu diễn bởi một phần tử `AGraph`, được viết

bên trong một hộp ghi chú và gắn với lớp hoạt động `EnrolmentMgmt` trong mô hình lớp hợp nhất.

3.4 Tổng kết chương

Trong chương này, luận án đã đề xuất các kỹ thuật tích hợp hành vi và ràng buộc OCL vào DM nhằm mở rộng mô hình DCSL, cho phép biểu diễn DM đầy đủ thông tin bao gồm cấu trúc, hành vi và các ràng buộc nghiệp vụ phức tạp, qua đó nâng cao khả năng biểu đạt theo phương pháp DDD. Các kỹ thuật được đề xuất dựa trên aDSL để xây dựng DM hợp nhất, tạo nền tảng cho việc thiết kế và phát triển phần mềm trong các bối cảnh ứng dụng thực tế.

Kỹ thuật tích hợp hành vi vào DM được giới thiệu lần đầu trong bài báo đăng trên tạp chí *Information and Software Technology [V1]*. Ngôn ngữ AGL được xây dựng nhằm biểu diễn các hành vi miền thông qua hệ thống các mẫu điều khiển tương ứng với các cấu trúc cốt lõi của biểu đồ hoạt động UML. AGL cho phép gắn trực tiếp hành vi vào DM thông qua cơ chế aDSL và được hiện thực trong khung làm việc JDA, qua đó đảm bảo cả tính biểu đạt và khả năng thực thi của DM trong quá trình sinh phần mềm.

Kỹ thuật tích hợp các ràng buộc OCL vào mô hình miền được đề xuất lần trong luận án. Phương pháp này dựa trên cơ chế các mẫu chú thích ràng buộc, qua đó mở rộng năng lực biểu diễn của ngôn ngữ DCSL bằng cách cung cấp các khuôn mẫu ràng buộc OCL được tham số hóa. Cách tiếp cận này cho phép khái quát hóa và tái sử dụng các ràng buộc miền có cấu trúc tương đồng, đồng thời đảm bảo tính nhất quán trong đặc tả. Mỗi CAP được ánh xạ một–một với các bất biến OCL tương ứng và được tích hợp trực tiếp vào mô hình miền hợp nhất. Nhờ đó, mô hình không chỉ nâng cao khả năng biểu đạt các ràng buộc nghiệp vụ phức tạp mà còn hỗ trợ sinh tự động bản mẫu phần mềm có khả năng kiểm tra tính đúng đắn trên dữ liệu thực, qua đó đảm bảo đồng thời tính biểu đạt và tính khả thi trong thiết kế phần mềm theo phương pháp DDD.

Chương 4

PHƯƠNG PHÁP TÍCH HỢP CÁC MỐI QUAN TÂM VÀO MÔ HÌNH MIỀN

Trong chương này, luận án trình bày phương pháp tích hợp các mối quan tâm vào mô hình miền có khả năng thực thi theo tiếp cận thiết kế hướng miền, hình thành một mô hình hợp nhất. Phương pháp nhằm giải quyết bài toán phân mảnh mô hình khi các mối quan tâm (cấu trúc, hành vi, bảo mật) được đặc tả bằng nhiều DSL chuyên biệt theo mối quan tâm, đồng thời bảo đảm khả năng thực thi và kiểm chứng của mô hình miền. Luận án đề xuất hai cơ chế tích hợp: (i) tích hợp dựa trên siêu mô hình và (ii) tích hợp dựa trên cây cú pháp trừu tượng. Trên cơ sở đó, mô hình miền hợp nhất được đưa vào khung kiểm chứng ngữ nghĩa hình thức, nâng mô hình từ đặc tả mô tả lên một tạo tác có thể thực thi và được bảo đảm đúng đắn bằng lập luận toán học.

4.1 Giới thiệu

Thiết kế hướng miền (DDD) [41] nhấn mạnh việc đặt mô hình miền vào vị trí trung tâm của quá trình phát triển phần mềm, với kỳ vọng rằng các mô hình này có thể phản ánh chính xác tri thức miền và định hướng cho việc triển khai hệ thống. Trong thực tế, các mô hình miền hiếm khi chỉ tập trung vào một khía cạnh duy nhất. Thay vào đó, chúng thường bao hàm nhiều

mối quan tâm khác nhau, bao gồm khía cạnh cấu trúc, lô-gic hành vi, cũng như các mối quan tâm xuyên suốt như bảo mật [100, 103].

Trong bối cảnh đó, DM không chỉ là một đặc tả cấu trúc, mà còn là một tạo tác ngữ nghĩa nắm bắt các ràng buộc và quy tắc chi phối các trạng thái và hành vi hợp lệ của hệ thống. Khi DM đồng thời bao hàm nhiều mối quan tâm, thách thức đặt ra là xây dựng các DM vừa có khả năng thực thi vừa nhất quán về mặt ngữ nghĩa. Mặc dù các mối quan tâm cấu trúc và hành vi thường được mô hình hóa tách rời, ngữ nghĩa thực thi của chúng vốn phụ thuộc lẫn nhau; sự hiện diện của các mối quan tâm bổ sung như kiểm soát truy cập càng ràng buộc khi nào và bằng cách nào hành vi miền được phép thực thi. Khi thiếu một nền tảng ngữ nghĩa hợp nhất, việc tích hợp nhiều mối quan tâm có nguy cơ tạo ra bất nhất, mơ hồ hoặc các hành vi không mong muốn tại thời gian chạy.

Các biểu diễn truyền thống của DM, chẳng hạn biểu đồ lớp UML kết hợp với OCL [94], cho phép đặc tả cấu trúc và các ràng buộc tĩnh nhưng không cung cấp một ngữ nghĩa chuyển trạng thái tường minh để diễn giải sự tương tác giữa các mối quan tâm khác nhau. Tương tự, các DSL DSL ngoại sinh: có thể nâng cao khả năng biểu đạt ở mức cú pháp, song vẫn thiếu một nền tảng ngữ nghĩa thống nhất cho phép phân tích hành vi và kiểm chứng khi nhiều mối quan tâm cùng tồn tại trong mô hình. Một số hướng nghiên cứu gần đây [46, 135] đề xuất sử dụng DSL nội sinh và các aDSL nhúng trong OOPL [14, 33, 73, 98, 122] nhằm xây dựng các DM có khả năng thực thi. Mặc dù các tiếp cận này thu hẹp khoảng cách giữa mô hình và hiện thực, ngữ nghĩa thực thi chủ yếu vẫn được suy diễn từ ngôn ngữ lập trình chủ, dẫn đến khó khăn trong việc diễn giải và hợp thành ngữ nghĩa khi nhiều mối quan tâm được tích hợp đồng thời thông qua cơ chế chú thích.

Bài toán tích hợp nhiều mối quan tâm đã được thảo luận trong nhiều công trình về kết hợp DSL và siêu mô hình [92, 102, 123, 129, 131]. Tuy nhiên, các tiếp cận này chưa cung cấp được một nền tảng ngữ nghĩa thống nhất cho phép biểu diễn mô hình miền, và cũng chưa xác lập một nền tảng ngữ nghĩa hình thức cho các mối quan tâm khác nhau tương tác trong quá trình thực thi. Đặc biệt, ngữ nghĩa của hành vi miền dưới tác động của các ràng buộc bổ sung như: các chính sách bảo mật hoặc phân quyền vẫn còn mang tính ngầm định và phụ thuộc vào công cụ.

Trong các hệ thống hướng miền, các cơ chế RBAC [10, 61] không chỉ giới hạn quyền truy cập mà còn ràng buộc trực tiếp các chuyển trạng thái hợp lệ của DM; do đó, bảo mật trở thành một phần của ngữ nghĩa hành vi. Tuy nhiên, các phương pháp kiểm chứng hiện nay chủ yếu dựa trên kiểm thử hoặc cưỡng chế tại thời gian chạy, chưa cung cấp được các đảm bảo hình thức về tính đúng đắn của DM tích hợp bảo mật.

Mặc dù các kỹ thuật hình thức như Alloy, Petri nets hay ngữ nghĩa hoạt động dựa vào cấu trúc [68, 112, 116] cho phép kiểm chứng mạnh, chúng hiếm khi được tích hợp một cách có hệ thống vào quy trình mô hình hóa theo DDD hoặc DSL, đặc biệt là trong bối cảnh tích hợp mối quan tâm dựa trên chú thích.

Xuất phát từ các hạn chế trên, luận án đề xuất phương pháp tích hợp các mối quan tâm vào mô hình miền thông qua ngôn ngữ UDML (*Unified Domain Model Language*). Thực hiện định nghĩa một nền tảng ngữ nghĩa hình thức thống nhất cho UDML, cho phép tích hợp một cách có hệ thống các mối quan tâm miền dựa trên DSL dạng chú thích. Event-B [2, 3, 83, 101] được lựa chọn làm nền tảng ngữ nghĩa cho UDML, cung cấp một khung ngữ nghĩa cho các mô hình miền hợp nhất trong UDML. Các khía cạnh hành vi được đặc tả bằng AGL (trình bày trong Mục 3.3) xác định các chuyển trạng thái hợp lệ của miền và đóng vai trò là trụ cột ngữ nghĩa của mô hình miền hợp nhất. Trên cơ sở đó, luận án định nghĩa RBACDom, một mối quan tâm bảo mật được biểu diễn dưới dạng DSL dựa trên chú thích cho mô hình phân quyền theo vai trò (RBAC), trong đó các quy tắc ủy quyền được diễn giải như các điều kiện bảo vệ ràng buộc các chuyển trạng thái đã được xác định về mặt hành vi. Như vậy, các mối quan tâm về cấu trúc, hành vi và bảo mật tương ứng được đặc tả bởi DCSL, AGL và RBACDom được đồng bộ hóa trên một trạng thái hệ thống mối quan tâm chung, tạo thành một hệ chuyển trạng thái hợp nhất.

Luận án sử dụng hai phương pháp biểu diễn UDML nhằm khai thác tính bổ sung giữa hai hướng tiếp cận trong mô hình hóa miền. Phương pháp siêu mô hình cho phép đặc tả các mối quan tâm dưới dạng các DSL trên một nền tảng khái niệm thống nhất, qua đó cung cấp mức trừu tượng cao, bảo đảm tính chặt chẽ về ngữ nghĩa và tạo cơ sở cho phân tích, kiểm chứng hình thức cũng như mở rộng mô hình. Trong khi đó, phương pháp cây cú pháp trừu tượng (AST) nhấn mạnh khía cạnh hợp thành và hiện thực hóa, cho phép

tích hợp trực tiếp các DSL ở mức cấu trúc, hỗ trợ thao tác linh hoạt, kiểm tra tính nhất quán và sinh tự động mã nguồn, từ đó hình thành mô hình miền hợp nhất có khả năng thực thi. Trên cơ sở kết hợp hai phương pháp này, chương trình bày (i) một khung ngữ nghĩa nhằm giải thích ngữ nghĩa thao tác của mô hình miền hợp nhất; (ii) một DSL mang tên RBACDom để biểu diễn mối quan tâm phân quyền theo vai trò; và (iii) một phép chuyển đổi từ UDML sang Event-B, hỗ trợ mô phỏng và kiểm chứng hình thức thông qua nền tảng Rodin [2, 3, 83, 101].

Các mục còn lại của chương này được cấu trúc như sau. Mục 4.2 trình bày phương pháp tích hợp các mối quan tâm vào UDML dựa trên siêu mô hình và khung ngữ nghĩa và hỗ trợ công cụ cho các mô hình miền hợp nhất có thể thực thi trong UDML. Phương pháp tích hợp các mối quan tâm vào UDML dựa trên AST được trình bày trong Mục 4.3. Cuối cùng, tổng kết các kết quả đạt được được trình bày trong Mục 4.5 của luận án.

4.2 Phương pháp biểu diễn mô hình miền hợp nhất dựa vào siêu mô hình

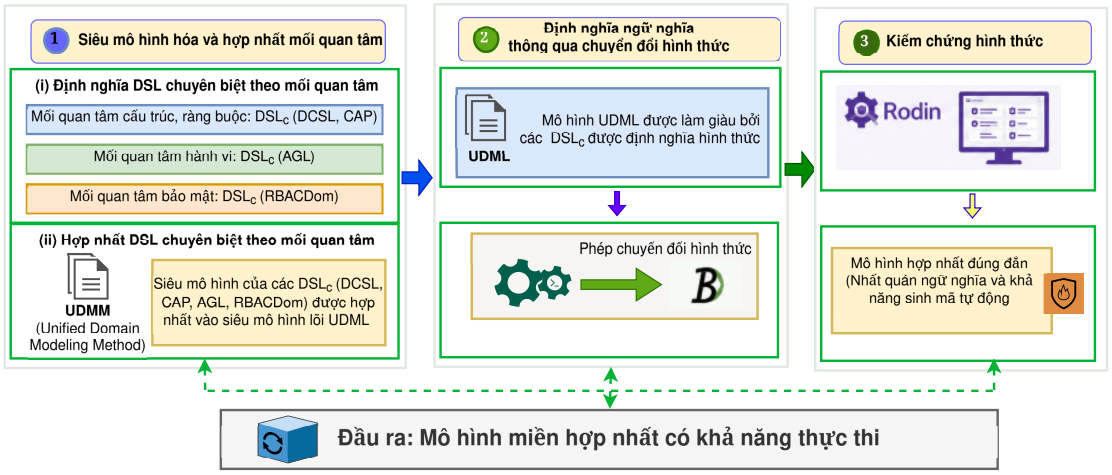
Trong phần này, luận án giới thiệu một khung ngữ nghĩa cho mô hình miền, trong đó nhiều mối quan tâm khác nhau, bao gồm mối quan tâm về cấu trúc, hành vi và bảo mật, có thể được tích hợp.

4.2.1 Tổng quan về phương pháp đề xuất

Hình 4.1 minh họa phương pháp được đề xuất, mang tên UDMM (Unified Domain Modeling Method), được tổ chức như một quy trình gồm ba bước. Phương pháp này nhận đầu vào là các đặc tả của các miền mối quan tâm và tạo ra đầu ra là các mô hình miền hợp nhất có khả năng thực thi và đã được kiểm chứng, được biểu diễn bằng UDML.

Bước 1: Siêu mô hình hóa và hợp nhất mối quan tâm. Với mỗi mối quan tâm, nhà thiết kế định nghĩa một DSL chuyên biệt theo mối quan tâm (*Concern-specific DSL* – DSL_c), để biểu diễn miền của mối quan tâm đó. Quá trình này bao gồm việc định nghĩa các siêu khái niệm trong miền dựa trên mô tả của nó, từ đó tạo ra một siêu mô hình cho DSL_c sau đó được

hợp nhất vào UDML (*Unified Domain Modeling Language*) thông qua cơ chế chú thích.



Hình 4.1: Tổng quan phương pháp đề xuất nhằm mô hình hóa miền hợp nhất.

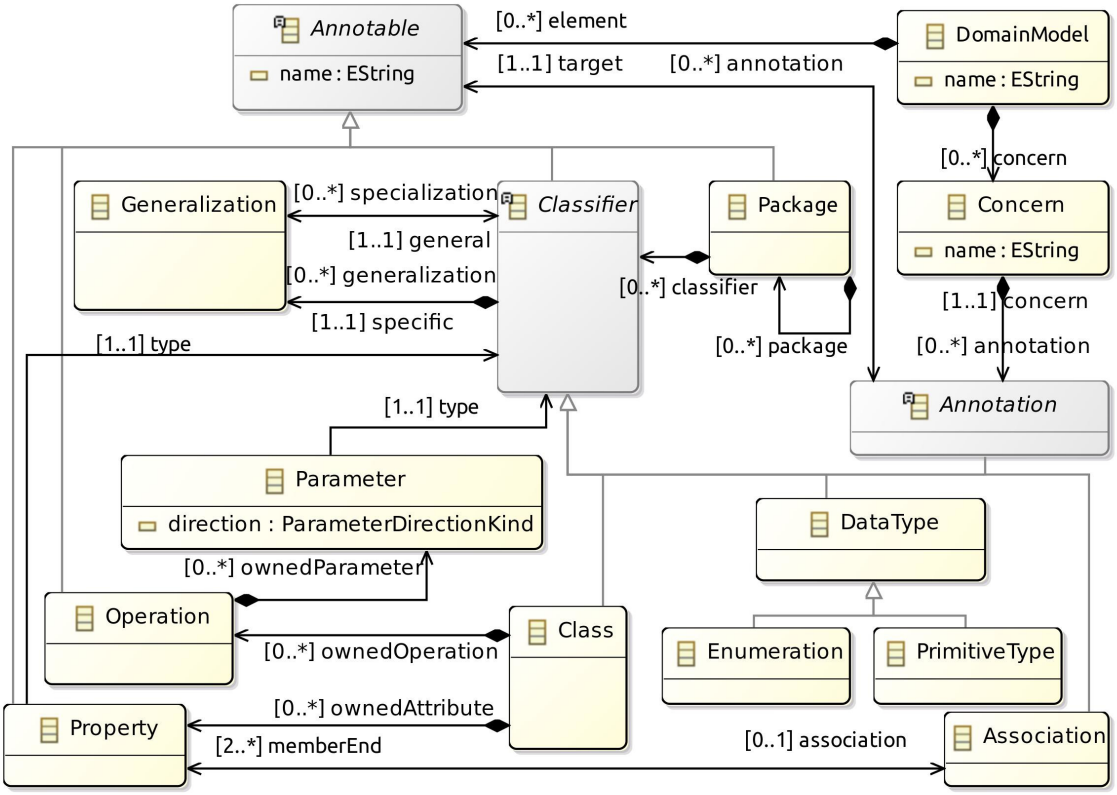
Bước 2: Định nghĩa ngữ nghĩa thông qua chuyển đổi hình thức. Bước này xác lập ngữ nghĩa hình thức cho mô hình miền hợp nhất. Một phép chuyển đổi được định nghĩa từ mô hình UDML được làm giàu bởi các DSL_c (bao gồm DCSL, AGL và RBACDom) sang Event-B, qua đó cung cấp một diễn giải toán học chính xác cho mô hình hợp nhất.

Trong phép chuyển đổi này, ngữ nghĩa hình thức của từng DSL_c được xác định dựa trên mô hình hệ chuyển trạng thái. Các đặc tả hành vi được biểu diễn bằng AGL quyết định các chuyển trạng thái hợp lệ, trong khi các ràng buộc cấu trúc từ DCSL và các ràng buộc ủy quyền từ RBACDom được diễn giải dưới dạng các bất biến và điều kiện bảo vệ nhằm giới hạn các chuyển trạng thái đó.

Bước 3: Kiểm chứng hình thức. Bước này thực hiện kiểm chứng hình thức bằng nền tảng Rodin [3]. Rodin tự động sinh ra các nghĩa vụ chứng minh tương ứng với ngữ nghĩa thực thi hợp nhất được làm giàu bởi các mối quan tâm đã được hợp thành. Việc giải quyết thành công các nghĩa vụ chứng minh này xác lập tính đúng đắn của mô hình miền hợp thành, bao gồm tính nhất quán xuyên mối quan tâm và việc bảo toàn các thuộc tính an toàn và bảo mật quan trọng.

Luận án định nghĩa siêu mô hình của UDML lõi dựa trên các siêu khái niệm của UML [94], như được minh họa trong Hình 4.2. Bốn siêu khái niệm

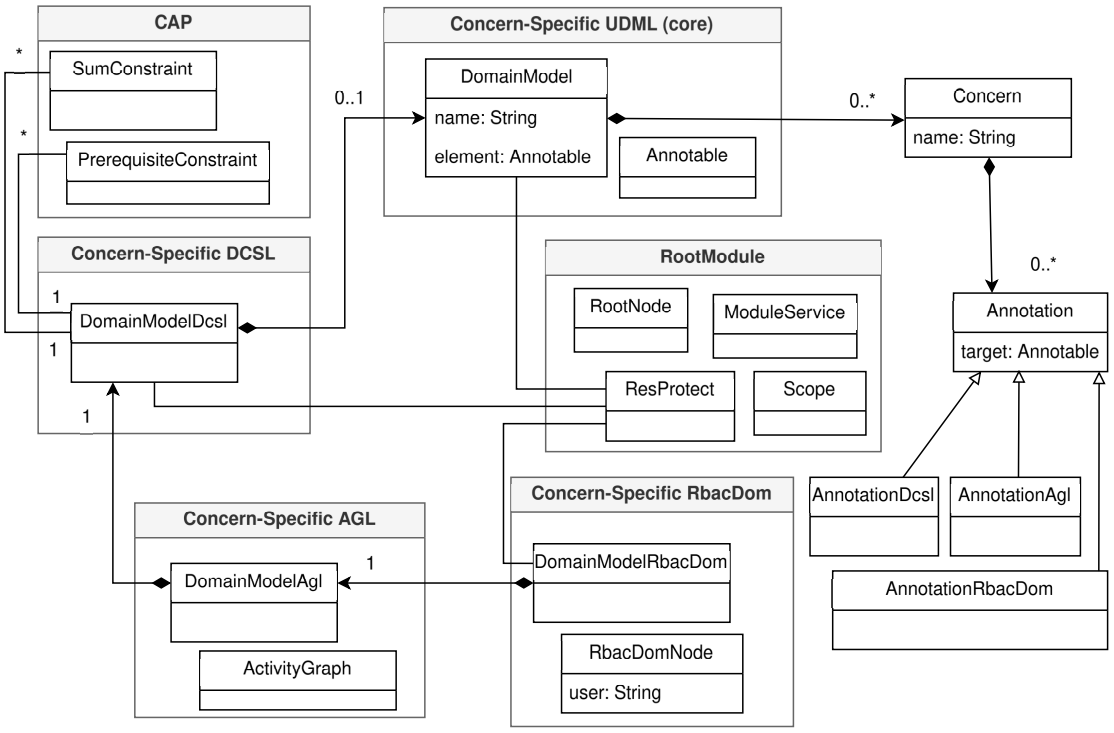
mới: `DomainModel`, `Concern`, `Annotation` và `Annotable` được bổ sung nhằm hỗ trợ việc hợp thành các mối quan tâm với DM và hợp nhất DSL_c trong UDML. Cụ thể, một `DomainModel` bao gồm các phần tử `Annotable` và các `Concern` là các thể hiện của siêu lớp `Concern`. Một `Concern` bao gồm các `Annotation`. Mỗi phần tử `Annotable` là một thể hiện của một siêu lớp trong siêu mô hình UML dành cho biểu đồ lớp, bao gồm `Package`, `Class`, `Property`, `Association` và `Operation`.



Hình 4.2: Siêu mô hình UDML lõi cho mô hình miền hợp nhất.

4.2.2 Biểu diễn mô hình miền hợp nhất

Hình 4.3 trình bày siêu mô hình rút gọn của UDML dùng để biểu diễn các mô hình miền hợp nhất. Siêu mô hình này được thiết kế có chủ đích nhằm hỗ trợ một cách diễn giải chính xác và được xác định rõ ràng cho các mô hình miền hợp nhất, trong đó các mối quan tâm về cấu trúc, hành vi và bảo mật có thể được tích hợp.



Hình 4.3: Siêu mô hình rút gọn của UDML.

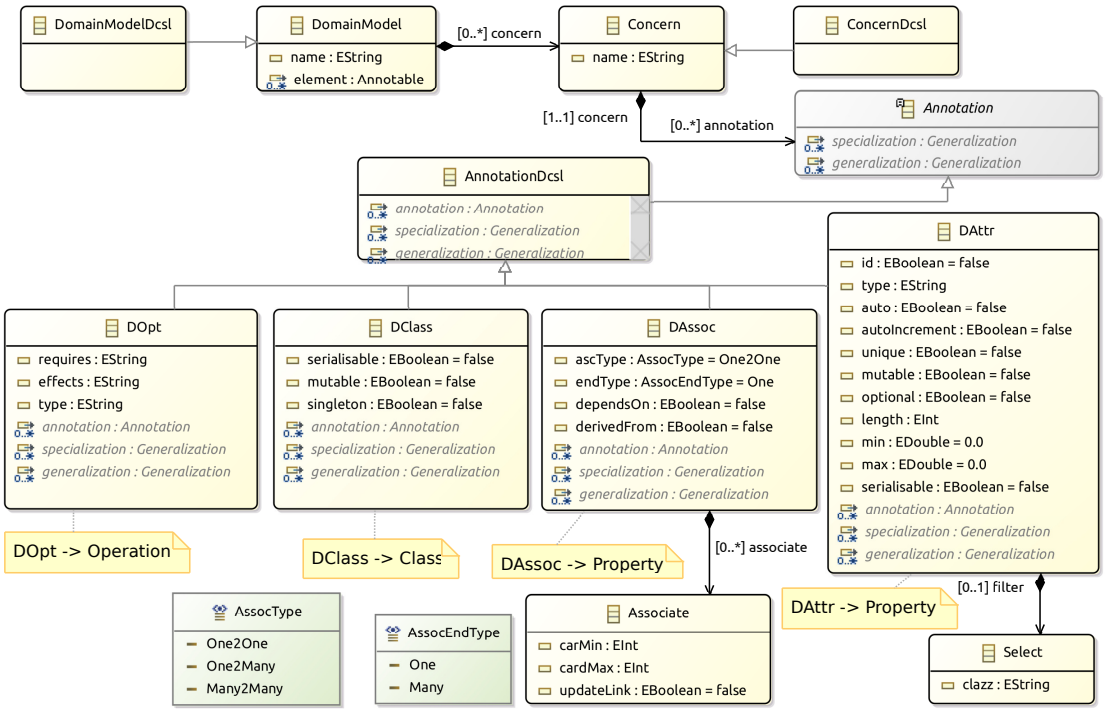
Siêu mô hình rút gọn của UDML cung cấp một lõi tối giản nhưng giàu khả năng biểu đạt của siêu khái niệm của UDML lõi đóng vai trò như một lớp hợp thành trung gian cho nhiều mối quan tâm khác. Cách biểu diễn này hỗ trợ tường minh việc hợp thành có hệ thống các mối quan tâm miền vào một mô hình miền hợp nhất.

Trong cách tiếp cận hợp thành dựa trên ngôn ngữ này, mỗi DSL_c, chẳng hạn như DCSL [73] cho các ràng buộc cấu trúc, hay AGL trình bày trong Mục 3.3 cho mô hình hóa hành vi, đều được định nghĩa bởi siêu mô hình riêng của nó và vẫn giữ tính độc lập về mặt khái niệm. UDML áp dụng cách tiếp cận hợp thành dẫn dắt bởi siêu mô hình, trong đó các mối quan tâm được tích hợp thông qua các tương ứng tường minh, được trung gian hóa bởi lõi UDML. Cách tiếp cận này làm cho các tương tác nhiều mối quan tâm trở nên tường minh, có thể phân tích và phù hợp cho việc diễn giải ngữ nghĩa hình thức. Cụ thể, các mô hình miền chuyên biệt theo mối quan tâm (ví dụ `DomainModelDcsl`, `DomainModelAgl` và `DomainModelRbacDom`) được liên kết với một thực thể `DomainModel` chung, qua đó bảo đảm một ngữ cảnh mô hình hóa thống nhất.

4.2.2.1 Tích hợp các ràng buộc cấu trúc

Để biểu diễn các ràng buộc cấu trúc và tích hợp chúng vào mô hình miền hợp nhất, luận án dựa trên siêu mô hình của DCSL trong [73], để xác định ngữ nghĩa hình thức của ngôn ngữ này nhằm cho phép việc tích hợp DCSL vào mô hình miền hợp nhất.

Cú pháp trừu tượng. Hình 4.4 trình bày siêu mô hình của DCSL, được giới thiệu trong công trình [73], đồng thời minh họa mối quan hệ ánh xạ giữa DCSL và UDML.



Hình 4.4: Siêu mô hình DCSL cho mô hình miền hợp nhất.

Siêu mô hình DCSL giới thiệu các siêu khái niệm lõi `DomainModelDcsl`, `ConcernDcsl` và `AnnotationDcsl`, cho phép các ràng buộc cấu trúc được đặc tả và hợp thành cùng với lõi UDML. Các ràng buộc cấu trúc cụ thể được biểu diễn thông qua các kiểu chú thích `DClass`, `DAttr`, `DAssoc` và `DOpt`, tương ứng được ánh xạ tới các khái niệm `Class`, `Property`, `Property` và `Operation`. Để bảo đảm tính đúng đắn của các ánh xạ này, các quy tắc hợp lệ hình thức được đặc tả dưới dạng các ràng buộc OCL gắn với siêu mô hình DCSL:

```
1     context DClass inv mapToCls:
2     self.target.oclIsKindOf(Class)
```

Các khái niệm khác, cùng với thuộc tính của tất cả các khái niệm được định nghĩa để biểu diễn các ràng buộc OCL thiết yếu, được tóm tắt trong Bảng 2.1.

Ngữ nghĩa. DCSL được xem như một DSL chuyên biệt theo mối quan tâm, dựa trên chú thích, nhằm nắm bắt khía cạnh cấu trúc của mô hình miền. Thay vì giới thiệu một siêu mô hình cấu trúc độc lập, DCSL đóng góp một tập các chú thích cấu trúc cùng với các quy tắc hợp lệ hình thức, được gắn kết trực tiếp với các phần tử lõi của UDML.

Ngữ nghĩa hình thức của DCSL được xác định theo cách tiếp cận sau:

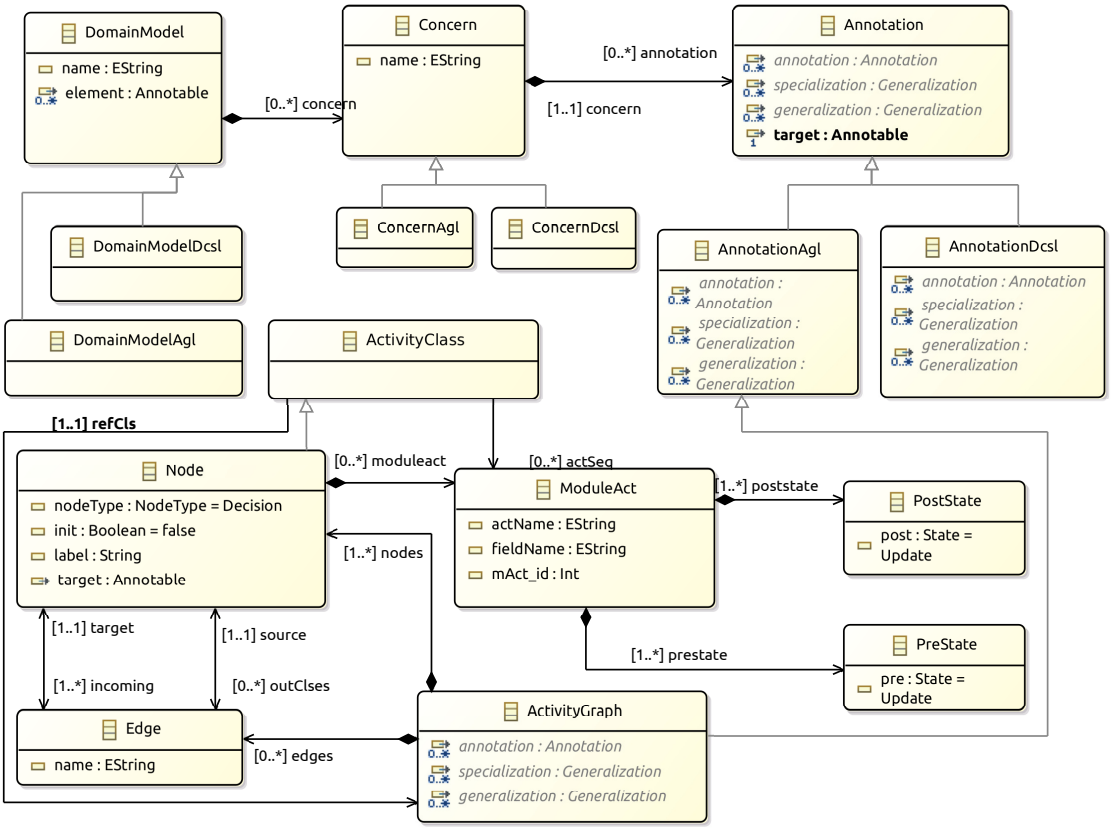
- *Liên kết từ chú thích đến UDML lõi:* Giả sử các tập mang được cho như sau. $AN_{dcsl} \subseteq AN$; DCL, DAT, DAS, $DOP \subseteq AN_{dcsl}$, trong đó DCL biểu thị các thể hiện của `DClass`; DAT biểu thị các thể hiện của `DAttr`; DAS biểu thị các thể hiện của `DAssoc`; DOP biểu thị các thể hiện của `DOpt`. Các chú thích của mối quan tâm về cấu trúc được liên kết với các phần tử lõi của UDML thông qua hàm UDML $target : AN \to AE$, với các ràng buộc kiểu sau đây:
  (D1) $\forall\, a \in DCL \cdot target(a) \in Class$;
  (D2) $\forall\, a \in DAT \cdot target(a) \in Property$;
  (D3) $\forall\, a \in DAS \cdot target(a) \in Property$;
  (D4) $\forall\, a \in DOP \cdot target(a) \in Operation$.
  Các ràng buộc này hình thức hóa ánh xạ dự định giữa các siêu khái niệm của DCSL và các siêu khái niệm lõi của UDML, đảm bảo rằng thông tin về mối quan tâm cấu trúc được gắn vào đúng các phần tử cấu trúc tương ứng.

- *Các ràng buộc tính đúng đắn của cấu trúc:* Các ràng buộc tính đúng đắn cấu trúc được định nghĩa nhằm đảm bảo tính nhất quán của miền cấu trúc. Gọi R_{dcsl} là tập các ràng buộc cấu trúc do DCSL đóng góp (ví dụ: các ràng buộc không gian trạng thái trên các trường miền và các đầu mút liên kết). Mô hình mối quan tâm cấu trúc được xem là đúng đắn khi và chỉ khi tất cả các ràng buộc trong R_{dcsl} đều thỏa mãn đối với các mục tiêu tương ứng trong mô hình UDML: $\bigwedge_{\varphi \in R_{dcsl}} \varphi$.
  Các ràng buộc này tạo thành các điều kiện đúng đắn về mặt cấu trúc, những điều kiện phải được bảo toàn trong quá trình thực thi của mô hình hợp nhất.

4.2.2.2 Tích hợp hành vi miền

Để biểu diễn các hành vi miền và tích hợp chúng vào mô hình miền hợp nhất, phần này tiếp tục tinh chỉnh AGL bằng cách xây dựng siêu mô hình của nó và định nghĩa ngữ nghĩa hình thức tương ứng.

Cú pháp trừu tượng. Hình 4.5 minh họa siêu mô hình AGL, mô hình này nắm bắt hành vi miền có khả năng thực thi bằng cách đặc tả việc thực thi hành vi dưới dạng các đồ thị hoạt động, các nút và các quan hệ luồng điều khiển. Định nghĩa siêu mô hình AGL được trình bày trong Mục 3.3 trong bối cảnh các mô hình miền hợp nhất có khả năng thực thi; một định nghĩa tường minh như vậy là cần thiết nhằm hỗ trợ việc tích hợp ngữ nghĩa một cách chính xác với các mối quan tâm về cấu trúc và bảo mật.



Hình 4.5: Siêu mô hình AGL dùng để nắm bắt hành vi miền có khả năng thực thi.

Các siêu khái niệm lõi của AGL bao gồm: `ActivityGraph` biểu diễn một đơn vị hành vi miền có khả năng thực thi và tổ chức hành vi dưới dạng một đồ thị có hướng. Mỗi `ActivityGraph` được liên kết với một `RootNode`, xác định điểm vào dùng để kích hoạt luồng hành vi tương ứng tại thời điểm

thực thi. `Node` biểu diễn một đơn vị hành vi nguyên tử và xác định một ranh giới thực thi tường minh. Mỗi nút tham chiếu đến một lớp miền thông qua thuộc tính `refCls`, qua đó liên kết việc thực thi hành vi với mô hình cấu trúc miền. `Edge` xác định các quan hệ luồng điều khiển hợp lệ giữa các nút và ràng buộc sự tiến triển có thể của quá trình thực thi. `ModuleAct` đặc tả các hành động cụ thể được thực thi tại một nút. Mỗi `ModuleAct` được liên kết với một `ModuleService` tương ứng, được định nghĩa trong `RootModule`, nhằm đảm bảo rằng các hành động hành vi được hiện thực một cách nhất quán trong kiến trúc thực thi theo mô-đun. `PreState` và `PostState` lần lượt đặc trưng cho trạng thái miền mong đợi trước và sau khi thực thi một hành động mô-đun.

Trong siêu mô hình này, mỗi `ActivityGraph` sở hữu các nút và các cạnh của nó, hình thành một ngữ cảnh thực thi hành vi có phạm vi xác định rõ ràng. Việc thực thi hành vi được khởi tạo tại `RootNode`, tiến triển theo các cạnh hợp lệ và gọi các dịch vụ mô-đun thông qua các phần tử `ModuleAct`. Bằng cách tham chiếu tới các lớp miền ở mức nút, AGL cung cấp một liên kết tường minh giữa hành vi và mô hình cấu trúc miền, tạo nền tảng cho ngữ nghĩa thực thi trong UDML.

Ngữ nghĩa. Khía cạnh hành vi của các mô hình UDML được định nghĩa trong AGL thông qua các đồ thị hoạt động được tích hợp với mô hình cấu trúc miền. Tiến hành định nghĩa ngữ nghĩa hình thức của AGL như sau. Gọi: AG, ND, ED, MA và AC lần lượt là các tập đồ thị hoạt động, các nút hoạt động, các cạnh, các hành động mô-đun và các lớp hoạt động.

- *Đồ thị hoạt động:* Một đồ thị hoạt động $g \in AG$ được định nghĩa là một bộ: $g = \langle N_g,\ E_g,\ src_g,\ tgt_g,\ init_g,\ actSeq,\ refActCls \rangle$, trong đó $N_g \subseteq ND$ là tập hữu hạn các nút hoạt động; $E_g \subseteq ED$ là tập hữu hạn các cạnh; $src_g, tgt_g : E_g \to N_g$ xác định nút nguồn và nút đích của mỗi cạnh; $init_g \in N_g$ là nút khởi đầu của đồ thị hoạt động; $actSeq : ND \to \mathcal{P}(MA)$ ánh xạ mỗi nút tới một tập (có thể rỗng) các hành động mô-đun được thực thi tại nút đó; $refActCls : ND \to AC$ ánh xạ mỗi nút tới lớp hoạt động mà nó tham chiếu.

- *Các lớp hoạt động:* Các lớp hoạt động đóng vai trò là phần mở rộng hành vi của các lớp miền, cung cấp ngữ cảnh cấu trúc trong đó các đồ thị hoạt động được diễn giải. Mỗi đồ thị hoạt động được gắn với đúng

một lớp hoạt động, và các nút hoạt động tham chiếu tới lớp này khi thực thi các hành động mô-đun.

- *Các nút có thể gắn chú thích:* Một quyết định thiết kế then chốt trong UDML là các nút hoạt động là các phần tử có thể được gắn chú thích. $ND \subseteq AE$. Điều này cho phép các ngữ nghĩa đặc thù theo mối quan tâm, bao gồm các ràng buộc bảo mật được đặc tả trong RBACDom (sẽ trình bày trong phần tiếp theo), được gắn trực tiếp vào các ranh giới thực thi hành vi.

- *Các ràng buộc tính đúng đắn của AGL:* Các ràng buộc tính đúng đắn sau đây phải được thỏa mãn đối với mọi đồ thị hoạt động $g \in AG$:
(A1) $N_g \neq \varnothing$; (A2) $\forall\, e \in E_g \cdot src_g(e) \in N_g \wedge tgt_g(e) \in N_g$;
(A3) $init_g \in N_g$; (A4) $\forall\, n \in N_g \cdot refActCls(n)$ được xác định.

Trong Mục 3.3 trình bày một phép thực thi của mô hình AGL được định nghĩa là một chuỗi hợp lệ các hành động nguyên tử được kích hoạt, chuỗi này gây ra một dãy tương ứng các trạng thái hệ thống có thể đạt được, tuân theo luồng điều khiển của đồ thị hoạt động và tôn trọng các điều kiện trạng thái trước và sau của mỗi hành động.

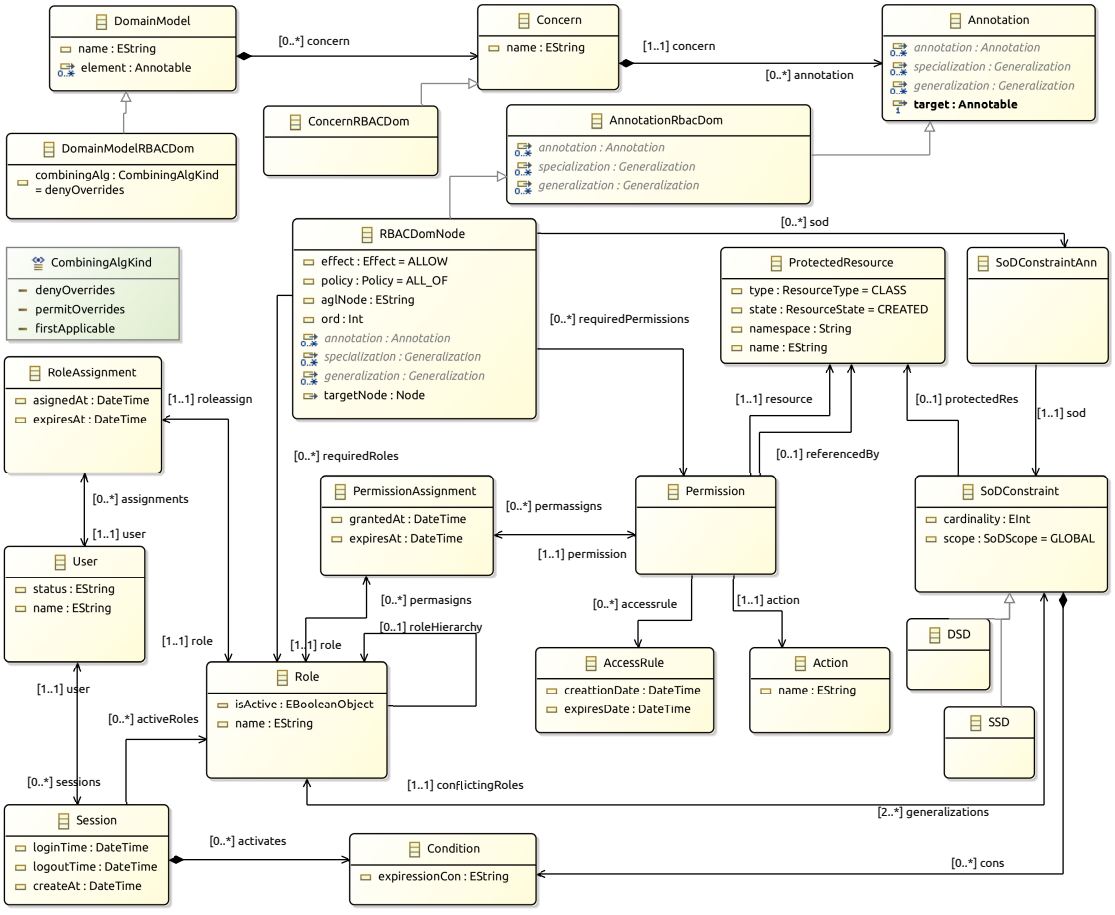
Định nghĩa 4.1 (Thực thi của một mô hình AGL)**.** *Cho $\mathcal{G} = (N, E)$ là một đồ thị hoạt động AGL, trong đó mỗi nút $n \in N$ được gắn với một hành động nguyên tử có cấu trúc $SAA(n)$, biểu diễn một tập các chuỗi hành động nguyên tử cho phép (ASEs). Một phép thực thi của $\mathcal{G}$ là một chuỗi (hữu hạn hoặc vô hạn) các nút $\pi = (n_0, n_1, \ldots, n_k)$ sao cho: (i) với mọi $i < k$, $(n_i, n_{i+1}) \in E$; và (ii) tồn tại một dãy các chuỗi hành động nguyên tử $S_0, S_1, \ldots, S_k$, với $S_i \in SAA(n_i)$, mà các phép biến đổi trạng thái do chúng gây ra là tương thích, tức là trạng thái hậu của S_i thỏa mãn trạng thái tiền của S_{i+1}. Tương đương, một phép thực thi tương ứng với một chuỗi hành động nguyên tử hợp lệ tạo ra một dãy các trạng thái hệ thống có thể đạt được.*

Ví dụ. Xét đồ thị hoạt động AGL cho Hệ thống CourseMan gồm các nút n_0 = Student, n_1 = DHelpOrSClass, n_2 = SClassRegistration, với các cạnh (n_0, n_1) và (n_1, n_2). Một phép thực thi hữu hạn của đồ thị là: $\pi = (n_0, n_1, n_2)$. Tại mỗi nút n_i, chọn một chuỗi hành động nguyên tử $S_i \in SAA(n_i)$: (i) S_0: tạo đối tượng `Student` (kết thúc ở trạng thái `Created`); (ii) S_1: thực hiện quyết định, chọn nhánh không yêu cầu hỗ trợ; (iii) S_2: tạo đối tượng

`SClassRegistration`, sử dụng `Student` đã tạo làm đầu vào. Các chuỗi S_0, S_1, S_2 là tương thích trạng thái, vì trạng thái hậu của mỗi S_i thỏa mãn trạng thái tiền của S_{i+1}. Do đó, π là một phép thực thi hợp lệ.

4.2.2.3 Tích hợp mối quan tâm bảo mật

Để hỗ trợ việc mô hình hóa tường minh cơ chế kiểm soát truy cập, luận án giới thiệu RBACDom như một DSL chuyên biệt theo mối quan tâm, chuyên biệt cho RBAC trong các mô hình miền hợp nhất. RBACDom được thiết kế phù hợp với các nguyên lý hướng mối quan tâm của UDML, cho phép các chính sách bảo mật được đặc tả một cách độc lập trong khi vẫn có khả năng kết hợp với các mối quan tâm về cấu trúc và hành vi. Phù hợp với định hướng thiết kế ngôn ngữ của UDML, RBACDom được định nghĩa thông qua cú pháp trừu tượng, cú pháp cụ thể và ngữ nghĩa hình thức, qua đó cho phép các ràng buộc ủy quyền được tích hợp một cách có hệ thống vào các mô hình miền hợp nhất.



Hình 4.6: Siêu mô hình RBACDom nắm bắt mối quan tâm bảo mật.

Cú pháp trừu tượng. Hình 4.6 minh họa cú pháp trừu tượng của RBACDom dùng để nắm bắt mối quan tâm bảo mật. RBACDom được định nghĩa bởi một siêu mô hình chuyên biệt dựa trên các siêu khái niệm UML [94]. Siêu mô hình này nắm bắt các khái niệm lõi của RBAC và các quan hệ cấu trúc giữa chúng, đồng thời đặc tả cách thức thông tin kiểm soát truy cập được biểu diễn và liên kết với các mô hình UDML.

RBACDom được đặc tả như một DSL độc lập theo mối quan tâm, trong đó mô hình miền của nó, `DomainModelRbacDom`, được kết hợp với các mô hình miền theo mối quan tâm khác thông qua lõi UDML. Cụ thể, `DomainModelRbacDom` được liên kết với `DomainModelAgl` ở mức UDML, qua đó duy trì sự phân tách rõ ràng giữa mô hình hóa hành vi và đặc tả kiểm soát truy cập.

Siêu khái niệm `DomainModelRbacDom` định nghĩa một thuộc tính *combiningAlg* nhằm xác định chiến lược kết hợp luật gắn với mô hình RBACDom. Thuộc tính này quy định cách thức đánh giá nhiều chú thích áp dụng đồng thời trên cùng một nút hành vi (ví dụ: denyOverrides, permitOverrides, firstApplicable). Việc tích hợp ở mức nút được thực hiện thông qua siêu khái niệm `RbacDomNode`. Một thể hiện của `RbacDomNode` biểu diễn một đặc tả chính sách RBAC và được khai báo trong một `ActivityGraph` như một phần của `DomainModelRbacDom`. Mỗi `RbacDomNode` thiết lập một ánh xạ tường minh tới một nút hành vi được định nghĩa trong AGL thông qua một thuộc tính tham chiếu (ví dụ: `aglNode`), dùng để xác định nút đích theo nhãn của nó. Ánh xạ này mang tính cấu trúc và không làm thay đổi cú pháp trừu tượng của `AGL :: Node`.

Siêu khái niệm `RbacDomNode` đặc tả các thuộc tính đặc thù của RBAC, bao gồm các vai trò và/hoặc quyền yêu cầu, chế độ đánh giá chính sách (`ALL_OF` hoặc `ANY_OF`), hiệu lực (`ALLOW` hoặc `DENY`), và các ràng buộc phân tách nhiệm vụ (separation-of-duty) tùy chọn. Một vị từ phạm vi tùy chọn có thể được sử dụng để giới hạn phạm vi áp dụng của chính sách đối với một tập con các đối tượng miền, qua đó hỗ trợ kiểm soát truy cập theo ngữ cảnh ở mức mô hình.

Ngoài ra, `RbacDomNode` định nghĩa một thuộc tính tùy chọn *ord*, biểu diễn thứ tự đánh giá của các chú thích. Thuộc tính này chỉ được sử dụng khi chiến lược kết hợp là firstApplicable, trong đó các chú thích được đánh giá

theo thứ tự đã gán.

Các tham số kết hợp luật có thể cấu hình. Trong các hệ thống kiểm soát truy cập thực tế, nhiều luật phân quyền có thể đồng thời áp dụng cho cùng một thao tác được bảo vệ. Do đó, các hệ thống chính sách khác nhau sử dụng các chiến lược kết hợp luật khác nhau để xác định quyết định phân quyền cuối cùng. Để phản ánh tính linh hoạt này ở mức mô hình, RBACDom cung cấp một tham số tường minh nhằm xác định cách kết hợp các chú thích áp dụng đồng thời.

Ký hiệu $CA = \{\mathsf{denyOverrides}, \mathsf{permitOverrides}, \mathsf{firstApplicable}\}$ là tập các chiến lược kết hợp luật hợp lệ.
Thuộc tính $combiningAlg : DomainModelRbacDom \to CA$ gán một chiến lược kết hợp cho mỗi mô hình RBACDom. Nếu không được chỉ định tường minh, chiến lược mặc định là denyOverrides.

Gọi $R \in DomainModelRbacDom$ là một mô hình RBACDom. Đối với việc đánh giá phụ thuộc thứ tự theo chiến lược firstApplicable, ta xét thêm một hàm riêng phần: $ord : RbacDomNode \rightharpoonup \mathbb{N}$, trong đó $ord(rna)$ biểu diễn thứ tự đánh giá của chú thích rna.

Gọi $Ann(n) \subseteq RbacDomNode$ là tập các chú thích RBACDom gắn với nút $n \in ND$. Điều kiện well-formedness sau đảm bảo tính xác định trong việc đánh giá các chính sách cục bộ tại nút: $(RC1)$ $combiningAlg(R) = \mathsf{firstApplicable} \Rightarrow \forall\, n \in ND \cdot \forall\, rna_1, rna_2 \in Ann(n) \cdot (ord(rna_1) = ord(rna_2) \Rightarrow rna_1 = rna_2)$.

RBACDom còn định nghĩa các khái niệm RBAC chuẩn `User`, `Role`, `Permission` và `Session` một cách độc lập với các phần tử miền của UDML. Các quan hệ gán người dùng–vai trò và vai trò–quyền được biểu diễn tường minh dưới dạng các quan hệ, trong khi các quyền được mô hình hóa như các cặp ⟨hành động, tài nguyên được bảo vệ⟩. Các tài nguyên được bảo vệ có thể tùy chọn được liên kết với các phần tử có thể gắn chú thích của UDML nhằm hỗ trợ việc tích hợp với các mô hình miền.

Để biểu diễn các ràng buộc cho RBACDom, mô hình này giới thiệu siêu khái niệm trừu tượng ràng buộc phân tách nhiệm vụ `SoDConstraint`, với các chuyên biệt hóa cho phân tách nhiệm vụ tĩnh `SSD` và phân tách nhiệm vụ động `DSD`. Việc mô hình hóa tường minh các ràng buộc SoD giúp cải

thiện tính mô-đun và khả năng phân tích cấu trúc của các đặc tả kiểm soát truy cập.

Bảng 4.1: Các quy tắc hợp lệ cấu trúc của siêu mô hình RBACDom

Ràng buộc	**Mô tả**
WF-S1: Định danh RBAC duy nhất	Mọi thực thể RBAC (người dùng, vai trò, quyền, hành động và tài nguyên được bảo vệ) phải có định danh duy nhất trong một mô hình RBACDom.
WF-S2: Gán kết vai trò hợp lệ	Mỗi gán kết người dùng–vai trò phải tham chiếu đến một người dùng và một vai trò tồn tại. Không cho phép các gán kết vai trò không xác định hoặc bị treo.
WF-S3: Gán kết quyền hợp lệ	Mỗi gán kết vai trò–quyền phải tham chiếu đến một vai trò tồn tại và một quyền hợp lệ được định nghĩa trong mô hình.
WF-S4: Định nghĩa quyền hợp lệ	Mỗi quyền phải được định nghĩa chính xác bởi một hành động áp dụng lên đúng một tài nguyên được bảo vệ, theo diễn giải RBAC coi quyền như các cặp ⟨thao tác, đối tượng⟩.
WF-S5: Bộ đôi quyền duy nhất	Không có hai quyền nào được phép định nghĩa cùng một cặp hành động–tài nguyên.
WF-S6: Gắn kết tài nguyên tùy chọn	Một tài nguyên được bảo vệ có thể được gắn hoặc không gắn với một phần tử miền cụ thể của UDML. Việc không gắn được cho phép nhằm hỗ trợ gắn kết.
WF-S7: Tương thích kiểu tài nguyên–miền	Nếu một tài nguyên được bảo vệ được gắn với một phần tử miền, kiểu của tài nguyên phải tương thích với loại phần tử UDML mà nó được gắn vào.
WF-S8: Ràng buộc phân tách nhiệm vụ hợp lệ	Mỗi ràng buộc SoD phải tham chiếu đến ít nhất hai vai trò khác nhau và xác định một lực lượng hợp lệ ($n \geq 2$).
WF-S9: Nhất quán phân tách nhiệm vụ tĩnh	Các ràng buộc SoD tĩnh không được bị vi phạm bởi các gán kết người dùng–vai trò.
WF-S10: Gắn kết chú thích và phạm vi	Mọi chú thích RBAC phải được gắn vào các phần tử UDML được xác định rõ ràng.

Siêu mô hình RBACDom được trang bị một tập các quy tắc tính đúng đắn cấu trúc, được tóm tắt trong Bảng 4.1. Các quy tắc này đảm bảo tính nhất quán nội tại của các đặc tả RBAC và tính hợp lệ của việc gắn chúng

vào các mô hình UDML trước khi tiến hành diễn giải ngữ nghĩa hoặc kiểm chứng hình thức.

Cú pháp cụ thể. Để hỗ trợ hiện thực hóa thực tiễn và tích hợp công cụ, phần này định nghĩa một cú pháp cụ thể dạng văn bản dựa trên chú thích cho RBACDom, phù hợp với các nguyên lý kết hợp hướng mối quan tâm và điều khiển bởi siêu mô hình của UDML. Cú pháp cụ thể này được dẫn xuất từ siêu mô hình cú pháp trừu tượng của RBACDom bằng cách định nghĩa một siêu mô hình cú pháp cụ thể tương ứng, thích hợp để nhúng vào một ngôn ngữ lập trình hướng đối tượng chủ (ví dụ: Java).

Phù hợp với phong cách tích hợp dựa trên chú thích của UDML, RBACDom được hiện thực như một DSL bên ngoài dựa trên chú thích. Các chính sách RBAC được đặc tả dưới dạng văn bản thông qua các chú thích, các chú thích này khởi tạo các phần tử `RbacDomNode` của siêu mô hình RBACDom, trong khi mối liên kết ngữ nghĩa của chúng với hành vi có khả năng thực thi được thiết lập thông qua sự tương ứng tường minh ở mức nút với các nút AGL. Thiết kế này bảo toàn sự phân tách giữa các mối quan tâm về hành vi và bảo mật, đồng thời cho phép kết hợp chúng một cách có hệ thống ở mức mô hình hóa.

Đặc tả 4.1 thể hiện cú pháp cụ thể của RBACDom được xây dựng xoay quanh chú thích `@RBACDomNode`, dùng để đặc tả các chính sách ủy quyền tại từng nút hành vi. Mỗi thể hiện của chú thích này tương ứng với một phần tử `RbacDomNode` trong cú pháp trừu tượng và chứa các thuộc tính của chính sách RBAC, bao gồm vai trò hoặc quyền yêu cầu, chế độ đánh giá chính sách, hiệu lực, các vị từ phạm vi và các ràng buộc SoD (nếu có). Các thuộc tính này xác định chủ thể được phép thực thi nút hành vi, cách kết hợp các yêu cầu ủy quyền và phạm vi áp dụng của chính sách. Đoạn liệt kê dưới đây minh họa cú pháp văn bản dựa trên chú thích dùng để đặc tả các chính sách RBACDom.

```
1     @RBACDomNode(
2     id     = "U.ID",
3     roles  = {<RoleName>},
4     perms  = {(<Op>, <Res>), (<...>)},
5     policy = ALL_OF | ANY_OF,
6     effect = ALLOW | DENY,
7     scope  = {<ScopePredicate>,<...>},
8     sod    = { SSD | DSD | ... },
```

```
    ord    = <Integer>
    )
```

Đặc tả 4.1: Biểu diễn RBACDom dưới dạng chú thích

Ví dụ. Đặc tả 4.2 minh họa một biểu diễn RBACDom cho vai trò **Author** trong miền OJS:

```
@RBACDomNode(
    id     = "U1.Author",
    roles  = {"Author"},
    perms  = {
      ("CreateSubmission", "Submission"),
      ("UploadManuscriptFile","ManuscriptFile")
    },
    policy = ANY_OF,
    effect = ALLOW,
    sod    = {DSD}1,
    ord    = 1
)
```

Đặc tả 4.2: Ví dụ RBACDom cho vài trò `Author`

Chú thích này được khai báo như một phần của mô hình RBACDom và được liên kết với một nút hành vi thông qua một sự tương ứng tường minh tới nhãn nút được định nghĩa trong đồ thị hoạt động AGL.

4.2.2.4 Ánh xạ giữa AGL và RBACDom

Các mối quan tâm về bảo mật được nắm bắt một cách độc lập trong RBACDom thông qua siêu khái niệm `RbacDomNode`, siêu khái niệm này đặc tả các ràng buộc ủy quyền, yêu cầu về vai trò và các thuộc tính bảo mật liên quan. Thay vì nhúng trực tiếp bảo mật vào các nút hành vi, các nút RBACDom được kết hợp với các nút AGL thông qua một quan hệ tương ứng tường minh thường được hiện thực bằng một tham chiếu hoặc nhãn (ví dụ: `aglNode`) nhằm xác định nút hành vi mà việc thực thi của nó bị ràng buộc.

Cách ánh xạ này bảo toàn tính mô-đun của các DSL chuyên biệt theo mối quan tâm, đồng thời căn chỉnh ngữ nghĩa của chúng trong khuôn khổ UDML. AGL xác định các chuyển tiếp hành vi hợp lệ, trong khi RBACDom ràng buộc khả năng thực thi bằng cách hạn chế việc kích hoạt các chuyển tiếp dưới các điều kiện ủy quyền. Các mối quan tâm này được hợp nhất

thông qua lõi UDML trên một diễn giải hành vi chung, mà không trộn lẫn các cú pháp trừu tượng của chúng. Việc hiện thực khả thi được hỗ trợ bởi *RootModule*, thành phần này đóng gói các đặc tả DCSL, AGL và RBACDom đã được kết hợp thành một đơn vị nhất quán. Được điều phối bởi `ModuleService` và trung gian bởi `ResProtect` cùng các định nghĩa phạm vi, cấu trúc này đảm bảo việc thực thi nhất quán cả hành vi lẫn kiểm soát truy cập, từ đó tạo ra một cú pháp trừu tượng hợp nhất làm nền tảng cho ngữ nghĩa thực thi và kiểm chứng hình thức.

Đặc tả 4.3 minh họa cách biểu diễn RBACDom cùng tồn tại với các định nghĩa hành vi trong một ngôn ngữ chủ dựa trên chú thích. Các nút hành vi được định nghĩa bằng các chú thích của AGL, trong khi các nút RBACDom được khai báo độc lập và được liên kết với các nút hành vi thông qua các quan hệ tương ứng thay vì nhúng cú pháp trực tiếp. Khi chiến lược kết hợp là `firstApplicable`, các chú thích được đánh giá theo thuộc tính `ord`. Trong ví dụ này, luật đầu tiên thỏa mãn sẽ quyết định kết quả cuối cùng. Nếu người dùng thỏa mãn luật thứ nhất (ví dụ: có vai trò `Guest`), quyền truy cập vào hoạt động *Submission* sẽ bị từ chối. Ngược lại, luật thứ hai sẽ được đánh giá, cho phép thực thi đối với những người dùng có vai trò `Author` và thỏa mãn các quyền yêu cầu.

```

@AGraph(nodes = {
    @ANode(label = "Submission",
    nodeType = NodeType.Action,
    init = true,
    refCls = Submission.class,
    outNodes = {"SubmissionQueue"},
    moduleact = {@MAct(
        actName = ActNames.newObject,
        poststate = {State.Created})}
    )
    @RBACDomNode(
    id = "U1.GuestDeny",
    aglNode = "Submission",
    effect  = Effect.DENY,
    roles   = {Role.Guest},
    ord     = 1
    ),
    @RBACDomNode(
    id = "U1.AuthorAllow",
    aglNode = "Submission",
```

```
22      effect  = Effect.ALLOW,
23      roles   = {Role.Author},
24      perms   = {
25          ("CreateSubmission","Submission"),
26          ("UploadManuscriptFile","ManuscriptFile")
27      },
28      policy  = Policy.ANY_OF,
29      sod     = {DSD},
30      ord     = 2
31      ),
32      ... }
33  )
34  public class SubmissionMng{
35      ...
36  }
```

Đặc tả 4.3: Đặc tả RBACDom và AGL nhúng vào OOPL (Java)

Ngữ nghĩa cho RBACDom. Các khía cạnh bảo mật được nắm bắt bởi RBACDom, một DSL chuyên biệt theo mối quan tâm dựa trên chú thích, hình thức hóa RBAC cùng với các ràng buộc SoD và liên kết chúng với các phần tử UDML có thể gắn chú thích, đặc biệt là các nút hoạt động AGL. Thực hiện định nghĩa ngữ nghĩa hình thức cho RBACDom như sau.

Giả sử các tập mang sau đây được cho: USR, R, P, SES, ACT, RES, SC, $RbacDomNode$ lần lượt biểu diễn người dùng, vai trò, quyền, phiên, hành động, tài nguyên được bảo vệ, các ràng buộc SoD và các chú thích RBAC ở mức nút.

- *Gán vai trò và quyền:* RBACDom nắm bắt các phép gán dưới dạng các quan hệ. $UA \subseteq USR \times R$; $PA \subseteq R \times P$, trong đó $(u, r) \in UA$ có nghĩa là người dùng u được gán vai trò r, và $(r, p) \in PA$ có nghĩa là vai trò r được cấp quyền p.

- *Phiên và các vai trò đang kích hoạt:* Các phiên được liên kết với người dùng và các vai trò đang kích hoạt bởi: $userOf : SES \rightarrow USR$; $activeRoles \subseteq SES \times R$, trong đó $(s, r) \in activeRoles$ có nghĩa là vai trò r đang hoạt động trong phiên s.

- *Quyền và tài nguyên được bảo vệ:* Mỗi quyền được đặc trưng bởi một hành động và một tài nguyên được bảo vệ. $permAction : P \rightarrow ACT$; $permRes : P \rightarrow RES$. Các tài nguyên được bảo vệ có thể được liên kết

với các phần tử UDML có thể gắn chú thích nhằm hỗ trợ tham chiếu ở mức miền: $bindsTo : RES \rightharpoonup AE$.

- *Phân tách nhiệm vụ (SoD):* Một ràng buộc SoD $c \in SC$ được định nghĩa là một bộ: $c = \langle conflictSet(c),\ card(c),\ kind(c),\ scope(c)\rangle$, trong đó $conflictSet(c) \subseteq R$, $card(c) \in \mathbb{N}$, $kind(c) \in \{SSD, DSD\}$, và $scope(c)$ biểu thị phạm vi của ràng buộc (ví dụ: $GLOBAL$).

- *Các chú thích RBACDom gắn với nút:* Các chính sách RBACDom được gắn với các ranh giới thực thi hành vi thông qua các chú thích ở mức nút. Gọi $ND \subseteq AE$ là tập các nút hoạt động có thể gắn chú thích được định nghĩa bởi AGL. Mỗi chú thích RBACDom $rna \in RbacDomNode$ được gắn với đúng một nút hoạt động thông qua hàm:
  $targetNode : RbacDomNode \rightarrow ND$. Mỗi chú thích cung cấp các thành phần cú pháp trừu tượng sau:
  $policy : RbacDomNode \rightarrow \{ALL\_OF, ANY\_OF\}$;
  $effect : RbacDomNode \rightarrow \{ALLOW, DENY\}$;
  $ReqRoles : RbacDomNode \rightarrow \mathcal{P}(R)$;
  $ReqPerms : RbacDomNode \rightarrow \mathcal{P}(P)$;
  $SoD : RbacDomNode \rightarrow \mathcal{P}(SC)$.

Ánh xạ cho RBACDom. Trong UDML, RBACDom được diễn giải như một mối quan tâm về bảo mật, có vai trò ràng buộc khả năng thực thi hành vi thông qua các tương ứng ở mức nút với các nút hoạt động trong AGL. Mỗi `RbacDomNode` tạo ra một điều kiện phân quyền trên nút đích của nó, qua đó giới hạn việc kích hoạt nút mà không làm thay đổi cấu trúc hành vi hay luồng điều khiển. Do đó, một nút hoạt động chỉ có thể thực thi khi nó được kích hoạt theo AGL và đồng thời thỏa mãn tất cả các ràng buộc RBACDom liên quan trong ngữ cảnh phân quyền hiện tại.

Nói chung, nhiều chú thích RBACDom có thể được gắn với cùng một nút hành vi. Vì vậy, quyết định phân quyền cuối cùng phụ thuộc vào chiến lược kết hợp luật được lựa chọn trong mô hình RBACDom. Hàm $combiningAlg : DomainModelRbacDom \rightarrow CA$ xác định cách đánh giá các chú thích áp dụng đồng thời. Do đó, ngữ nghĩa dưới đây được tham số hóa theo chiến lược kết hợp luật này. Trong ngữ nghĩa này, AGL xác định tiến hóa hành vi, trong khi RBACDom loại bỏ các chuyển trạng thái không hợp lệ, qua

đó cung cấp cơ sở hợp thành cho chuyển đổi bảo toàn ngữ nghĩa và kiểm chứng hình thức.

Định nghĩa 4.2 (Tương ứng nút RBACDom)**.** *Giả sử $G = (N, E)$ là một đồ thị hoạt động AGL với $N \subseteq ND$. Việc tích hợp RBACDom ở mức nút vào G được định nghĩa thông qua ánh xạ toàn phần $targetNode : RbacDomNode \to ND$ ánh xạ mỗi $rna \in RbacDomNode$ tới một nút đích duy nhất. Một nút $n \in N$ được kích hoạt trong một ngữ cảnh thực thi khi và chỉ khi tất cả các ràng buộc RBAC và ràng buộc phân tách nhiệm vụ được đặc tả bởi mọi rna sao cho $targetNode(rna) = n$ đều được thỏa mãn.*

Ngữ nghĩa kích hoạt nút cho các đồ thị được gắn RBACDom. Phần này định nghĩa ngữ nghĩa hình thức của việc kích hoạt nút đối với các đồ thị hoạt động được gắn RBACDom. Việc ủy quyền được diễn giải như một ràng buộc ngữ nghĩa trên thực thi hành vi: một nút hoạt động chỉ có thể được kích hoạt và thực thi trong một phiên cho trước nếu các chính sách RBAC liên kết với nó được thỏa mãn. Ngữ nghĩa này được định nghĩa ở mức các nút của đồ thị hoạt động, vốn đóng vai trò là các ranh giới thực thi của hành vi miền.

Một nút n được kích hoạt trong một hình chụp trạng thái hệ thống $\sigma \in \Sigma$ khi và chỉ khi nó được kích hoạt bởi ngữ nghĩa luồng điều khiển của AGL và tất cả các ràng buộc RBACDom được thỏa mãn đối với ngữ cảnh ủy quyền chứa trong $S_{DSL_c}(\sigma)$.

Các hàm phụ trợ và kiểu. Các hàm và vị từ phụ trợ được sử dụng trong ngữ nghĩa kích hoạt được định nghĩa và gán vào Ann dạng:
$Ann : ND \to \mathcal{P}(RbacDomNode)$ ánh xạ mỗi nút hoạt động tới tập các chú thích RBACDom của nó. Với mỗi chú thích $rna \in RbacDomNode$, các hàm ánh xạ nút RbacDom được định nghĩa như sau:

$policy : RbacDomNode \to \{ALL\_OF, ANY\_OF\}$

$effect : RbacDomNode \to \{item, DENY\}$

$ReqRoles : RbacDomNode \to \mathcal{P}(R)$

$ReqPerms : RbacDomNode \to \mathcal{P}(P)$

$SoD : RbacDomNode \to \mathcal{P}(SC)$

Các quyền được kích hoạt bởi một tập vai trò được suy ra bởi:
$Perms : \mathcal{P}(R) \to \mathcal{P}(P)$, trong đó $Perms(R') = \{p \in P \mid \exists\, r \in R' \cdot (r, p) \in PA\}$.

Vị từ $Violates : SC \times SES \to \mathbb{B}$ chỉ ra liệu một ràng buộc phân tách nhiệm vụ có bị vi phạm trong một phiên cho trước hay không.

Một chú thích RBACDom gắn với nút $rna \in Ann(n)$ được xem là *áp dụng* trong một phiên $s \in SES$ khi và chỉ khi tất cả các điều kiện về vai trò, quyền và SoD của nó đều được thỏa mãn: $Applies(rna, s) \quad s \in SES \triangleq RolesSat(rna, s) \wedge PermsSat(rna, s) \wedge SoDSat(rna, s)$.

Các vị từ thành phần được định nghĩa như sau:
$RolesSat(rna, s) \triangleq \big(policy(rna) = ALL\_OF \Rightarrow ReqRoles(rna) \subseteq activeRoles(s)\big)$
$\wedge\ \big(policy(rna) = ANY\_OF \Rightarrow ReqRoles(rna) \cap activeRoles(s) \neq \varnothing\big)$
$PermsSat(rna, s) \triangleq ReqPerms(rna) \subseteq Perms(activeRoles(s))$
$SoDSat(rna, s) \triangleq \forall\, c \in SoD(rna) \cdot \neg\, Violates(c, s)$

Ngữ nghĩa kết hợp luật. Gọi $s \in SES$ là một trạng thái phiên, trong đó SES là tập các trạng thái phiên. Đối với một nút hành vi $n \in ND$, một chú thích RBACDom $rna \in RbacDomNode$ được gọi là áp dụng trong trạng thái s nếu các ràng buộc về vai trò, quyền và ngữ cảnh của nó được thỏa mãn, vị từ này ký hiệu là: $Applies(rna, s)$. Tập các chú thích RBACDom áp dụng cho nút n trong trạng thái s được định nghĩa như sau: $App(n, s) = \{\, rna \in Ann(n) \mid Applies(rna, s) \,\}$. Gọi $effect : RbacDomNode \to \{ALLOW, DENY\}$ là hàm biểu diễn hiệu lực phân quyền gắn với một chú thích RBACDom. Quyết định phân quyền cho nút n trong trạng thái s được xác định bởi một hàm quyết định phụ thuộc chiến lược $Dec : ND \times SES \to \{ALLOW, DENY\}$. Trong đó, việc xác định $Dec(n, s)$ phụ thuộc vào chiến lược kết hợp luật được chọn bởi $combiningAlg(R)$.

Deny-Overrides. Nếu $combiningAlg(R) =$ denyOverrides, quyết định được xác định như sau:

$$Dec(n, s) = \begin{cases} DENY & \text{nếu } \exists\, rna \in App(n, s) \cdot effect(rna) = DENY \\ ALLOW & \text{nếu } \exists\, rna \in App(n, s) \cdot effect(rna) = ALLOW \\ DENY & \text{trong các trường hợp còn lại.} \end{cases}$$

Permit-Overrides. Nếu $combiningAlg(R) =$ permitOverrides, quyết định được xác định như sau:

$$Dec(n,s) = \begin{cases} ALLOW & \text{nếu } \exists\, rna \in App(n,s) \cdot effect(rna) = ALLOW \\ DENY & \text{nếu } \neg \exists\, rna \in App(n,s) \cdot effect(rna) = ALLOW \\ & \quad \wedge \exists\, rna \in App(n,s) \cdot effect(rna) = DENY \\ DENY & \text{trong các trường hợp còn lại.} \end{cases}$$

First-Applicable. Nếu $combiningAlg(R) = \mathsf{firstApplicable}$, các luật được đánh giá theo thứ tự được xác định bởi quan hệ ord.

$$first(n,s) = \arg\, min_{rna \in App(n,s)}\, ord(rna)$$

Khi đó, quyết định được xác định như sau:

$$Dec(n,s) = \begin{cases} effect(first(n,s)) & \text{nếu } App(n,s) \neq \varnothing \\ DENY & \text{trong các trường hợp còn lại.} \end{cases}$$

Cuối cùng, một nút hành vi được xem là được cấp quyền thực thi nếu $Enable_{RBAC}(n,s) \triangleq Dec(n,s) = ALLOW$.

Do đó, thành phần RBACDom trong việc kích hoạt nút được biểu diễn bởi $Enable_{RBAC}(n,s)$. Một nút hành vi n chỉ có thể thực thi khi nó được kích hoạt theo ngữ nghĩa hành vi của AGL và đồng thời điều kiện phân quyền $Enable_{RBAC}(n,s)$ được thỏa mãn.

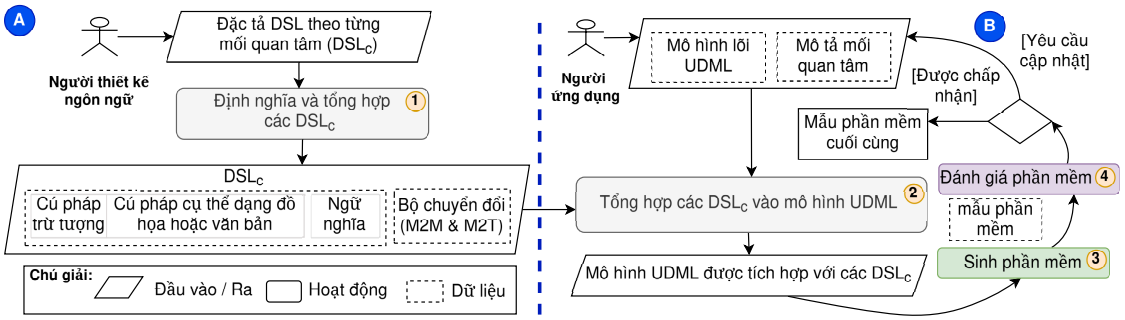
4.3 Phương pháp biểu diễn mô hình miền hợp nhất dựa vào cây cú pháp

Phần này trình bày một cách tiếp cận khác để hợp nhất các DSL được thiết kế chuyên biệt cho từng mối quan tâm vào một mô hình miền hợp nhất, thông qua cơ chế tích hợp dựa trên cây cú pháp trừu tượng. Cách tiếp cận này hướng tới việc xây dựng một miền kỹ thuật có khả năng thực thi, trong đó các mối quan tâm được kết hợp trực tiếp ở mức cú pháp trừu tượng và được gắn với ngữ nghĩa hình thức.

4.3.1 Tổng quan về phương pháp đề xuất

Phương pháp tổng hợp các DSL theo mối quan tâm vào một DM hợp nhất, như minh họa trong Hình 4.7, được tổ chức thành một quy trình lặp gồm

bốn bước, nhằm tích hợp và tinh chỉnh các DSL hướng tới việc sinh tự động bản mẫu phần mềm.



Hình 4.7: Tổng quan phương pháp đề xuất, được tổ chức thành hai giai đoạn: (A) thiết kế ngôn ngữ và (B) ứng dụng ngôn ngữ.

Thứ nhất, ở giai đoạn thiết kế ngôn ngữ (nhãn A), nhà thiết kế đặc tả miền mối quan tâm bằng một DSL chuyên biệt cho mối quan tâm (DSL_c). Đối với mỗi DSL_c, cú pháp trừu tượng (AS), cú pháp cụ thể (CS), và ngữ nghĩa được định nghĩa một cách hình thức, tạo thành một DSL_c mô tả chính xác mối quan tâm mục tiêu. *Thứ hai,* ở giai đoạn ứng dụng (nhãn B), nhà thiết kế sử dụng đầu ra của bước một cùng với mô hình lõi UDML hiện có. Kết hợp với mô tả về các yêu cầu mới hoặc thay đổi của mối quan tâm, các đầu vào này cho phép tổng hợp các DSL_c vào mô hình UDML theo cách thống nhất và nhất quán. Cú pháp cụ thể dạng văn bản hoặc đồ họa có thể được sử dụng để xây dựng mô hình chi tiết biểu diễn thể hiện cụ thể của mối quan tâm. Kết quả đầu ra của bước này là một mô hình UDML đã được tích hợp đầy đủ với mối quan tâm tương ứng. *Thứ ba,* mô hình UDML đã tích hợp đóng vai trò làm nền tảng cho việc sinh phần mềm. Tạo tác phần mềm thu được sau đó được nhà thiết kế đánh giá để thu thập phản hồi. *Cuối cùng,* nếu có phản hồi, mô hình UDML và đặc tả mối quan tâm tương ứng sẽ được cập nhật. Quy trình sau đó được lặp lại, hỗ trợ chu trình cải tiến liên tục cho đến khi hệ thống phần mềm thỏa mãn đầy đủ các yêu cầu đã đề ra.

4.3.2 Biểu diễn và tích hợp các mối quan tâm

Phần này định nghĩa cú pháp và ngữ nghĩa của UDML và giới thiệu thuật toán tổng hợp các DSL vào một DM hợp nhất.

Cây cú pháp trừu tượng của UDML

Phương pháp được trình bày trong mục này, thực hiện định nghĩa UDML tuân theo một khuôn khổ ba lớp có cấu trúc, bao gồm cú pháp trừu tượng, cú pháp cụ thể và ngữ nghĩa hình thức, cùng với một chiến lược tích hợp.

Ngôn ngữ mô hình miền hợp nhất (UDML) được xây dựng từ một lõi chung (UDML core) và một tập các DSL theo từng mối quan tâm $DSL_{i=1}^{n}$, trong đó mỗi DSL_i đặc tả một khía cạnh riêng biệt của hệ thống. Mỗi DSL được đặc trưng bởi ba thành phần: cú pháp trừu tượng (AS_i), cú pháp cụ thể (CS_i) và ngữ nghĩa (Sem_i), qua đó cho phép biểu diễn và tích hợp các mối quan tâm không đồng nhất vào một mô hình miền hợp nhất.

Trên cây cú pháp trừu tượng của UDML được xác định đầy đủ về AS, CS và ngữ nghĩa. AS của UDML được hình thức hóa dưới dạng một cấu trúc cây phân cấp, trong đó mỗi nút biểu diễn một thể hiện của khái niệm ngôn ngữ; các cạnh mô tả quan hệ tổng hợp hoặc tham chiếu; và các thuộc tính ghi nhận đặc điểm cục bộ của từng nút. Các concern DSL mở rộng cây này một cách tăng dần bằng cách bổ sung các khái niệm và quan hệ mới mà vẫn duy trì khung xương chung được xác định trong UDML lõi.

Định nghĩa 4.3 (Cây cú pháp trừu tượng của UDML)**.** *Cho C_{UDML} là tập hợp các siêu khái niệm của UDML và A_{UDML} là tập thuộc tính tương ứng. Với mỗi khái niệm $c \in C_{UDML}$, $A(c) \subseteq A_{UDML}$ là tập thuộc tính của c. Cú pháp trừu tượng **AS** của UDML được định nghĩa bởi bộ*
$AS_{UDML} = (N, root, child, refCons, label, attr, P_{UDML})$*, trong đó:*

- *N: tập hữu hạn các nút,*
- *$root \in N$: nút gốc biểu diễn DM hợp nhất, được thể hiện trong UDML dưới dạng* `DomainModel`*,*
- *$child \subseteq N \times N$: quan hệ cây xác định các nút con có thứ tự,*
- *$refCons \subseteq N \times N$: tập các cạnh tham chiếu liên kết các nút giữa các concern,*
- *$label : N \to C_{UDML}$: hàm gán nhãn xác định siêu khái niệm của mỗi nút,*
- *$attr : N \to (A(label(n)) \to V)$: hàm gán giá trị thuộc tính cho từng nút, và*
- *P_{UDML}: tập các quy tắc và ràng buộc bảo đảm tính đúng đắn của cây.*

CS được định nghĩa như một phép chiếu từ cây AST sang các ký hiệu hướng người dùng. Mỗi góc nhìn cú pháp cung cấp cách biểu diễn phù hợp với miền, trong khi tất cả các góc nhìn đều thao tác trên cùng một AST nhằm bảo đảm tính nhất quán và đồng bộ.

Định nghĩa 4.4 (Cú pháp cụ thể của UDML)**.** *Cú pháp cụ thể* ***CS*** *của UDML được định nghĩa bởi bộ* $CS_{UDML} = (V, M, \pi, H, P_{CS})$*, trong đó:*

- V*: tập các ký hiệu trực quan (hộp, mũi tên, bảng, nhãn...),*
- M*: tập quy tắc ánh xạ liên kết đến nút AST hoặc cạnh được biểu diễn trực quan,*
- $\pi : N \cup (N \times N) \to V$*: hàm chiếu ánh xạ nút/cạnh sang ký hiệu cụ thể,*
- H*: tập các bộ xử lý thao tác (tạo, sửa, kéo/thả, gán chú thích,...), và*
- P_{CS}*: tập ràng buộc bảo đảm tính hợp lệ của phép biểu diễn.*

Ngoài ra, hàm chiếu có thể được chỉ số hóa theo từng góc nhìn, $\pi_v : N \cup (N \times N) \to V$, với $v \in Views$, cho phép nhiều ký pháp khác nhau mô tả cùng một AST.

Ngữ nghĩa của UDML được xác định bằng cách gán ý nghĩa trực tiếp cho từng nút và cạnh trong AST. Mỗi concern DSL đóng góp một ánh xạ ngữ nghĩa độc lập, và các ánh xạ này được tổng hợp dựa theo cấu trúc AST.

Định nghĩa 4.5 (Ngữ nghĩa của UDML)**.** *Ngữ nghĩa* ***Sem*** *của UDML được định nghĩa bởi hàm* $\langle Sem \rangle_{UDML} : N \cup (N \times N) \to D$*, trong đó:*

- N*: tập các nút AST,*
- $N \times N$*: tập các cạnh (cây hoặc tham chiếu),*
- D*: miền ngữ nghĩa, là tích của các miền mối quan tâm* $D = D_{dcsl} \times D_{agl} \times D_{rbacDom}$*,*
- $\langle Sem(n) \rangle$*: ngữ nghĩa tại nút* n*, do concern DSL tương ứng xác định, và*
- $\langle Sem(n_i, n_j) \rangle$*: ngữ nghĩa của cạnh, cho biết cách ngữ nghĩa được lan truyền theo quan hệ cha–con hoặc tham chiếu.*

Các Định nghĩa 4.3, 4.4 và 4.5 thiết lập cơ sở hình thức cho ngôn ngữ UDML, bao gồm cú pháp trừu tượng, cú pháp cụ thể và ngữ nghĩa. Cụ thể, cú pháp trừu tượng xác định cấu trúc của mô hình miền hợp nhất, cú pháp cụ thể hỗ trợ biểu diễn và thao tác mô hình, trong khi ngữ nghĩa quy định cách diễn giải các phần tử mô hình trong miền ngữ nghĩa. Trên cơ sở đó,

các định nghĩa này đóng vai trò nền tảng cho việc xây dựng biểu diễn miền hợp nhất, thiết lập các liên kết ngữ nghĩa giữa các DSL theo mối quan tâm, cũng như hỗ trợ các phép chuyển đổi mô hình và kiểm chứng hình thức trong các phần tiếp theo của luận án.

Ngữ nghĩa toàn cục của UDML được xác định thông qua việc tổng hợp ngữ nghĩa địa phương của các nút và cạnh trong mô hình. Mỗi DSL bảo toàn quy tắc diễn giải riêng trong cấu trúc hợp nhất, đồng thời bảo đảm tính nhất quán ngữ nghĩa xuyên mối quan tâm. Quá trình hợp nhất được thực hiện theo cách tiếp cận tăng dần thông qua thuật toán hợp nhất các DSL theo mối quan tâm vào UDML, trong đó các khái niệm được tích hợp dựa trên cơ chế kế thừa và liên kết ngữ nghĩa.

Cơ chế tích hợp các DSL theo mối quan tâm vào UDML
Một AST hợp nhất được xây dựng từ nhiều DSL theo mối quan tâm dựa trên UDML lõi và các phần mở rộng của nó (như DCSL, AGL, CAP). Mỗi nút biểu diễn một thể hiện siêu khái niệm (`Student:DClass`) và các cạnh biểu diễn quan hệ tổng hợp hoặc tham chiếu (`AGraph` $\rightarrow$ `DClass`).

Thuật toán 4.1 mô tả quy trình hợp nhất tăng dần các DSL theo mối quan tâm vào UDML. Thuật toán khởi tạo UDML với các thành phần lõi (dòng 1), sau đó lần lượt hợp nhất từng DSL trong tập đầu vào (dòng 2). Đầu tiên, cú pháp trừu tượng được tích hợp bằng cách thêm các khái niệm mới và quan hệ mới (dòng 3–16). Tiếp theo là hợp nhất cú pháp cụ thể (dòng 17–19), rồi hợp nhất ngữ nghĩa gửi từ từng DSL theo từng mối quan tâm vào ngữ nghĩa tổng thể của UDML (dòng 20–21). Sau khi tất cả các DSL được xử lý, thuật toán trả về UDML hợp nhất (dòng 22), tạo thành mô hình thực thi thống nhất, bảo tồn tính mô-đun và truy vết xuyên suốt các mối quan tâm.

4.4 Ngữ nghĩa của mô hình miền hợp nhất

Mục này trình bày khung ngữ nghĩa hình thức cho các mô hình miền hợp nhất trong UDML. Một mô hình UDML được xem như sự tích hợp có hệ thống giữa mô hình miền lõi và các DSL chuyên biệt theo mối quan tâm, trong đó mỗi mối quan tâm đóng góp các khía cạnh ngữ nghĩa riêng nhưng có liên kết chặt chẽ. Ngữ nghĩa của mô hình được đặc trưng theo quan điểm

Thuật toán 4.1 Thuật toán hợp nhất các DSL theo mối quan tâm vào UDML

Đầu vào: $D = \{DSL_1, DSL_2, \ldots, DSL_n\}$: Tập các mối quan tâm $DSL_i = (AS_i, CS_i, Sem_i)$; $UDML_{core} = (AS_{core}, CS_{core}, Sem_{core})$

Đầu ra : $UDML_{unified}$: DSL biểu diễn mô hình nhất bao gồm cú pháp trừu tượng AS, cú pháp cụ thể CS và ngữ nghĩa Sem

$AS_{UDML} \leftarrow AS_{core}$, $CS_{UDML} \leftarrow CS_{core}$, $Sem_{UDML} \leftarrow Sem_{core}$
foreach *mỗi* $DSL_i = (AS_i, CS_i, Sem_i) \in D$ **do**
    // Xử lý các khái niệm trong AS
    foreach *mỗi khái niệm* $c \in AS_i$ *sao cho* $c \notin AS_{UDML}$ **do**
        if *c mở rộng một khái niệm lõi* **then**
            Tích hợp quan hệ kế thừa vào AS_{UDML}
        end if
        Thêm các thuộc tính, quan hệ và ràng buộc của c vào AS_{UDML}
    end foreach
    foreach *mỗi cạnh* $(n_i, n_j) \in child_i \cup refCons_i$ **do**
        if $(n_i, n_j) \notin (child_{UDML} \cup refCons_{UDML})$ **then**
            Thêm cạnh (n_i, n_j) vào AS_{UDML}
        end if
        else
            Giải quyết xung đột bằng cách áp dụng quy tắc ánh xạ hoặc quy tắc ưu tiên
        end if
    end foreach
    // Xử lý cú pháp cụ thể
    foreach *mỗi khái niệm mới hoặc khái niệm được mở rộng* c **do**
        Thêm các ký hiệu đồ họa hoặc văn bản tương ứng vào CS_{UDML} Mở rộng các bộ xử lý UI (ví dụ: kéo-thả, chú thích, thao tác chỉnh sửa)
    end foreach
    // Tích hợp ngữ nghĩa
    Tích hợp Sem_i vào Sem_{UDML} theo kiểu mô-đun hợp nhất các ràng buộc mới vào P_{UDML} Kiểm tra tính nhất quán giữa các ngữ nghĩa xuyên mối quan tâm của Sem_i và Sem_{UDML}
end foreach
return $UDML_{unified} = (AS_{UDML}, CS_{UDML}, Sem_{UDML})$

thực thi, dựa trên các dãy hình chụp trạng thái phản ánh sự tiến hóa của hệ thống theo thời gian. Mỗi hình chụp biểu diễn một trạng thái hợp nhất, bao gồm trạng thái của mô hình lõi và các trạng thái do các DSL theo mối quan tâm đóng góp $(S_{\text{core}}, S_{DSL_c})$, qua đó thiết lập một ngữ nghĩa thao tác chung mà không phụ thuộc vào cách hiện thực cụ thể của từng DSL. Trên cơ sở đó, mục này xây dựng các định nghĩa hình thức cho mô hình UDML, điều kiện hợp lệ của trạng thái, và ngữ nghĩa thực thi, làm nền tảng cho việc phân tích, kiểm chứng và chuyển đổi mô hình bảo toàn ngữ nghĩa.

4.4.1 Định nghĩa hình thức các mô hình UDML

Định nghĩa hình thức các mô hình UDML như các mô hình miền có khả năng thực thi hợp nhất, thu được bằng cách kết hợp nhiều mối quan tâm

trực giao nhưng có liên hệ ngữ nghĩa với nhau.

Định nghĩa 4.6 (Mô hình UDML)**.** *Một mô hình UDML được định nghĩa là một bộ:* $\mathcal{M} = \langle \mathcal{U}, \mathcal{S}, \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$ *trong đó:*

- $\mathcal{U}$ *là siêu mô hình kết hợp lõi của UDML, cung cấp các trừu tượng cho mô-đun hóa mối quan tâm và liên kết chú thích, đồng thời độc lập với ngữ nghĩa thao tác,*
- $\mathcal{S}$ *là mô hình mối quan tâm cấu trúc DCSL, bao gồm các chú thích và các ràng buộc của DCSL được gắn với các phần tử cấu trúc của mô hình miền,*
- $\mathcal{B}$ *là mô hình mối quan tâm hành vi AGL, bao gồm các đồ thị hoạt động mà các nút và cạnh của chúng đặc trưng cho sự tiến hóa hành vi cho phép và các ranh giới thực thi cho các hành động miền,*
- $\mathcal{R}$ *là mô hình mối quan tâm bảo mật RBACDom, đặc tả các ràng buộc ủy quyền và SoD nhằm hạn chế khả năng thực thi của các phần tử hành vi (đặc biệt là việc thực thi nút), và*
- $\mathcal{C}$ *là tập các ràng buộc toàn cục, bao gồm các điều kiện hợp lệ và các ràng buộc nhất quán xuyên mối quan tâm phải được bảo toàn bởi mọi phép thực thi mô hình.*

Sự phân rã này làm rõ vai trò của từng mối quan tâm mô hình hóa, đồng thời cho phép tích hợp chúng một cách có hệ thống trong một mô hình miền có khả năng thực thi duy nhất. Cụ thể, mô hình hành vi $\mathcal{B}$ quyết định không gian các chuyển tiếp trạng thái cho phép, mô hình bảo mật $\mathcal{R}$ ràng buộc những chuyển tiếp nào có thể thực thi dưới các điều kiện ủy quyền cho trước, và tập ràng buộc $\mathcal{C}$ xác định các thuộc tính đúng đắn toàn cục phải được duy trì trong suốt quá trình thực thi.

Hình thức hóa lõi UDML. Lõi UDML cung cấp các trừu tượng nền tảng để kết hợp nhiều DSL chuyên biệt theo mối quan tâm theo cách thống nhất và không xâm lấn. Lõi này định nghĩa "kết dính" cấu trúc cho phép tích hợp các mối quan tâm cấu trúc, hành vi và bảo mật mà bản thân nó không áp đặt ngữ nghĩa thực thi. Việc định nghĩa ngữ nghĩa hình thức của UDML như sau.

- *Các phần tử miền lõi:* Lõi UDML được hình thức hóa như một bộ: $\mathcal{U} = \langle DM,\ CN,\ AN,\ AE,\ elements,\ concerns,\ annotations,\ target \rangle$. Giả sử các tập mang đôi một rời nhau sau đây được cho: $DM,\ CN,\ AN,\ AE$, trong đó DM là tập các mô hình miền; CN là tập các mối quan tâm được gắn với các mô hình miền; AN là tập các chú thích được dùng để biểu diễn thông tin đặc thù theo mối quan tâm; AE là tập các phần tử có thể gắn chú thích, tức là các phần tử của mô hình miền có thể được mở rộng bởi chú thích.

  Cho các phần tử miền lõi sau đây:

  $$Class,\ Property,\ Operation,\ Association \subseteq AE$$

  Các phần tử này cấu thành cú pháp trừu tượng của mô hình miền trong UDML. Chúng biểu diễn các khái niệm mô hình hóa cấu trúc cơ bản tương tự các lớp, thuộc tính, thao tác và liên kết trong UML mà trên đó hành vi miền và các ngữ nghĩa đặc thù theo mối quan tâm được định nghĩa. Vì vậy, các phần tử có thể gắn chú thích tạo thành một không gian đích chung cho việc tích hợp mối quan tâm: mọi DSL chuyên biệt theo mối quan tâm như DCSL, AGL và RBACDom đóng góp ngữ nghĩa bằng cách gắn chú thích lên các phần tử trong AE thay vì tái định nghĩa chính cấu trúc mô hình miền.

- *Các quan hệ lõi:* Các quan hệ lõi của $\mathcal{U}$ được định nghĩa như sau: $elements \subseteq DM \times AE$ liên kết mỗi mô hình miền với các phần tử có thể gắn chú thích của nó; $concerns \subseteq DM \times CN$ liên kết mỗi mô hình miền với các mối quan tâm được khai báo; $annotations \subseteq CN \times AN$ liên kết mỗi mối quan tâm với các chú thích của nó; $target : AN \to AE$ ánh xạ mỗi chú thích tới phần tử có thể gắn chú thích đích.

- *Các ràng buộc hợp lệ:* Các ràng buộc sau đây phải được thoả mãn.
  (U1) $\forall\, a \in AN \cdot target(a) \in AE$;
  (U2) $\forall\, a \in AN \cdot \exists! c \in CN : (c, a) \in annotations$;
  (U3) $\forall\, c \in CN \cdot \exists! d \in DM : (d, c) \in concerns$.
  (U4) $\forall\, a \in AN \cdot \exists\, d \in DM : (d, target(a)) \in elements \wedge \exists\, c \in CN : (d, c) \in concerns \wedge (c, a) \in annotations$.
  Các ràng buộc này đảm bảo cấu trúc sở hữu và gắn kết của các mối quan tâm và chú thích trong một mô hình miền được xác định rõ ràng.

Định nghĩa 4.7 (Hình chụp hệ thống (snapshot))**.** *Cho $\mathcal{M}$ là một mô hình UDML. Ký hiệu Σ là tập các hình chụp do $\mathcal{M}$ sinh ra. Mỗi hình chụp $\sigma \in \Sigma$ được định nghĩa như một trạng thái có cấu trúc: $\sigma \triangleq \langle S_{core}(\sigma),\ S_{DSL_c}(\sigma)\rangle$, trong đó $S_{core}(\sigma)$ biểu diễn trạng thái của mô hình miền lõi, còn $S_{DSL_c}(\sigma)$ biểu diễn trạng thái của các DSL theo mối quan tâm (ví dụ: hành vi từ AGL và ngữ cảnh ủy quyền từ RBACDom). Sự tương thích giữa các thành phần này được xác định thông qua các liên kết ngữ nghĩa trong lõi UDML.*

Dựa trên khái niệm hình chụp hệ thống, ta xác định điều kiện hợp lệ của một hình chụp như sau.

Định nghĩa 4.8 (Hình chụp hợp lệ)**.** *Cho $\mathcal{M}$ là một mô hình UDML và $\sigma \in \Sigma$. Hình chụp σ được gọi là* hợp lệ*, ký hiệu (σ), khi và chỉ khi σ thỏa mãn đồng thời: (i) các ràng buộc cấu trúc suy ra từ DCSL; (ii) các bất biến bảo mật suy ra từ RBACDom; (iii) các bất biến do các DSL theo mối quan tâm khác đóng góp; và (iv) các điều kiện liên kết ngữ nghĩa trong lõi UDML nhằm đảm bảo sự nhất quán giữa $S_{core}(\sigma)$ và $S_{DSL_c}(\sigma)$.*

Điều kiện hợp lệ này xác định các trạng thái có ý nghĩa về ngữ nghĩa, và sẽ được sử dụng để ràng buộc quá trình thực thi của mô hình.

Định nghĩa 4.9 (Ngữ nghĩa thực thi)**.** *Ngữ nghĩa của một mô hình UDML $\mathcal{M}$ được định nghĩa như một hệ chuyển: $\mathcal{M} = \langle \Sigma, \Sigma_0, \rightarrow \rangle$, trong đó Σ là tập các hình chụp, $\Sigma_0 \subseteq \Sigma$ là tập các hình chụp khởi tạo, và $\rightarrow \subseteq \Sigma \times ND \times \Sigma$ là quan hệ chuyển trạng thái. Một phép thực thi (hay một đường đi) của $\mathcal{M}$ là một dãy (hữu hạn hoặc vô hạn) $\sigma_0, \sigma_1, \ldots$ sao cho $\sigma_0 \in \Sigma_0$, (σ_0), và với mỗi $i \geq 0$, $\sigma_i \xrightarrow{n_i} \sigma_{i+1}$ với (σ_{i+1}).*

Một phép thực thi biểu diễn một lần chạy khả dĩ của mô hình $\mathcal{M}$, bắt đầu từ một hình chụp khởi tạo và tiến hóa qua các lần thực thi nút liên tiếp. Mỗi chuyển trạng thái tương ứng với việc thực thi một nút làm biến đổi trạng thái hệ thống từ hình chụp này sang hình chụp kế tiếp. Ở mỗi bước, điều kiện kích hoạt nút $Enable(n_i, s)$ được đánh giá trên hình chụp hiện tại σ_i, sử dụng thành phần phiên $s \in SES$ chứa trong S_{DSL_c}. Việc chuyển sang σ_{i+1} chỉ được phép nếu nó bảo toàn tất cả các ràng buộc cấu trúc do DCSL suy ra và tất cả các bất biến do các DSL_c đã được kết hợp đóng góp trong hình chụp kết quả.

Trong phần tiếp theo, luận án làm rõ một cách tường minh phép ánh xạ từ các phép thực thi của AGL sang ngữ nghĩa thực thi của mô hình UDML thông qua các hình chụp trạng thái hệ thống.

4.4.2 Ánh xạ thực thi trong AGL sang UDML

Để thiết lập mối liên hệ ngữ nghĩa giữa hành vi được đặc tả trong AGL và trạng thái của mô hình miền hợp nhất UDML, cần xác định một cơ chế ánh xạ giữa các trạng thái thực thi ở hai mức biểu diễn này. Trong cách tiếp cận đề xuất, mỗi hình chụp của UDML không chỉ bao gồm thông tin cấu trúc và các mối quan tâm liên quan, mà còn chứa thành phần hành vi được đặc tả bởi AGL. Do đó, hành vi AGL được xem như một phép chiếu của trạng thái hợp nhất trong UDML. Trên cơ sở đó, ta định nghĩa một ánh xạ giữa các hình chụp của AGL và UDML thông qua phép chiếu π_{AGL}.

Định nghĩa 4.10 (Ánh xạ các hình chụp)**.** *Cho $\mathcal{M}$ là một mô hình UDML và Σ_{AGL} là tập các hình chụp do đặc tả AGL của nó sinh ra. Một hình chụp của AGL $q \in \Sigma_{AGL}$ tương ứng với một hình chụp của UDML $\sigma \in \Sigma$, ký hiệu $q \sim \sigma$, khi và chỉ khi: $q = \pi_{AGL}(\sigma)$, trong đó $\pi_{AGL} : \Sigma \to \Sigma_{AGL}$ là phép chiếu trích xuất thành phần hành vi AGL từ $S_{DSL_c}(\sigma)$. Ánh xạ này cho phép liên kết ngữ nghĩa giữa trạng thái hành vi của AGL và trạng thái hợp nhất của UDML.*

Ví dụ. Ví dụ sau minh họa một phép thực thi của hệ thống OJS như một dãy các hình chụp do đặc tả AGL của nó sinh ra. $q_0 \xrightarrow{\text{submit}} q_1 \xrightarrow{\text{approve}} q_2$ tương ứng với phép thực thi UDML $\sigma_0 \xrightarrow{\text{submit}} \sigma_1 \xrightarrow{\text{approve}} \sigma_2$. Ta có các ánh xạ hình chụp $q_i \sim \sigma_i$, (i=0,2). Nếu RBACDom yêu cầu vai trò Manager cho thao tác approve, thì bước thứ hai chỉ được kích hoạt khi điều kiện ủy quyền được thỏa mãn; nếu không, phép thực thi sẽ dừng tại σ_1. Trong cả hai trường hợp, sự tương ứng về hành vi vẫn được bảo toàn.

Định lý 4.1 (Đồng thực thi trên UDML và AGL)**.** *Cho $\mathcal{M}$ là một mô hình UDML kết hợp AGL với một tập các DSL chuyên biệt theo mối quan tâm. Giả sử rằng mỗi DSL mối quan tâm được kết hợp chỉ ràng buộc hành vi bằng cách hạn chế việc kích hoạt nút (tức là bằng cách bổ sung các điều kiện bảo vệ và/hoặc các bất biến trạng thái), và không làm thay đổi hình chụp AGL. Khi đó: (i) mọi phép thực thi UDML khi được chiếu qua π_{AGL} sẽ cho ra một*

phép thực thi AGL hợp lệ; và (ii) mọi phép thực thi AGL thỏa mãn các ràng buộc kích hoạt và các bất biến do các mối quan tâm được kết hợp sinh ra đều có thể được "nâng" lên thành một phép thực thi UDML tương ứng.

Chứng minh *(Ánh xạ một - một).* Cho $\mathcal{M}$ là một mô hình UDML đảm báo tính hợp lệ. Theo cấu trúc xây dựng, mỗi hình chụp của UDML $\sigma \in \Sigma$ chứa một thành phần hành vi AGL, và phép chiếu π_{AGL} thiết lập một ánh xạ từ các hình chụp của UDML sang các hình chụp của AGL.

Tính đúng. Mỗi bước UDML $\sigma_i \xrightarrow{n_i} \sigma_{i+1}$ thực thi một nút hành vi n_i được định nghĩa bởi AGL. Vì các DSL mối quan tâm được kết hợp không làm thay đổi trạng thái hành vi AGL và chỉ hạn chế khi nào một nút được kích hoạt (thông qua các điều kiện bảo vệ) và những trạng thái nào là cho phép (thông qua các bất biến), mỗi bước UDML tương ứng với một bước AGL cho phép trên các trạng thái đã được chiếu. Do đó, chiếu một phép thực thi UDML qua π_{AGL} sẽ thu được một phép thực thi AGL hợp lệ.

Tính đầy đủ. Ngược lại, xét một phép thực thi AGL thỏa mãn tất cả các hạn chế kích hoạt và các bất biến do các mối quan tâm được kết hợp sinh ra. Bắt đầu từ một hình chụp UDML khởi tạo σ_0 với $\pi_{AGL}(\sigma_0) = q_0$, mỗi bước AGL có thể được hiện thực như một chuyển tiếp UDML vì cùng một nút được kích hoạt và hình chụp kết quả vẫn là cho phép. Lặp lại lập luận này theo từng bước sẽ xây dựng được một phép thực thi UDML mà phép chiếu của nó khớp với phép thực thi AGL đã cho.

Vì vậy, dọc theo các phép thực thi, các hình chụp của UDML và các hình chụp của AGL tương ứng một-một theo từng bước thông qua π_{AGL}.

4.4.3 Định nghĩa ngữ nghĩa UDML sử dụng Event-B

Mục này tập trung vào việc chuyển đổi mô hình UDML sang môi trường Event-B nhằm phục vụ cho mục đích mô phỏng và kiểm chứng các mô hình miền hợp nhất. Công việc tiếp theo là định nghĩa một phép biến đổi bảo toàn ngữ nghĩa, trong đó các mô hình UDML đã được tích hợp được chuyển dịch thành các ngữ cảnh và máy trạng thái của Event-B. Trên cơ sở các Định nghĩa 4.7, 4.8, 4.9 và 4.10, không gian trạng thái của UDML được đặc trưng bởi tập các hình chụp Σ, trong đó mỗi trạng thái của máy Event-B tương ứng với một hình chụp hợp lệ của mô hình UDML. Thông qua phép

chiếu π_{AGL}, mỗi hình chụp $\sigma \in \Sigma$ xác định một hình chụp hành vi $q \in \Sigma_{AGL}$, qua đó bảo toàn ngữ nghĩa thực thi của các phép chuyển trạng thái được đặc tả trong AGL. Ngược lại, mỗi sự kiện Event-B biểu diễn một chuyển trạng thái khả dĩ trong quan hệ $\rightarrow$, đồng thời phản ánh sự tiến hóa của thành phần hành vi được ánh xạ từ AGL. Do đó, ánh xạ từ UDML sang Event-B không chỉ bảo toàn cấu trúc trạng thái và các điều kiện bất biến, mà còn bảo toàn liên kết ngữ nghĩa giữa hành vi AGL và trạng thái hợp nhất của UDML, qua đó tạo cơ sở cho kiểm chứng hình thức và chuyển đổi bảo toàn ngữ nghĩa.

Thuật toán chuyển đổi mô hình UDML2Event-B 4.2 về cơ bản được xác định bởi ánh xạ, các phần tử cấu trúc và các ràng buộc toàn cục được chuyển dịch thành các "ngữ cảnh" trong Event-B, trong khi hành vi có khả năng thực thi và cơ chế cưỡng chế bảo mật được chuyển dịch thành các "máy trạng thái" của Event-B. Sự phân tách này phù hợp với kỹ thuật mô hình hóa của Event-B và hỗ trợ suy luận mô-đun về cấu trúc miền, hành vi và kiểm soát truy cập.

Ánh xạ các phần tử cấu trúc. Các thành phần cấu trúc được đặc tả bằng DCSL bao gồm: các lớp miền, thuộc tính, liên kết và các bất biến cấu trúc được ánh xạ sang các *ngữ cảnh - Context* của Event-B. Các lớp miền được biểu diễn dưới dạng các tập mang, các thuộc tính và liên kết được biểu diễn dưới dạng các biến hoặc hằng với các bất biến kiểu tương ứng, và các ràng buộc cấu trúc được chuyển dịch thành các bất biến của Event-B. Phép ánh xạ này xác lập không gian trạng thái miền tĩnh, trên đó hành vi được thực thi.

Ánh xạ các mô hình hành vi. Hành vi được đặc tả bằng AGL được ánh xạ sang các *máy trạng thái - machine* của Event-B. Mỗi đồ thị hoạt động được chuyển thành một *máy trạng thái*, trong đó các biến trạng thái mã hóa trạng thái điều khiển hiện tại của đồ thị. Các nút của đồ thị hoạt động được ánh xạ thành các sự kiện của Event-B, còn các cạnh xác định quan hệ luồng điều khiển giữa các sự kiện. Việc kích hoạt một sự kiện tương ứng với việc thực thi nút hoạt động liên quan, dẫn đến một chuyển trạng thái phản ánh cả sự tiến triển của luồng điều khiển và các cập nhật trạng thái miền do các hành động mô-đun tương ứng gây ra.

Ánh xạ các ràng buộc RBACDom. Các mối quan tâm bảo mật đặc tả

Thuật toán 4.2 UDML2Event-B: Chuyển đổi UDML sang Event-B và kiểm chứng.

Đầu vào: Một mô hình UDML hợp lệ $M = \langle \mathcal{U}, \mathcal{S}, \mathcal{B}, \mathcal{R}, \mathcal{C} \rangle$.
Đầu ra : Một phát triển Event-B $\mathcal{D}(M) = \langle C, \mathsf{Mach} \rangle$ và thống kê chứng minh.

$\mathcal{T}_D$ — các hàm chuyển cho cấu trúc ; $\mathcal{T}_A$ — các hàm chuyển cho hành vi ; $\mathcal{T}_S$ — các hàm chuyển cho bảo mật ; $\mathsf{WF}(\cdot)$ — bộ kiểm tra hợp lệ DM hợp nhất ; $\mathsf{AnnRbac}(n)$ — `RbacDomNode` (tùy chọn) gắn với nút n

assert $\mathsf{WF}(M)$ `// Khởi tạo phát triển Event-B`

Khởi tạo một *context* Event-B rỗng $C \leftarrow \langle \mathsf{SETS}, \mathsf{CONSTS}, \mathsf{AXMS} \rangle$
Khởi tạo một *machine* Event-B rỗng $\mathsf{Mach} \leftarrow \langle \mathsf{VARS}, \mathsf{INVS}, \mathsf{EVTS} \rangle$
Liên kết Mach với C (tức là Mach **sees** C)

`// Dịch phần cấu trúc`

$(\mathsf{SETS}_D, \mathsf{CONSTS}_D, \mathsf{AXMS}_D) \leftarrow \mathcal{T}_D^{ctx}(M.\mathcal{S})$
$C.\mathsf{SETS} \leftarrow C.\mathsf{SETS} \cup \mathsf{SETS}_D$
$C.\mathsf{CONSTS} \leftarrow C.\mathsf{CONSTS} \cup \mathsf{CONSTS}_D$
$C.\mathsf{AXMS} \leftarrow C.\mathsf{AXMS} \cup \mathsf{AXMS}_D$
$\mathsf{Mach.INVS} \leftarrow \mathsf{Mach.INVS} \cup \mathcal{T}_D^{inv}(M.\mathcal{S})$

`// Dịch phần hành vi`

$C.\mathsf{SETS} \leftarrow C.\mathsf{SETS} \cup \{\texttt{NODES}, \texttt{EDGES}, \texttt{GRAPHS}\}$
$C.\mathsf{CONSTS} \leftarrow C.\mathsf{CONSTS} \cup \{src, tgt, nodesOf, edgesOf, initOf\}$
$C.\mathsf{AXMS} \leftarrow C.\mathsf{AXMS} \cup \mathcal{T}_A^{ctx}(M.\mathcal{B})$

Thêm các biến máy trạng thái cho thực thi: $\mathsf{Mach.VARS} \leftarrow \mathsf{Mach.VARS} \cup \{curG, pc\}$; Thêm các bất biến hành vi: $\mathsf{Mach.INVS} \leftarrow \mathsf{Mach.INVS} \cup \mathcal{T}_A^{inv}(M.\mathcal{B}, curG, pc)$; Sinh các sự kiện hành vi: $\mathsf{Mach.EVTS} \leftarrow \mathsf{Mach.EVTS} \cup \mathcal{T}_A^{evt}(M.\mathcal{B}, curG, pc)$

`// Dịch phần bảo mật (RBAC/SoD)`

$(\mathsf{SETS}_S, \mathsf{CONSTS}_S, \mathsf{AXMS}_S) \leftarrow \mathcal{T}_S^{ctx}(M.\mathcal{R})$
$C.\mathsf{SETS} \leftarrow C.\mathsf{SETS} \cup \mathsf{SETS}_S$
$C.\mathsf{CONSTS} \leftarrow C.\mathsf{CONSTS} \cup \mathsf{CONSTS}_S$
$C.\mathsf{AXMS} \leftarrow C.\mathsf{AXMS} \cup \mathsf{AXMS}_S$

Thêm các hằng $permAction \in \texttt{PERMISSIONS} \to \texttt{ACTIONS}$ và $permRes \in \texttt{PERMISSIONS} \to \texttt{RESOURCES}$; Thêm các tiên đề gán kiểu cho $permAction, permRes$; Thêm các biến trạng thái RBAC: $\mathsf{Mach.VARS} \leftarrow \mathsf{Mach.VARS} \cup \mathcal{T}_S^{vars}(M.\mathcal{R})$; Thêm các bất biến RBAC (bao gồm SoD): $\mathsf{Mach.INVS} \leftarrow \mathsf{Mach.INVS} \cup \mathcal{T}_S^{inv}(M.\mathcal{R})$; Thêm hằng (hoặc biến) $mapRes \in \texttt{RESOURCES} \rightharpoonup \texttt{AE}$; Thêm tiên đề/bất biến $\mathsf{Typing}(mapRes)$

`// Vị từ suy diễn dùng trong guard`

$EnabledRoles(sess) \triangleq active(sess)$
$EnabledPerms(sess) \triangleq \{p \mid \exists\, r \in active(sess) \cdot (r, p) \in pa\}$

foreach *event thực thi nút* $ev_n \in \mathsf{Mach.EVTS}$ *tương ứng với nút* $n \in ND$ **do**
  Đặt $Allow \leftarrow \varnothing$ và $Deny \leftarrow \varnothing$ **foreach** $rna \in Ann(n)$ **do**
    Tính $Applies(rna, sess) \equiv RolesSat(rna, sess) \wedge PermsSat(rna, sess) \wedge SoDSat(rna, sess)$ **if** $effect(rna) = ALLOW$ **then** $Allow \leftarrow Allow \cup \{Applies(rna, sess)\}$;
    else $Deny \leftarrow Deny \cup \{Applies(rna, sess)\}$;
  end foreach
  Định nghĩa $AllowApplies(n, sess) \equiv \bigvee Allow$; Định nghĩa $DenyApplies(n, sess) \equiv \bigvee Deny$; Tăng cường guard(ev_n) với $AllowApplies(n, sess) \wedge \neg DenyApplies(n, sess)$
end foreach

Sinh *INITIALISATION* để khởi tạo nhất quán mọi biến máy theo các bất biến: đặt $curG$ bằng thể hiện đồ thị được chọn (hoặc đồ thị mặc định) ; đặt $pc \leftarrow initOf(curG)$; khởi tạo $ua, pa, sess, active$ (và mọi biến trạng thái miền) sao cho mọi bất biến đều thỏa

`// Sinh và giải nghĩa vụ chứng minh`

Sinh các nghĩa vụ chứng minh $PO(\mathcal{D}(M))$; Giải $PO(\mathcal{D}(M))$ bằng các bộ chứng minh Rodin và ghi nhận thống kê chứng minh

return $(\mathcal{D}(M), PO\text{-statistics})$

bằng RBACDom được tích hợp trực tiếp vào ngữ nghĩa hành vi thông qua các điều kiện bảo vệ của sự kiện và các bất biến. Mỗi `RbacDomNode` tương ứng với một nút AGL (thông qua thuộc tính `aglNode`) được chuyển dịch thành các vị từ gác trên sự kiện Event-B tương ứng. Các vị từ gác này mã hóa các điều kiện kích hoạt nút, được suy ra từ yêu cầu về vai trò, kiểm tra quyền và các ràng buộc phân tách nhiệm vụ (SoD). Theo cách này, ủy quyền được cưỡng chế như một điều kiện tiên quyết cho việc thực thi sự kiện, thay vì là một kiểm tra bên ngoài áp đặt sau cùng.

Các ràng buộc SoD trải rộng trên nhiều nút hoặc phiên được chuyển thành các bất biến trong Event-B, bảo đảm mọi trạng thái đạt được đều thỏa mãn các thuộc tính bảo mật, qua đó duy trì các ràng buộc RBACDom trong toàn bộ quá trình thực thi và tinh chỉnh. Bảng 4.2 tóm tắt các quy tắc ánh xạ giữa UDML và Event-B. Theo đó, ngữ nghĩa thực thi của UDML được đặc trưng bởi hành vi của các *context*, *machine* Event-B và các nghĩa vụ chứng minh tương ứng.

Thuật toán 4.2 hiện thực hóa phép chuyển bảo toàn của UDML được định nghĩa hình thức ở trên, đồng thời xem UDML như một tầng tích hợp thống nhất cho các mối quan tâm cấu trúc, hành vi và bảo mật. Quá trình biên dịch được tổ chức thành các pha liên tiếp, mỗi pha tương ứng với một khối dòng được xác định rõ trong thuật toán.

Biên dịch cấu trúc (Dòng 1–4). Kiểm tra tính hợp lệ của mô hình đầu vào và khởi tạo ngữ cảnh và máy trạng thái Event-B. Sau đó, đặc tả cấu trúc được biểu diễn trong DCSL được chuyển đổi.

Biên dịch hành vi (Dòng 5–6). Chuyển dịch đặc tả hành vi trong AGL. Các tập mang và các hằng được đưa vào để biểu diễn các nút, cạnh và các quan hệ luồng điều khiển của đồ thị hoạt động. Một biến trạng thái điều khiển *pc* được thêm vào để ghi nhận nút đang hoạt động.

Biên dịch bảo mật (Dòng 7–8). Biên dịch đặc tả bảo mật RBACDom. Các khái niệm RBAC cốt lõi, bao gồm người dùng, vai trò, quyền hạn và phiên, được chuyển dịch thành các tập mang và các hằng. Các quan hệ gán và kích hoạt vai trò được biên dịch thành các biến của máy trạng thái, trong khi các ràng buộc phân tách nhiệm vụ và các ràng buộc RBAC khác được chuyển dịch thành các bất biến phải được bảo toàn bởi mọi sự kiện.

Tích hợp liên-mối quan tâm (Dòng 9). Tích hợp các mối quan tâm

Bảng 4.2: UDML2Event-B: Ánh xạ từ UDML sang Event-B

Cấu trúc UDML	**Hiện vật Event-B**	**Mô tả**
Mô hình miền (DomainModel)	Ngữ cảnh	Định nghĩa miền tĩnh của hệ thống, bao gồm các tập kiểu cơ sở và các hằng số toàn cục.
Mối quan tâm (Concern)	Ngữ cảnh	Dùng để tổ chức các hằng, tiên đề và bất biến liên quan đến một mối quan tâm mô hình hóa cụ thể.
Lớp miền (DClass)	Tập mang / Hằng	Mỗi lớp miền được ánh xạ thành một tập mang hoặc một tập trừu tượng biểu diễn các thể hiện của nó.
Thuộc tính (DAttr)	Biến / Hàm	Thuộc tính được ánh xạ thành các biến trạng thái hoặc các hàm, trong đó miền xác định và miền giá trị được ràng buộc bởi các bất biến.
Liên kết (DAssoc)	Quan hệ / Bất biến	Liên kết được ánh xạ thành các quan hệ giữa các tập mang, với các ràng buộc cơ chế thực thi dưới dạng bất biến.
Đồ thị hoạt động (AGraph)	Máy trạng thái	Mỗi đồ thị hoạt động được ánh xạ thành một machine Event-B nắm bắt ngữ nghĩa thực thi của nó.
Nút khởi tạo	Khởi tạo sự kiện	Nút khởi tạo của đồ thị hoạt động xác định giá trị ban đầu của biến trạng thái điều khiển.
Nút (ANode)	Sự kiện	Mỗi nút được ánh xạ thành một sự kiện Event-B biểu diễn một bước thực thi nguyên tử.
Cạnh	Điều kiện bảo vệ	Các cạnh được ánh xạ thành các điều kiện ràng buộc bảo vệ sự kiện-nút nào có thể bắn dựa trên trạng thái điều khiển hiện tại.
Trạng thái điều khiển (nút hiện tại)	Biến	Nút đang hoạt động được biểu diễn bởi một biến trạng thái trong máy trạng thái.
Mô-đun hoạt động (ModuleAct)	Hành động sự kiện	Các hành động mô-đun được chuyển dịch thành các phép thay thế Event-B cập nhật trạng thái của máy trạng thái.
Nút bảo mật (RbacDomNode)	Điều kiện bảo vệ sự kiện	Các ràng buộc RBAC được đặc tả bởi `RbacDomNode` (tương ứng với các nút AGL) được ánh xạ thành các điều kiện bảo vệ nhằm cưỡng chế các yêu cầu về vai trò và quyền hạn.
Các vai trò yêu cầu	Vị từ bảo vệ	Yêu cầu vai trò được mã hóa dưới dạng các vị từ trên tập vai trò đang kích hoạt của phiên hiện tại.
Các quyền yêu cầu	Vị từ bảo vệ	Yêu cầu quyền hạn được mã hóa dưới dạng các vị từ suy ra từ các quan hệ gán vai trò–quyền hạn.
Phiên	Biến	Ngữ cảnh thực thi hiện tại được biểu diễn như một biến phiên.
Ràng buộc SoD	Bất biến	Các ràng buộc SoD được mã hóa thành các bất biến hạn chế việc gán vai trò hoặc kích hoạt vai trò.
Chính sách (ALL_OF / ANY_OF)	Lô-gic bảo vệ	Ngữ nghĩa của chính sách chi phối cách kết hợp các điều kiện bảo vệ RBAC theo phép AND hoặc OR.
Tác động (ALLOW / DENY)	Ngữ nghĩa bảo vệ	Hiệu lực xác định liệu điều kiện bảo vệ cho phép hay chặn việc thực thi sự kiện.
Chiến lược kết hợp các quy tắc (DENY_OVERRIDES)	Xây dựng điều kiện bảo vệ	Các điều kiện bảo vệ phân quyền được xây dựng dưới dạng $AllowApplies \wedge \neg DenyApplies$, đảm bảo rằng các chính sách từ chối có thể áp dụng sẽ chặn thực thi sự kiện ngay cả khi tồn tại các chính sách cho phép có thể áp dụng.
Các quy tắc hợp lệ	Bất biến / Tiên đề	Các ràng buộc hợp lệ được chuyển dịch thành các bất biến hoặc tiên đề để đảm bảo tính nhất quán của mô hình.

hành vi và bảo mật. Với mỗi sự kiện Event-B tương ứng với một nút đồ thị hoạt động được bảo vệ, điều kiện bảo vệ của sự kiện được tăng cường bằng một vị từ phân quyền được suy ra từ `RbacDomNode` liên kết.

Tính áp dụng của luật (Dòng 10–16). Định nghĩa tính áp dụng của các luật phân quyền và việc xây dựng các điều kiện bảo vệ phân quyền cho các chính sách từ chối. Để mã hóa đúng ngữ nghĩa kết hợp luật, chúng tôi áp dụng chiến lược `DENY_OVERRIDES`, theo đó việc thực thi một nút chỉ được cho phép khi tồn tại ít nhất một chính sách cho phép có thể áp dụng và không có chính sách từ chối có thể áp dụng nào.

Khởi tạo và kiểm chứng (Dòng 17–18). Sinh sự kiện khởi tạo, thiết lập một trạng thái ban đầu nhất quán cho mọi biến của máy trạng thái. Cuối cùng, các nghĩa vụ chứng minh được sinh ra và được giải bằng nền tảng Rodin.

Để cho phép các DM có khả năng thực thi và hỗ trợ sinh tự động các tạo tác phần mềm, các mối quan tâm bảo mật cần được được ánh xạ 1-1 từ DSL ngoại sinh sang DSL nội sinh nhúng vào ngôn ngữ lập trình. Cụ thể, RBACDom đòi hỏi một cú pháp cụ thể cho phép các chính sách kiểm soát truy cập được biểu diễn một cách tường minh, tích hợp liền mạch với các DM và hành vi, đồng thời có thể được xử lý bởi các chuỗi công cụ hướng mô hình.

4.5 Tổng kết chương

Trong chương này, luận án đã trình bày phương pháp tổng hợp các ngôn ngữ chuyên biệt miền theo mối quan tâm vào một DM hợp nhất có khả năng thực thi theo DDD, cùng với một khung kiểm chứng ngữ nghĩa hình thức cho UDML. Phương pháp đề xuất cho phép tích hợp có hệ thống các mối quan tâm cấu trúc, hành vi và bảo mật trong một DM thống nhất. Trên cơ sở đó, chương đã xây dựng và tích hợp DSL bảo mật RBACDom vào UDML, đồng thời thực hiện kiểm chứng hình thức DM hợp nhất ở giai đoạn thiết kế nhằm đảm bảo tính nhất quán và tính đúng đắn ngữ nghĩa của các đặc tả bảo mật động trước khi sinh bản mẫu phần mềm.

Các kết quả nghiên cứu cốt lõi của chương này đã được công bố trong bài báo hội thảo quốc tế RIVF 2024 *[V3]*, được mở rộng trong SOICT 2025 *[V5]*, và tiếp tục phát triển theo hướng tích hợp khía cạnh bảo mật và kiểm chứng hình thức cho UDML trong bài báo tạp chí JCSCE 2026 *[V6]*.

Chương 5

SINH TỰ ĐỘNG BẢN MẪU PHẦN MỀM DỰA VÀO MÔ HÌNH MIỀN HỢP NHẤT

Trong chương này, luận án trình bày một bộ chuyển đổi mô hình nhằm sinh tự động bản mẫu phần mềm từ mô hình miền hợp nhất trong bối cảnh thiết kế hướng miền. Trọng tâm là các kỹ thuật chuyển đổi có hệ thống, góp phần thu hẹp khoảng cách giữa đặc tả yêu cầu và hiện thực phần mềm, đồng thời bảo đảm tính nhất quán ngữ nghĩa xuyên suốt quá trình chuyển đổi. Cụ thể, luận án đề xuất kỹ thuật sinh mô hình miền hợp nhất *UDM* từ mô hình yêu cầu *RM* thông qua bộ chuyển RM2UDM, trong đó *RM* được đặc tả bằng biểu đồ lớp và biểu đồ hoạt động UML/OCL, còn *UDM* tuân thủ siêu mô hình *UDML*. Trên cơ sở đó, *UDM* được chuyển đổi thành đặc tả miền thực thi AGL^+ dưới dạng mã nguồn bằng thông qua kỹ thuật chuyển đổi mô hình–sang–văn bản với bộ chuyển UDM2AGL. Các kỹ thuật này được đặc tả dưới dạng thuật toán và các luật ánh xạ tương ứng, bảo đảm quá trình chuyển đổi từ mô hình yêu cầu đến mô hình thực thi diễn ra một cách có hệ thống và nhất quán. Từ đặc tả AGL^+, bản mẫu phần mềm được sinh tự động trên nền tảng JDA, hỗ trợ đánh giá và tinh chỉnh mô hình theo quy trình phát triển lặp.

5.1 Giới thiệu

Trong quá trình phát triển phần mềm, việc xây dựng bản mẫu phần mềm đóng vai trò quan trọng trong việc diễn đạt ý tưởng, xác minh yêu cầu và cải thiện giao tiếp giữa các bên liên quan. Bản mẫu phần mềm cho phép kiểm tra và tinh chỉnh các tính năng trước khi triển khai, từ đó giảm thiểu rủi ro và góp phần bảo đảm sản phẩm đáp ứng đúng nhu cầu người dùng [120]. Tuy nhiên, việc tự động hóa hoạt động này vẫn là một thách thức do khoảng cách ngữ nghĩa giữa các đặc tả yêu cầu bằng UML/OCL [94] và mã nguồn dùng để hiện thực hệ thống.

Nhiều nghiên cứu đã đề xuất các phương pháp sinh tự động bản mẫu phần mềm từ đặc tả yêu cầu, chủ yếu tập trung vào sinh giao diện người dùng hoặc các thành phần riêng lẻ từ mô hình [11, 15, 104, 134]. Một số công trình khai thác mô hình miền hoặc các biểu đồ UML để sinh bản mẫu phần mềm [70, 87, 98, 122]. Gần đây, thiết kế hướng miền (DDD) [41, 124] được xem là một hướng tiếp cận hiệu quả để thu hẹp khoảng cách giữa mô hình và hiện thực. Tuy nhiên, hai thách thức chính vẫn còn tồn tại: (i) đặc tả chính xác mô hình yêu cầu theo hướng miền; và (ii) xây dựng các phép chuyển đổi để sinh tự động bản mẫu phần mềm từ các đặc tả này.

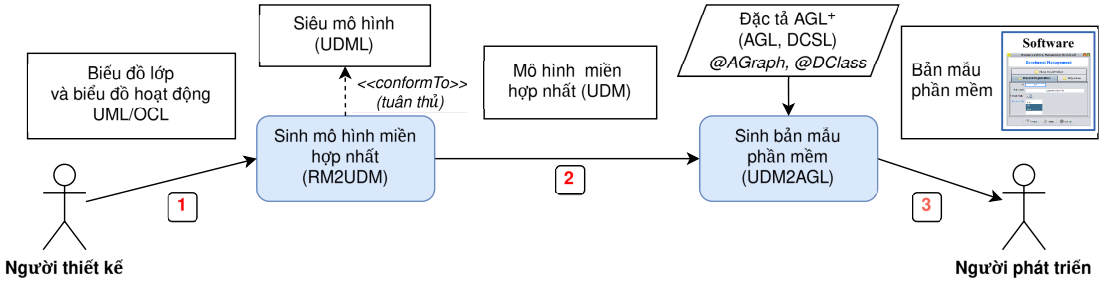
Trong nghiên cứu trình bày tại Mục 3.3, hành vi miền đã được tích hợp vào mô hình miền nhằm hợp nhất các khía cạnh cấu trúc và hành vi. Tuy nhiên, việc chuyển đổi từ đặc tả yêu cầu mức cao (biểu đồ lớp và biểu đồ hoạt động UML) sang một biểu diễn miền có khả năng thực thi, và tiếp tục sinh bản mẫu phần mềm, vẫn là một thách thức đáng kể.

Trong bối cảnh đó, luận án đề xuất một bộ chuyển đổi mô hình nhằm sinh tự động bản mẫu phần mềm từ mô hình yêu cầu theo hướng thiết kế hướng miền. Bộ chuyển đổi gồm hai bước chính: (i) chuyển đổi mô hình yêu cầu *RM* sang mô hình miền hợp nhất *UDM* thông qua bộ chuyển RM2UDM; và (ii) hiện thực hóa *UDM* thành đặc tả miền thực thi AGL^+ để nhúng vào ngôn ngữ lập trình hướng đối tượng (như Java) bằng kỹ thuật chuyển đổi mô hình sang văn bản UDM2AGL, từ đó sinh bản mẫu phần mềm trên nền tảng JDA [74]. Cách tiếp cận này thiết lập một chuỗi chuyển đổi thống nhất từ mô hình yêu cầu đến hệ thống thực thi, góp phần thu hẹp khoảng cách giữa đặc tả mức cao và hiện thực phần mềm.

Các mục còn lại của chương được tổ chức như sau. Mục 5.2 trình bày tổng quan phương pháp. Mục 5.3 trình bày kỹ thuật sinh mô hình miền hợp nhất từ mô hình yêu cầu. Mục 5.4 trình bày quá trình hiện thực hóa mô hình miền và sinh bản mẫu phần mềm. Cuối cùng, tổng kết chương được trình bày trong Mục 5.5.

5.2 Tổng quan phương pháp

Phần này trình bày tổng quan phương pháp sinh tự động bản mẫu phần mềm từ mô hình yêu cầu dựa trên các kỹ thuật chuyển đổi mô hình trong bối cảnh thiết kế hướng miền. Như minh họa trong Hình 5.1, phương pháp được tổ chức thành một bộ chuyển đổi mô hình thống nhất, vận hành theo cơ chế lặp, nhằm từng bước thu hẹp khoảng cách giữa đặc tả yêu cầu và hiện thực phần mềm. Phương pháp được tổ chức thành ba giai đoạn chính, tương ứng với các bước chuyển đổi từ mô hình yêu cầu đến bản mẫu phần mềm.



Hình 5.1: Tổng quan phương pháp sinh tự động bản mẫu phần mềm dựa vào mô hình miền hợp nhất.

Giai đoạn 1 – Chuyển đổi mô hình yêu cầu sang mô hình miền hợp nhất. Đầu vào của giai đoạn này là mô hình yêu cầu được biểu diễn dưới dạng biểu đồ lớp và biểu đồ hoạt động UML/OCL. Thông qua phương pháp RM2UDM, mô hình yêu cầu được chuyển đổi thành mô hình miền hợp nhất *UDM* tuân thủ siêu mô hình *UDML*. Kết quả thu được là một biểu diễn trung gian thống nhất, trong đó khía cạnh cấu trúc được đặc tả bằng DCSL, khía cạnh hành vi được biểu diễn ở mức khái niệm bằng AGL, đồng thời tích hợp các cấu hình mô-đun phục vụ cho các bước hiện thực hóa tiếp theo.

Giai đoạn 2 – Hiện thực hóa mô hình miền. Từ mô hình miền hợp nhất *UDM*, phép chuyển đổi mô hình–sang–văn bản UDM2AGL được áp dụng để sinh đặc tả miền thực thi AGL^+ dưới dạng mã nguồn có chú thích. Trong đó, các thành phần cấu trúc miền được ánh xạ sang các lớp với chú thích như `@DClass`, trong khi hành vi miền được biểu diễn thông qua các cấu trúc tương ứng với `@AGraph` và các chú thích liên quan. Kết quả của giai đoạn này là một mô hình miền thực thi ở mức mã nguồn, cho phép ánh xạ trực tiếp sang các thành phần phần mềm.

Giai đoạn 3 – Sinh bản mẫu phần mềm. Trên cơ sở đặc tả miền thực thi AGL^+, nền tảng khung phần mềm JDA được sử dụng để sinh tự động bản mẫu phần mềm. Các chú thích trong AGL^+ được khai thác để tạo ra các thành phần như lớp dữ liệu, giao diện người dùng và cơ chế điều khiển, qua đó hình thành một hệ thống phần mềm có thể thực thi. Bản mẫu sinh ra được sử dụng để đánh giá với chuyên gia miền; nếu có phản hồi, mô hình được cập nhật và chuỗi chuyển đổi được lặp lại cho đến khi đáp ứng yêu cầu.

5.3 Sinh mô hình miền hợp nhất từ mô hình yêu cầu

Phần này trình bày bộ chuyển RM2UDM, là một kỹ thuật chuyển đổi từ mô hình yêu cầu sang mô hình miền hợp nhất, làm cơ sở cho việc sinh mô hình miền thực thi và bản mẫu phần mềm ở các bước tiếp theo. Ý tưởng cốt lõi là đề xuất một kỹ thuật chuyển đổi có hệ thống từ đặc tả yêu cầu ở mức khái niệm sang một biểu diễn trung gian hợp nhất, qua đó thu hẹp khoảng cách ngữ nghĩa giữa mô hình yêu cầu và hiện thực phần mềm trong bối cảnh thiết kế hướng miền.

Trong phương pháp này, mô hình yêu cầu được biểu diễn dưới dạng một cặp: $RM = \langle CD, AD \rangle$, trong đó CD là biểu đồ lớp UML/OCL đặc tả khía cạnh cấu trúc của miền, còn AD là biểu đồ hoạt động UML đặc tả khía cạnh hành vi. Mục tiêu là xây dựng một biểu diễn miền hợp nhất: $UDM = \langle DM, AG, MC \rangle$, tuân thủ siêu mô hình *UDML*, DM biểu diễn phần cấu trúc của mô hình miền dưới dạng DCSL, AG biểu diễn hành vi miền dưới dạng đồ thị hoạt động, và MC biểu diễn cấu hình mô-đun phục vụ tổ chức và sinh các thành phần phần mềm.

UDML là siêu mô hình quy định cú pháp và ngữ nghĩa của mô hình miền hợp nhất, trong khi *UDM* là một thể hiện cụ thể tuân thủ *UDML* và được xây dựng từ mô hình yêu cầu. Do đó, *UDM* đóng vai trò là biểu diễn trung gian hợp nhất, tích hợp các khía cạnh cấu trúc, hành vi và cấu hình trong một mô hình thống nhất. Trên cơ sở này, kỹ thuật chuyển đổi RM2UDM thực hiện chuyển đổi và tích hợp mô hình yêu cầu thành *UDM*, bảo toàn thông tin cấu trúc và hành vi, đồng thời chuẩn hóa theo khuôn khổ *UDML*. Mô hình thu được là đầu vào cho bước hiện thực hóa mô hình miền và sinh bản mẫu phần mềm trong Mục 5.4.

5.3.1 Tổng quan chuyển đổi từ RM sang UDM (RM2UDM)

Việc xây dựng thể hiện UDM từ mô hình yêu cầu theo UDML gồm hai thành phần: (i) Biểu diễn khía cạnh cấu trúc, tương ứng với DCSL; (ii) Biểu diễn khía cạnh hành vi, tương ứng với AGL.

Biểu diễn khía cạnh cấu trúc. Khía cạnh cấu trúc của UDM được xây dựng từ biểu đồ lớp đầu vào và được biểu diễn dưới dạng mô hình DCSL kết hợp với cấu hình mô-đun. Thành phần này bảo toàn các lớp miền, các thuộc tính, các quan hệ kết hợp và các ràng buộc cấu trúc được đặc tả trong mô hình yêu cầu. Cụ thể:

- Một lớp miền hoạt động c_a được xác định tương ứng với mỗi biểu đồ hoạt động đầu vào nhằm biểu diễn ngữ cảnh thực thi của hành vi miền.
- Các lớp miền $c_1, \ldots, c_n$ được xác định tương ứng với các lớp trong biểu đồ lớp đầu vào.
- Với mỗi tập lớp thành phần $c_{i1}, \ldots, c_{ik} \in \{c_1, \ldots, c_n\}$ tham gia thực hiện các hành động miền, các liên kết tương ứng được bổ sung vào lớp hoạt động và các lớp liên quan để phản ánh các quan hệ cấu trúc cần thiết cho quá trình thực thi.
- Mỗi lớp miền sau khi được ánh xạ vào mô hình DCSL tiếp tục được gắn với một mô-đun cấu hình tương ứng nhằm hỗ trợ tổ chức phần mềm và sinh mã bản mẫu.

Biểu diễn khía cạnh hành vi. Khía cạnh hành vi của UDM được xây dựng bằng cách ánh xạ biểu đồ hoạt động đầu vào sang đồ thị hoạt động AGL.

Trong đó, hành vi được tích hợp vào mô hình miền thông qua lớp hoạt động, đóng vai trò trung tâm để liên kết các lớp dữ liệu, lớp điều khiển và các quan hệ thực thi. Các thành phần chính bao gồm:

- Lớp hoạt động: là lớp miền đại diện cho một hoạt động nghiệp vụ.
- Lớp dữ liệu: là lớp miền đại diện cho các kho dữ liệu hoặc thực thể dữ liệu tham gia vào hoạt động.
- Lớp điều khiển: là lớp nắm bắt trạng thái miền gắn với các nút điều khiển của biểu đồ hoạt động, chẳng hạn như nút quyết định, nút rẽ nhánh, nút hợp nhất hoặc nút đồng bộ.
- Lớp điều phối: là lớp đại diện cho một nhóm nhiệm vụ điều phối trong luồng thực thi.
- Các liên kết dành riêng cho hoạt động: là các liên kết được bổ sung giữa lớp hoạt động và các lớp dữ liệu, lớp điều khiển hoặc các lớp liên quan khác nhằm phản ánh đúng luồng điều khiển và luồng dữ liệu của biểu đồ hoạt động đầu vào.

Như vậy, UDM thu được là một mô hình miền hợp nhất cụ thể, trong đó cấu trúc và hành vi được tích hợp trong cùng một biểu diễn tuân thủ UDML. Đây là cơ sở trực tiếp để thực hiện bộ chuyển đổi RM2UDM.

5.3.2 Đặc tả bộ chuyển đổi mô hình RM2UDM

Bộ chuyển đổi RM2UDM được xây dựng nhằm tự động chuyển mô hình yêu cầu RM sang mô hình miền hợp nhất UDM. Về bản chất, đây là một phép chuyển đổi mô hình–sang–mô hình (M2M), trong đó mô hình nguồn là RM và mô hình đích là một thể hiện UDM tuân thủ siêu mô hình UDML.

Thuật toán 5.1 được thực hiện trên đồ thị hoạt động theo một thứ tự duyệt xác định (duyệt theo chiều sâu – DFS) và sử dụng các ánh xạ trung gian nhằm bảo đảm tính nhất quán giữa mô hình nguồn và mô hình đích. Thuật toán được tổ chức thành ba pha chính. Pha thứ nhất xây dựng phần cấu trúc của mô hình đích từ biểu đồ lớp đầu vào, bao gồm các lớp miền, thuộc tính, quan hệ, ràng buộc và các mô-đun cấu hình tương ứng. Pha thứ hai xây dựng phần hành vi từ biểu đồ hoạt động, trong đó các nút

Thuật toán 5.1 Sinh mô hình UDM từ mô hình RM bằng bộ chuyển RM2UDM

Đầu vào: Mô hình yêu cầu $RM = \langle \mathsf{CD}, \mathsf{AD} \rangle$

- CD: Biểu đồ lớp UML/OCL đặc tả khía cạnh cấu trúc;
- AD: Biểu đồ hoạt động UML đặc tả khía cạnh hành vi.

Đầu ra : Mô hình miền hợp nhất $\mathsf{UDM} = \langle \mathsf{DM}, \mathsf{AG}, \mathsf{MC} \rangle$.

Khởi tạo DM $\leftarrow \varnothing$, AG $\leftarrow \varnothing$, MC $\leftarrow \varnothing$
Khởi tạo ánh xạ mapNode : $AD \rightarrow AG$
// Pha 1: Ánh xạ khía cạnh cấu trúc từ CD sang DM và MC
foreach class $\in$ CD **do**
  genDCSL(class, DM) genModule(class, MC)
end foreach
// Pha 2: Ánh xạ khía cạnh hành vi từ AD sang AG
genActivityClass(AD, DM) initActivityGraph(AD, AG) order $\leftarrow$ DFS(AD)
foreach node $\in$ *order* **do**
  if node *là nút hành động* **then**
    $n' \leftarrow$ genActionNode(node) AG $\leftarrow$ AG$\cup\{n'\}$ mapNode[node] $\leftarrow n'$ bindRefCls(n', DM) bindOutCls(n', DM)
  else if node *là nút điều khiển kiểu Decision* **then**
    $n' \leftarrow$ genDecisionNode(node) AG $\leftarrow$ AG $\cup \{n'\}$ mapNode[node] $\leftarrow n'$
  else if node *là nút điều khiển kiểu Sequential* **then**
    $n' \leftarrow$ genSequentialNode(node) AG $\leftarrow$ AG $\cup \{n'\}$ mapNode[node] $\leftarrow n'$
  else if node *là nút điều khiển kiểu Fork* **then**
    $n' \leftarrow$ genForkNode(node) AG $\leftarrow$ AG $\cup \{n'\}$ mapNode[node] $\leftarrow n'$
  else if node *là nút điều khiển kiểu Merge* **then**
    $n' \leftarrow$ genMergeNode(node) AG $\leftarrow$ AG $\cup \{n'\}$ mapNode[node] $\leftarrow n'$
  else if node *là nút điều khiển kiểu Join* **then**
    $n' \leftarrow$ genJoinNode(node) AG $\leftarrow$ AG $\cup \{n'\}$ mapNode[node] $\leftarrow n'$
  else
    $n' \leftarrow$ genCoordinateNode(node) AG $\leftarrow$ AG $\cup \{n'\}$ mapNode[node] $\leftarrow n'$
end foreach
foreach edge $= (u, v) \in$ AD **do**
  genEdge(mapNode[u], mapNode[v], AG)
end foreach
// Pha 3: Tích hợp hành vi với mô hình miền
foreach node $\in$ AG *với* node *là nút hành động* **do**
  bindActSeq(node, DM)
end foreach
genActivityLinks(DM, AG)
// Kiểm tra tính phù hợp
if $\neg$*conformsToUDML*(DM, AG, MC) **then**
  error "UDM không hợp lệ"
end if
return UDM

hành động và nút điều khiển được ánh xạ sang các thành phần tương ứng của đồ thị hoạt động AGL. Pha thứ ba tích hợp hai phần này thông qua các tham chiếu như `refCls`, `outCls` và `actSeq`, đồng thời bổ sung các liên kết hoạt động để thu được một mô hình miền hợp nhất hoàn chỉnh. Trên

cơ sở ba pha này, RM2UDM được đặc tả bằng một tập các luật chuyển đổi chính như sau.

Luật 1. Mỗi biểu đồ hoạt động trong mô hình yêu cầu được ánh xạ sang một lớp hoạt động (`ActivityClass`) trong UDM. Lớp này là phần tử trung tâm để biểu diễn đồ thị hoạt động tương ứng.

Luật 2. Mỗi nút hành động trong biểu đồ hoạt động được ánh xạ sang một thể hiện `Node` trong UDM. Các tham chiếu `refCls` và `outCls` được xác định trên cơ sở lớp miền tương ứng trong biểu đồ lớp.

Luật 3. Mỗi nút điều khiển trong mô hình yêu cầu được ánh xạ sang một mẫu miền (`DomainPattern`) tương ứng trong UDM, chẳng hạn như `Decision`, `Fork`, `Merge`, `Join`, `Sequential` hoặc `Coordinate`.

Luật 4. Với mỗi `Node` được tạo từ một nút hành động, chuỗi hành động thực thi được biểu diễn thông qua thuộc tính `actSeq` và được gắn với lớp miền tương ứng nhằm đồng bộ trạng thái hành vi với trạng thái miền.

Luật 5. Mỗi lớp, thuộc tính và ràng buộc OCL trong biểu đồ lớp được ánh xạ sang các thành phần DCSL tương ứng trong UDM, qua đó bảo toàn khía cạnh cấu trúc của mô hình đầu vào.

Luật 6. Mỗi lớp miền trong mô hình yêu cầu được ánh xạ sang một mô-đun cấu hình tương ứng trong UDM nhằm hỗ trợ tổ chức phần mềm và sinh bản mẫu.

Như vậy, các luật chuyển đổi trên là cơ sở hình thức cho thuật toán RM2UDM. Trên phương diện phương pháp, chúng xác định mối tương ứng giữa các phần tử của mô hình nguồn và mô hình đích; trên phương diện hiện thực, chúng làm nền tảng cho việc xây dựng bộ chuyển đổi và công cụ hỗ trợ ở Chương 6.2.4.

Cơ sở hình thức của phép chuyển đổi. Trong mô hình miền hợp nhất UDM, khía cạnh hành vi được biểu diễn dưới dạng một đồ thị hoạt động AG, phản ánh cấu trúc điều khiển và luồng dữ liệu của các hoạt động miền. Đồ thị này có thể được mô hình hóa hình thức như một bộ sáu phần tử: $AG = \langle N, E, eve, gua, var, obj \rangle$. Trong đó, AG là thể hiện của ngôn ngữ AGL trong mô hình UDM, N là tập các nút, E là tập các cạnh, eve là tập

các sự kiện, *gua* là tập các điều kiện gác, *var* là tập các biến cục bộ, và *obj* là tập các đối tượng tham gia trong quá trình thực thi.

Các nút trong AG được phân thành hai loại chính: (i) *ActionNode*, biểu diễn các thao tác nghiệp vụ; và (ii) *ControlNode*, bao gồm các nút điều khiển như *decision*, *sequential*, *fork*, *merge* và *join*, dùng để biểu diễn cấu trúc điều khiển của luồng thực thi. Sự phân loại này cho phép xác định rõ vai trò và ngữ nghĩa của từng loại nút trong mô hình hành vi miền.

Trên cơ sở biểu diễn hình thức này, các luật ánh xạ hành vi được xác định nhằm dẫn xuất đặc tả hành vi tương ứng từ đồ thị hoạt động AG. Cụ thể, mỗi nút $n \in N$ và mỗi cạnh $e \in E$ được ánh xạ sang các thành phần hành vi tương ứng trong biểu diễn đích, bảo đảm rằng cấu trúc điều khiển và luồng dữ liệu của mô hình miền được bảo toàn trong quá trình chuyển đổi. Phép ánh xạ này có thể được ký hiệu tổng quát như sau: $\phi : AG \rightarrow AGL$, trong đó ϕ là ánh xạ từ đồ thị hoạt động ở mức mô hình sang biểu diễn hành vi tương ứng.

Bảng 5.1 trình bày mối tương ứng giữa các phần tử hình thức của đồ thị hoạt động AG trong UDM và các thành phần tương ứng trong siêu mô hình AGL. Bảng này làm rõ mối liên hệ giữa biểu diễn hành vi ở mức mô hình và biểu diễn hành vi trong mô hình đích, cho thấy phép chuyển đổi không chỉ là một ánh xạ cú pháp mà còn bảo toàn cấu trúc ngữ nghĩa của hành vi miền.

Bảng 5.1: Tương ứng phần tử giữa mô hình AGL và đặc tả đồ thị AG

Phần tử trong siêu mô hình AGL	**Thành phần trong đặc tả hình thức**
ActivityGraph	AG
ActivityNode	N
ActivityEdge	E
Event	*eve*
Guard	*gua*
Variable	*var*
Object	*obj*

Việc sử dụng cơ sở hình thức này cho phép đặc tả các luật ánh xạ một cách rõ ràng và có hệ thống, đồng thời bảo đảm rằng đặc tả hành vi thu được phản ánh chính xác hành vi của mô hình miền trong *UDM*. Đây là nền tảng cho bước hiện thực hóa mô hình miền được trình bày trong Mục 5.4, trong đó các luật ánh xạ này được sử dụng để sinh đặc tả miền thực thi dưới dạng mã nguồn.

Đặc tả trên được hiện thực bằng công cụ ATL và được trình bày chi tiết trong Mục 6.2.4. Cần lưu ý rằng thứ tự duyệt DFS chỉ mang tính khái niệm; trong hiện thực, các phép ánh xạ được thực hiện dựa trên quan hệ phụ thuộc giữa các phần tử mô hình.

5.4 Sinh mã nguồn bản mẫu phần mềm

Trên cơ sở mô hình miền hợp nhất *UDM* đã được xây dựng trong Mục 5.3, phần này trình bày quá trình hiện thực hóa mô hình miền để sinh bản mẫu phần mềm trong bối cảnh thiết kế hướng miền. Mô hình *UDM* được biểu diễn ở mức cú pháp trừu tượng, tập trung đặc tả cấu trúc, hành vi và các mối quan tâm của miền một cách độc lập với nền tảng thực thi; do đó, để có thể triển khai, mô hình này cần được chuyển đổi sang dạng cú pháp cụ thể dưới dạng văn bản, dưới dạng mã nguồn với các chú thích đặc tả. Quá trình này được xây dựng dưới dạng một bộ chuyển đổi từ mô hình sang mã (M2T), kết hợp với cơ chế sinh phần mềm dựa trên khung phần mềm JDA, nhằm hiện thực hóa trực tiếp các khái niệm của mô hình miền trong môi trường lập trình hướng đối tượng.

Quá trình hiện thực hóa gồm hai bước chính: (i) chuyển đổi mô hình miền hợp nhất *UDM* thành đặc tả miền thực thi AGL^+ dưới dạng mã nguồn thông qua bộ chuyển đổi UDM2AGL; và (ii) sử dụng đặc tả AGL^+ trong khung phần mềm JDA để sinh tự động bản mẫu phần mềm tương ứng. Trong cách tiếp cận này, AGL^+ được xem như một đặc tả miền thực thi dưới dạng mã nguồn có chú thích, trong đó các thành phần cấu trúc và hành vi được đặc tả thông qua các chú thích như `@DClass`, `@AGraph`, ... Biểu diễn này cho phép ánh xạ trực tiếp sang các thành phần phần mềm trong môi trường thực thi.

Luận án đề xuất bộ chuyển đổi UDM2AGL thực hiện kỹ thuật mô hình–sang–văn bản từ $UDM = \langle DM, AG, MC \rangle$ sang mã nguồn AGL^+. Trong đó, DM mô tả cấu trúc miền, còn AG nắm bắt các khía cạnh hành vi. Kết quả của phép biến đổi là một đặc tả miền có khả năng thực thi, trong đó các yếu tố cấu trúc và hành vi được tích hợp thống nhất trong mã nguồn nhúng vào ngôn ngữ lập trình hướng đối tượng (ví dụ, Java).

Quá trình chuyển đổi được thực hiện bằng cách duyệt các phần tử của UDM và áp dụng các luật sinh mã tương ứng. Mỗi lớp trong DM được ánh xạ thành lớp miền với chú thích `@DClass`, qua đó thể hiện cấu trúc dữ liệu và các ràng buộc. Các yếu tố hành vi trong AG được chuyển đổi thành đặc tả thông qua `@AGraph` và các chú thích đặc tả hành vi liên quan, cho phép mô hình hóa luồng điều khiển và tương tác giữa các hành động miền. Đối với MC, các phần tử được ánh xạ thành các thành phần trong kiến trúc JDA, bao gồm mô-đun chức năng, lớp điều khiển, giao diện người dùng và các cấu hình điều hướng. Quá trình hiện thực được hỗ trợ bởi hệ thống chú thích cấu hình và các khuôn mẫu mã, đảm bảo tính nhất quán giữa mô hình và triển khai. Sự tích hợp của ba thành phần DM, AG và MC trong cùng một mã nguồn tạo nên đặc tả AGL^+ hoàn chỉnh, trong đó cấu trúc, hành vi và kiến trúc được hợp nhất. Đặc tả này có thể được thực thi trực tiếp trên nền tảng JDA, qua đó hỗ trợ sinh tự động các bản mẫu phần mềm tương ứng.

Thuật toán 5.2 mô tả quy trình tổng quát để sinh đặc tả miền thực thi AGL^+ từ mô hình miền hợp nhất UDM. Thuật toán được tổ chức thành ba pha chính, tương ứng với việc chuyển đổi DM, AG và hợp thành đặc tả trong một biểu diễn mã nguồn thống nhất. Trong pha thứ nhất, các phần tử cấu trúc của mô hình miền được xử lý. Mỗi lớp miền trong DM được duyệt và ánh xạ thành một đơn vị mã nguồn tương ứng thông qua hàm `genDClass`. Kết quả của pha này là tập các lớp miền dưới dạng mã nguồn được chú thích, phản ánh cấu trúc dữ liệu và các ràng buộc của miền trong đặc tả AGL^+. Pha thứ hai tập trung vào việc sinh đặc tả hành vi từ đồ thị hoạt động AG. Trước hết, cấu trúc tổng thể của đồ thị được thiết lập thông qua hàm `genAGraph`. Sau đó, các nút trong đồ thị được xử lý theo một thứ tự phụ thuộc (được biểu diễn khái niệm bằng duyệt DFS) nhằm bảo đảm tính nhất quán giữa các thành phần hành vi. Tùy theo loại nút (như *ActionNode*, *DecisionNode*, *ForkNode*, ...), các hàm sinh mã tương

Thuật toán 5.2 Sinh đặc tả AGL^+ từ *UDM* bằng bộ chuyển UDM2AGL

Đầu vào: UDM $= \langle DM, AG, MC \rangle$ - mô hình miền hợp nhất
Đầu ra : Code - mã nguồn biểu diễn đặc tả AGL^+

```
1  Khởi tạo Code ← ∅
   // Pha 1: Sinh mã cấu trúc miền từ DM
2  foreach class ∈ DM.classes do
3  |  Code ← Code ∪ genDClass(class)
4  end foreach
   // Pha 2: Sinh mã hành vi miền từ AG
5  Code ← Code ∪ genAGraph(AG)  order ← DFS(AG.nodes)
6  foreach node ∈ order do
7  |  if node là ActionNode then
8  |  |  Code ← Code ∪ genActionNode(node)
9  |  else if node là DecisionNode then
10 |  |  Code ← Code ∪ genDecisionNode(node)
11 |  else if node là SequentialNode then
12 |  |  Code ← Code ∪ genSequentialNode(node)
13 |  else if node là ForkNode then
14 |  |  Code ← Code ∪ genForkNode(node)
15 |  else if node là MergeNode then
16 |  |  Code ← Code ∪ genMergeNode(node)
17 |  else if node là JoinNode then
18 |  |  Code ← Code ∪ genJoinNode(node)
19 end foreach
   // Pha 3: Hoàn thiện đặc tả AGL+
20 Code ← composeAGLPlus(Code)
21 return Code
```

ứng được áp dụng để tạo ra các đoạn mã đặc tả hành vi, qua đó ánh xạ có hệ thống các cấu trúc điều khiển của AG sang các cấu trúc thực thi trong AGL^+. Trong pha thứ ba, các thành phần mã nguồn được hợp nhất thông qua hàm `composeAGLPlus` để tạo thành một đặc tả AGL^+ hoàn chỉnh. Bước này đảm bảo rằng các yếu tố cấu trúc và hành vi được tích hợp nhất quán trong cùng một biểu diễn mã nguồn, đồng thời đáp ứng các yêu cầu về tổ chức và cấu hình của hệ thống theo kiến trúc khung phần mềm JDA.

Thuật toán trên được hiện thực bằng công cụ Acceleo chuyển đổi mô hình–sang–văn bản và được trình bày chi tiết trong Mục 6.2.4. Kết quả của phép chuyển đổi là một đặc tả miền thực thi dưới dạng mã nguồn có chú thích, sử dụng khung JDA để sinh tự động bản mẫu phần mềm.

5.5 Tổng kết chương

Chương này đã trình bày một chuỗi chuyển đổi mô hình nhằm sinh tự động bản mẫu phần mềm từ mô hình yêu cầu trong bối cảnh thiết kế hướng miền. Trên cơ sở đó, luận án đề xuất một cách tiếp cận nhất quán, kết nối chặt chẽ giữa mô hình yêu cầu, mô hình miền và hiện thực phần mềm. Cụ thể, luận án đã đề xuất phương pháp RM2UDM để chuyển đổi mô hình yêu cầu UML/OCL sang mô hình miền hợp nhất *UDM* tuân thủ siêu mô hình *UDML*. Mô hình *UDM* đóng vai trò là biểu diễn trung gian, trong đó các khía cạnh cấu trúc và hành vi được tích hợp trong một mô hình thống nhất, làm cơ sở cho các bước hiện thực hóa tiếp theo.

Trên cơ sở đó, chương đã trình bày quá trình hiện thực hóa mô hình miền thông qua phép chuyển đổi mô hình–sang–văn bản UDM2AGL, nhằm sinh đặc tả miền thực thi AGL^+ dưới dạng mã nguồn có chú thích. Đặc tả này tích hợp các thành phần cấu trúc và hành vi của mô hình miền trong một biểu diễn có khả năng thực thi, cho phép ánh xạ trực tiếp sang các thành phần phần mềm. Từ đặc tả AGL^+, nền tảng khung phần mềm JDA được sử dụng để sinh tự động bản mẫu phần mềm, qua đó thiết lập một cầu nối từ mô hình miền đến hiện thực phần mềm. Chuỗi chuyển đổi tổng thể từ mô hình yêu cầu đến bản mẫu phần mềm góp phần thu hẹp khoảng cách giữa đặc tả mức cao và hệ thống thực thi, đồng thời hỗ trợ đánh giá sớm và tinh chỉnh yêu cầu trong quy trình phát triển lặp.

Các kết quả nghiên cứu trong chương này đã được công bố trong các công trình khoa học liên quan, bao gồm: cách tiếp cận chuyển đổi đặc tả mức cao sang mô hình miền được trình bày trong bài báo hội thảo quốc tế KSE 2023 *[V2]*, và phương pháp sinh tự động bản mẫu phần mềm từ mô hình yêu cầu được công bố trong bài báo hội thảo quốc gia FAIR 2024 *[V4]*.

Chương 6

THỰC NGHIỆM VÀ ĐÁNH GIÁ

Chương này trình bày việc áp dụng và đánh giá các phương pháp đề xuất của luận án thông qua các ca nghiên cứu điển hình, bao gồm CourseMan, OrderMan, ProcessMan và hệ thống OJS. Các hệ thống này đại diện cho các miền ứng dụng khác nhau với sự đa dạng về cấu trúc, hành vi và các mối quan tâm liên quan. Mục tiêu của chương là đánh giá tính khả thi và hiệu quả của cách tiếp cận đề xuất thông qua việc xây dựng mô hình miền hợp nhất, thiết lập các liên kết ngữ nghĩa giữa các mối quan tâm, và thực hiện các kỹ thuật chuyển đổi mô hình nhằm hỗ trợ phân tích, kiểm chứng hình thức và sinh tự động các bản mẫu phần mềm.

Các ca nghiên cứu được sử dụng với vai trò khác nhau tùy theo nội dung đánh giá. Cụ thể, CourseMan, OrderMan và ProcessMan được dùng để đánh giá các kỹ thuật biểu diễn và chuyển đổi mô hình, trong khi OJS được sử dụng để minh họa và đánh giá tích hợp bảo mật dựa trên vai trò và kiểm chứng hình thức. Trên cơ sở đó, các kết quả thực nghiệm được phân tích nhằm làm rõ tính khả thi, tính nhất quán ngữ nghĩa và khả năng áp dụng của phương pháp đề xuất.

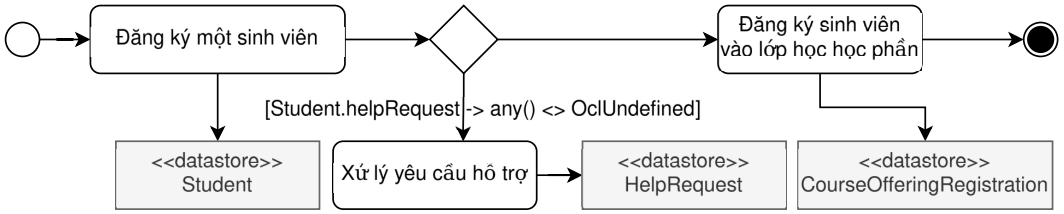
6.1 Giới thiệu

Luận án sử dụng hệ thống quản lý khóa học (CourseMan) làm ca nghiên cứu xuyên suốt cho thực nghiệm và đánh giá. Miền vấn đề của hệ thống được thể hiện thông qua mô hình miền trong Hình 1.2. Biểu đồ lớp UML trong hình này mô tả khía cạnh cấu trúc của mô hình miền CourseMan. Để đặc tả các

quy tắc nghiệp vụ mà UML không thể biểu đạt đầy đủ, các ràng buộc OCL được sử dụng; ràng buộc minh họa trong hình đảm bảo rằng tổng số tín chỉ sinh viên đăng ký không vượt quá mức tối đa theo chương trình đào tạo.

Mục tiêu của luận án là tích hợp mô hình miền UML/OCL này vào bối cảnh DDD. Tuy nhiên, điều này đặt ra thách thức vì DDD không chỉ yêu cầu mô hình hóa cấu trúc mà còn đòi hỏi mô hình miền phải có ngữ nghĩa vận hành rõ ràng, tức là có khả năng thực thi hoặc diễn giải như một chương trình. Như đã phân tích trong Như đã trình bày trong Mục 3.3 và Mục 3.2, mô hình miền trong DDD cần biểu diễn đầy đủ các khía cạnh cấu trúc, hành vi và ràng buộc nghiệp vụ, đồng thời đóng vai trò là cơ sở chung để các bên liên quan trao đổi, kiểm chứng và hiện thực hóa hệ thống.

Hình 6.1 trình bày biểu đồ hoạt động UML cho quy trình quản lý đăng ký, bao gồm đăng ký sinh viên, đăng ký học phần (`CourseOffering`) và xử lý các yêu cầu hỗ trợ trong quá trình đăng ký.



Hình 6.1: Biểu đồ hoạt động UML mô tả hoạt động quản lý đăng ký lớp học phần.

Để tạo ra một phiên bản hệ thống có thể thực thi, các hành vi miền cần được tích hợp chặt chẽ với mô hình cấu trúc miền (Hình 1.2). Trong cách tiếp cận truyền thống, hành vi thường được đặc tả riêng biệt bằng UML (chẳng hạn biểu đồ hoạt động), trong khi mô hình cấu trúc được biểu diễn bằng biểu đồ lớp. Sự tách biệt này dẫn đến việc bảo đảm tính nhất quán giữa hai khía cạnh chủ yếu chỉ được thực hiện ở giai đoạn cài đặt, làm gia tăng khoảng cách ngữ nghĩa giữa đặc tả và hiện thực, đồng thời gây khó khăn cho việc phân tích và kiểm chứng ở mức mô hình.

Nhằm khắc phục hạn chế này, luận án đề xuất xem hành vi miền như một phần mở rộng nội tại của mô hình miền thiết yếu, từ đó hình thành một mô hình miền hợp nhất tích hợp đồng thời cấu trúc và hành vi. Cách tiếp cận này phù hợp với nguyên lý thiết kế hướng miền (DDD) trong [73], theo đó mô hình miền không chỉ phản ánh chính xác tri thức nghiệp vụ mà

còn phải khả thi về mặt kỹ thuật, đáp ứng yêu cầu hiệu năng và có khả năng được hiểu và xác nhận bởi chuyên gia miền.

Công cụ được trình bày trong chương này nhằm hiện thực hóa cách tiếp cận trên bằng việc tích hợp các kỹ thuật AGL và CAP vào một quy trình phát triển cụ thể. Công cụ không chỉ minh họa cách áp dụng các kỹ thuật đề xuất trong thực tiễn, mà còn cho phép đánh giá khả năng sử dụng của chúng trong phát triển phần mềm cho các miền vấn đề thực tế. Đồng thời, chương này cũng mô tả kiến trúc và hiện thực của công cụ, làm rõ các quyết định thiết kế chủ chốt và các công nghệ được sử dụng.

Mục tiêu của thực nghiệm là chỉ ra rằng AGL và CAP vừa có khả năng biểu đạt cần thiết, vừa có tính khả dụng trong thực tiễn. Việc đánh giá được tiến hành theo các hướng dẫn nghiên cứu tình huống được đề xuất trong [107] nhằm đảm bảo tính chặt chẽ về phương pháp luận. Ngoài ra, các tiêu chí đánh giá DSL được đề xuất trong [121] cũng được sử dụng để phân tích AGL và CAP, tập trung vào các câu hỏi nghiên cứu sau:

- RQ1. Làm thế nào có thể định nghĩa ngữ nghĩa vận hành hình thức cho các mô hình miền hợp nhất trong UDML?
- RQ2. Làm thế nào các mối quan tâm xuyên suốt động, tiêu biểu là RBAC, có thể được đặc tả dưới dạng các DSL dựa trên chú thích và được tích hợp vào các mô hình miền hợp nhất?
- RQ3. Mức độ biểu đạt của CAPs trong việc mô hình hóa các khái niệm miền so với các phương pháp DDD hiện có là như thế nào?
- RQ4 CAP có thể được áp dụng ở mức độ nào trong thực tiễn phát triển phần mềm?
- RQ5. Phương pháp tích hợp khía cạnh hành vi vào mô hình miền hiệu quả như thế nào so với các phương pháp DDD hiện có?
- RQ6. Mức độ nỗ lực cần thiết để định nghĩa một mô hình miền hợp nhất bằng aDSL (bao gồm DCSL và AGL) nhằm sinh phần mềm theo DDD là bao nhiêu?
- RQ7. Làm thế nào có thể tích hợp một mối quan tâm nền tảng vào mô hình miền?

- RQ8. Làm thế nào có thể biểu diễn miền của mối quan tâm, vốn có thể được xem như một ngôn ngữ chuyên biệt miền dựa trên chú thích?
- RQ9. Làm thế nào các DSL theo mối quan tâm có thể được hợp nhất một cách có hệ thống vào một cây cú pháp trừu tượng hợp nhất?
- RQ10. Những cơ chế hợp thành ngôn ngữ và các phương pháp có sự hỗ trợ của công cụ nào là phù hợp để tạo điều kiện cho việc tích hợp và tiến hóa các mối quan tâm không đồng nhất trong các mô hình miền mô-đun và có khả năng mở rộng?

Phần tiếp theo trình bày cấu trúc của công cụ hỗ trợ trong Mục 6.2. Các kết quả thực nghiệm và thảo luận đánh giá hiệu quả của các phương pháp đề xuất được trình bày trong Mục 6.3.

6.2 Công cụ hỗ trợ

Luận án phát triển công cụ thực nghiệm (UDML) nhằm hiện thực hóa các kỹ thuật biểu diễn, hợp nhất và chuyển đổi mô hình được đề xuất. Cài đặt của công cụ được công bố dưới dạng mã nguồn mở trên GitHub[1].

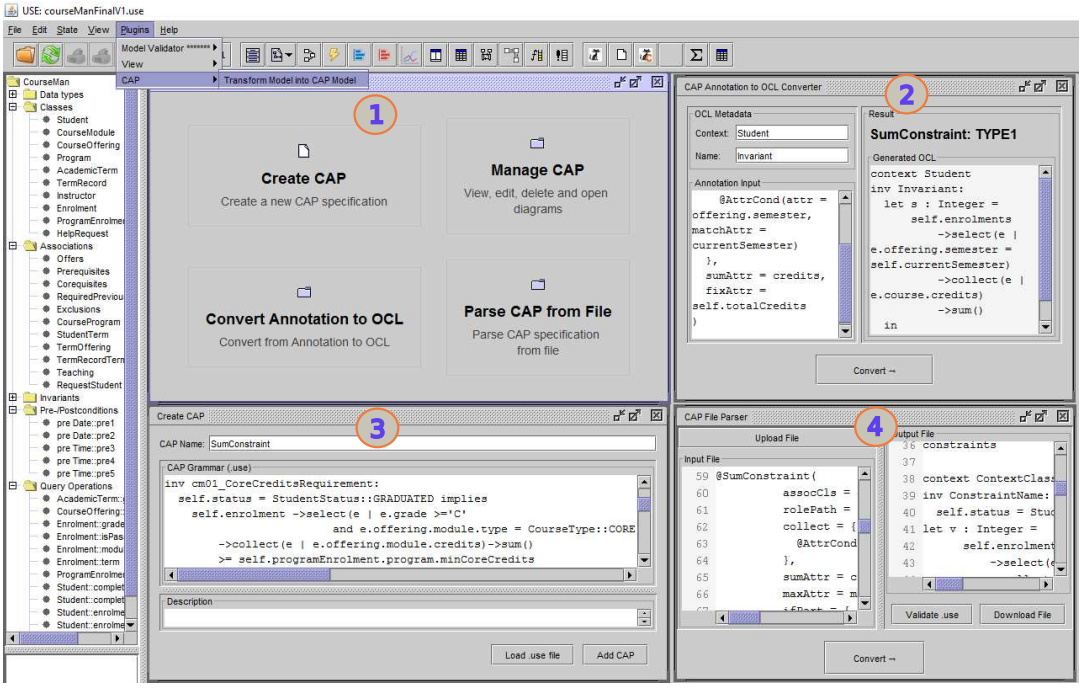
6.2.1 Công cụ tích hợp ràng buộc vào mô hình miền

Kiến trúc công cụ: Mở rộng CAP/UDML của USE

Trong tiểu mục này, luận án trình bày phương pháp xây dựng công cụ hỗ trợ, được gọi là CAP/UDML. Công cụ này được hiện thực như một phần mở rộng của môi trường đặc tả dựa trên UML USE (*UML-based Specification Environment*) [24], mở rộng khả năng hỗ trợ sẵn có của USE về mô hình hóa cấu trúc UML và kiểm chứng ràng buộc OCL bằng cách bổ sung cơ chế đặc tả CAP dựa trên chú thích và tái tạo tự động các bất biến.

Để hỗ trợ việc tích hợp các CAPs vào mô hình miền hợp nhất (UDM), CAP/UDML mở rộng ngữ pháp và pipeline xử lý của USE nhằm: (i) nhận diện các chú thích CAP, (ii) liên kết chúng với các khuôn mẫu CAP trong danh mục, và (iii) tự động sinh các bất biến OCL tương ứng theo ánh xạ sinh hình thức được định nghĩa trong Mục 3.2.2.3. Các bất biến được sinh ra sau đó được kiểm chứng bằng bộ máy OCL của USE nhằm đảm bảo tính đúng đắn cú pháp và tính nhất quán cấu trúc với mô hình miền.

[1] https://github.com/vinhskv/udml



Hình 6.2: Công cụ CAP/UDML quản lý và ứng dụng mẫu CAP

Hình 6.2 cung cấp cái nhìn tổng quan về kiến trúc của công cụ CAP/UDML, minh họa các lớp mở rộng được xây dựng trên nền USE cũng như quy trình từ phân tích chú thích đến tái tạo OCL và tích hợp vào UDM.

Ở phía trên bên trái (nhãn (1)) của Hình 6.2, giao diện hiển thị bốn chức năng chính của công cụ CAP/UDML, bao gồm: (i) Tạo CAP, (ii) Chuyển đổi chú thích độc lập thành CAP, (iii) Phân tích các chú thích từ tệp của USE (`.use`), và (iv) Quản lý CAP: mỗi CAP có thể bao gồm nhiều kiểu khác nhau; chức năng này cho phép người dùng định nghĩa và quản lý các kiểu này trong một CAP đã được xác định.

Ở phía trên bên phải (nhãn (2)) của Hình 6.2, giao diện hiển thị chức năng **Chuyển đổi chú thích thành ràng buộc OCL**. Chức năng này tập trung vào việc phân tích các chú thích độc lập, tức là các chú thích không gắn với một ngữ cảnh mô hình cụ thể. Để chuyển đổi các chú thích này thành ràng buộc OCL, người dùng cần xác định thủ công ngữ cảnh (ví dụ: lớp áp dụng như `Student` hoặc `CourseModule`) và cung cấp tên bất biến, đóng vai trò là định danh cho ràng buộc OCL được sinh ra.

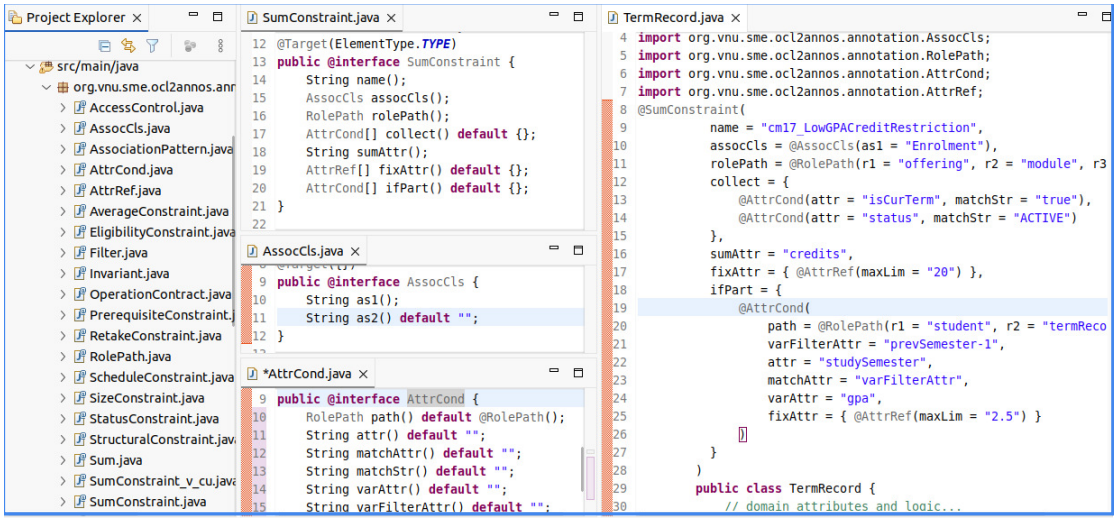
Ở phía dưới bên trái (nhãn (3)) của Hình 6.2, giao diện hiển thị chức năng **Tạo CAP**. Chức năng này cho phép người dùng định nghĩa một kiểu CAP mới. Khi tạo CAP, người dùng cần chỉ định tên CAP (ví dụ: SumConstraint)

và cung cấp một tệp (`.use`) tương ứng mô tả cấu trúc biểu đồ lớp liên quan đến CAP đó. Hệ thống cho phép kiểm tra tệp (`.use`) nhằm đảm bảo ngữ pháp và cú pháp tuân thủ đặc tả của USE.

Ở phía dưới bên phải (nhãn (4)) của Hình 6.2, giao diện cung cấp chức năng Phân tích các chú thích từ tệp (`.use`). Chức năng này phân tích các chú thích và ngữ cảnh tương ứng trong tệp mô hình (`.use`), tái tạo các ràng buộc OCL và sinh một tệp (`.use`) mới, trong đó các chú thích được thay thế bằng các ràng buộc OCL tương ứng. Các ràng buộc được tích hợp vào UDM, đồng thời tệp kết quả có thể được kiểm chứng để bảo đảm tính đúng đắn về ngữ pháp và cú pháp.

Sinh bản mẫu phần mềm

Việc sinh bản mẫu phần mềm được minh họa trong Hình 6.3. Bản mẫu được hiện thực trên nền tảng Spring Boot kết hợp với khung JDA [74], được phát hành dưới dạng mã nguồn mở trên GitHub[2]. Từ mô hình miền đã được bổ sung các ràng buộc OCL tái tạo, JDA tự động sinh các thành phần của phần mềm thực thi, bao gồm các lớp miền, lô-gic kiểm tra ràng buộc, cấu hình lưu trữ dữ liệu và giao diện người dùng (GUI).



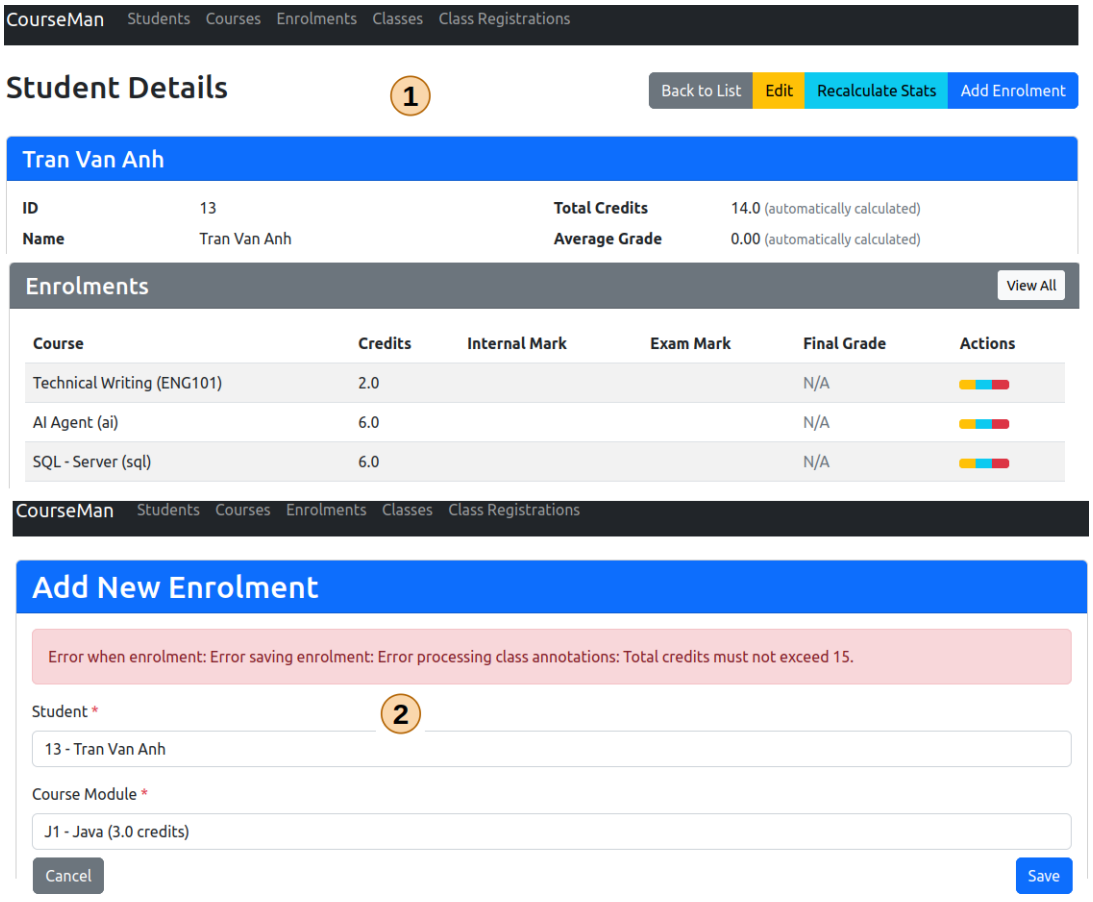
Hình 6.3: Hiện thực công cụ và khả năng sử dụng của khung làm việc CAP.

Giao diện bên trái của Hình 6.3 liệt kê các lớp mô-đun và các lớp miền trong mô hình cấu trúc. Đối với mỗi lớp, vùng giao diện ở giữa hiển thị các chú thích CAP đã được áp dụng, chẳng hạn `@SumConstraint` và

[2] https://github.com/vinhskv/udml/tree/main/CAP/frameWorkGenCode

`@PrerequisiteConstraint`, phản ánh các ràng buộc được đặc tả theo phương pháp CAP.

Ứng dụng CourseMan được sinh tự động với giao diện đồ họa (GUI), như minh họa trong Hình 6.4. Ví dụ này minh họa cơ chế kiểm tra ràng buộc tại thời gian chạy nhằm đảm bảo tổng số tín chỉ mà một sinh viên đăng ký không vượt quá 15. Nhãn (1) thể hiện trường hợp hợp lệ khi sinh viên *"Trần Văn Anh"* đăng ký 14 tín chỉ, trong khi Nhãn (2) minh họa trường hợp vi phạm khi số tín chỉ vượt quá giới hạn cho phép. Khi người dùng thực hiện thao tác lưu đăng ký học phần mới, hệ thống tự động kích hoạt cơ chế kiểm tra ràng buộc theo phương pháp đề xuất.



Hình 6.4: GUI của phần mềm CourseMan được sinh tự động bởi công cụ.

6.2.2 Công cụ tích hợp hành vi vào mô hình miền

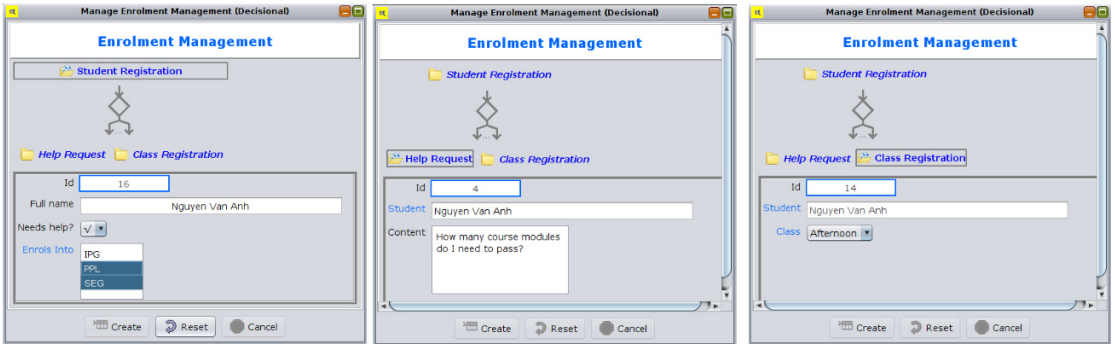
Nhằm đánh giá khả năng hiện thực của phương pháp đề xuất, một công cụ là khung phần mềm JDA đã được phát triển để xây dựng hỗ trợ tích hợp khía cạnh hành vi vào mô hình hợp nhất và sinh tự động bản mẫu phần mềm. Công cụ biểu diễn năm mẫu hành vi miền, gồm: mẫu tuần tự, mẫu quyết định, mẫu phân nhánh, mẫu hợp nhất và mẫu gộp. Đối với mỗi mẫu, công cụ sinh một mô hình hợp nhất tương ứng cùng các giao diện phần mềm (GUI) được cài đặt cho các ca nghiên cứu: CourseMan, OrderMan và ProcessMan. Do các mẫu có quy trình hiện thực tương tự nhau, luận án lựa chọn mẫu Quyết định để trình bày chi tiết.

Mẫu quyết định

Phần dưới của Hình 3.7 minh họa cách mẫu này được áp dụng cho hoạt động quản lý ghi danh của CourseMan. Thực hiện cài đặt cụ thể như sau: $\texttt{C}_\texttt{a}$ = `EnrolmentMgmt`, $\texttt{C}_\texttt{d}$ = `Student`, `D` = `DHelpOrSClass`, $n = 2$, $\texttt{C}_1$ = `HelpRequest`, và $\texttt{C}_2$ = `CourseOffering`. Nút điều khiển c_k không được chỉ định.

Hình 6.5 trình bày ba ảnh chụp GUI của ví dụ: GUI thứ nhất dành cho việc đăng ký sinh viên; GUI thứ hai và thứ ba lần lượt dành cho hai trường hợp sinh viên có yêu cầu hỗ trợ và không có yêu cầu hỗ trợ. Giao diện của hoạt động chứa giao diện của cả ba hành động trong các "thẻ" riêng biệt.

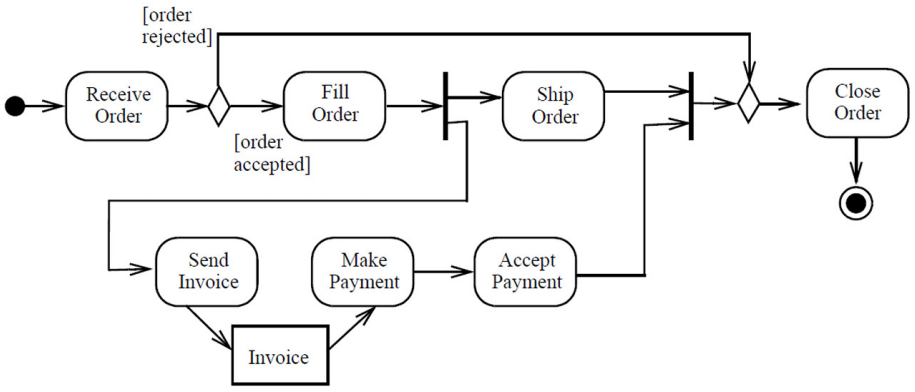
Ở cả hai nhánh quyết định, đối tượng `Student` được tạo từ hành động đầu tiên được truyền sang hành động kế tiếp và được hiển thị trong trường dữ liệu của thuộc tính `student` của lớp miền tương ứng.



Hình 6.5: Biểu diễn theo mẫu quyết định của hoạt động quản lý ghi danh.

Một nghiên cứu tình huống phức tạp khác cũng được trình bày—miền quản lý đơn hàng (OrderMan) được điều chỉnh từ đặc tả OMG/UML [94].

Mục đích là minh họa khả năng áp dụng AGL trong thực tiễn và khảo sát cách phương pháp đã đề xuất có thể được sử dụng để phát triển phần mềm cho các miền vấn đề trong thế giới thực. Ngoài ra, phần này còn trình bày công cụ hỗ trợ được sử dụng, tập trung giải thích các quyết định thiết kế chính và công nghệ được áp dụng.

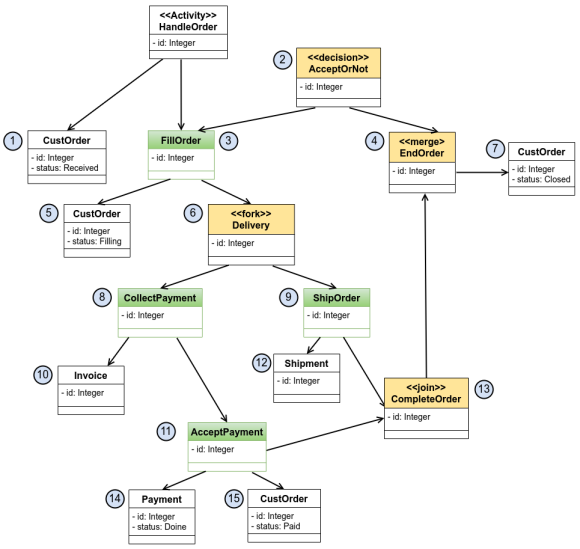


Hình 6.6: Biểu đồ hoạt động UML cho quy trình xử lý đơn hàng, được trích từ [94].

Hình 6.6 minh họa biểu đồ hoạt động UML mô tả quy trình xử lý đơn hàng trong OrderMan. Để tạo ra mô hình miền hợp nhất cho OrderMan, như được trình bày trong Hình 6.7, thực hiện áp dụng toàn bộ các mẫu hành vi miền. Hình 6.7(A) trình bày một đồ thị hoạt động, trong đó các nút được gán số thứ tự tương ứng với hoạt động `HandleOrder`. Hình cũng mô tả một phần của mô hình lớp hợp nhất, bao gồm một lớp hoạt động (`HandleOrder`), bốn lớp dữ liệu chính (`CustOrder`, `Invoice`, `Shipment`, `Payment`), bốn lớp điều khiển (`AcceptOrNot`, `Delivery`, `CompleteOrder`, `EndOrder`), và bốn lớp còn lại liên quan đến các coordinator nodes (`FillOrder`, `CollectPayment`, `ShipOrder`, `AcceptPayment`).

Các nút điều phối được sử dụng để cung cấp một cái nhìn đầy đủ về một nhóm nhiệm vụ. Chẳng hạn, `FillOrder` (nút 3) điều phối hai nhiệm vụ: `UpdateOrder` (nút 5) và `DeliveryOrder` (nút 6). `FillOrder` chỉ làm nhiệm vụ điều phối để đảm bảo rằng `UpdateOrder` được thực hiện trước `DeliveryOrder`. Nó không đóng góp dữ liệu vào luồng xử lý, nhưng cho phép người dùng quan sát và thực thi luồng nhiệm vụ trên giao diện người dùng.

Hình 6.7(B) minh họa cách mỗi nút trong đồ thị hoạt động được ánh xạ đến một mô-đun tương ứng (so với lớp miền được tham chiếu bởi `refCls`), các nút kết quả (cũng dựa trên `refCls`), và các đối tượng `ModuleAct` đặc tả



1:AG (HandleOrder)

Node	label	nodeType	refCls	serviceCls	outNodes	actSeq
1	MOrder	-	CustOrder	default	[2:Node]	[1:MAct]
2	MAcceptOrNot	Decision	AcceptOrNot	-	[3:Node, 4:Node]	-
3	MFillOrder	Coordinator	FillOrder	default	[5:]	[2:, 3:, 4:]
4	MEndOrder	Merge	EndOrder	-	[7:]	-
5	MOrder	-	CustOrder	default	[6:]	[5:]
6	MDelivery	Fork	Delivery	default	[8:, 9:]	[2:, 4:]
7	MOrder	-	CustOrder	default		[6:, 7:]
8	MCollectPayment	Coordinator	CollectPayment	default	[10:]	[2:, 3:, 4:]
9	MShipOrder	Coordinator	ShipOrder	default	[12:]	[2:, 3:]
10	MInvoice	-	Invoice	default	[11:]	[2:, 3:]
11	MAccept Payment	Coordinator	AcceptPayment	default	[14:]	[2:, 4:, 8:]
12	MShipment	-	Shipment	default	[13:]	[2:, 3:]
13	MComplete Order	Join	CompleteOrder	-	[4:]	-
14	MPayment	-	Payment	default	[15:]	[2:, 8:]
15	MOrder	-	CustOrder	default	[13:]	[9:, 5:]

MAct

Id	actName	postStates	filterType	fieldNames
1	newObject	{Created}		
2	newObject	{NewObject}		
3	setDataFieldValues	{Created}		{"order"}
4	createObject	{Created}		
5	showObject	{Updated}		
6	showObject			
7	updateObject	{Updated}		
8	setDataFieldValues	{Created}		{"invoice"}
9	filterInput		FilterCustOrder FromPayment	

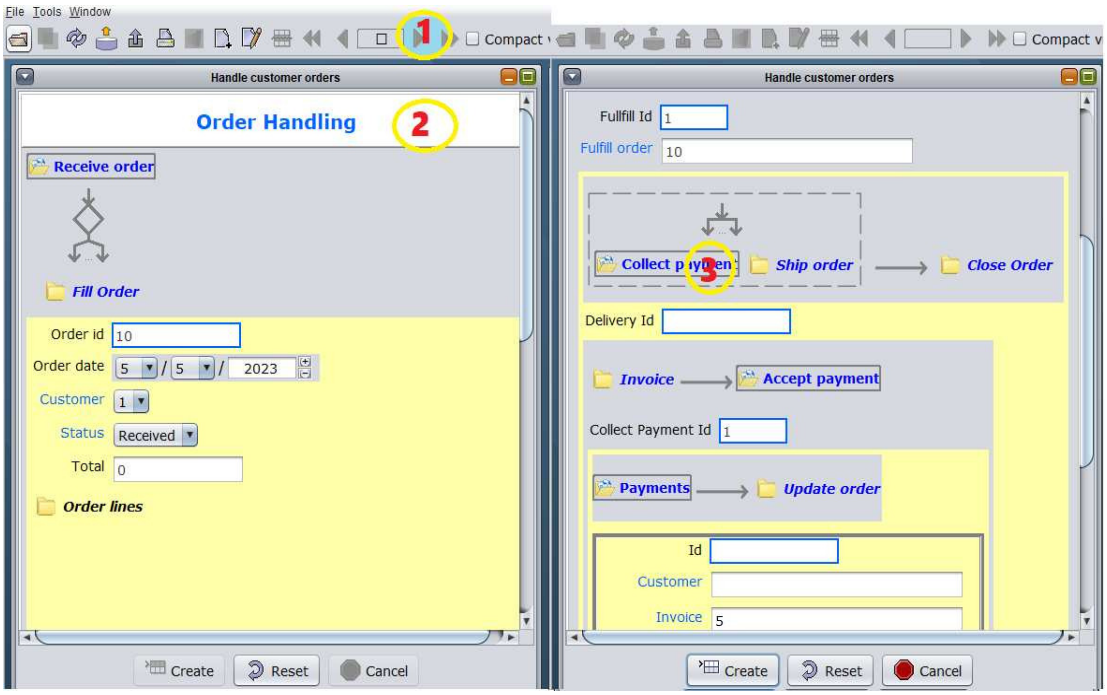
Hình 6.7: (A: Trái) Đồ thị hoạt động với các nút được gán nhãn bằng các lớp hoạt động và lớp thành phần; (B: Trên–phải) Các đối tượng `Node`; (C: Dưới–phải) Các đối tượng `ModuleAct` được tham chiếu bởi các `Node`.

các SAA cho hành vi của mô-đun. Các đối tượng `ModuleAct` này, cùng với các SAA tương ứng, được liệt kê trong Hình 6.7(C).

Để phát triển phần mềm OrderMan với giao diện GUI, như thể hiện trong Hình 6.8, mô hình lớp hợp nhất tích hợp đặc tả AGL cho mô hình hoạt động cần được hiện thực hóa bằng mã Java. Việc hiện thực cho OrderMan có thể được tìm thấy trong kho mã nguồn[3]. Việc hiện thực phương pháp được minh họa trong Hình 6.9, sử dụng một công cụ hỗ trợ được xây dựng trên JDA, một khung phần mềm Java mà đã trình bày trong [73].

Để sinh phần mềm từ mô hình miền hợp nhất đầu vào, lập trình viên cần mã hóa mô hình miền hợp nhất bằng DCSL và AGL, tạo ra một chương trình Java bao gồm hai thành phần chính: (i) Mô hình hợp nhất DCSL, bao gồm các lớp thành phần và lớp hoạt động; (ii) Đặc tả AGL cho các đồ thị hoạt động gắn với các lớp hoạt động. Cần lưu ý rằng cả đặc tả DCSL và AGL đều được mã hóa trong Java, như được minh họa ở hai khung giữa và phải của Hình 6.9. Công cụ bao gồm ba thành phần chính: bộ quản lý mô

[3] https://github.com/vinhskv/udml/tree/main/AGL/jda-agl-orderman

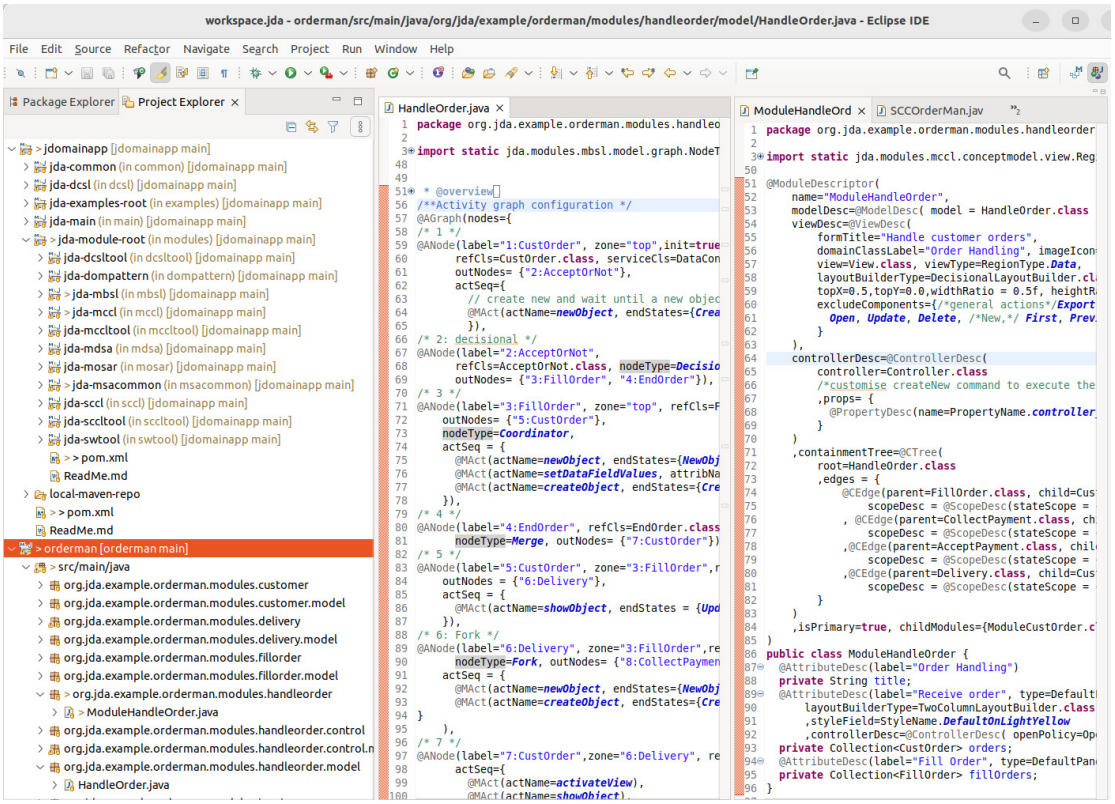


Hình 6.8: Giao diện người dùng của OrderMan được tạo ra bởi JDA.

hình, bộ quản lý giao diện và bộ quản lý đối tượng. Quản lý mô hình chịu trách nhiệm đăng ký và cung cấp mô hình lớp hợp nhất cho các thành phần khác. Cửa sổ bên trái của Hình 6.9 hiển thị danh sách các lớp mô-đun và các lớp miền tương ứng, được gắn chú thích `AGraph` nhằm biểu diễn đặc tả AGL, như được trình bày trong cửa sổ ở giữa. Quản lý giao diện tự động sinh giao diện người dùng của phần mềm và xử lý tương tác của người dùng dựa trên mô tả cấu hình của các mô-đun được đặc tả trong MOSA [75].

Ví dụ, lớp mô-đun `ModuleHandleOrder` chứa lớp hoạt động `HandleOrder` được đặc tả bằng mô tả mô-đun trong MCCL, như thể hiện trong cửa sổ bên phải của hình. Cuối cùng, quản lý bộ đối tượng thời gian chạy cho mỗi lớp miền và cung cấp một thành phần lưu trữ đối tượng tổng quát hỗ trợ cả lưu trữ dựa trên tệp và cơ sở dữ liệu quan hệ. Mô hình dữ liệu quan hệ được sinh tự động từ mô hình lớp hợp nhất khi phần mềm chạy lần đầu tiên.

Xét tình huống cụ thể với mô-đun `ModuleHandleOrder` và lớp hoạt động `HandleOrder`, được hiện thực như trong Listing 6.1. Khi phần mềm chạy, một thể hiện của mô-đun `ModuleHandleOrder` được gọi. Dựa trên mô tả cấu hình của mô-đun, một đối tượng `ActivityModel` được tạo dưới dạng tổ hợp giữa đối tượng `HandleOrder` và đồ thị hoạt động được mô tả bằng AGL tương ứng với lớp hoạt động này.



Hình 6.9: Minh họa việc hiện thực và khả năng sử dụng AGL dựa trên nền tảng JDA.

```
1     /*Activity graph configuration in AGL */
2     @AGraph(nodes={...
3         /* Node 14 */
4         @ANode(label="14:Payment", zone="11:AcceptPayment",
5         refCls=Payment.class, serviceCls=DataController.class,
6         outNodes={"15:CustOrder"},
7         actSeq={
8             @MAct(actName=newObject, endStates={NewObject}),
9             @MAct(actName=setDataFieldValues, attribNames={"
    invoice"},
10            endStates={Created})
11        }), ...
12    })
13    /*END: activity graph configuration */
14    public class HandleOrder {...}
```

Đặc tả 6.1: Lớp hoạt động `HandleOrder` trong Java

Nhờ cơ chế chú thích trong Java, đối tượng `HandleOrder` có thể được xử lý như một đối tượng `AGraph`, cho phép nó biểu diễn và quản lý đồ thị hoạt động. Đối tượng `AGraph` này cho phép mô-đun `ModuleHandleOrder` xử

lý từng `ANode` của nó chẳng hạn: `ANode` tương ứng với nút 14 trong đoạn mã 6.1, cũng như lớp miền được tham chiếu bởi `ANode` đó, trong ví dụ này là lớp `Payment`.

Cách AGL và các chú thích đi kèm được hiện thực trong Java. Đặc tả AGL của một lớp hoạt độngnhằm được biểu diễn dưới dạng chú thích ngay trên lớp đó trong mã Java. Ví dụ, đoạn mã 6.1 minh họa cách chú thích `AGraph` nhằm biểu diễn đặc tả AGL của lớp hoạt động `HandleOrder`. Tất cả các chú thích của AGL, như tóm tắt trong Hình 3.9, đều phải được định nghĩa trong Java. Đoạn mã 6.2 cung cấp ví dụ về định nghĩa chú thích `AGraph`, và các chú thích khác trong AGL cũng được định nghĩa tương tự.

```
1 package jda.modules.mbsl.model.graph.meta;
2 import java.lang.annotation.*;
3 import jda.modules.mbsl.model.graph.ActivityGraph;
4     @Retention(RetentionPolicy.RUNTIME)
5     @Target(value=java.lang.annotation.ElementType.TYPE)
6     @Documented
7 public @interface AGraph {ANode[] nodes();}
```

Đặc tả 6.2: Hiện thực Java của chú thích `AGraph`

6.2.3 Công cụ hỗ trợ phương pháp tích hợp các mối quan tâm vào mô hình miền

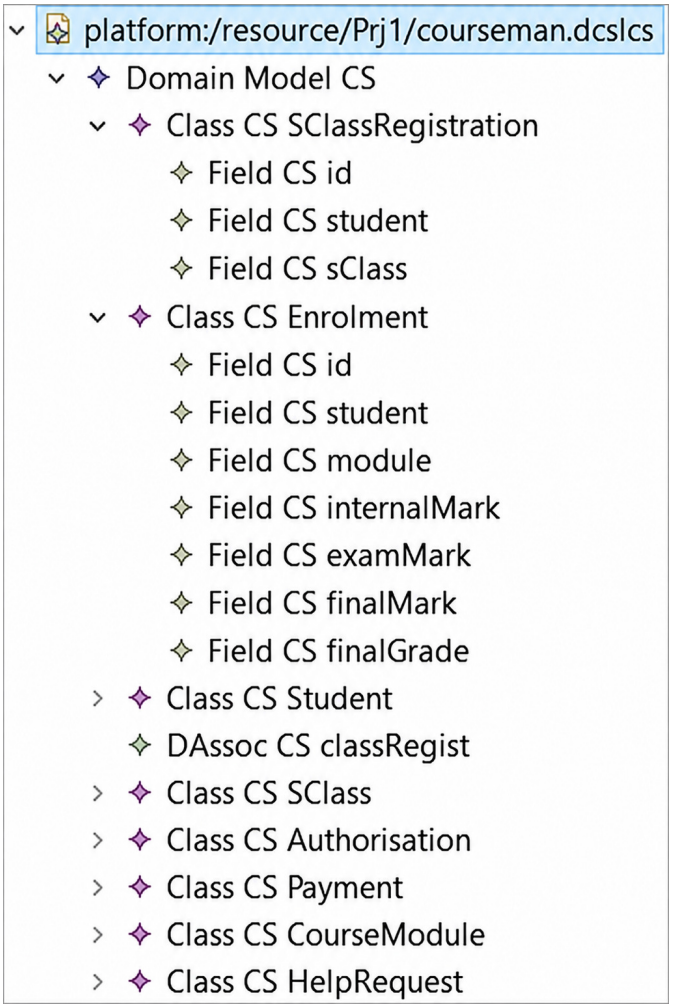
Trong phần này trình bày công cụ hỗ trợ có tên UDMM và đánh giá kết quả thực nghiệm phương pháp đề xuất trên nghiên cứu điển hình CourseMan

Công cụ hỗ trợ tích hợp theo phương pháp siêu mô hình

Công cụ hỗ trợ UDMM bao gồm ba hoạt động chính: (i) định nghĩa và kết hợp các DSL theo mối quan tâm để tích hợp vào mô hình UDML; (ii) sinh mã nguồn; và (iii) tạo phần mềm hoặc bản mẫu, theo phương pháp được đề xuất trong Hình 4.1.

Việc xây dựng công cụ tập trung vào xác định các thành phần DSL cần biểu diễn đồ họa (lớp, thuộc tính, quan hệ), định nghĩa siêu mô hình UDML bằng EMF nhằm đảm bảo tính chặt chẽ và khả năng mở rộng, và thiết kế giao diện kéo và thả trực quan bằng Sirius. Trên cơ sở đó, cây cú pháp đồ họa được xây dựng để tổ chức và biểu diễn trực quan các thành phần của DSL.

Tích hợp với quy trình phát triển phần mềm: Mô hình đồ họa do Sirius tạo ra có thể được chuyển đổi thành mã thực thi, hỗ trợ trực tiếp quá trình phát triển ứng dụng.



Hình 6.10: Giao diện kéo và thả để tạo mô hình UDML.

Quá trình ánh xạ từ mô hình sang mã nguồn Java được định nghĩa bằng các luật chuyển đổi trong các tệp **.mtl** thuộc Acceleo. Các luật này quy định cách các thành phần trong mô hình dcslcs được chuyển thành mã nguồn Java tương ứng. Lớp (ClassCS) trong mô hình được ánh xạ thành các lớp Java với khai báo thuộc tính, phương thức và các mối quan hệ. Thuộc tính (FieldCS) được ánh xạ thành các biến thành viên trong lớp Java với các phương thức getter và setter tương ứng. Phương thức (MethodCS) được ánh xạ thành các phương thức, hàm tương ứng. Liên kết (DAssocCS) được chuyển đổi thành quan hệ giữa các đối tượng, sử dụng kiểu dữ liệu phù hợp như danh sách hoặc tham chiếu. Các kiểu dữ liệu (như, PrimitiveType, ReferenceTypeCS) được ánh xạ sang các kiểu dữ liệu tương ứng trong Java

Sau khi xây dựng, nhà thiết kế có thể thực hiện việc kéo thả để tạo mô hình. Công cụ tiến hành thực nghiệm trên CourseMan Hình 1.2, được biểu diễn bằng biểu đồ lớp UML cùng với các ràng buộc OCL. Sử dụng giao diện kéo thả tạo mô hình và thực hiện chuyển đổi mô hình sang mô hình mô hình miền hợp nhất dạng cụ thể (gần với ngôn ngữ OOPL) có thể tạo mã bằng công cụ ATL Hình 6.10. Giúp nâng cao khả năng sử dụng và hiểu biết về hệ

thống là cú pháp cụ thể dạng đồ họa. Giúp các bên liên quan, bao gồm nhà phát triển, kiến trúc sư phần mềm và chuyên gia miền, dễ dàng tiếp cận và phân tích mô hình, hỗ trợ trực quan hóa các thành phần của hệ thống một cách rõ ràng. Mô hình miền hợp nhất dạng cụ thể được tạo ra chuyển sang bước tiếp theo là sinh mã nguồn tự động, sử dụng Acceleo để ánh xạ các thành phần trong mô hình thành mã nguồn.

Công cụ hỗ trợ phương pháp tích hợp mối quan tâm dựa vào cây cú pháp

Công cụ được được xây dựng bằng MPS [20, 128] có cấu trúcsau bao gồm bốn lớp như sau:

Lớp lõi: Định nghĩa cú pháp của DSL lõi (`UDML.core`), tạo nền tảng cho việc tích hợp các mối quan tâm khác vào UDML. Lớp này bao gồm các khái niệm cơ bản như `Annotable`, `Annotation`, `Class` và `Association`.

Lớp mối quan tâm: Định nghĩa cú pháp trừu tượng cho từng mối quan tâm tùy theo mục tiêu phát triển. Ví dụ: `UDM.rbac`, `UDM.gui`, `UDM.dcsl`, và `UDM.conc` cho các khái niệm tổng quát khác.

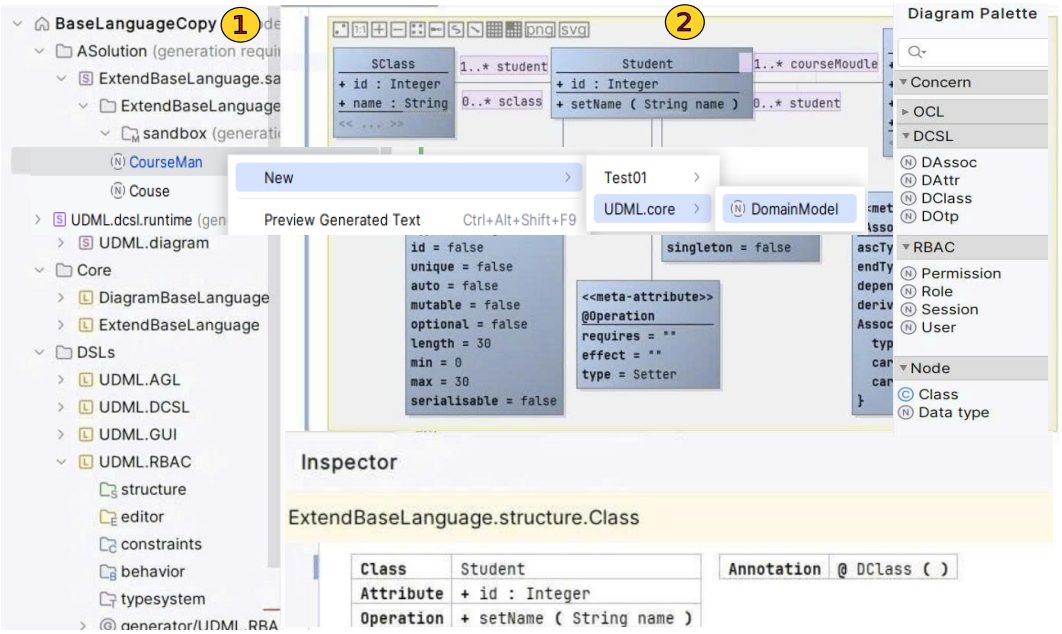
Lớp hợp nhất: Hỗ trợ việc hợp nhất các DSL theo mối quan tâm vào mô hình UDML sử dụng cơ chế hợp nhất được mở rộng, tạo thành một cây cú pháp trừu tượng (AST) thống nhất và nhất quán.

Lớp dịch vụ: Cung cấp các dịch vụ giao diện đồ hoạ cho phép nhà thiết kế thực hiện các tác vụ như chỉnh sửa trình bày, kiểm tra ràng buộc, sinh phần mềm/nguyên mẫu, cùng với các dịch vụ hỗ trợ như kiểm tra bảo mật.

Phương pháp được hiện thực bằng JetBrains MPS [20, 128], một nền tảng linh hoạt cho việc định nghĩa và hợp nhất DSL dựa trên AST. Mỗi DSL theo mối quan tâm định nghĩa một tập các khái niệm (tương ứng với các kiểu nút trong AST) có thể được tái sử dụng và kết hợp trong mô hình. Các trình chỉnh sửa dạng chiếu cho phép mỗi DSL có giao diện chuyên biệt riêng, trong khi tất cả cùng đóng góp vào một mô hình thống nhất. MPS cũng hỗ trợ tích hợp vào quy trình phát triển thông qua sinh mã và công cụ build bên ngoài, dù thường yêu cầu điều chỉnh thay vì tích hợp hoàn toàn tự động.

Mỗi concern DSL được định nghĩa dựa trên bốn thành phần chính: (i) Cú pháp trừu tượng (AS): Xác định cấu trúc ngôn ngữ bằng các khái niệm tương ứng với các nút trong AST. (ii) Cú pháp cụ thể: Xác định cách biểu diễn các khái niệm (dạng bảng, văn bản, ký hiệu...). (iii) Hệ thống kiểu và ràng buộc: Đảm bảo tính đúng đắn của mô hình qua kiểm tra và ràng buộc ngữ nghĩa. (iv) Bộ sinh mã: Chuyển đổi mô hình DSL thành mã đích, thường nhúng vào một ngôn ngữ lập trình hướng đối tượng (OOPL).

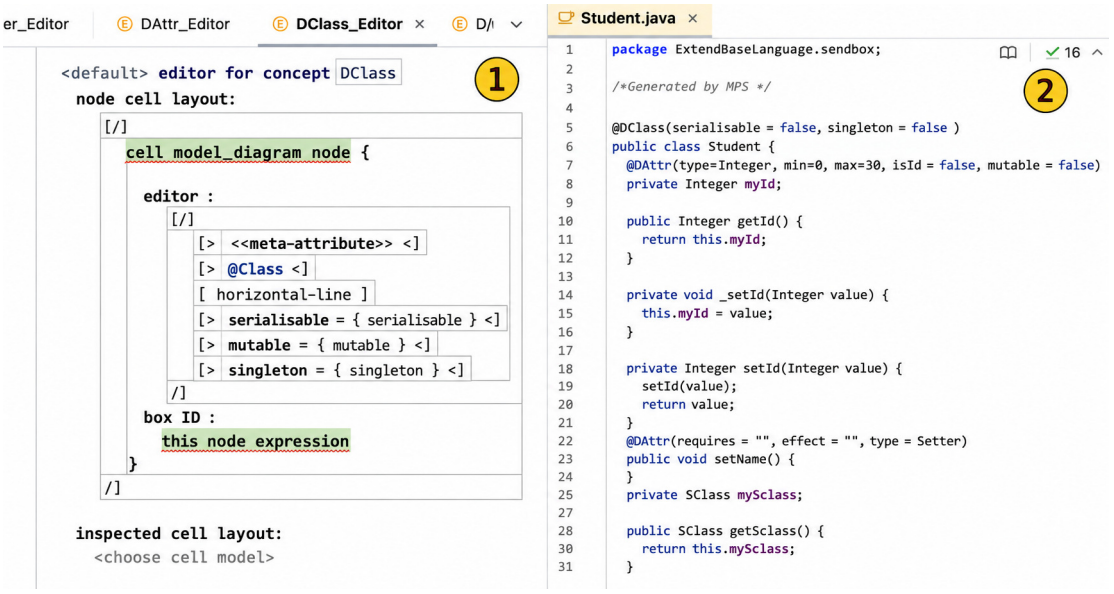
Công cụ hỗ trợ việc thiết kế, mở rộng và tích hợp các concern DSL, đồng thời kiểm chứng tính biểu đạt, khả thi và mức độ thỏa mãn của phương pháp.



Hình 6.11: Hiện thực hóa dựa trên MPS và tích hợp thực tiễn các DSL theo mối quan tâm trong UDML.

Tính khả thi của phương pháp được minh họa trong Hình 6.11, phía bên trái thể hiện ba thành phần lõi: (i) Chỉ dẫn: mô tả các siêu khái niệm cho UDML và các DSL theo mối quan tâm, (ii) Soạn thảo: cung cấp giao diện cú pháp cụ thể với hỗ trợ kéo và thả, và (iii) Bộ sinh mã: biến đổi AST thành mã thực thi dựa trên khung phần mềm JDA [75]. Ngôn ngữ UDML được thiết kế thành các gói mô-đun: `UDML.core` đóng vai trò nền tảng tích hợp; `UDML.dcsl`, `UDML.agl`, và `UDML.rbac` lần lượt mô tả các mối quan tâm về cấu trúc, hành vi và bảo mật. Khu vực trung tâm cung cấp giao diện mô hình hóa miền với hỗ trợ dựng mô hình trực quan.

Luận án áp dụng công cụ vào nghiên cứu tình huống CourseMan. Trong Hình 6.11, nhãn (2) minh hoạ biểu đồ lớp UML cùng các DSL theo mối quan tâm được tích hợp: DCSL, AGL và RBAC. Công cụ sau đó sinh mã nguồn và tạo ra nguyên mẫu phần mềm bằng khung làm việc JDA, với các chi tiết kỹ thuật trình bày trong Hình 6.12.



Hình 6.12: Sử dụng MPS và sinh mã để áp dụng vào khung JDA.

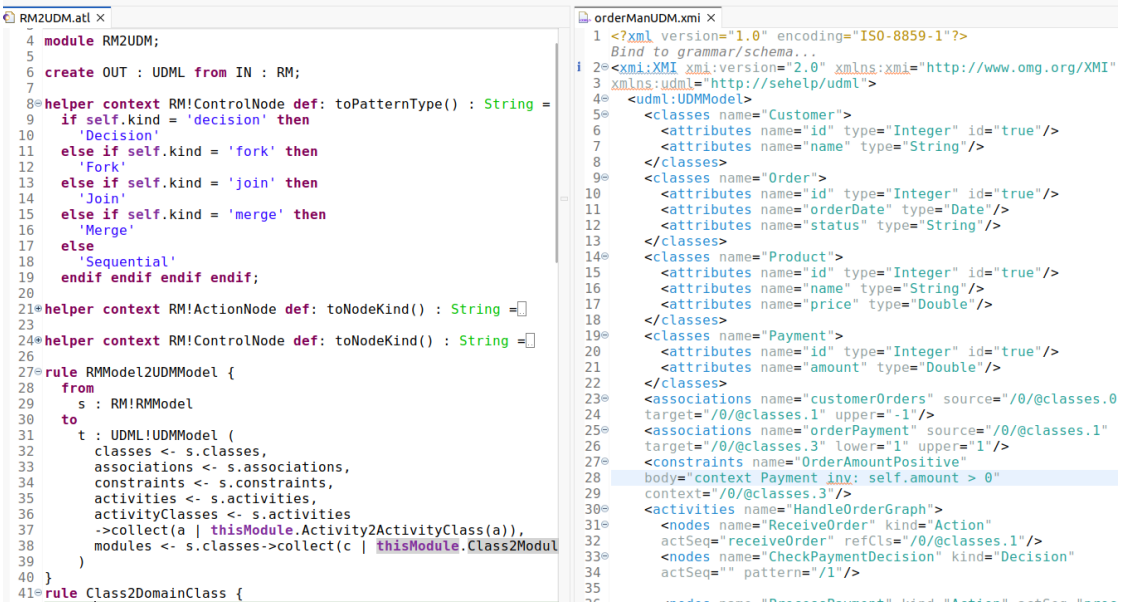
Trong Hình 6.12, phía trái (nhãn (1)) cho thấy định nghĩa tệp mẫu trong MPS, trong khi phía phải (nhãn (2)) thể hiện mô hình miền `Student` của hệ thống quản lý khóa học. Mô hình này sau đó được nhúng vào JDA để sinh các tạo tác phần mềm.

6.2.4 Công cụ hỗ trợ chuyển đổi mô hình

Sinh mô hình miền hợp nhất từ mô hình yêu cầu (RM2UDM)

Công cụ được hiện thực nhằm hỗ trợ quá trình chuyển đổi mô hình từ RM sang UDM theo bộ chuyển đổi RM2UDM. Để minh họa cách thức hoạt động của công cụ, luận án sử dụng hệ thống CourseMan như một ví dụ điển hình. Kết quả thực nghiệm cho thấy công cụ hỗ trợ hiệu quả việc xây dựng biểu diễn miền hợp nhất và thực hiện các phép chuyển đổi mô hình. Trên cơ sở đó, các kết quả được phân tích trong bối cảnh các ca nghiên cứu của chương, đồng thời làm rõ các yếu tố ảnh hưởng đến tính đúng đắn và hiệu quả của phương pháp.

Để hiện thực các chuyển đổi mô hình, các luật chuyển được đặc tả bằng ngôn ngữ chuyển đổi mô hình sang mô hình (M2M), trong đó ATL [65] được sử dụng để xây dựng bộ chuyển đổi RM2UDM như đã trình bày trong Mục 5.3. Hình 6.13 (A) minh họa một phần các luật chuyển được cài đặt bằng ATL, trong khi Hình 6.13 (B) thể hiện kết quả ánh xạ từ mô hình yêu cầu sang mô hình miền hợp nhất UDM của OrderMan.



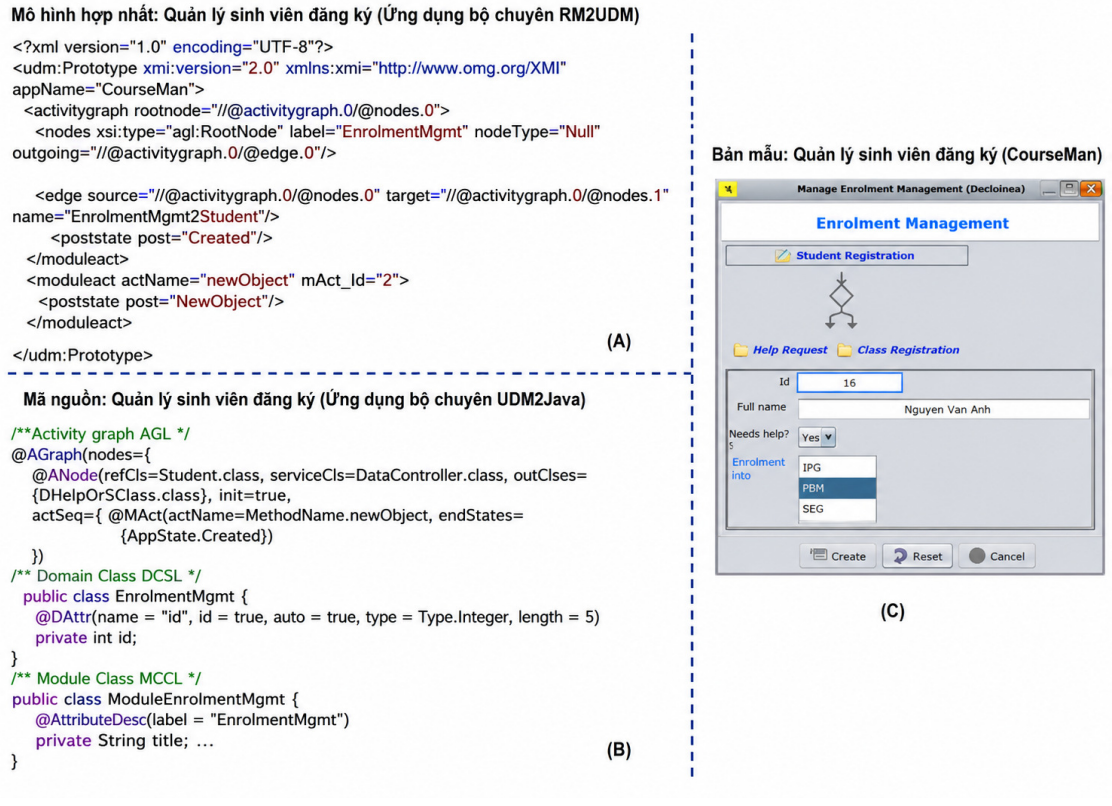
Hình 6.13: Các luật chuyển đổi của ATL và kết quả mô hình UDM.

Thực nghiệm. Công cụ hỗ trợ phương pháp đề xuất được phát triển nhằm sinh tự động bản mẫu phần mềm từ mô hình yêu cầu. Người thiết kế sử dụng các công cụ mô hình hóa trực quan để xây dựng các biểu đồ lớp và biểu đồ hoạt động UML/OCL. Bộ chuyển đổi RM2UDM được hiện thực bằng ATL trên nền tảng Eclipse để chuyển đổi mô hình yêu cầu sang mô hình miền hợp nhất UDM.

Trên cơ sở đó, mô hình UDM được chuyển đổi sang đặc tả miền thực thi AGL^+ dưới dạng mã nguồn sử dụng phép chuyển đổi UDM2AGL. Các đặc tả này sau đó được sử dụng trong khung phần mềm JDA để sinh tự động bản mẫu phần mềm theo thiết kế hướng miền. Hình 6.14 (A) hiển thị một phần UDM dạng XMI là kết quả của bộ chuyển đổi RM2UDM, Hình 6.14 (B) hiển thị một phần mã nguồn Java biểu diễn đặc tả AGL^+ được sinh ra từ UDM sử dụng bộ chuyển UDM2AGL, và Hình 6.14 (C) biểu diễn kết quả GUI bản mẫu phần mềm cho CourseMan. Để đánh giá hiệu quả của công cụ trên hệ thống CourseMan, các kết quả thực nghiệm cho thấy: đối với bộ

chuyển đổi RM2UDM, mô hình miền hợp nhất UDM đạt mức độ bao phủ và khả năng diễn đạt trên 95% so với mô hình yêu cầu đầu vào.

Đối với bước chuyển đổi từ mô hình UDM sang đặc tả miền thực thi AGL^+, toàn bộ các lớp miền trong mô hình yêu cầu đều được chuyển đổi tương ứng thành các lớp trong mã nguồn và các mô-đun phần mềm. Cụ thể, để xây dựng bản mẫu phần mềm của CourseMan theo cách thủ công cần 549 dòng mã lệnh chính, trong khi phương pháp đề xuất sinh tự động được 477 dòng mã. Tỷ lệ mã nguồn được sinh tự động đạt khoảng 87%, trong khi phần chỉnh sửa và bổ sung thủ công chiếm dưới 13%. Nguyên nhân của các chỉnh sửa này được phân tích trong phần thảo luận tiếp theo.



Hình 6.14: Công cụ hỗ trợ sinh tự động mô hình miền hợp nhất và bản mẫu phần mềm từ đặc tả yêu cầu.

Thảo luận. Phương pháp đề xuất đã được áp dụng thành công cho hệ thống CourseMan, cho thấy tiềm năng ứng dụng của phương pháp trong thực tiễn. Tuy nhiên, vẫn tồn tại một số mối đe dọa đối với tính hợp lệ của phương pháp.

Thứ nhất, phương pháp tiếp nhận đặc tả yêu cầu đầu vào dưới dạng các biểu đồ UML/OCL, do đó tính đúng đắn của kết quả phụ thuộc vào khả năng diễn đạt và mức độ chính xác của ngôn ngữ mô hình hóa này.

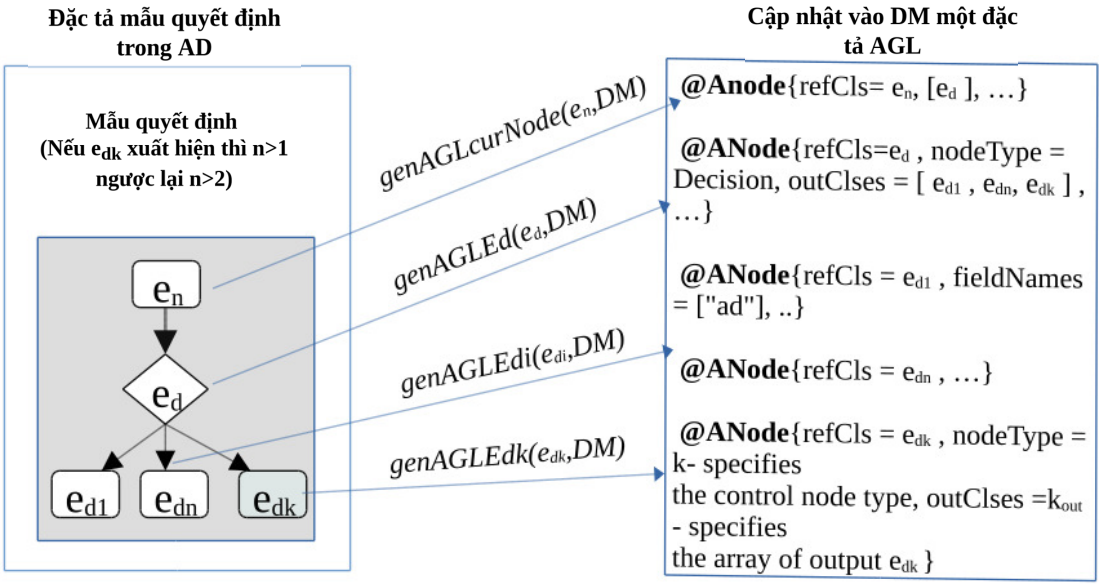
Thứ hai, phương pháp hiện tại chưa đảm bảo rằng mọi mô hình UML/OCL đầu vào đều có thể được chuyển đổi tự động sang một mô hình hợp nhất UDM. Bên cạnh đó, các ràng buộc OCL trong mô hình yêu cầu cần được diễn đạt tương ứng bằng DCSL trong mô hình UDM thông qua các thao tác thủ công. Những hạn chế này liên quan trực tiếp đến phạm vi áp dụng và tính đúng đắn của bộ chuyển đổi RM2UDM. Trong phạm vi phương pháp đề xuất, tính đúng đắn ở khía cạnh này được đảm bảo bằng việc rà soát các luật chuyển đổi và thực hiện kiểm thử cho bộ chuyển đổi.

Thứ ba, bản mẫu thực thi sinh ra trong Java có thể chưa hoàn toàn tương thích với đặc tả yêu cầu đầu vào nếu tồn tại sai sót trong các ánh xạ tự động từ mô hình UDM sang đặc tả AGL^{+}. Tương tự như bộ chuyển đổi RM2UDM, độ chính xác của bước sinh mã được đảm bảo bằng cách kiểm tra, rà soát các luật ánh xạ và tiến hành kiểm thử đối với bộ chuyển đổi này.

Sinh mã nguồn bản mẫu phần mềm (UDM2AGL)

Trong phần này, luận án sử dụng Hệ thống quản lý đơn hàng OrderMan làm ca nghiên cứu để minh họa việc hiện thực hóa bộ chuyển đổi từ mô hình miền hợp nhất (UDM) sang đặc tả miền thực thi AGL^{+}. Trên cơ sở mô hình UDM đã được xây dựng, các phần tử hành vi biểu diễn dưới dạng AG được chuyển đổi sang các đặc tả AGL^{+} tương ứng theo các quy tắc chuyển đổi đã đề xuất. Quá trình này làm rõ cách thức bảo toàn ngữ nghĩa của hành vi miền trong quá trình chuyển đổi, đồng thời tạo đầu vào cho các bước sinh mã nguồn và hiện thực phần mềm. Ca nghiên cứu OrderMan được lựa chọn nhằm minh họa tính khả thi của phương pháp trên một hệ thống có quy trình nghiệp vụ đặc trưng và nhiều nhánh điều khiển.

Hình 6.15 minh họa một nút quyết định trong đồ thị hoạt động AG của mô hình miền hợp nhất UDM và cách ánh xạ nút này sang đặc tả AGL tương ứng. Trong đồ thị nguồn, nút quyết định được sử dụng để biểu diễn cấu trúc rẽ nhánh của luồng thực thi, trong đó mỗi cạnh đi ra được gắn với một điều kiện cảnh báo. Trong quá trình chuyển đổi, khi duyệt đến một nút quyết định $n \in N$, bộ chuyển đổi UDM2AGL áp dụng hàm `genDecisionNode(`n`)` để sinh phần tử tương ứng trong đặc tả AGL.



Hình 6.15: Ánh xạ nút quyết định từ đặc tả AG sang AGL.

Hàm này thực hiện việc ánh xạ cấu trúc rẽ nhánh của nút nguồn sang biểu diễn điều khiển tương ứng trong AGL, đồng thời bảo toàn các điều kiện gác gắn với các cạnh đi ra.

Cụ thể, mỗi nhánh xuất phát từ nút quyết định trong *AG* được chuyển thành một nhánh điều kiện trong đặc tả AGL, qua đó phản ánh đầy đủ cấu trúc điều khiển của hành vi miền. Việc ánh xạ này bảo đảm rằng ngữ nghĩa của cấu trúc rẽ nhánh trong mô hình nguồn được duy trì trong biểu diễn đích, đồng thời phù hợp với cơ chế thực thi của AGL. Ví dụ này minh họa cách các nút điều khiển trong *AG* được hiện thực hóa thành các cấu trúc điều khiển tương ứng trong đặc tả AGL, làm rõ các luật ánh xạ được áp dụng trong bộ chuyển đổi UDM2AGL.

Hình 6.16 minh họa quá trình chuyển đổi mô hình sang văn bản từ mô hình miền hợp nhất *UDM* sang đặc tả AGL^+ sử dụng Acceleo. Quá trình này dựa trên các mẫu chuyển đổi được định nghĩa trên siêu mô hình, trong đó các nút hành vi trong đồ thị hoạt động *AG* được duyệt, phân loại theo kiểu và ánh xạ tương ứng sang các cấu trúc trong AGL, qua đó hình thành biểu diễn miền có khả năng thực thi. Mẫu trong Acceleo để chuyển đổi một nút *DecisionNode* trong *AG*, với hàm `genDecisionNode(curNode)`, bao gồm các bước sau: Bước 1, biểu diễn cấu hình mẫu của đồ thị hoạt động cho mẫu quyết định trong đặc tả AGL^+; Bước 2, đọc tất cả các cạnh đi ra từ *DecisionNode*, ánh xạ chúng vào template tương ứng và sinh ra các `ANode`

generateAD2AGL.mtl

```
1  [module generateAD2AGL('http://www.eclipse.org/uml2/5.0.0/UML')]
2  [comment]-------- Generate AGL ----------------------[/comment]
3  [template public generateElementAGL(aModel : Model)]
4  [file (aModel.name.concat('.java'), false, 'UTF-8')]
5  /**Activity graph configuration*/
6  @AGraph(nodes={
7      [let nodeElements : Sequence(ActivityNode) =
8                  packagedElement->select(e | e.oclIsKindOf(Activity))
9                  ->collect(e | e.oclAsType(Activity).node)
10                 ->sortedBy(n | n.name)]
11     [for (aActivityNode : ActivityNode | nodeElements)]
12                 [let nextNode: ActivityNode=aActivityNode.outgoing->
13                                          any(true).target]
14                   [if (nextNode.oclIsKindOf(OpaqueAction))]
15                       [callSequentialAGL(aActivityNode)/]
16                   [/if]
17                   [if (nextNode.oclIsKindOf(DecisionNode))]
18                       [callDecisionAGL(aActivityNode)/]
19                   [/if]
20                   [if (nextNode.oclIsKindOf(ForkNode))]
21                       [callForkAGL(aActivityNode)/]
22                   [/if]
23                   [if (nextNode.oclIsKindOf(MergeNode))]
24                       [callMergeAGL(aActivityNode)/]
25                   [/if]
26                   [if (nextNode.oclIsKindOf(JoinNode))]
27                       [callJoinAGL(aActivityNode)/]
28                   [/if]
29                 [/let]
30     [/for]
31     [/let]
```

Updates to DM an AGL with Sequential pattern

Updates to DM an AGL with Decision pattern

Updates to DM an AGL with Fork pattern

Updates to DM an AGL with Merge pattern

Updates to DM an AGL with Join pattern

Hình 6.16: Các luật (mẫu trong Acceleo) để chuyển đổi từ *UDM* sang đặc tả AGL^+.

phản ánh cấu trúc rẽ nhánh trong đặc tả AGL. Dựa trên các điều kiện gác, mỗi `ANode` được liên kết đến nút kế tiếp phù hợp; Bước 3, sử dụng các mẫu truy vấn để trích xuất thông tin từ mô hình *UDM*, qua đó cập nhật đặc tả AGL của mô hình miền.

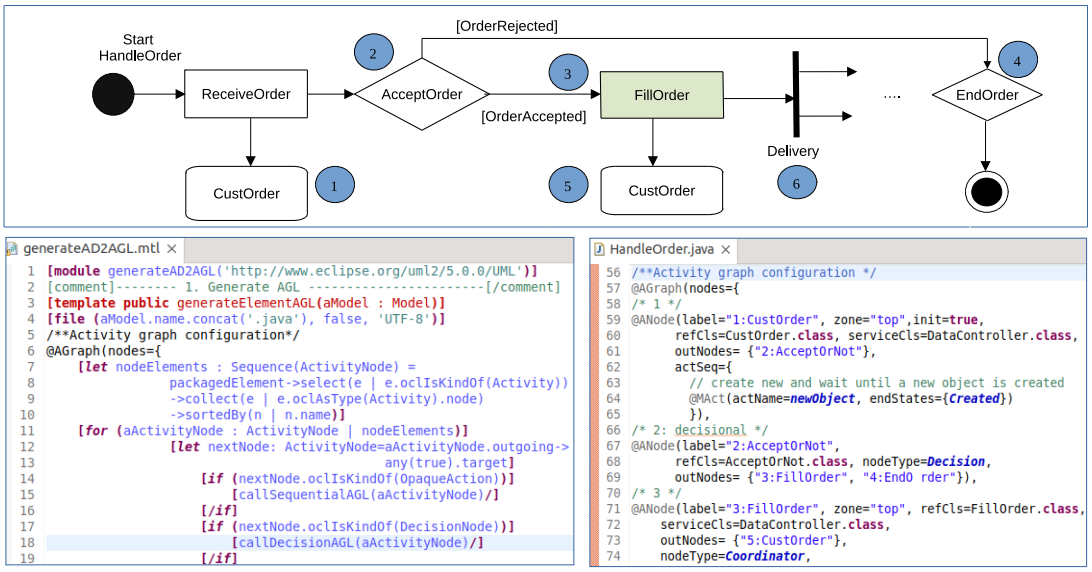
Mẫu trong Acceleo để sinh đặc tả DCSL được thực hiện bằng hàm *genDCSL(model:Model)*. Đầu vào của hàm là phần cấu trúc miền *DM* trong mô hình miền hợp nhất *UDM*, bao gồm các lớp, thuộc tính, quan hệ kết hợp và các ràng buộc được kế thừa từ mô hình yêu cầu. Hàm này thực thi các bước sau: Bước 1, biểu diễn mẫu DCSL trong đặc tả AGL^+; Bước 2, duyệt tất cả các lớp trong *DM* và ánh xạ chúng vào các mẫu tương ứng. Dựa trên các ràng buộc về lớp, thuộc tính và quan hệ, tiến hành sinh các phần tử cấu trúc trong đặc tả DCSL; Bước 3, sử dụng các mẫu truy vấn để trích xuất thông tin từ mô hình nhằm ánh xạ các thành phần *DClass, DAttr, DOpt* và *DAssoc*, qua đó cập nhật đặc tả AGL^+ của mô hình miền.

Đoạn mã 6.3 minh họa các mẫu truy vấn dành cho **@MAct** và **@DClass**, từ đó sinh ra các thành phần tương ứng trong đặc tả AGL^+.

```
1     /**actName = newObject,pstStates=\{Created\}*/
2     [query public genMActcreateObject(): Set(String) = '@MAct(
      actName = newObject,pstStates = [Created])'
3     /]
4     /** @DClass(serialisable = true, singleton = true)*/
5     [query public genDClass(serialisable: String,singleton:
      String): Set(String) = '@DClass(serialisable = '.concat(
      serialisable).concat(',singleton = ').concat(singleton).
      concat(')')
6     /]
```

Đặc tả 6.3: The template generate the *@MAct* and *@DClass*.

Hình 6.17 minh họa biểu diễn hành vi trong mô hình miền của hệ thống OrderMan ở phía trên. Phần bên trái phía dưới hiển thị khuôn mẫu bằng mã Acceleo thực hiện phép chuyển đổi sang mã `HandleOrder`, trong khi kết quả đầu ra của phương pháp đặc tả AGL^+ được trình bày ở phía dưới bên phải.

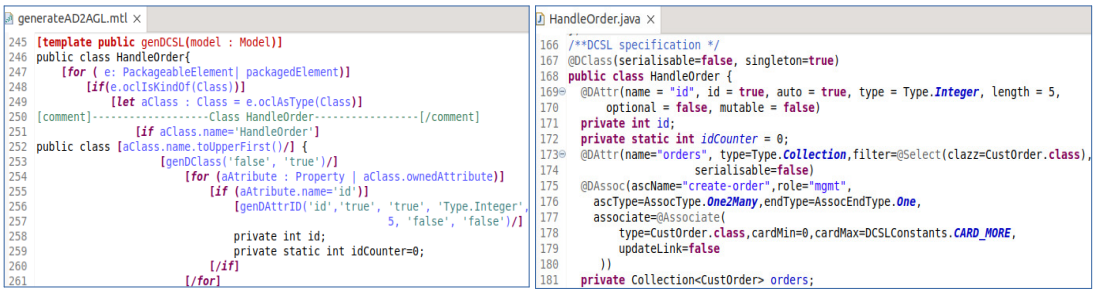


Hình 6.17: Sử dụng Acceleo để chuyển đổi mô hình *UDM* sang đặc tả AGL^+, lớp `HandleOrder` được hiện thực trong Java.

Ngoài ra, Acceleo cũng được sử dụng để sinh đặc tả DCSL từ phần cấu trúc miền trong *UDM*. Trong Hình 6.18, khuôn mẫu bằng mã Acceleo dùng để sinh đặc tả DCSL được trình bày ở bên trái, trong khi kết quả đặc tả DCSL thu được được hiển thị ở bên phải.

Các thành phần hành vi (AGL) và cấu trúc (DCSL) được kết hợp để hình thành đặc tả miền thực thi AGL^+. Đặc tả này sau đó được sử dụng

làm đầu vào để tự động sinh và hiện thực hệ thống với sự hỗ trợ của khung phần mềm JDA.



Hình 6.18: Sử dụng Acceleo để sinh đặc tả DCSL từ mô hình miền trong *UDM*.

Thảo luận. Nghiên cứu tình huống với hệ thống OrderMan cho thấy phép chuyển đổi từ mô hình miền hợp nhất sang đặc tả miền thực thi là khả thi và có tính ứng dụng thực tiễn. Việc sử dụng Acceleo để hiện thực hóa các luật chuyển đổi từ *UDM* sang AGL^+ chứng minh rằng mô hình miền có thể được tự động chuyển hóa thành một biểu diễn có khả năng thực thi trong bối cảnh thiết kế hướng miền.

Tuy nhiên, phương pháp đề xuất chịu ảnh hưởng bởi chất lượng và mức độ đầy đủ của mô hình đầu vào. Các biểu diễn hành vi chưa đầy đủ thông tin ngữ nghĩa (chẳng hạn điều kiện gác hoặc luồng dữ liệu) có thể dẫn đến đặc tả AGL^+ chưa hoàn chỉnh và cần hiệu chỉnh thủ công. Điều này phản ánh hạn chế chung của các tiếp cận sinh mã dựa trên mô hình khi mô hình nguồn chưa chứa đủ thông tin cho thực thi.

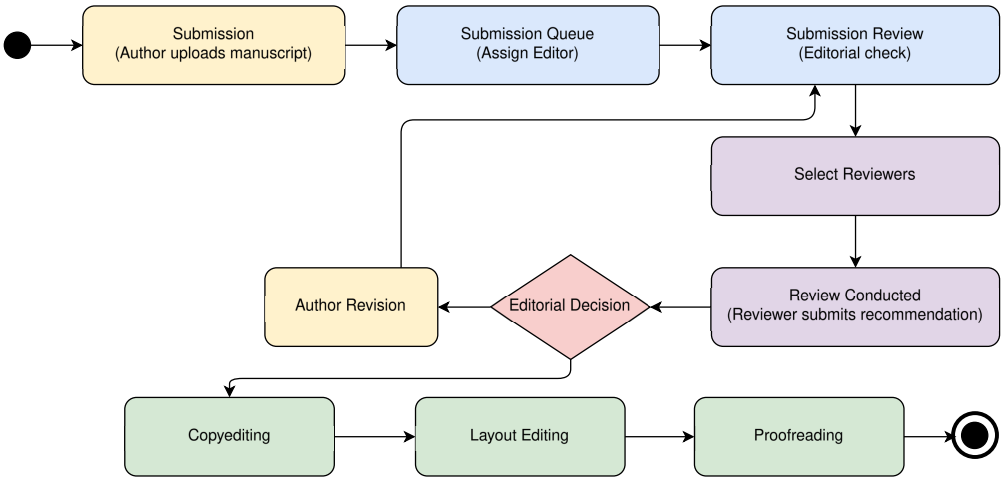
Việc kết hợp các thành phần hành vi và cấu trúc trong đặc tả AGL^+ khẳng định vai trò trung tâm của mô hình miền trong quá trình sinh phần mềm theo thiết kế hướng miền, trong đó mô hình miền đóng vai trò cầu nối giữa đặc tả yêu cầu và hiện thực hệ thống.

Công cụ hỗ trợ và đánh giá cho UDML-to-Event-B

Phần này trình bày quy trình làm việc có sự hỗ trợ của công cụ nhằm hiện thực hóa và đánh giá khung ngữ nghĩa được đề xuất cho UDML. Rodin được sử dụng như một hậu kiểm chứng để giải quyết các nghĩa vụ chứng minh (POs) của các mô hình Event-B được sinh ra, qua đó xác nhận tính

đúng đắn của ngữ nghĩa hình thức đối với các mô hình UDML được hợp thành cùng RBACDom.

Cấu hình thực nghiệm. Phần này trình bày cấu hình thực nghiệm được sử dụng để đánh giá tính khả thi và hiệu quả của phương pháp đề xuất. Việc đánh giá được thực hiện thông qua một ca nghiên cứu điển hình trên miền *Open Journal Systems* (OJS). Hình 6.19 minh họa quy trình quản lý bài nộp của OJS, được mô hình hóa dưới dạng đồ thị hoạt động AGL trong UDML và được hợp thành với các chính sách RBACDom tương ứng tại từng nút hành vi. Việc thực nghiệm xem xét một tập cố định gồm bảy vai trò:



Hình 6.19: Quy trình xử lý bài báo được nộp trong OJS.

Roles={Author, Reviewer, JournalManager, Editor, Copyeditor, LayoutEditor, Proofreader}.

Tổng cộng, chín chính sách RBACDom ở mức nút $\langle P_{N1}, \ldots, P_{N9} \rangle$ được đặc tả dưới dạng các thể hiện của `RbacDomNode` trong mô hình RBACDom. Các chính sách này được liên kết với các bước trong quy trình xử lý bài báo được nộp của OJS thông qua sự tương ứng với các nhãn nút AGL, được xác định bởi thuộc tính `aglNode`.

Ánh xạ các chính sách RBAC sang AGL. Mỗi chính sách P_{Ni} được đặc tả như một quy tắc kiểm soát truy cập ở mức nút trong RBACDom và được biểu diễn bởi một `RbacDomNode`. Một chính sách ràng buộc việc thực thi của một bước cụ thể trong quy trình quản lý bài nộp bằng cách áp đặt một điều kiện ủy quyền lên nút AGL tương ứng, được xác định bởi `aglNode`. Cụ thể, một chính sách xác định: (i) các vai trò được phép, (ii) các vị từ

phạm vi gắn với đối tượng miền, (iii) chế độ và hiệu lực của chính sách, và (iv) các ràng buộc SoD và tuân thủ tùy chọn. Các chính sách thu được cho quy trình quản lý bài nộp OJS được tổng hợp trong Bảng 6.1.

Bảng 6.1: Chính sách RBACDom dựa trên tương ứng nút cho OJS

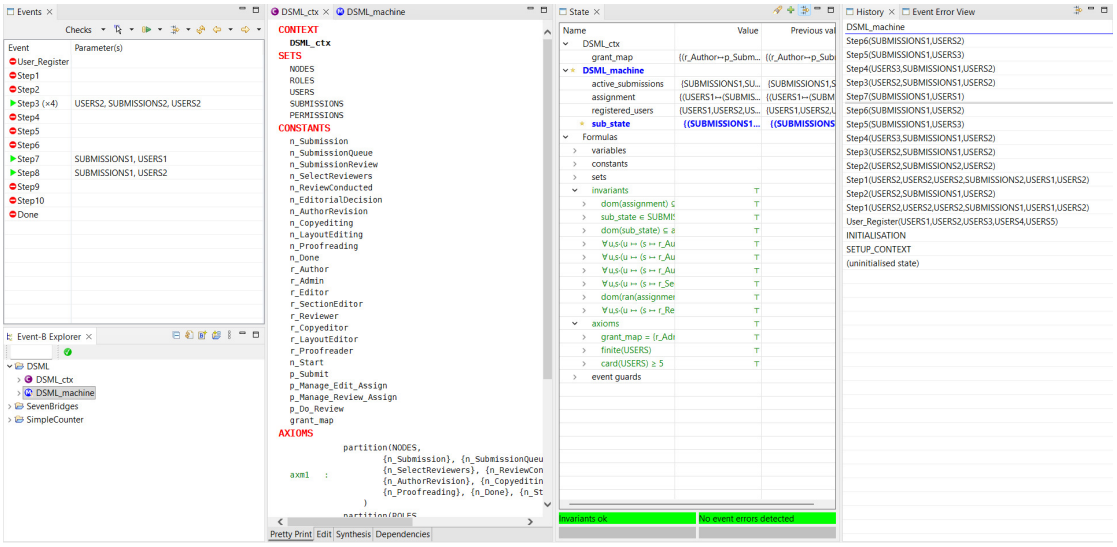
Nút hoạt động	**Chính sách**	**Ràng buộc RBACDom (vai trò, phạm vi, SoD, tuân thủ)**
SubmitPaper	«rbac:P_N1»	{roles={Author}, policy=ALL_OF, effect=ALLOW, scope=owns(u, sub) theo xây dựng (trường tác giả được gán bằng u), SoD=SSD/DSD(Author≠Reviewer trên *sub*, Author≠Editor trên *sub*), compliance=log(createSubmission), hideReviewerIdentities}
AssignEditor	«rbac:P_N2»	{roles={JournalManager}, policy=ALL_OF, effect=ALLOW, scope=state(sub) = SUBMITTED, SoD=DSD(JournalManager≠Author trên *sub*) [tùy chọn], compliance=log(assignEditor)}
EditorialCheck	«rbac:P_N3»	{roles={Editor}, policy=ALL_OF, effect=ALLOW, scope=assignedEditor(u, sub) ∧ state(sub) = SUBMITTED, SoD=DSD(Editor≠Author trên *sub*), compliance=log(editorialCheck)}
SelectReviewer	«rbac:P_N4»	{roles={Editor}, policy=ALL_OF, effect=ALLOW, scope=assignedEditor(u, sub) ∧ state(sub) ∈ {EDITORIAL_CHECKED, UNDER_REVIEW}, SoD=DSD(Editor≠Author trên *sub*)∧DSD(Editor≠Reviewer trên *sub*), compliance=log(assignReviewer), enforceBlindReview}
DoReview	«rbac:P_N5»	{roles={Reviewer}, policy=ALL_OF, effect=ALLOW, scope=assignedReviewer(u, sub) ∧ state(sub) = UNDER_REVIEW ∧ ¬coi(u, sub), SoD=DSD(Reviewer≠Author trên *sub*)∧DSD(Reviewer≠Editor trên *sub*), compliance=log(submitReview), deadlineLogged}
EditorialDecision (Nút quyết định)	«rbac:P_N6»	{roles={Editor}, policy=ALL_OF, effect=ALLOW, scope=assignedEditor(u, sub) ∧ state(sub) = UNDER_REVIEW ∧ allReviewsSubmitted(sub), SoD=DSD(Editor≠Author trên *sub*)∧DSD(Editor≠Reviewer trên *sub*), compliance=log(editorDecision), rationaleRequired [tùy chọn]}
CopyEditing	«rbac:P_N7»	{roles={Copyeditor}, policy=ALL_OF, effect=ALLOW, scope=assignedCopyeditor(u, sub) ∧ state(sub) = ACCEPTED, SoD=DSD(Copyeditor≠Author trên *sub*) [tùy chọn], compliance=log(copyedit), noAccessToReviewData}
LayoutEditing	«rbac:P_N8»	{roles={LayoutEditor}, policy=ALL_OF, effect=ALLOW, scope=assignedLayoutEditor(u, sub) ∧ state(sub) = COPYEDITED, SoD=DSD(LayoutEditor≠Copyeditor trên *sub*) [tùy chọn], compliance=log(layout), checksumVersioning [tùy chọn]}
Proofreading	«rbac:P_N9»	{roles={Proofreader}, policy=ALL_OF, effect=ALLOW, scope=assignedProofreader(u, sub) ∧ state(sub) = LAYOUT_DONE, SoD=DSD(Proofreader≠LayoutEditor trên *sub*) [tùy chọn], compliance=log(proofApproval), readOnlyContent}

Đặc tả ở mức nút này cho phép cục bộ hóa kiểm soát truy cập tại các ranh giới thực thi hành vi, đồng thời vẫn bảo toàn sự tách biệt giữa DCSL, AGL và RBACDom.

Kết quả thực nghiệm. Để ngăn chặn các trạng thái không hợp lệ, hệ thống xác định một tập các bất biến đóng vai trò như một "lưới an toàn" toán học, bao trùm từ các ràng buộc kiểu dữ liệu cơ bản đến các chính sách DSD. Như một ví dụ tiêu biểu, quy tắc nghiệp vụ "Tác giả không được đảm nhận vai trò phản biện cho chính bài nộp của mình" được hình thức hóa bằng vị từ bắt buộc sau:

$$\begin{aligned} &\forall\, u, s \cdot (u \mapsto (s \mapsto r\_Author)) \in assignment \\ &\Rightarrow (u \mapsto (s \mapsto r\_Reviewer)) \notin assignment \end{aligned} \tag{6.1}$$

Tính đúng đắn của mô hình Event-B thu được được kiểm chứng bằng bộ kiểm tra mô hình ProB trong môi trường Rodin; kết quả kiểm chứng được minh họa trong Hình 6.20.



Hình 6.20: Kiểm chứng thực nghiệm bằng Rodin/ProB sử dụng hệ thống OJS làm ca nghiên cứu.

Như thể hiện trong Hình 6.20, quá trình kiểm chứng được trực quan hóa sử dụng ba vùng giao diện. Bảng điều khiển bên trái trình bày mối quan hệ cấu trúc giữa ngữ cảnh và máy Event-B, trong khi bảng bên phải ghi lại một vết thực thi cụ thể của một bài báo từ khởi tạo đến hoàn tất, cho thấy quy trình không rơi vào bế tắc. Bảng trung tâm cung cấp bằng chứng rõ ràng về tính an toàn: các chỉ báo "T" (True) màu xanh xác nhận rằng, trong suốt kịch bản thực thi, tất cả các ràng buộc bảo mật bao gồm cả quy tắc DSD trong Công thức (6.1) đều được bảo toàn. Mô hình Event-B được sinh ra, bao gồm các thành phần ngữ cảnh và máy, được nhập vào Rodin, nơi các nghĩa vụ chứng minh được tự động sinh và giải quyết nhằm xác nhận tính khả thi thực thi, các ràng buộc bảo mật và tính nhất quán xuyên mối quan tâm.

RQ1: *Làm thế nào có thể định nghĩa ngữ nghĩa vận hành hình thức cho các mô hình miền hợp nhất trong UDML?*

Kết quả cho thấy ngữ nghĩa vận hành của UDML có thể được xác định hình thức bằng cách diễn giải mô hình miền hợp nhất như một *hệ chuyển trạng thái thống nhất*, trong đó các mối quan tâm chia sẻ trạng thái hệ thống chung nhưng đóng góp các ràng buộc ngữ nghĩa riêng. AGL xác định các chuyển trạng thái hành vi, trong khi DCSL và RBACDom được diễn giải như các bất biến và điều kiện kích hoạt.

Ánh xạ UDML sang Event-B hiện thực hóa ngữ nghĩa này: các nút AGL được biên dịch thành các sự kiện Event-B, còn các ràng buộc cấu trúc và bảo mật thành bất biến và điều kiện cảnh báo. Do đó, mỗi phép thực thi UDML tương ứng với một chuỗi sự kiện Event-B bảo toàn các bất biến. Các thống kê trong Bảng 6.2 cung cấp bằng chứng rằng ngữ nghĩa vận hành này được xác định rõ ràng và có thể kiểm chứng bằng các cơ chế hình thức tiêu chuẩn.

Để đánh giá mức độ nỗ lực kiểm chứng, luận án phân tích phân bố các nghĩa vụ chứng minh theo trạng thái hoàn thành. Bảng 6.3 tóm tắt các kết quả được sinh ra bởi nền tảng Rodin.

Mức độ tự động hóa kiểm chứng. Tổng cộng có 381 nghĩa vụ chứng minh được sinh ra, trong đó 369 nghĩa vụ (96,8%) được tự động hoàn tất bởi các bộ chứng minh tích hợp trong Rodin, trong khi 12 nghĩa vụ còn lại (3,2%) yêu cầu một số bước chứng minh tương tác đơn giản.

Lưu ý rằng không phải tất cả các loại nghĩa vụ chứng minh trong Event-B đều xuất hiện trong ca nghiên cứu này. Cụ thể, các nghĩa vụ liên quan đến tinh chỉnh (FIS, SIM, WIT) và các nghĩa vụ dựa trên biến thể (VAR, NAT) không được sinh ra, do mô hình hiện tại không bao gồm các bước tinh chỉnh hoặc các sự kiện hội tụ. Do đó, Bảng 6.3 chỉ báo cáo các loại nghĩa vụ chứng minh thực sự được sinh ra bởi nền tảng Rodin.

RQ2: *Làm thế nào các mối quan tâm xuyên suốt động, tiêu biểu là RBAC, có thể được đặc tả dưới dạng các DSL dựa trên chú thích và được tích hợp vào các mô hình miền hợp nhất?*

Trong đó các chính sách ủy quyền, phân tách nhiệm vụ và ngữ nghĩa phiên được gắn lên các biên thực thi hành vi thay vì mã hóa trực tiếp trong mô hình hành vi. RBACDom được định nghĩa độc lập với AGL nhưng được tích hợp bằng các liên kết hình thức ở mức cú pháp trừu tượng, qua đó bảo toàn tính mô-đun của các mối quan tâm.

Bảng 6.2: Suy diễn thống kê kiểm chứng từ các tạo tác hình thức của OJS và RBACDom

Thàn phần mô hình	**Cơ sở hình thức**	**Bất biến**	**Sự kiện**	**Nghĩa vụ chứng minh**
Mô hình RBAC lõi	Các ràng buộc nhất quán vai trò quyền hạn, loại trừ lẫn nhau và hợp dạng RBAC được suy ra từ RBACDom, độc lập với luồng công việc OJS	14	8	96
Gán vai trò & ràng buộc SoD	Các ràng buộc phân tách nhiệm vụ tĩnh và động được suy ra từ đặc tả RBACDom cho các vai trò Tác giả, Phản biện, Biên tập viên và các vai trò sản xuất	11	6	78
Ủy quyền thời gian chạy	Ngữ nghĩa phiên và kích hoạt vai trò, bảo đảm chỉ các vai trò đã được gán và đang hoạt động mới có thể thực thi các thao tác được bảo vệ	9	5	64
Tích hợp chính sách RBAC vào mô hình OJS	Các bất biến liên kết các ràng buộc ủy quyền RBAC với các bước trong luồng công việc OJS và các trạng thái miền thông qua các chú thích RBACDom	12	7	102
Ràng buộc theo ngữ cảnh /trạng thái	Các ràng buộc về thứ tự luồng công việc và tiến hóa trạng thái, giới hạn các chuyển trạng thái miền hợp lệ trong vòng đời OJS	6	4	41
Tổng cộng		**52**	**30**	**381**

Các bất biến và sự kiện liên quan đến tích hợp chính sách RBAC vào mô hình OJS trong Bảng 6.2 cho thấy phần lớn nỗ lực kiểm chứng phát sinh từ việc ràng buộc ngữ nghĩa ủy quyền với hành vi và trạng thái miền, phản ánh bản chất động của RBAC trong các hệ thống hướng quy trình.

Bảng 6.3: Các nghĩa vụ chứng minh và trạng thái hoàn thành

Loại PO	Tổng	Tự động	Thủ công
WD	120	120	0
INV	140	132	8
GRD	80	78	2
THM	41	39	2
Tổng	**381**	**369**	**12**

Việc diễn giải RBACDom như các điều kiện bảo vệ trên sự kiện Event-B cho phép áp dụng nhất quán và kiểm chứng được các ràng buộc bảo mật mà không làm thay đổi cấu trúc hành vi.

Để đánh giá mức độ nỗ lực kiểm chứng, luận án phân tích phân bố các nghĩa vụ chứng minh theo trạng thái hoàn thành. Bảng 6.3 tóm tắt các kết quả được sinh ra bởi nền tảng Rodin.

Mức độ tự động hóa kiểm chứng. Tổng cộng có 381 nghĩa vụ chứng minh được sinh ra, trong đó 369 nghĩa vụ (96,8%) được tự động chứng minh bởi các bộ chứng minh tích hợp trong Rodin, trong khi 12 nghĩa vụ còn lại (3,2%) yêu cầu các bước chứng minh tương tác ở mức đơn giản. Phần lớn các nghĩa vụ được chứng minh tự động tương ứng với các điều kiện xác định tốt (WD) và tăng cường điều kiện bảo vệ (GRD), vốn được các bộ chứng minh tự động xử lý hiệu quả. Các nghĩa vụ cần chứng minh tương tác chủ yếu liên quan đến việc bảo toàn bất biến và chỉ yêu cầu các bước suy luận đơn giản, chẳng hạn như lựa chọn giả thuyết phù hợp hoặc khởi tạo các biến lượng hóa. Không cần sử dụng các chiến lược chứng minh phức tạp hay thay đổi cấu trúc mô hình; tất cả các nghĩa vụ còn lại đều được hoàn tất bằng bộ chứng minh mệnh đề tích hợp.

Lưu ý rằng không phải tất cả các loại nghĩa vụ chứng minh trong Event-B được liệt kê đều xuất hiện trong ca nghiên cứu này. Cụ thể, các nghĩa vụ liên quan đến tinh chỉnh (FIS, SIM, WIT) và các nghĩa vụ dựa trên biến thể (VAR, NAT) không được sinh ra, do mô hình hiện tại không bao gồm các bước tinh chỉnh hoặc các sự kiện hội tụ. Do đó, Bảng 6.3 chỉ báo cáo các loại nghĩa vụ chứng minh thực sự được sinh ra bởi nền tảng Rodin.

Các kết quả trên OJS xác nhận rằng UDML, khi hợp thành với AGL và RBACDom, hỗ trợ đặc tả các mô hình miền hợp nhất có khả năng thực thi

và ngữ nghĩa hình thức rõ ràng, trong đó AGL xác định sự tiến hóa hành vi còn RBACDom giới hạn khả năng thực thi sử dụng các điều kiện ủy quyền dựa trên tương ứng nút. Khối lượng nghĩa vụ chứng minh tuân theo các dạng chuẩn của Event-B, cho thấy phương pháp đề xuất có thể tái sử dụng trực tiếp các cơ chế kiểm chứng hiện có mà không cần mở rộng đặc thù, qua đó khẳng định UDML như một nền tảng ngữ nghĩa hình thức cho các mô hình miền hợp nhất có thể phân tích và kiểm chứng.

Thảo luận. Rodin được sử dụng trong bài báo này nhằm xác nhận khung ngữ nghĩa đề xuất, thay vì để đánh giá hiệu năng của công cụ. Cơ chế sinh và chứng minh tự động các nghĩa vụ chứng minh (PO) cho phép phát hiện sớm các bất nhất ngữ nghĩa và các vi phạm chính sách, những vấn đề vốn có thể bị ẩn trong các mô hình miền có khả năng thực thi.

Kết quả trên hệ thống OJS cho thấy các mô hình UDML được hợp thành với RBACDom có thể được diễn giải như các hệ chuyển trạng thái hợp nhất với khả năng thực thi và các thuộc tính bảo mật có thể phân tích hình thức. Trong ngữ cảnh này, AGL xác định sự tiến hóa hành vi của mô hình miền bằng cách đặc tả khi nào một nút có thể được thực thi theo ngữ nghĩa luồng điều khiển, trong khi RBACDom quyết định liệu nút đó có được phép thực thi hay không. Các thống kê kiểm chứng cho thấy nỗ lực kiểm chứng phần lớn được xử lý bởi các cơ chế chuẩn của Event-B, đạt được mức độ tự động hóa cao mà không cần các mở rộng kiểm chứng đặc thù.

Một khía cạnh thiết kế quan trọng khác là chiến lược kết hợp luật được sử dụng để đánh giá nhiều chú thích RBACDom gắn với cùng một nút hành vi. Trong khung được mở rộng, việc kết hợp luật được mô hình hóa như một tham số cấu hình ở mức mô hình sử dụng hàm *combiningAlg* : $DomainModelRbacDom \rightarrow CA$. Điều này cho phép đặc tả trực tiếp các ngữ nghĩa phân quyền khác nhau, bao gồm denyOverrides, permitOverrides và firstApplicable, ngay trong mô hình RBACDom, đồng thời vẫn bảo toàn cơ chế chuyển đổi UDML sang Event-B. Mở rộng này nâng cao tính tổng quát của khung phương pháp mà không ảnh hưởng đến ngữ nghĩa cấu trúc của DCSL hay ngữ nghĩa hành vi của AGL.

Hạn chế và khả năng sử dụng. Đánh giá hiện tại tập trung vào một quy trình có tính thực tế nhưng quy mô trung bình với tập vai trò cố định; việc khái quát rộng hơn đòi hỏi thêm các ca nghiên cứu bổ sung. Bên cạnh đó,

các lựa chọn mô hình hóa như mức độ phân rã của các nút và độ biểu đạt của các vị từ phạm vi có ảnh hưởng trực tiếp đến kích thước của mô hình Event-B được sinh ra cũng như số lượng nghĩa vụ chứng minh.

Hơn nữa, tỷ lệ cao các nghĩa vụ chứng minh được tự động hoàn tất trong ca nghiên cứu OJS (96,8%) cho thấy phần lớn các nghĩa vụ tương ứng với các dạng nghĩa vụ chuẩn của Event-B, vốn được các bộ chứng minh Rodin xử lý hiệu quả. Điều này cung cấp bằng chứng ban đầu cho thấy phương pháp có khả năng mở rộng mà không làm tăng tương ứng nỗ lực kiểm chứng thủ công. Tuy nhiên, việc đánh giá toàn diện trên các hệ thống công nghiệp quy mô lớn vẫn là hướng nghiên cứu trong tương lai.

Nguyên mẫu hiện tại sử dụng các đặc tả dựa trên chú thích để liên kết các chính sách RBACDom với các nút hành vi. Mặc dù cách biểu diễn này cho phép tích hợp trực tiếp với cơ sở mã hiện có, việc viết chú thích thủ công có thể trở nên dễ sai sót đối với các quy trình lớn như OJS. Do đó, các nghiên cứu tiếp theo sẽ tập trung phát triển các công cụ hỗ trợ theo hướng đồ họa và hướng mô hình cho RBACDom, cho phép đặc tả các chính sách bảo mật trực tiếp ở mức mô hình miền và tự động chuyển đổi sang các chú thích và hiện vật hình thức tương ứng. Các công cụ này sẽ giúp giảm công sức đặc tả thủ công và nâng cao khả năng sử dụng cho các hệ thống quy mô lớn.

6.3 Thực nghiệm và đánh giá

Phần này trình bày, thực nghiệm và đánh giá về các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền.

6.3.1 Kỹ thuật biểu diễn mô hình miền tích hợp ràng buộc

Trong phần này, luận án tiến hành đánh giá hai khía cạnh chính: (i) mức độ biểu đạt của CAP trong việc mô hình hóa các khái niệm miền, và (ii) khả năng áp dụng CAP trong thực tiễn phát triển phần mềm. Để đảm bảo tính khách quan và độ tin cậy của kết quả, luận án trình bày chi tiết phương pháp đánh giá cùng với các ca nghiên cứu được sử dụng trong quá trình thực nghiệm.

Phương pháp đánh giá. Để củng cố các kết quả và hạn chế tính chủ quan trong diễn giải, luận án tiến hành các thực nghiệm tái xây dựng có kiểm soát nhằm so sánh CAP với bốn nền tảng: DCSL [73], Apache Causeway [122], OpenXava [98], và Actifsource [1].

Các ca nghiên cứu. Đối với mỗi nền tảng, luận án tái xây dựng các mô hình miền tương đương về chức năng cho ba hệ thống nghiên cứu: CourseMan, ProcessMan, và OrderMan được lựa chọn nhằm đảm bảo tính đa dạng về miền, sự phong phú của các ràng buộc, cũng như mức độ trưởng thành khác nhau của mô hình hóa. CourseMan là một miền học thuật chuẩn, có nhiều ràng buộc cấu trúc và hành vi phong phú. ProcessMan cho phép so sánh có kiểm soát với các mô hình dựa trên DCSL trước đó nhằm đánh giá lợi ích gia tăng của CAP. OrderMan, được xây dựng dựa trên các kịch bản quy trình nghiệp vụ trong doanh nghiệp, đại diện cho một miền không đồng nhất, giàu luồng công việc, nhằm đánh giá khả năng mở rộng và tính tổng quát của phương pháp.

Bảng 6.4: Tổng hợp các ca nghiên cứu điển hình

Ca nghiên cứu	Loại miền	Số lượng ràng buộc OCL	Loại ràng buộc OCL theo ngữ nghĩa	Vai trò trong đánh giá
CourseMan	Học thuật	184	14	Mốc chuẩn về khả năng biểu đạt
ProcessMan	Quản lý quy trình	140	12	So sánh có kiểm soát với DCSL
OrderMan	Quy trình nghiệp vụ	76	11	Khả năng mở rộng và tổng quát

Bảng 6.4 tóm tắt các đặc trưng định lượng của ba hệ thống, bao gồm số lượng ràng buộc và các loại ràng buộc OCL theo ngữ nghĩa được sử dụng trong đánh giá.

Quy trình tái xây dựng có kiểm soát. Đối với mỗi ca nghiên cứu và mỗi nền tảng, luận án thực hiện các bước sau:

i. Tái xây dựng mô hình miền (biểu đồ lớp) bằng các cơ chế đặc tả gốc của từng nền tảng;

ii. Đặc tả các ràng buộc sao cho phù hợp nhất với đặc tả tham chiếu;

iii. Ghi nhận các chỉ số có thể đo lường: số lượng ràng buộc được hỗ trợ trực tiếp bởi chú thích hoặc các cấu trúc dựng sẵn; số lượng ràng buộc cần hiện thực thủ công; kích thước mã kiểm tra thủ công (LOC) cho mỗi ràng buộc; thời gian hiện thực; và mức độ hỗ trợ sinh tự động (sinh mã từ mô hình và sinh lô-gic kiểm tra).

iv. Kiểm chứng độc lập kết quả tái xây dựng bởi hai nhà nghiên cứu; các sai khác được xử lý thông qua việc rà soát lại các bản mẫu phần mềm và đạt được sự đồng thuận.

RQ3: Mức độ biểu đạt của CAPs trong việc mô hình hóa các khái niệm miền so với các phương pháp DDD hiện có là như thế nào?

Tính biểu đạt. Phân tích khả năng sử dụng chú thích để đặc tả và kiểm tra các khía cạnh của mô hình miền, từ đó xác định các cấu trúc ngôn ngữ và lượng hóa mức độ tương ứng. Mười hai tiêu chí đánh giá được rút ra từ khả năng biểu đạt của hệ thống chú thích và được áp dụng để đánh giá sử dụng ba mức: (i) Hỗ trợ đầy đủ, (ii) hỗ trợ một phần hoặc gián tiếp, (iii) Không hỗ trợ. Kết quả tổng hợp thể hiện trong Bảng 6.5.

Bảng 6.5: So sánh các công cụ dựa trên mức độ biểu đạt

Các công cụ sử dụng so sánh	**Hỗ trợ đầy đủ**	**Hỗ trợ một phần hoặc gián tiếp**	**Không hỗ trợ**
CAP	10/12	2/12	0
DCSL	4/12	5/12	3/12
OPenXAVA	2/12	7/12	3/12
Apache Causeway	2/12	7/12	3/12
Actifsource	9/12	3/12	0

Kết quả cho thấy: CAP đạt mức hài lòng 100%, tức toàn bộ 12/12 tiêu chí đều được hỗ trợ đầy đủ hoặc gián tiếp. Điều này chứng minh rằng các mô hình miền xây dựng bằng CAP có khả năng biểu đạt toàn bộ các ràng

buộc nghiệp vụ trong cùng một ngôn ngữ giao tiếp chung. Ngược lại, DCSL, OpenXAVA và Apache Causeway còn 25% tiêu chí không được hỗ trợ (3/12). Trong khi đó, Actifsource, do được thiết kế chủ yếu cho mô hình hóa cấu trúc và sinh mã, cũng đạt mức 100% hỗ trợ (đầy đủ hoặc gián tiếp) đối với các tiêu chí liên quan đến mô hình miền.

Mức độ mã hóa thủ công. Các tiêu chí này đánh giá mức độ lập trình thủ công cần thiết để hiện thực và kiểm tra các quy tắc trong mô hình miền, đặc biệt đối với các ràng buộc phức tạp trên năm tiêu chí như sau: (i) *Tự động hóa sinh mã:* Mức độ mà công cụ có thể tự động sinh mã, từ đó giảm thiểu nỗ lực lập trình thủ công. (ii) *Hỗ trợ ràng buộc phức tạp:* Khả năng xử lý các ràng buộc nâng cao (ví dụ: biểu thức OCL) mà không cần viết mã tùy chỉnh. (iii) *Tích hợp mô hình:* Mức độ dễ dàng khi tích hợp ràng buộc vào mô hình miền với ít điều chỉnh thủ công nhất. (iv) *Độ khó học:* Nỗ lực cần thiết để sử dụng công cụ hiệu quả; đường cong học tập càng cao thì nhu cầu lập trình thủ công càng tăng. (v) *Tính linh hoạt:* Khả năng tùy biến hoặc mở rộng lô-gic miền mà không cần nhiều mã bổ sung. Sử dụng ba mức đánh giá là:

Mức độ mã hóa thủ công. Đánh giá này tập trung vào các *ràng buộc OCL cấu trúc thiết yếu* được định nghĩa trong Bảng 2.1 (Mục 2.1.4, trang 24), trong đó tổng hợp tập các ràng buộc miền cốt lõi được xem xét trong nghiên cứu. Mục tiêu là định lượng mức độ công sức hiện thực thủ công cần thiết để thực thi các ràng buộc này khi sử dụng DCSL mở rộng với CAP so với các nền tảng khác.

Phạm vi đánh giá. Đối với mỗi trong ba ca nghiên cứu (CourseMan, ProcessMan, và OrderMan), luận án tái xây dựng các mô hình miền tương đương và hiện thực cùng một tập các loại ràng buộc OCL thiết yếu, bao gồm: (i) các ràng buộc giá trị thuộc tính; (ii) các ràng buộc bội số và lực lượng; (iii) các ràng buộc phụ thuộc; (iv) các ràng buộc dựa trên phép tổng hợp (ví dụ: tổng, đếm); và (v) các bất biến cấu trúc liên kết chéo.

Chỉ các ràng buộc thiết yếu được đưa vào đo lường, vì chúng đại diện cho các quy tắc toàn vẹn cốt lõi của miền mà CAP hướng tới. Các ràng buộc phụ trợ và ở mức khung làm việc được loại trừ do thường được xử lý tự động và không phản ánh chính xác mức giảm công sức mô hình hóa thủ công.

Quy trình đo lường. Đối với mỗi nền tảng và mỗi thể hiện ràng buộc thiết yếu, luận án ghi nhận: (i) liệu ràng buộc có thể được thực thi theo cách khai báo mà không cần mã kiểm tra thủ công hay không; (ii) số lượng phương thức kiểm tra thủ công cần thiết; (iii) số dòng mã kiểm tra thủ công (LOC), không bao gồm chú thích và dòng trống; và (iv) số lượng lớp validator/service bổ sung được tạo ra chỉ để thực thi ràng buộc.

Lưu ý rằng các khai báo lớp cấu trúc không được tính là mã kiểm tra, vì chúng thuộc đặc tả mô hình miền chứ không phải lô-gic kiểm tra ràng buộc là trọng tâm của CAP.

Bảng 6.6: Mức độ mã hóa thủ công các ràng buộc OCL thiết yếu

Các công cụ sử dụng so sánh	**LOC thủ công trung bình/ràng buộc**	**Tỷ lệ % được tự động thực thi**	**Số lớp validator bổ sung**
CAP-extended DCSL	6.2	92%	0
Apache Causeway	54.3	22%	6
OpenXava	37.5	45%	4
Actifsource	31.8	52%	3

Bảng 6.6 tóm tắt các kết quả tổng hợp trung bình trên ba ca nghiên cứu. DCSL mở rộng với CAP đạt tỷ lệ tự động hóa cao nhất (92%) đối với các ràng buộc OCL thiết yếu và yêu cầu số dòng mã kiểm tra thủ công trung bình thấp nhất (6.2 LOC trên mỗi ràng buộc), đồng thời không cần bổ sung các lớp kiểm tra ràng buộc. Kết quả này cho thấy các bất biến thiết yếu được đặc tả theo cách khai báo sử dụng chú thích CAP và được tái tạo tự động thành lô-gic kiểm tra OCL có thể thực thi. DCSL gốc cung cấp hỗ trợ một phần cho các ràng buộc thiết yếu nhưng vẫn yêu cầu các phương thức thủ công đối với các ràng buộc dựa trên tổng hợp và phụ thuộc, dẫn đến gia tăng LOC. Trong Apache Causeway và OpenXava, các ràng buộc cấu trúc phức tạp thường được hiện thực theo kiểu mệnh lệnh, dẫn đến số LOC thủ công cao hơn đáng kể và cần thêm các lớp kiểm tra ràng buộc/dịch vụ. Actifsource hỗ trợ cấu hình luật ở mức mô hình; tuy nhiên, các ràng buộc liên kết chéo và dựa trên tổng hợp thường đòi hỏi thêm các tảo tác phần mềm hoặc cấu hình bổ sung.

Diễn giải. Các kết quả cho thấy việc tích hợp CAP vào DCSL giúp giảm đáng kể công sức lập trình thủ công trong việc thực thi các ràng buộc OCL thiết yếu ở mức mô hình miền. Mức giảm này phản ánh công sức thực thi trên mỗi ràng buộc, thay vì việc dịch chuyển lô-gic tương đương, do ngữ nghĩa OCL được định nghĩa một lần trong các khuôn mẫu CAP có thể tái sử dụng, thay vì phải hiện thực lại cho từng trường hợp cụ thể. Tất cả các phép đo được thực hiện dựa trên các thực nghiệm tái xây dựng có kiểm soát, sử dụng cùng một tập ràng buộc thiết yếu trên các nền tảng.

RQ4: CAP có thể được áp dụng ở mức độ nào trong thực tiễn phát triển phần mềm?

Để đánh giá khả năng áp dụng của CAP trong thực tế, thực hiện triển khai và kiểm thử các mẫu ràng buộc trên ba hệ thống CourseMan, ProcessMan và OrderMan. Kết quả được tổng hợp trong Bảng 6.7, bao gồm mười bốn nhóm ràng buộc OCL. Được đề xuất một danh mục CAPs, bao gồm nhiều mẫu cung cấp một khuôn khổ cú pháp và ngữ nghĩa hình thức để đặc tả các ràng buộc nghiệp vụ trong OCL, bao gồm: tính hợp lệ cấu trúc, giới hạn định lượng, quan hệ phụ thuộc, v.v.

Tuy nhiên, ba nhóm ràng buộc mang tính hành vi sau đây được loại khỏi danh mục CAPs: (i) tiền điều kiện và hậu điều kiện, vốn đòi hỏi ngữ nghĩa tác vụ và suy luận trên các trạng thái trước và sau; (ii) kiểm soát truy cập, phụ thuộc vai trò và chính sách nền tảng; (iii) phản ứng tự động, liên quan đến ràng buộc sự kiện, tác vụ và thứ tự thực thi.

Ba nhóm này mang tính ràng buộc theo ngữ cảnh hơn là các bất biến cấu trúc, ít có khả năng tái sử dụng như các mẫu bất biến và phụ thuộc mạnh vào kiến trúc triển khai; vì vậy chúng được loại bỏ để giữ cho danh mục CAPs súc tích và mang tính đại diện.

Các thí nghiệm trên ba miền bài toán CourseMan, OrderMan và ProcessMan cho thấy rằng các CAPs được đề xuất có thể được biểu diễn bằng DSL dựa trên chú thích như DCSL và được tích hợp vào mô hình miền thống nhất nhằm hỗ trợ sinh tự động và tạo nguyên mẫu phần mềm.

Trong nghiên cứu tình huống CourseMan, 184 ràng buộc được khảo sát và phân loại vào 14 nhóm. CAPs biểu diễn thành công 172 ràng buộc thuộc 11 nhóm; 12 ràng buộc còn lại nằm ngoài phạm vi hiện tại do phụ thuộc vào trạng thái tại thời gian chạy, sự kiện hoặc điều kiện kích hoạt theo ngữ

Bảng 6.7: Các nhóm ràng buộc và các mẫu CAP

Nhóm ràng buộc	**Danh mục CAP**	**Khả năng áp dụng CAP**		
		CourseMan	OrderMan	ProcessMan
Ràng buộc tính hợp lệ (well-formedness)	DCSL trong Bảng 2.1	11/11	11/11	11/11
Giới hạn định lượng	SumConstraint	19/19	4/4	7/7
Ràng buộc phụ thuộc và tiên quyết	PrerequisiteConstraint	10/10	12/12	13/13
Ràng buộc lập lịch và xung đột	ScheduleConstraint	17/17	6/6	10/10
Ràng buộc điều kiện đủ của thực thể	EligibilityConstraint	20/20	5/5	12/12
Quy tắc thi lại / thực hiện lại	RetakeConstraint	8/8	2/2	8/8
Ràng buộc sức chứa	SizeConstraint	17/17	3/3	15/15
Ràng buộc thời gian và hạn chót	TimeConstraint	10/10	6/6	13/13
Quy tắc tính toán và dẫn xuất	SumProduct	15/15	8/8	8/8
Ràng buộc dựa trên trạng thái	StatusConstraint	12/12	6/6	15/15
Ràng buộc cấu trúc tổ chức	StructuralConstraint	29/29	6/6	16/16
Ràng buộc tiền điều kiện / hậu điều kiện	×	0/4	0/4	0/4
Ràng buộc kiểm soát truy cập	×	0/4	0/2	0/4
Ràng buộc phản ứng tự động	×	0/4	0/2	0/4
	Tổng	**172/184**	**66/76**	**128/140**

cảnh. Các ràng buộc này chủ yếu thuộc nhóm tiền điều kiện/hậu điều kiện, kiểm soát truy cập và phản ứng tự động. Tổng tỷ lệ hỗ trợ đạt 93.5%.

Đối với OrderMan, 76 ràng buộc được phân tích, trải rộng từ ràng buộc cấu trúc đến các quy tắc nghiệp vụ và hành vi phức tạp. Trong số này, 66 ràng buộc được hỗ trợ bởi CAPs; các ràng buộc còn lại có đặc điểm động tương tự CourseMan, dẫn đến tỷ lệ áp dụng 86.8%.

Đánh giá trên ProcessMan cũng cho kết quả tương đồng: trong số 140 ràng buộc được khảo sát, 128 được biểu diễn hiệu quả bằng CAPs, số còn lại liên quan tới hành vi động hoặc cơ chế kích hoạt theo sự kiện vốn nằm ngoài khả năng hiện tại của khuôn khổ, đạt tỷ lệ áp dụng 91.4%.

Bên cạnh tính biểu đạt và khả năng áp dụng, tính khả thi của phương pháp được khẳng định bằng việc sinh và thực thi các nguyên mẫu phần mềm

trực tiếp từ mô hình miền thống nhất có tích hợp CAPs. Điều này chứng minh rằng phương pháp được đề xuất có thể được hiện thực một cách trơn tru dưới dạng các hiện vật phần mềm vận hành thực tế.

Thảo luận. Các mối đe dọa đối với tính hợp lệ của phương pháp đề xuất và quá trình đánh giá, theo phân loại của Runeson et al. [107].

Nghiên cứu giả định rằng các yêu cầu miền được thu thập và diễn giải đầy đủ, chính xác. Nguy cơ diễn giải sai được giảm thiểu thông qua việc biểu diễn tường minh mô hình lớp miền, các ràng buộc OCL và các chú thích tham số hóa, sau đó đóng gói chúng trong các mẫu CAP để hợp nhất vào mô hình miền hợp nhất có khả năng thực thi theo DDD.

Các mối đe dọa chính xuất phát từ việc chuyển đổi đặc tả UML/OCL sang các mẫu CAP và từ khả năng bao phủ chưa đầy đủ của các ràng buộc OCL. Luận án giảm thiểu các rủi ro này bằng cách cung cấp hướng dẫn áp dụng CAP, đánh giá trên nhiều hệ thống có độ phức tạp khác nhau (CourseMan, OrderMan, ProcessMan), và kiểm thử, rà soát kỹ lưỡng công cụ hiện thực trong khung phần mềm JDA.

6.3.2 Kỹ thuật biểu diễn mô hình miền tích hợp hành vi

Luận án sử dụng ba ứng dụng phản ánh yêu cầu thực tế: CourseMan, ProcessMan và OrderMan để đánh giá tính biểu đạt và khả năng áp dụng của kỹ thuật biểu diễn mô hình miền tích hợp hành vi.

Tính biểu đạt được đánh giá bằng cách đối chiếu AGL tích hợp DCSL với các mẫu DDD [41] gọi là AGL$^+$ và so sánh với các khung làm việc DDD dựa trên chú thích: AL [122] và XL [98]. Việc xem xét mô hình hóa cấu trúc (`DomainClass`, `DomainField`, `AssociativeField`, `DomainMethod`), mô hình hoá hành vi, và mức độ mã hoá cần thiết (RCL).

RQ5: *Hiệu quả biểu diễn mô hình miền so với DDD hiện có?*

Bảng 6.8(A) cho thấy AGL$^+$ và các phương pháp DDD khác đều hiện thực một phần các mẫu DDD. Bảng 6.8(B) cho thấy AGL$^+$ có tính biểu đạt cao nhất trong cả cấu trúc lẫn hành vi. AL và XL chỉ hỗ trợ một phần cấu trúc và không hỗ trợ hành vi. AD mô tả hành vi tốt nhưng thiếu gắn kết với mô hình cấu trúc.

Bảng 6.8: (A–trái) Tiêu chí 1: Các tiêu chí về tính biểu đạt dựa trên các mẫu DDD; (B–phải) Tiêu chí 2: Các tiêu chí về tính biểu đạt dựa trên các siêu khái niệm của miền

	Tiêu chí 1					**Tiêu chí 2**			
	AGL+	AL	XL	AD		AGL+	AL	XL	AD
DomainClass (DC)	√	√	√	×	DC	**1/1**	1/1	0/1	×
DomainField (DF)	√	√	√	×	DF	**8/8**	4/8	5/8	×
AssociativeField (AF)	√	√	√	×	AF	**7/7**	0/7	1/7	×
DomainMethod (DM)	√	√	√	×	DM	√	×	×	×
ActivityDomainClass (ADC)	√	×	×	√	ADC	√	×	×	√

RQ6: *Nỗ lực cần thiết để định nghĩa mô hình miền thống nhất?*

Từ Figure 3.8 và 3.9, ta có: Bảng 6.9 trình bày số lượng các mẫu hành vi miền được sử dụng (được định nghĩa) để xây dựng các ứng dụng CourseMan, ProcessMan và OrderMan. Trong đó, số lượng mô-đun phản ánh mức độ phức tạp của phần mềm, còn số lượng các mẫu hành vi miền thể hiện mức độ áp dụng của bộ mẫu đầu tiên cũng như lượng công sức được giảm bớt cho lập trình viên trong việc hiện thực phần mềm thủ công.

Bảng 6.9: Thống kê các mẫu hành vi và mô-đun cho CourseMan, ProcessMan và OrderMan

	Mẫu	Mô-đun
CourseMan	3	6
ProcessMan	5	37
OrderMan	5	13

Thảo luận. Kết quả đánh giá cho thấy phương pháp dựa trên AGLD-CSL có khả năng tích hợp hiệu quả vào các quy trình phát triển lặp và linh hoạt, trong đó chuyên gia miền và lập trình viên có thể cộng tác xây dựng mô hình miền hợp nhất và sinh phần mềm trực tiếp từ mô hình. Việc kết hợp với các kỹ thuật kỹ nghệ phần mềm hướng mô hình, đặc biệt là ánh xạ mô hình yêu cầu và mô hình miền trừu tượng sang mô hình miền hợp nhất phụ thuộc nền tảng (AGL$^+$/DCSL trên Java), qua đó thu hẹp khoảng cách từ mô hình đến hiện thực thực thi.

Tính khả dụng của phương pháp chịu ảnh hưởng đáng kể bởi giao diện GUI, vốn được thiết kế đơn giản và nhất quán, phản ánh trực tiếp cấu trúc mô hình miền và hỗ trợ hiệu quả tương tác với chuyên gia miền. Bên cạnh đó, việc AGL được nhúng trong ngôn ngữ lập trình hướng đối tượng cho phép tận dụng các cơ chế kiểm tra tĩnh, tái cấu trúc và hỗ trợ IDE, qua đó nâng cao khả năng xây dựng và bảo trì đặc tả.

Cuối cùng, dựa trên các mẫu hành vi miền và kiến trúc mô-đun độc lập ngôn ngữ, phương pháp có tính tổng quát cao và có thể được triển khai trên các nền tảng khác ngoài Java hoặc sử dụng qua các DSL ngoại sinh sử dụng các khung ngôn ngữ hiện có. Điều này cho thấy tiềm năng mở rộng và khả năng áp dụng của phương pháp trong các bối cảnh phát triển phần mềm đa dạng.

6.3.3 Kỹ thuật tích hợp các DSL theo mối quan tâm vào mô hình miền

Trong phần này, thực hiện đánh giá hiệu quả của phương pháp tích hợp các mối quan tâm vào mô hình miền hợp nhất UDML thông qua các ca nghiên cứu đã được cài đặt và thực nghiệm thành công. Việc đánh giá lần lượt trả lời các câu hỏi nghiên cứu RQ7–RQ10.

RQ7: *Làm thế nào có thể tích hợp một mối quan tâm nền tảng vào mô hình miền?*

UDML trả lời câu hỏi nghiên cứu **RQ7** bằng cơ chế tích hợp dựa trên siêu mô hình, trong đó mỗi mối quan tâm được mô hình hóa và quản lý một cách độc lập, đồng thời được gắn vào mô hình miền bằng các siêu khái niệm mở rộng: `DomainModel`, `Concern`, `Annotation` và `Annotable`. Cách tổ chức này cho phép (i) gom nhóm các chú thích của một mối quan tâm trong một `Concern` riêng; (ii) gắn các chú thích đó lên các phần tử miền lõi (gói, lớp, thuộc tính, kết hợp, thao tác) thông qua `Annotable`; và (iii) duy trì ranh giới rõ ràng giữa miền lõi và miền mối quan tâm nhưng vẫn đảm bảo khả năng tích hợp mối quan tâm vào mô hình miền ở đúng vị trí cần thiết. Qua các ca nghiên cứu đã triển khai, việc tích hợp mối quan tâm theo cơ chế này cho thấy mô hình UDML sau hợp nhất vẫn giữ được cấu

trúc miền, đồng thời bổ sung được thông tin theo mối quan tâm dưới dạng chú thích mà không làm biến dạng mô hình miền lõi.

RQ8. Làm thế nào có thể biểu diễn miền của mối quan tâm như một aDSL?

Để biểu diễn miền của mối quan tâm như một aDSL bằng cách cách thiết kế miền mối quan tâm theo hai mức: (i) một DSL ngoại sinh DSL_c có siêu mô hình riêng để biểu diễn cú pháp trừu tượng và ràng buộc miền của mối quan tâm; và (ii) một aDSL tương ứng (DSL nội sinh dựa trên chú thích) được xây dựng bằng ánh xạ một–một từ DSL_c. Cách tiếp cận này bảo đảm đồng thời hai yêu cầu: *tính hình thức* ở mức siêu mô hình (dễ kiểm tra hợp dạng, dễ phân tích và chuyển đổi trong MDE) và *tính khả thi* ở mức hiện thực (aDSL nhúng trực tiếp trong OOPL để tham gia sinh mã/thi hành). Các ca thực nghiệm cho thấy cơ chế ánh xạ song ánh giữa mô hình DSL ngoại sinh và mô hình aDSL cho phép duy trì nhất quán giữa mô hình hóa và hiện thực, đồng thời tạo tuyến chuyển đổi M2M/M2T rõ ràng để đưa mối quan tâm vào chuỗi sinh phần mềm.

RQ9: *Làm thế nào các DSL theo mối quan tâm có thể được hợp nhất có hệ thống vào một AST hợp nhất?*

Bằng cơ chế hợp nhất dựa trên AST, mỗi DSL theo mối quan tâm được chuẩn hóa dưới dạng bộ ba (AS, CS, Sem) và được tích hợp tăng dần vào UDML lõi áp dụng thuật toán hợp nhất cây. Ở mức cú pháp trừu tượng, việc hợp nhất bao gồm mở rộng hoặc kế thừa khái niệm lõi, bổ sung thuộc tính và ràng buộc, và thiết lập các liên kết xuyên mối quan tâm qua các cạnh tham chiếu (`refCons`). Ở mức cú pháp cụ thể, các ký pháp khác nhau được ánh xạ về cùng một AST nhằm bảo đảm đồng bộ đa góc nhìn. Ở mức ngữ nghĩa, mỗi DSL đóng góp một ánh xạ mô-đun và được tổng hợp theo cấu trúc AST hợp nhất.

R10: *Những cơ chế hợp thành ngôn ngữ và phương pháp có hỗ trợ công cụ nào phù hợp để tích hợp và tiến hóa các mối quan tâm không đồng nhất trong mô hình miền mô-đun, mở rộng?*

Kết quả thực nghiệm cho thấy hai hướng tiếp cận bổ trợ lẫn nhau là phù hợp cho UDML. Thứ nhất, hợp thành dựa trên siêu mô hình và chuyển đổi mô hình (MDE) phù hợp cho việc định nghĩa chính xác miền mối quan tâm,

kiểm tra hợp dạng và xây dựng bộ chuyển đổi M2M/M2T nhằm đưa mô hình về dạng thực thi; các công cụ như ATL và Acceleo hỗ trợ hiệu quả việc duy trì ánh xạ giữa DSL ngoại sinh và aDSL, cũng như tiến hóa mô hình thông qua cập nhật luật chuyển đổi. Thứ hai, hợp thành dựa trên AST và chú thích phù hợp khi cần tích hợp các DSL không đồng nhất và mở rộng theo vòng đời, trong đó AST đóng vai trò cấu trúc trung tâm của mô hình hợp nhất.

Đối với các mối quan tâm nhạy cảm như bảo mật, việc sử dụng Event-B như một hậu-end kiểm chứng cung cấp cơ chế kiểm chứng hình thức cho mô hình UDML hợp nhất, cho phép phát hiện xung đột và sai lệch chính sách ở mức ngữ nghĩa.

Thảo luận. Việc áp dụng phương pháp đề xuất trong Mục 4.2 cho các miền bài toán thực tế cho thấy tính khả thi và hiệu quả, song vẫn tồn tại một số mối đe dọa đến tính hợp lệ.

Thứ nhất, phương pháp phụ thuộc vào năng lực biểu đạt của các miền mối quan tâm và khả năng tích hợp chúng vào một mô hình miền hợp nhất có thể thực thi; do đó, UDMM cần tiếp tục được mở rộng để bao quát thêm các mối quan tâm khác.

Thứ hai, tính đúng đắn của mô hình UDM phụ thuộc vào độ chính xác của các phép chuyển đổi từ cú pháp trừu tượng sang cú pháp cụ thể, hiện được đảm bảo thông qua rà soát và kiểm thử các luật chuyển đổi ATL và Acceleo.

Cuối cùng, mã nguồn sinh tự động bằng Java có thể chịu ảnh hưởng từ sai sót trong khung sinh mã; vì vậy, việc rà soát và kiểm thử mã nguồn tiếp tục được thực hiện nhằm nâng cao độ tin cậy của kết quả sinh tự động.

Bên cạnh các kết quả đạt được, một hướng mở rộng tiềm năng được gợi mở từ phân tích ở Mục 2.5 để khai thác các kỹ thuật AI/LLM nhằm hỗ trợ giai đoạn khởi tạo mô hình miền, đặc biệt trong việc sinh nháp đặc tả hoặc đề xuất các thành phần mô hình từ mô tả nghiệp vụ. Cách tiếp cận này có thể góp phần gia tăng mức độ tự động hóa ở bước đầu của quá trình mô hình hóa và giảm chi phí xây dựng mô hình ban đầu. Tuy nhiên, khác với các phép biểu diễn và chuyển đổi có đặc tả tường minh được đề xuất trong luận án, các kết quả do AI/LLM sinh ra mang tính suy diễn và chưa được bảo đảm về ngữ nghĩa. Vì vậy, các đặc tả được sinh ra cần được chuẩn hóa

và biểu diễn lại trong các DSL của luận án, sau đó tích hợp vào mô hình miền hợp nhất và kiểm chứng bằng các cơ chế hình thức đã đề xuất, nhằm bảo đảm tính nhất quán và đúng đắn của mô hình. Hướng nghiên cứu này sẽ được tiếp tục thảo luận trong Chương 7.

6.4 Tổng kết chương

Trong chương này, luận án đã xây dựng và triển khai các công cụ hỗ trợ cho các phương pháp đề xuất, bao gồm kỹ thuật biểu diễn mô hình miền có khả năng thực thi, kỹ thuật tích hợp hành vi và ràng buộc vào mô hình miền hợp nhất, cũng như kỹ thuật sinh tự động bản mẫu phần mềm theo thiết kế hướng miền. Các công cụ này hiện thực hóa chuỗi xử lý hoàn chỉnh từ đặc tả mô hình đầu vào đến các tạo tác phần mềm đầu ra, qua đó minh chứng tính khả thi của các phương pháp đề xuất. Toàn bộ quá trình mô hình hóa, chuyển đổi và sinh mã được đánh giá thông qua các ca nghiên cứu và thực nghiệm nhằm đảm bảo tính đúng đắn, tính nhất quán và khả năng ứng dụng trong thực tiễn phát triển phần mềm.

Chương 7

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Luận án đã nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền nhằm thu hẹp khoảng cách giữa mô hình miền và phần mềm thực thi. Trên cơ sở phân tích những hạn chế của các hướng tiếp cận hiện có trong biểu diễn mô hình miền, tích hợp các mối quan tâm và tự động hóa phát triển phần mềm, luận án đã đề xuất một hướng tiếp cận thống nhất cho phép biểu diễn mô hình miền không chỉ phản ánh cấu trúc của miền nghiệp vụ mà còn tích hợp các khía cạnh ràng buộc, hành vi và bảo mật trong cùng một khuôn khổ.

Điểm cốt lõi của hướng tiếp cận đề xuất là việc xem mô hình miền như một tạo tác trung tâm trong toàn bộ quá trình phát triển phần mềm theo thiết kế hướng miền. Thay vì đặc tả các khía cạnh khác nhau của hệ thống bằng các DSL theo từng mối quan tâm một cách tách biệt, luận án xây dựng một mô hình miền hợp nhất, trong đó các đặc tả về cấu trúc, ràng buộc, hành vi và bảo mật được tích hợp có hệ thống trong ngôn ngữ mô hình miền hợp nhất UDML. Trên cơ sở đó, mô hình miền không chỉ đóng vai trò là phương tiện giao tiếp giữa chuyên gia miền và kỹ sư phần mềm, mà còn trở thành một tạo tác có khả năng thực thi, được kiểm chứng hình thức, chuyển đổi và sử dụng trực tiếp trong quá trình sinh phần mềm.

Bên cạnh việc đề xuất các kỹ thuật biểu diễn mô hình miền, luận án còn xây dựng các bộ chuyển đổi mô hình cho phép từng bước đưa đặc tả yêu cầu, mô hình miền hợp nhất và các DSL theo mối quan tâm đến các tạo tác

thực thi. Các bộ chuyển đổi này góp phần nâng cao mức độ tự động hóa trong phát triển phần mềm, đồng thời hỗ trợ duy trì tính nhất quán giữa các mức trừu tượng khác nhau của hệ thống. Trên nền tảng đó, các công cụ hỗ trợ đã được xây dựng nhằm hiện thực hóa các kỹ thuật đề xuất và đánh giá tính khả thi của phương pháp thông qua các ca nghiên cứu thực tế.

Từ góc độ khoa học, các kết quả của luận án góp phần làm rõ khả năng kết hợp giữa các nguyên lý của thiết kế hướng miền, kỹ nghệ phần mềm hướng mô hình và các ngôn ngữ chuyên biệt miền. Đặc biệt, việc đề xuất ngôn ngữ UDML cho thấy khả năng xây dựng một mô hình miền hợp nhất có thể đóng vai trò như một tầng gian giàu ngữ nghĩa giữa yêu cầu và phần mềm thực thi. Mô hình này không chỉ hỗ trợ đặc tả các khía cạnh khác nhau của miền trong một biểu diễn thống nhất mà còn tạo nền tảng cho các hoạt động kiểm chứng, chuyển đổi mô hình và sinh tự động phần mềm.

Từ góc độ ứng dụng, hướng tiếp cận đề xuất có tiềm năng áp dụng cho các hệ thống phần mềm có miền nghiệp vụ phức tạp, nhiều quy tắc nghiệp vụ và yêu cầu mức độ nhất quán cao giữa thiết kế và hiện thực. Bằng việc tích hợp các khía cạnh cấu trúc, hành vi, ràng buộc và bảo mật trong một mô hình miền hợp nhất, UDML góp phần giảm sự phân tán tri thức, nâng cao tính nhất quán của hệ thống và hỗ trợ tái sử dụng các tạo tác mô hình trong quá trình phát triển phần mềm. Bên cạnh đó, với cách tiếp cận dựa trên tích hợp các DSL theo mối quan tâm, UDML có khả năng mở rộng để hỗ trợ các khía cạnh mới như giao diện người dùng, luồng tương tác, quyền riêng tư, tuân thủ quy định và các yêu cầu phi chức năng khác. Điều này cho thấy UDML không chỉ là một giải pháp cho phạm vi nghiên cứu của luận án mà còn có tiềm năng trở thành nền tảng cho việc xây dựng các mô hình miền có khả năng thực thi trong nhiều miền ứng dụng khác nhau.

Trên cơ sở các kết quả đạt được, phần còn lại của chương này trình bày các đóng góp chính của luận án và thảo luận các hướng phát triển tiếp theo nhằm tiếp tục hoàn thiện cơ sở lý thuyết, mở rộng khả năng biểu diễn của mô hình miền hợp nhất, nâng cao hiệu quả của các kỹ thuật chuyển đổi mô hình và tăng cường khả năng áp dụng của phương pháp trong thực tiễn phát triển phần mềm.

7.1 Các kết quả đạt được

Sau quá trình nghiên cứu và giải quyết bài toán đặt ra, luận án đã đạt được các kết quả phù hợp với mục tiêu nghiên cứu ban đầu, đồng thời góp phần làm rõ hướng tiếp cận biểu diễn và chuyển đổi mô hình trong thiết kế hướng miền. Các kết quả này tập trung vào việc nâng cao khả năng biểu diễn mô hình miền, hợp nhất các mối quan tâm khác nhau trong một khuôn khổ thống nhất, và hỗ trợ tự động hóa quá trình sinh phần mềm từ mô hình miền. Cụ thể như sau:

i. **Về kỹ thuật biểu diễn mô hình miền.** Luận án đã đề xuất các kỹ thuật biểu diễn mô hình miền theo hướng có khả năng thực thi, cho phép đặc tả một cách tường minh và nhất quán các khía cạnh cấu trúc, ràng buộc và hành vi của miền bài toán. Đối với ràng buộc nghiệp vụ, luận án đề xuất kỹ thuật tích hợp các ràng buộc OCL phức tạp vào mô hình miền bằng các mẫu chú thích ràng buộc CAP, qua đó mở rộng khả năng biểu diễn của mô hình miền và tạo điều kiện cho việc tái sử dụng, kiểm tra và xử lý tự động các ràng buộc nghiệp vụ. Đối với hành vi, luận án đề xuất kỹ thuật tích hợp hành vi miền thông qua ngôn ngữ AGL, cho phép gắn trực tiếp các đặc tả hành vi vào mô hình miền, làm cho mô hình không chỉ phản ánh cấu trúc tĩnh mà còn thể hiện được lô-gic vận hành của hệ thống.

ii. **Về kỹ thuật hợp nhất các mối quan tâm vào mô hình miền hợp nhất.** Trên cơ sở các kỹ thuật biểu diễn đã xây dựng, luận án đã đề xuất phương pháp hợp nhất các DSL theo các mối quan tâm khác nhau vào một mô hình miền hợp nhất có khả năng thực thi. Kết quả này góp phần giải quyết tình trạng phân tán mô hình khi các khía cạnh cấu trúc, hành vi, ràng buộc và bảo mật thường được đặc tả bằng các ngôn ngữ khác nhau. Luận án không chỉ xây dựng mô hình miền hợp nhất ở mức cú pháp mà còn thiết lập nền tảng ngữ nghĩa thực thi cho mô hình này, từ đó hỗ trợ kiểm chứng hình thức thông qua chuyển đổi mô hình UDML sang môi trường Event-B, nhằm bảo đảm tính nhất quán và tính đúng đắn của mô hình miền hợp nhất.

iii. **Về kỹ thuật chuyển đổi mô hình và sinh tự động bản mẫu phần mềm.** Luận án đã đề xuất các kỹ thuật chuyển đổi mô hình nhằm thao tác trên mô hình miền hợp nhất và từng bước đưa mô hình

đến phần mềm thực thi. Trên cơ sở đó, luận án xây dựng bộ chuyển đổi từ mô hình yêu cầu sang mô hình miền hợp nhất, cũng như các phép chuyển đổi tiếp theo để sinh đặc tả miền thực thi và mã nguồn bản mẫu phần mềm. Kết quả này cho thấy cách tiếp cận đề xuất có khả năng thu hẹp khoảng cách giữa đặc tả yêu cầu, mô hình miền và phần mềm thực thi, đồng thời góp phần nâng cao mức độ tự động hóa trong phát triển phần mềm theo thiết kế hướng miền.

Bên cạnh các kết quả cụ thể, luận án cho thấy tính khả thi của việc kết hợp các nguyên lý của thiết kế hướng miền với kỹ nghệ phần mềm hướng mô hình trong một khuôn khổ thống nhất. Trong đó, UDML đóng vai trò là mô hình miền hợp nhất trung tâm, hỗ trợ tích hợp các mối quan tâm khác nhau của hệ thống và làm cơ sở cho kiểm chứng hình thức, chuyển đổi mô hình và sinh tự động phần mềm.

Các kết quả nghiên cứu cũng cho thấy khả năng mở rộng của phương pháp đề xuất. Với cách tiếp cận dựa trên tích hợp các DSL theo mối quan tâm, UDML có thể tiếp tục được mở rộng để hỗ trợ các khía cạnh mới như giao diện người dùng, quyền riêng tư dữ liệu, tuân thủ quy định và các yêu cầu phi chức năng, qua đó nâng cao khả năng thích nghi của phương pháp trong nhiều miền ứng dụng khác nhau.

Từ góc độ thực tiễn, các kỹ thuật được đề xuất góp phần thu hẹp khoảng cách giữa mô hình và hiện thực, nâng cao tính nhất quán giữa các mức mô hình và phần mềm thực thi, đồng thời hỗ trợ tái sử dụng các tạo tác mô hình trong quá trình phát triển phần mềm.

7.2 Hướng phát triển tiếp theo

Mặc dù các kết quả đạt được cho thấy tính khả thi và hiệu quả của các kỹ thuật được đề xuất, nghiên cứu vẫn còn một số hạn chế nhất định về phạm vi biểu diễn, mức độ tự động hóa và khả năng hỗ trợ người dùng trong quá trình mô hình hóa miền. Trên cơ sở đó, một số hướng phát triển tiếp theo được xác định nhằm tiếp tục hoàn thiện cơ sở lý thuyết, mở rộng khả năng biểu diễn của mô hình miền hợp nhất, nâng cao hiệu quả của các bộ chuyển đổi mô hình và tăng cường khả năng áp dụng phương pháp trong thực tiễn phát triển phần mềm.

- *Hoàn thiện môi trường công cụ hỗ trợ mô hình hóa miền hợp nhất.* Một hướng phát triển tiếp theo là tiếp tục hoàn thiện công cụ UDML dưới dạng một môi trường phát triển tích hợp hoặc plugin cho các nền tảng phát triển phần mềm hiện đại. Cần nghiên cứu và xây dựng các cú pháp cụ thể dạng biểu diễn đồ họa cho các DSL theo mối quan tâm, đặc biệt đối với các mô hình hành vi, mô hình bảo mật và các mẫu chú thích ràng buộc. Việc này giúp nâng cao khả năng sử dụng và tạo điều kiện thuận lợi cho việc trao đổi giữa các bên liên quan tham gia vào quá trình xây dựng và hiệu chỉnh mô hình.

- *Mở rộng khả năng biểu diễn ràng buộc của mô hình miền.* Trong tương lai, cần tiếp tục mở rộng thư viện CAP nhằm hỗ trợ nhiều loại ràng buộc OCL phức tạp hơn, bao gồm các ràng buộc phụ thuộc ngữ cảnh, ràng buộc thời gian, ràng buộc liên lớp và các ràng buộc quy trình nghiệp vụ. Điều này sẽ góp phần nâng cao tính biểu đạt, khả năng tái sử dụng và khả năng áp dụng của mô hình miền trong các miền ứng dụng thực tế.

- *Mở rộng UDML cho các DSL theo mối quan tâm mới.* Hiện nay, UDML tập trung tích hợp các khía cạnh cấu trúc, ràng buộc, hành vi và bảo mật. Trong tương lai, cần mở rộng khung phương pháp để hỗ trợ các DSL cho các mối quan tâm khác như giao diện người dùng, quyền riêng tư dữ liệu, ghi nhật ký, tuân thủ quy định và các yêu cầu phi chức năng. Đồng thời, cần nghiên cứu các cơ chế tích hợp ngữ nghĩa giữa các DSL này trong mô hình miền hợp nhất. Đối với bảo mật, cần tiếp tục mở rộng hỗ trợ cho các cơ chế phân quyền động, kiểm soát truy cập phụ thuộc ngữ cảnh và phân quyền dựa trên thuộc tính.

- *Nâng cao mức độ tự động hóa của các bộ chuyển đổi mô hình.* Mặc dù luận án đã xây dựng các bộ chuyển đổi từ mô hình yêu cầu sang mô hình miền hợp nhất và từ mô hình miền hợp nhất sang các tạo tác thực thi, mức độ tự động hóa của quy trình vẫn có thể tiếp tục được cải thiện. Các nghiên cứu tiếp theo có thể tập trung vào việc mở rộng phạm vi các luật chuyển đổi, tăng cường khả năng kiểm tra tính nhất quán giữa các mức mô hình và xây dựng các cơ chế đánh giá chất lượng chuyển đổi. Bên cạnh đó, việc nghiên cứu các kỹ thuật kiểm chứng bảo toàn ngữ nghĩa trong quá trình chuyển đổi cũng là

một hướng cần thiết nhằm đảm bảo tính đúng đắn của các tạo tác được sinh ra.

- *Tích hợp các kỹ thuật AI/LLM vào quy trình mô hình hóa miền.* Sự phát triển nhanh chóng của các mô hình ngôn ngữ lớn (LLMs) mở ra nhiều cơ hội mới cho việc hỗ trợ kỹ sư phần mềm trong các hoạt động phân tích yêu cầu và mô hình hóa miền. Trong bối cảnh của luận án, LLM không được xem là sự thay thế cho các DSL, các kỹ thuật chuyển đổi mô hình hoặc các cơ chế kiểm chứng hình thức, mà được xem như một công nghệ hỗ trợ nhằm nâng cao khả năng sử dụng và mức độ tự động hóa của công cụ.

  Một hướng nghiên cứu tiềm năng là sử dụng LLM để hỗ trợ giai đoạn khởi tạo mô hình miền từ các mô tả yêu cầu bằng ngôn ngữ tự nhiên. Từ các tài liệu đặc tả nghiệp vụ, LLM có thể hỗ trợ nhận diện các thực thể miền, thuộc tính, quan hệ, vai trò người dùng, hành động nghiệp vụ và các tài nguyên cần bảo vệ, từ đó sinh ra các bản nháp ban đầu của mô hình miền hoặc các DSL theo mối quan tâm. Các kết quả này có thể đóng vai trò như một bước tiền xử lý nhằm giảm khối lượng công việc thủ công trong quá trình xây dựng mô hình.

  Bên cạnh đó, LLM có thể được sử dụng để hỗ trợ sinh nháp các đặc tả DSL như CAP, AGL hoặc RBACDom từ mô tả nghiệp vụ. Ví dụ, từ một quy tắc nghiệp vụ được diễn đạt bằng ngôn ngữ tự nhiên, LLM có thể đề xuất các mẫu CAP phù hợp hoặc gợi ý các ràng buộc OCL tương ứng. Đối với hành vi miền, LLM có thể hỗ trợ xây dựng các đặc tả hành vi sơ bộ dưới dạng AGL. Đối với bảo mật, LLM có thể hỗ trợ nhận diện vai trò, quyền và chính sách truy cập từ các tài liệu yêu cầu.

  Một hướng nghiên cứu khác là sử dụng LLM như một trợ lý tương tác trong công cụ UDML. Trong vai trò này, LLM có thể hỗ trợ giải thích mô hình bằng ngôn ngữ tự nhiên, phát hiện các điểm bất thường hoặc thiếu nhất quán trong mô hình, đề xuất các phần tử miền còn thiếu và hỗ trợ sinh tài liệu thiết kế từ các mô hình đã xây dựng. Khả năng này đặc biệt hữu ích trong quá trình trao đổi giữa các bên liên quan tham gia vào phát triển phần mềm, góp phần tăng cường vai trò của ngôn ngữ chung trong DDD.

  Tuy nhiên, các LLM hiện nay vẫn tồn tại nhiều hạn chế liên quan đến tính chính xác, tính nhất quán và khả năng bảo đảm ngữ nghĩa. Các

mô hình này có thể sinh ra các phần tử miền không tồn tại, diễn giải sai yêu cầu hoặc tạo ra các đặc tả không tuân thủ ngữ nghĩa của DSL. Vì vậy, các kết quả do LLM sinh ra cần được chuẩn hóa trong các DSL tương ứng và tiếp tục được kiểm tra bằng các quy tắc hợp lệ cấu trúc, các cơ chế chuyển đổi mô hình và các phương pháp kiểm chứng hình thức. Theo quan điểm này, LLM phù hợp nhất với vai trò hỗ trợ xây dựng và hoàn thiện mô hình, trong khi tính đúng đắn và tính nhất quán của mô hình vẫn cần được bảo đảm bởi các kỹ thuật mô hình hóa và kiểm chứng được đề xuất trong luận án.

Việc kết hợp giữa LLM và mô hình miền hợp nhất UDML được kỳ vọng sẽ tạo ra một hướng nghiên cứu mới, trong đó khả năng hiểu và xử lý ngôn ngữ tự nhiên của trí tuyệ nhân tạo được kết hợp với tính chính xác, khả năng truy vết và khả năng kiểm chứng của các kỹ thuật mô hình hóa hướng miền. Đây là một hướng phát triển có nhiều tiềm năng nhằm nâng cao mức độ tự động hóa và khả năng ứng dụng của phương pháp trong các môi trường phát triển phần mềm hiện đại.

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

1. [V1] Duc-Hanh Dang, Duc Minh Le, Van-Vinh Le. AGL: Incorporating behavioral aspects into domain-driven design. *Information and Software Technology*, 163, 107284, 2023.
DOI: 10.1016/j.infsof.2023.107284. (WoS/Scopus, Q1)

2. [V2] Van-Vinh Le, Nghia-Trong Be, Duc-Hanh Dang. On Automatic Generation of Executable Domain Models for Domain-Driven Design. *Proc. 15th Int. Conf. Knowledge and Systems Engineering (KSE)*, IEEE, 2023. *DOI: 10.1109/KSE59128.2023.10299453*. (WoS/Scopus)

3. [V3] Van-Vinh Le, Duc-Hanh Dang. An Approach to Composing Concerns for an Executable Unified Domain Model. *Proc. 18th Int. Conf. Research, Innovation and Vision for the Future (RIVF)*, IEEE, pp. 415-419,2024. *DOI: 10.1109/RIVF64335.2024.11009109*. (WoS/Scopus)

4. [V4] Le Van Vinh, Dang Duc Hanh. RM2UDM: A method for automatically generating functional prototypes from requirement models. *Proc. 17th National Conf. Fundamental and Applied Information Technology Research (FAIR)*, pp. 734–741, 2024.
ISBN: 978-604-357-304-6, DOI: 10.15625/vap.2024.0271.

5. [V5] Van-Vinh Le, Nhat-Hoang Nguyen, Duc-Quyen Nguyen, Duc-Hanh Dang. A Method for Composing Concerns into a Unified Domain Model in Domain-Driven Design. *Proc. 14th Int. Symposium on Information and Communication Technology (SOICT)*, Springer, 2025.
ISSN 1865-0929.

6. [V6] Van-Vinh Le, Nhat-Hoang Nguyen, Duc-Quyen Nguyen, Duc-Hanh Dang. A Semantic Framework and Tool Support for Unified Executable Domain Models in UDML: A Case Study on the RBAC Concern. *VNU Journal of Science: Computer Science and Communication Engineering. Vol 42 No 1, pages 79-114, 2026. ISSN: 2615-9260, DOI: 10.25073/2588-1086/vnucsce.6743.*

TÀI LIỆU THAM KHẢO

[1] *Actifsource*. Actifsource AG, Switzerland - all rights reserved., 2017. URL https://www.actifsource.com/_downloads/ActifsourceManual_ActifsourceUserManual.pdf.

[2] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, 2010. ISBN 978-0-521-89556-9.

[3] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466, November 2010. ISSN 1433-2779, 1433-2787. doi: 10.1007/s10009-010-0145-y.

[4] Seif Abukhalaf, Mohammad Hamdaqa, and Foutse Khomh. On Codex prompt engineering for OCL generation: An empirical study. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 2023.

[5] Muhammad Umar Aftab, Zhiguang Qin, Negalign Wake Hundera, Oluwasanmi Ariyo, Zakria, Ngo Tung Son, and Tran Van Dinh. Permission-based separation of duty in dynamic role-based access control model. *Symmetry*, 11(5):669, 2019.

[6] Fatimah Akeel, Asieh Salehi Fathabadi, Federica Paci, Andrew Gravell, and Gary Wills. Formal Modelling of Data Integration Systems Security Policies. *Data Science and Engineering*, 1(3):139–148, September 2016. ISSN 2364-1541. doi: 10.1007/s41019-016-0016-y.

[7] Hessa Ali Alhamad and Mohammad Mahdi Hassan. Aspect-Oriented Models-Based Framework to Secure Intelligent Systems. In *Proc. 2022 8th Int. Conf. Computer Technology Applications*, ICCTA '22, pages 249–262, New York, NY, USA, September 2022. Association for Computing Machinery. ISBN 978-1-4503-9622-6. doi: 10.1145/3543712.3543726.

[8] Shaukat Ali, Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel C. Briand. Generating Test Data from OCL Constraints with Search Techniques. *IEEE Transactions on Software Engineering*, 39(10):

1376–1402, October 2013. ISSN 1939-3520. doi: 10.1109/TSE.2013.17. Conference Name: IEEE Transactions on Software Engineering.

[9] Andrzej Wasowski and Thorsten Berger. *Domain-Specific Languages Effective Modeling, Automation, and Reuse*. Springer Cham, 2023. ISBN 978-3-031-23668-6.

[10] ANSI/INCITS. Information technology – role-based access control. Technical Report INCITS 359-2012, American National Standards Institute (ANSI), May 2012. URL https://webstore.ansi.org/standards/incits/incits3592012. Reaffirmed as INCITS 359-2012 (R2022).

[11] I. Antović, S. Vlajić, M. Milić, D. Savić, and V. Stanojević. Model and software tool for automatic generation of user interface based on use case and data model. *Institution of Engineering and Technology*, 6(6): 559 – 573, 2012. ISSN Print ISSN 1751-8806. Online ISSN 1751-8814. doi: 10.1049/iet-sen.2011.0060.

[12] David Basin, Jurgen Doser, and Torsten Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, January 2006. ISSN 1049-331X. doi: 10.1145/1125808.1125810.

[13] Nelly Bencomo, Jordi Cabot, et al. Abstraction engineering. *arXiv preprint arXiv:2408.14074*, August 2024.

[14] Alexander Bergmayr, Michael Grossniklaus, Manuel Wimmer, and Gerti Kappel. Leveraging annotation-based modeling with Jump. *Software & Systems Modeling*, 17(1):65–89, February 2018. ISSN 1619-1366, 1619-1374.

[15] Mario Luca Bernardi, Giuseppe Antonio Di Lucca, and Damiano Distante. Model-driven fast prototyping of RIAs: From conceptual models to running applications. In *Proc. 2014 Int. Conf. Advances in Computing, Communications and Informatics (ICACCI)*, pages 250–258, 2014. doi: 10.1109/ICACCI.2014.6968522.

[16] Dines Bjørner. Domain modelling: A foundation for software development. In *Theories of Programming and Formal Methods: Essays Dedicated to He Jifeng on the Occasion of His 80th Birthday*, pages 165–210. Springer, 2023. doi: 10.1007/978-3-031-40436-8_7.

[17] Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. Object-oriented analysis and design with applications, third edition. *ACM SIGSOFT Software Engineering Notes*, 33(5):29–29, August 2008. ISSN 0163-5948. doi: 10.1145/1402521.1413138.

[18] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan&Claypool, 2nd edition, 2017. ISBN 978-1-62705-708-0.

[19] Achim D. Brucker, Matthias P. Krieger, Delphine Longuet, and Burkhart Wolff. A Specification-Based Test Case Generation Method for UML/OCL. In *Models in Software Engineering*, pages 334–348. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-21210-9. doi: 10.1007/978-3-642-21210-9_33. ISSN: 1611-3349.

[20] Antonio Bucchiarone, Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. *Domain-specific languages in practice: with JetBrains MPS*. Springer Nature, 2021.

[21] Antonio Bucchiarone, Juri Di Rocco, Damiano Di Vincenzo, and Alfonso Pierantonio. From OCL to JSX: declarative constraint modeling in modern SaaS tools, September 2025.

[22] Frank Budinsky. *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley, 2004. ISBN 978-0-13-142542-2.

[23] Dennis M. Buede and William D. Miller. *The Engineering Design of Systems: Models and Methods*. Wiley, 4th edition, 2024. ISBN 978-1-119-98401-6.

[24] Fabian Büttner and Martin Gogolla. On ocl-based imperative model transformation. *Electronic Communications of the EASST*, 29, 2010.

[25] J. Cabot and E. Teniente. Transformation techniques for OCL constraints. *Science of Computer Programming*, 68(3):179–195, October 2007. ISSN 0167-6423. doi: 10.1016/j.scico.2007.05.001.

[26] Jordi Cabot and Martin Gogolla. Object Constraint Language (OCL): A Definitive Guide. In *Formal Methods for Model-Driven Engineering*, volume 7320, pages 58–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-30981-6 978-3-642-30982-3.

[27] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. On the assessment of generative AI in modeling tasks: An experience report with ChatGPT and UML. *Software and Systems Modeling*, 22 (3):781–793, 2023. doi: 10.1007/s10270-023-01105-5.

[28] Edmilson Campos, Uirá Kulesza, Marília Freire, and Eduardo Aranha. A Generative Development Method with Multiple Domain-Specific Languages. In *M. (eds) Product-Focused Software Process Improvement. PROFES 2014. Lecture Notes in Computer Science*, volume 8892, pages 178–193. Springer, Cham, 2014. ISBN 978-3-319-13834-3.

[29] Joanna Chimiak-Opoka, Birgit Demuth, Andreas Awenius, Dan Chiorean, Sebastien Gabel, Lars Hamann, and Edward Willink. OCL tools report based on the ide4OCL feature model. *Electronic Communications of the EASST*, 44, 2011.

[30] Tony Clark and Jos B. Warmer, editors. *Object modeling with the OCL: the rationale behind the Object Constraint Language*. Number 2263 in Lecture notes in computer science. Springer, Berlin ; New York, 2002. ISBN 978-3-540-43169-5.

[31] Benoit Combemale, Robert France, Jean-Marc Jézéquel, Bernhard Rumpe, James Steel, and Didier Vojtisek. *Engineering Modeling Languages: Turning Domain Knowledge into Tools*. CRC Press, November 2016. ISBN 978-1-315-38793-2.

[32] Krzysztof Czarnecki. Overview of Generative Software Development. In Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors, *Unconventional Programming Paradigms*, pages 326–341, Berlin, Heidelberg, 2005. Springer. ISBN 978-3-540-31482-0. doi: 10.1007/11527800_25.

[33] Irene Córdoba-Sánchez and Juan De Lara. Ann: A domain-specific language for the effective design and validation of Java annotations. *Computer Languages, Systems & Structures*, 45:164–190, April 2016. ISSN 14778424.

[34] Alberto Rodrigues da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, October 2015. doi: 10.1016/j.cl.2015.06.001.

[35] Carolina Dania and Manuel Clavel. OCL2MSFOL: a mapping to many-sorted first-order logic for efficiently checking the satisfiability of OCL constraints. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 65–75, Saint-malo France, October 2016. ACM. ISBN 978-1-4503-4321-3. doi: 10.1145/2976767.2976774.

[36] Istvan David, Kousar Aslam, Ivano Malavolta, and Patricia Lago. Collaborative Model-Driven Software Engineering — A systematic survey of practices and needs in industry. *Journal of Systems and Software*, 199:111626, May 2023. ISSN 0164-1212. doi: 10.1016/j.jss.2023.111626.

[37] Dr Birgit Demuth. OCL (Object Constraint Language) by Example. page 60, 2009.

[38] Qing Duan, Junhui Liu, Zhihong Liang, Hongwei Kang, and Xingping Sun. An Analysis of the Keys to the Executable Domain-Specific Model. In Srikanta Patnaik and Xiaolong Li, editors, *Proc. Int. Conf. Soft Computing Techniques and Engineering Application*, pages 567–574, New Delhi, 2014. Springer India. ISBN 978-81-322-1695-7. doi: 10.1007/978-81-322-1695-7_67.

[39] Edmilson Campos, Neto, Marília Aranha, Freire, Uirá, Kulesza, Adorilson, Bezerra, and Eduardo, Aranha. Composition of Domain Specific Modeling Languages-An Exploratory Study.. In *Proc. 1st Int. Conf. Model-Driven Engineering*, pages 149–156, 2013.

[40] Tobias Eisenreich, Husein Jusic, and Stefan Wagner. Automating domain-driven design: Experience with a prompting framework. In *Proceedings of the IEEE/ACM International Conference on Software Engineering Workshops (ICSE Workshops)*. IEEE, 2023.

[41] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004.

[42] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53. IEEE, 2023.

[43] Agung Fatwanto and Clive Boughton. Analysis, Specification and Modeling of Functional Requirements for Translative Model-Driven Development. In *Proc. 2008 Int. Symposium Conf. Knowledge Acquisition and Modeling*, pages 859–863. IEEE, October 2008. ISBN 978-0-7695-3488-6. doi: 10.1109/KAM.2008.185.

[44] Alessio Ferrari, Sallam Abualhaija, and Chetan Arora. Model Generation with LLMs: From Requirements to UML Sequence Diagrams. In *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, pages 291–300, June 2024. doi: 10.1109/REW61692.2024.00044. URL https://ieeexplore.ieee.org/document/10628665. ISSN: 2770-6834.

[45] Elvis Foster and Bradford Towle Jr. *Software Engineering: A Methodical Approach, 2nd Edition*. Auerbach Publications, New York, 2 edition, July 2021. ISBN 978-0-367-74602-5. doi: 10.1201/9780367746025.

[46] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 1st edition, September 2010. ISBN 978-0-13-139280-9.

[47] Alfonso Fuggetta and Elisabetta Di Nitto. Software process. In *Future of Software Engineering Proceedings*, FOSE 2014, pages 1–12, New York, NY, USA, May 2014. Association for Computing Machinery. ISBN 978-1-4503-2865-4. doi: 10.1145/2593882.2593883.

[48] Fernanda Gonzalez-Lopez, Guillermo Bustos, Jorge Munoz-Gama, and Marcos Sepulveda. Domain model based design of business process architectures. *Applied Sciences*, 12(5):2563, 2022.

[49] James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification, Java SE 8 Edition*. Addison-Wesley Professional, 1st edition, April 2014. ISBN 978-0-13-390069-9.

[50] Olga Goubali, Patrick Girard, Laurent Guittet, Alain Bignon, Djamal Kesraoui, Pascal Berruet, and Jean-Frédéric Bouillon. Designing Functional Specifications for Complex Systems. In *Human-Computer Interaction. Theory, Design, Development and Practice . HCI 2016. Lecture Notes in Computer Science()*, volume 9731, pages 166–177. Springer, Cham, 2016. ISBN 978-3-319-39509-8. doi: 10.1007/978-3-319-39510-4_16.

[51] Jesse Griffin. Domain-Driven Laravel, Learn to Implement Domain-Driven Design Using Laravel. January 2021. ISSN 978-1-4842-6022-7. doi: 10.1007/978-1-4842-6023-4.

[52] Raffaela Groner, Peter Bellmann, Stefan Höppner, Patrick Thiam, Friedhelm Schwenker, Hans A. Kestler, and Matthias Tichy. Enhanced performance prediction of ATL model transformations. *Performance Evaluation*, 164:102413, May 2024. ISSN 0166-5316. doi: 10.1016/j.peva.2024.102413. URL https://www.sciencedirect.com/science/article/pii/S016653162400018X.

[53] Arne Haber, Markus Look, Antonio Navarro Perez, Pedram Mir Seyed Nazari, Bernhard Rumpe, Steven Völkel, and Andreas Wortmann. Integration of heterogeneous modeling languages via extensible and composable language components. In *2015 3rd Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 19–31. IEEE, 2015.

[54] Lars Hamann, Oliver Hofrichter, and Martin Gogolla. OCL-Based Runtime Monitoring of Applications with Protocol State Machines. In *Modelling Foundations and Applications*, volume 7349, pages 384–399. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-31490-2 978-3-642-31491-9. doi: 10.1007/978-3-642-31491-9_29. Series Title: Lecture Notes in Computer Science.

[55] Fitash Ul Haq and Jordi Cabot. B-OCL: An Object Constraint Language Interpreter in Python, March 2025. arXiv:2503.00944 [cs].

[56] Lukas Heiland, Marius Hauser, and Justus Bogner. Design Patterns for AI-based Systems: A Multivocal Literature Review and Pattern Repository. In *2023 IEEE/ACM 2nd Int. Conf. AI Engineering – Software Engineering for AI (CAIN)*, pages 184–196, May 2023. doi: 10.1109/CAIN58948.2023.00035.

[57] Anders Hejlsberg, Mads Torgersen, Scott Wiltamuth, and Peter Golde. *The C# Programming Language (Covering C# 4.0)*. Addison-Wesley Professional, October 2010. ISBN 978-0-13-248172-4.

[58] Frank Hilken and Lars Hamann. History of the USE Tool 20 Years of UML/OCL Modeling Made in Germany. *The Journal of Object Technology*, 19(3):3:1, 2020. ISSN 1660-1769. doi: 10.5381/jot.2020.19.3.a20.

[59] Jeffrey A. Hoffer, Joey George, and Joseph A. Valacich. *Modern Systems Analysis and Design*. Pearson, 7a edition, 2013. ISBN 978-0-13-299130-8.

[60] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79, 2024. doi: 10.1145/3695988.

[61] Vincent C Hu, Rick Kuhn, and Dylan Yaga. Verification and test methods for access control policiesmodels. Technical Report NIST SP 800-192, National Institute of Standards and Technology, Gaithersburg, MD, June 2017.

[62] Sandeep Kumar Jaiswal and Rohit Agrawal. Domain-Driven Design (DDD)- Bridging the Gap between Business Requirements and Object-Oriented Modeling. *Int. Innovative Research in Engineering and Management*, 11(2):79–83, 2024. ISSN 2350-0557.

[63] Manfred Jeusfeld, Matthias Jarke, and John Mylopoulos. *Metamodeling for Method Engineering*. MIT Press, 2009. ISBN 978-0-262-10108-0. Google-Books-ID: i1kRtAEACAAJ.

[64] Zihan et al. Jiang. A survey on large language models for software engineering. *arXiv preprint arXiv:2308.10620*, 2023.

[65] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. *Science of computer programming*, 72(1-2):31–39, 2008.

[66] Stefan Kapferer and Olaf Zimmermann. Domain-Driven Service Design. In Schahram Dustdar, editor, *Service-Oriented Computing*, Communications in Computer and Information Science, pages 189–208, Cham, 2020. Springer International Publishing. ISBN 978-3-030-64846-6. doi: 10.1007/978-3-030-64846-6_11.

[67] Elavarasi Kesavan. The Evolution of Software Design Patterns: An In-Depth Review. *International Journal of Innovations in Science, Engineering And Management*, pages 163–167, 2025.

[68] Meriem Kherbouche and Balint Molnar. Formal Model Checking and Transformations of Models Represented in UML with Alloy. In *Modelling to Program*, pages 127–136, Cham, 2021. Springer International Publishing. ISBN 978-3-030-72696-6.

[69] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 1 edition, December 2008. ISBN 978-0-321-55345-4.

[70] Preyanoot Kluisritrakul and Yachai Limpiyakorn. Generation of Java Code from UML Sequence and Class Diagrams. In *Proc. 7th Int. Conf. Information Science and Applications (ICISA 2016)*, volume 376, pages 1117–1125. Springer. K.J. Kim and N. Joukov (eds.), 2016. doi: DOI:10.1007/978-981-10-0557-2_106.

[71] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson, Upper Saddle River, NJ, 2010. ISBN 978-0-13-148906-6.

[72] Michael Lawley and Jim Steel. Practical declarative model transformation with tefkat. In *Proc. int. conf. Satellite Events at the MoDELS*, MoDELS'05, pages 139–150, Berlin, Heidelberg, October 2005. Springer-Verlag. ISBN 978-3-540-31780-7. doi: 10.1007/11663430_15.

[73] Duc Minh Le, Duc-Hanh Dang, and Viet-Ha Nguyen. On design using annotation-based domain specific language. *Computer Languages, Systems & Structures*, 54:199–235, December 2018. ISSN 14778424.

[74] Duc Minh Le, Duc-Hanh Dang, and Ha Thanh Vu. jDomainApp: A Module-Based Domain-Driven Software Framework. In *Proc. 10th Int. Symposium on Information and Communication Technology (SoICT)*, pages 399–406, December 2019. ISBN 978-1-4503-7245-9.

[75] Duc Minh Le, Duc-Hanh Dang, and Viet-Ha Nguyen. Generative software module development for domain-driven design with annotation-based domain specific language. *Information and Software Technology*, 120:106–239, 2020.

[76] Yunliang Li, Zhiqiang Du, Yanfang Fu, and Liangxin Liu. Role-based access control model for inter-system cross-domain in multi-domain environment. *Applied Sciences*, 12(24):13036, 2022.

[77] Barbara Liskov and John Guttag. *Program development in Java: abstraction, specification, and object-oriented design*. Addison-Wesley Professional, 2000. ISBN 978-0-201-65768-5.

[78] Shugang Liu, Jinfeng Chen, Yifei Liu, and Pengrui Lv. Mapping of UML Diagrams to Executable Code. pages 9–16. Atlantis Press, April 2017. ISBN 978-94-6252-327-2. doi: 10.2991/icmse-17.2017.2.

[79] Wissam Mallouli, Jean-Marie Orset, Ana Cavalli, Nora Cuppens, and Frederic Cuppens. A formal approach for testing security rules. In *Proc. 12th ACM symposium on Access control models and technologies - SACMAT '07*, page 127, Sophia Antipolis, France, 2007. ACM Press. ISBN 978-1-59593-745-2. doi: 10.1145/1266840.1266860.

[80] Neel Mani, Markus Helfert, and Claus Pahl. A Domain-specific Rule Generation Using Model-Driven Architecture in Controlled Variability Model. *Procedia Computer Science*, 112:2354–2362, 2017. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2017.08.206.

[81] Ismail Mendil, Yamine Ait-Ameur, Neeraj Kumar Singh, Guillaume Dupont, Dominique Mery, and Philippe Palanque. Formal domain-driven system development in Event-B: Application to interactive critical systems. *Journal of Systems Architecture*, 135:102798, February 2023. ISSN 1383-7621. doi: 10.1016/j.sysarc.2022.102798.

[82] Marcelo Paternostro Ed Dave Steinberg Merks, Frank Budinsky. *EMF: Eclipse Modeling Framework Second Edition*. ISBN 978-0-321-33188-5.

[83] Dominique Mery. The Event B Modelling Method. *Erasmus*, 2011.

[84] Judith Michael, Loek Cleophas, and et al. Zschaler. Model-Driven Engineering for Digital Twins: Opportunities and Challenges. *Systems Engineering*, 28(5):659–670, 2025. ISSN 1520-6858. doi: 10.1002/sys.21815. _eprint: https://incose.onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21815.

[85] Aya Khaled Youssef Sayed Mohamed, Dagmar Auer, Daniel Hofer, and Josef Küng. A systematic literature review for authorization and access control: definitions, strategies and models. *International Journal of Web Information Systems*, 18(2-3):156–180, August 2022. ISSN 1744-0084. doi: 10.1108/IJWIS-04-2022-0077.

[86] Lionel Montrieux, Yijun Yu, Michel Wermelinger, and Zhenjiang Hu. Issues in representing domain-specific concerns in model-driven engineering. In *2013 5th International Workshop on Modeling in Software Engineering (MiSE)*, pages 1–6. IEEE, 2013.

[87] Maryam I. Mukhtar and Bashir S. Galadanci. Automatic code generation from UML diagrams: the state-of-the-art. *Science World Journal*, 13(4):47–60, 2019. ISSN 1597-6343.

[88] Iftikhar Azim Niaz and Jiro Tanaka. An Object-Oriented Approach to Generate Java Code from UML Statecharts. *International Journal of Computer & Information Science*, 6(2):83–98, 2005.

[89] Erik et al. Nijkamp. Codegen: An open large language model for code generation. *arXiv preprint arXiv:2203.13474*, 2022.

[90] Carlos Noguera and Laurence Duchien. Annotation Framework Validation Using Domain Models. In *Model Driven Architecture – Foundations and Applications*, pages 48–62. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-540-69100-6. doi: 10.1007/978-3-540-69100-6_4. ISSN: 1611-3349.

[91] Siegfried Nolte. *QVT-Relations Language*. Springer Science & Business Media, 2009.

[92] Milan Nosál', Matús Sulír, and Ján Juhár. Language composition using source code annotations. *Computer Science and Information Systems*, 13(3):707–729, 2016. ISSN 1820-0214, 2406-1018.

[93] OMG. *Object Constraint Language 4*. 2014.

[94] OMG. *Unified Modeling Language 2.5.1*. Object Management Group, 2017.

[95] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[96] Orcun Oruc. Role-based embedded domain-specific language for collaborative multi-agent systems through blockchain technology. In *9th Int. Conf. Security, Privacy and Trust Management (SPTM)*, 2021.

[97] Samir Ouchani and Mourad Debbabi. Specification, verification, and quantification of security in model-based systems. *Computing*, 97(7):691–711, July 2015. ISSN 0010-485X, 1436-5057. doi: 10.1007/s00607-015-0445-x.

[98] Javier Paniza. Openxava: Automatic frontend engine for java. https://openxava.org/, 2021. URL https://openxava.org/.

[99] Marijana Rackov, Sebastijan Kaplar, Milorad Filipović, and Gordana Milosavljević. Java code generation based on ocl rules. In *6th International Conference on Information Society and Technology*, pages 191–196, 2016.

[100] Qusai Ramadan, Mattia Salnitriy, Daniel Struber, Jan Jurjens, and Paolo Giorgini. From Secure Business Process Modeling to Design-Level Security Verification. In *2017 ACM/IEEE 20th Int. Conf. Model Driven Engineering Languages and Systems (MODELS)*, pages 123–133, Austin, TX, September 2017. IEEE. ISBN 978-1-5386-3492-9. doi: 10.1109/MODELS.2017.10.

[101] Abhishek Rawat, Raghuraj Singh Suryavanshi, Vishal Nagar, and Diwakar Yadav. Formal Development of Blockchain Enabled E Healthcare System Using Event-B. *Available at SSRN 5167558*, 2025.

[102] Carlos Rodríguez, Mario Sánchez, and Jorge Villalobos. Executable model composition: a multilevel approach. In *Proc. 2011 ACM Symposium on Applied Computing (SAC)*, 11, pages 877–884. Association for Computing Machinery, March 2011. ISBN 978-1-4503-0113-8.

[103] Markus Roggenbach, Antonio Cerone, Bernd-Holger Schlingloff, Gerardo Schneider, and Siraj Ahmed Shaikh. Correction to: Formal Methods for Software Engineering. In Markus Roggenbach, Antonio Cerone, Bernd-Holger Schlingloff, Gerardo Schneider, and Siraj Ahmed Shaikh, editors, *Formal Methods for Software Engineering: Languages, Methods, Application Domains*, pages C1–C1. Springer International Publishing, Cham, 2022. ISBN 978-3-030-38800-3.

[104] António Miguel Rosado da Cruz and João Faria. Automatic Generation of User Interface Models and Prototypes from Domain and Use Case Models. Intech, user interfaces, rita matrai (ed.) edition, 2010. ISBN 978-953-307-084-1. doi: 10.5772/9498.

[105] Djaber Rouabhia and Ismail Hadjadj. Behavioral Augmentation of UML Class Diagrams: An Empirical Study of Large Language Models for Method Generation, June 2025.

[106] Subhrojyoti Roy Chaudhuri, Swaminathan Natarajan, Amar Banerjee, and Venkatesh Choppella. Methodology to develop domain specific modeling languages. In *Proc. 17th ACM SIGPLAN Int. Workshop on Domain-Specific Modeling*, DSM 2019, pages 1–10, New York, NY, USA, October 2019. Association for Computing Machinery. ISBN 978-1-4503-6984-8. doi: 10.1145/3358501.3361235.

[107] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, April 2009. ISSN 1573-7616. doi: 10.1007/s10664-008-9102-8.

[108] Mark Sagar. Creating Models for Simulating the Face. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems*, pages 394–394, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-24485-8. doi: 10.1007/978-3-642-24485-8_28.

[109] Rubén Salado-Cid, Antonio Vallecillo, Kamram Munir, and José Raúl Romero. SWEL: A Domain-Specific Language for Modeling Data-Intensive Workflows. *Business & Information Systems Engineering*, 66(2):137–160, April 2024. ISSN 1867-0202. doi: 10.1007/s12599-023-00826-7.

[110] Josue Sangabriel-Alarcón, Jorge Octavio Ocharán-Hernández, Karen Cortés-Verdín, and Xavier Limón. Domain-Driven Design for Microservices Architecture Systems Development: A Systematic Mapping Study. pages 25–34. IEEE Computer Society, November 2023. ISBN 979-8-3503-2883-7. doi: 10.1109/CONISOFT58849.2023.00014.

[111] Shane Sendall and Wojtek Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(05):42–45, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231150.

[112] B. Shafiq, A. Masood, J. Joshi, and A. Ghafoor. A Role-Based Access Control Policy Verification Framework for Real-Time Systems. In *10th IEEE Inte. Work. Object-Oriented Real-Time Dependable Systems*, pages 13–20, Sedona, AZ, USA, 2005. IEEE. ISBN 978-0-7695-2347-7. doi: 10.1109/WORDS.2005.11.

[113] Anthony Simons. ReMoDeL: A Pure Functional Object-Oriented Concept Language for Models, Metamodels and Model Transformation:. In *Proc. 13th Int. Conf. Model-Based Software and Systems Engineering*, pages 242–249, Porto, Portugal, 2025. SCITEPRESS - Science and Technology Publications. ISBN 978-989-758-729-0. doi: 10.5220/0013184700003896.

[114] Colin Snook, Michael Butler, Thai Son Hoang, Asieh Salehi Fathabadi, and Dana Dghaym. Developing the UML-B Modelling Tools. In Paolo Masci, Cinzia Bernardeschi, Pierluigi Graziani, Mario Koddenbrock, and Maurizio Palmieri, editors, *Software Engineering and Formal Methods. SEFM 2022 Collocated Workshops*, pages 181–188, Cham, 2023. Springer International Publishing. ISBN 978-3-031-26236-4. doi: 10.1007/978-3-031-26236-4_16.

[115] K. Sohr, M. Drouineaud, G.-J. Ahn, and M. Gogolla. Analyzing and Managing Role-Based Access Control Policies. *IEEE Transactions on Knowledge and Data Engineering*, 20(7):924–939, July 2008. ISSN 1041-4347. doi: 10.1109/TKDE.2008.28.

[116] Frank P. M. Stappers, Michel A. Reniers, and Sven Weber. Transforming SOS Specifications to Linear Processes. In *Formal Methods for Industrial Critical Systems*, volume 6959, pages 196–211. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-24430-8 978-3-642-24431-5.

[117] James Gosling Bill Joy Guy Steele and Gilad Bracha Alex Buckley. *The Java language Specificition*. Bhaskarjyoti Saikia, 2020.

[118] Harald Störrle. Improving the Usability of OCL as an Ad-hoc Model Querying Language: 13th International Workshop on OCL, Model Constraint and Query Languages (OCL 2013). *Proc. MODELS 2013 OCL Workshop*, pages 83–92, 2013.

[119] E. V. Sunitha and Philip Samuel. Object constraint language for code generation from activity models. *Information and Software Technology*, 103:92–111, November 2018. ISSN 0950-5849. doi: 10.1016/j.infsof.2018.06.010.

[120] Witold Suryn. *Software Quality Engineering: A Practitioner's Approach*. John Wiley & Sons, December 2013. ISBN 978-1-118-83018-5.

[121] Thakur and U.S. Pandey. The Role of Model-View Controller in Object Oriented Software Development. *Nepal Journal of Multidisciplinary Research*, 2:1–6, November 2019. doi: 10.3126/njmr.v2i2.26279.

[122] The Apache Software Foundation. Apache Causeway: Framework for rapidly developing domain-driven apps in Java. https://causeway.apache.org/, 2023.

[123] Arie Van Deursen, Eelco Visser, and Jos Warmer. Model-driven software evolution: A research agenda. In *Proc. 1st int.l workshop on model-driven software evolution*, pages 41–49. MoDSE Nantes, France, 2007.

[124] Vaughn Vernon. *Implementing domain-driven design*. Addison-Wesley Professional, 1 er edition, 2013. ISBN 978-0-321-83457-7.

[125] Vaughn Vernon. *Domain-Driven Design Distilled*. Addison-Wesley Professional, 1st edition edition, May 2016. ISBN 978-0-13-443442-1.

[126] Inna Vistbakka and Elena Troubitsyna. Towards Integrated Modelling of Dynamic Access Control with UML and Event-B. *Electronic Proceedings in Theoretical Computer Science*, 271:105–116, May 2018. ISSN 2075-2180. doi: 10.4204/EPTCS.271.8.

[127] Sunitha Edacheril Viswanathan and Philip Samuel. Automatic code generation using unified modeling language activity and sequence models. *IET Software*, 10(6):164–172, 2016. ISSN 1751-8814. doi: 10.1049/iet-sen.2015.0138.

[128] Markus Voelter. Language and IDE Modularization and Composition with MPS. volume 7680 of *Lecture Notes in Computer Science (LNPSE)*, pages 383–430. Springer, Berlin, Heidelberg, 2013. ISBN 978-3-642-35992-7.

[129] M. Völter, S Benz, C Dietrich, B Engelmann, M Helander, LCL Kats, E Visser, and GH Wachsmuth. *DSL Engineering - Designing, implementing and using domain-specific languages*. M Volter / DSL-Book.org, Stuttgart, Germany, 2013.

[130] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[131] J. B. Warmer and A. G. Kleppe. Building a Flexible Software Factory Using Partial Domain Specific Models. In *6th OOPSLA Workshop on Domain-Specific Modeling, DSM*, pages 15–22. University of Jyvaskyla, 2006. ISBN 951-39-2631-1.

[132] E. D. Willink. An extensible OCL virtual machine and code generator. In *Proceedings of the 12th Workshop on OCL and Textual Modelling*, OCL '12, pages 13–18, New York, NY, USA, September 2012. Association for Computing Machinery. ISBN 978-1-4503-1799-3. doi: 10.1145/2428516.2428519.

[133] Edward D. Willink. Challenges for code generated OCL execution. In *Proc. 25th Int. Conf. Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, pages 872–881, New York, NY, USA, November 2022. Association for Computing Machinery. ISBN 978-1-4503-9467-3. doi: 10.1145/3550356.3561537.

[134] Willow. Model-Driven Rapid Prototyping for Control Algorithms with the GIPS Framework. In *Proc. 14th. Int. Workshop Graph Computation Models (GCM)*. STAFF/GCM Session 1, 2023.

[135] Andrzej Wasowski and Thorsten Berger. Internal Domain-Specific Languages. In *Domain-Specific Languages: Effective Modeling, Automation, and Reuse*, pages 357–394. Springer, Cham, 2023. ISBN 978-3-031-23669-3.

[136] Idriss Chana Mohammed Lahmer and Abdallah Rhattoy Yassine Rhazali, Youssef Hadi. A model transformation in model driven architecture from business model to web model. 2018.

[137] Chenxing Zhong, Shanshan Li, Huang Huang, Xiaodong Liu, Zhikun Chen, Yi Zhang, and He Zhang. Domain-Driven Design for Microservices: An Evidence-Based Investigation. *IEEE Transactions on Software Engineering*, 50(6):1425–1449, June 2024. ISSN 1939-3520.

[138] Ozan Özkan, Önder Babur, and Mark van den Brand. Domain-Driven Design in software development: A systematic literature review on implementation, challenges, and effectiveness. *Journal of Systems and Software*, 230:112537, 2025. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2025.112537.

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHẬN XÉT ĐÁNH GIÁ LUẬN ÁN TIẾN SĨ

Tên đề tài luận án: **Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền**

Ngành: Kỹ thuật phần mềm Mã số: 9480103

Nghiên cứu sinh: Lê Văn Vinh

Họ và tên người nhận xét: Phạm Ngọc Hùng Học hàm, Học vị: PGS.TS.

Chuyên ngành: CNPM

Cơ quan công tác: Trường Đại học Công nghệ, ĐHQGHN

Ý KIẾN NHẬN XÉT

1. Về các ý kiến, nội dung tiếp thu, chỉnh sửa, giải trình của NCS về nhận xét phản biện độc lập

   NCS cơ bản đã giải trình đầy đủ các ý kiến của các phản biện độc lập, nội dung chỉnh sửa giải trình phù hợp, khá thuyết phục.

2. Về tính cấp thiết, ý nghĩa khoa học của đề tài luận án

   Luận án đề xuất phương pháp biểu diễn và chuyển đổi mô hình trong DDD nhằm thu hẹp khoảng cách giữa thiết kế và hệ thống thực thi. Nghiên cứu thực hiện mở rộng mô hình miền để tích hợp toàn diện các yếu tố cấu trúc, hành vi, bảo mật và ràng buộc OCL. Đồng thời, luận án xây dựng ngôn ngữ mô hình miền hợp nhất (UDML) kết hợp có hệ thống các DSL theo mối quan tâm. Từ đó, bộ chuyển đổi mô hình được phát triển để tăng cường tự động hóa sinh phần mềm chất lượng cao và tuân thủ nguyên lý DDD. Đây là bài toán có tính cấp thiết cao, có ý nghĩa khoa học và khả năng ứng dụng trong thực tiễn.

3. Về mục đích và phương pháp nghiên cứu, sự hợp lý và độ tin cậy của các phương pháp nghiên cứu

   Mục tiêu của luận án được phát biểu rõ ràng, khoa học. Luận án đã tiến hành khảo sát các nghiên cứu liên quan, tìm ra các khoảng trống nghiên cứu, đề xuất mô hình, tiến hành thực nghiệm để đánh giá tính hiệu quả của mô hình đề xuất. Phương pháp nghiên cứu của luận án là phù hợp với nội dung nghiên cứu, có độ tin cậy cao và có tính hiện đại.

4. Đánh giá các kết quả đạt được, chỉ rõ những kết quả mới, đóng góp mới và giá trị khoa học của những đóng góp đó

   Luận án đã đạt được các kết quả chính như sau:

   - Đề xuất phương pháp biểu diễn mô hình miền có khả năng thực thi, cho phép tích hợp các khía cạnh cấu trúc, hành vi, bảo mật và ràng buộc nghiệp vụ, đồng thời xây dựng nền tảng ngữ nghĩa và cơ chế kiểm chứng hình thức cho mô hình
   - Đề xuất Ngôn ngữ mô hình miền hợp nhất (UDML), được xây dựng dựa trên tiếp cận siêu mô hình và cây cú pháp trừu tượng, và được kiểm chứng hình thức bằng Event-B
   - Đề xuất các kỹ thuật chuyển đổi mô hình cho phép sinh tự động các bản mẫu phần mềm từ mô hình miền, đồng thời đảm bảo bảo toàn ngữ nghĩa trong quá trình chuyển đổi.

   Các kết quả này đã được công bố tại 01 tạp chí Q1, 01 tạp chí trong nước, 03 báo cáo tại hội nghị quốc tế và 01 báo cáo tại hội nghị trong nước. Các kết quả của luận án có ý nghĩa khoa học, đóng góp các giải pháp biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền - một bài toán quan trọng trong phát triển phần mềm hướng miền.

5. Nhận xét về những ưu điểm và thiếu sót, những điểm cần bổ sung và sửa chữa, những câu hỏi và các vấn đề cần giải thích thêm

   Về ưu điểm: Đây là một luận án có các đóng góp rõ ràng, có công bố khá tốt, kết quả nghiên cứu có ý nghĩa khoa học và thực tiễn cao. Luận án được viết cấu trúc phù hợp và chất lượng khá tốt.

   Về nhược điểm: Quy mô các thực nghiệm còn hạn chế, xem xét bổ sung các thực nghiệm với quy mô lớn hơn để minh chứng tốt hơn cho khả năng áp dụng của các kết quả nghiên cứu của luận án trong thực tiễn. Xu hướng sử dụng AI trong bài toán này cũng đang được quan tâm lớn, luận án xem xét bổ sung các thảo luận, so sánh với hướng nghiên cứu này. Luận án chỉnh sửa văn phong nhằm tăng thêm tính phân tích đánh giá và rà soát lại một số lỗi trình bày.

6. Đánh giá chung về sự đáp ứng các yêu cầu đối với luận án tiến sĩ, về việc luận án có thể đưa ra đánh giá luận án ở Hội đồng đánh giá luận án tiến sĩ hay không?

   Luận án đáp ứng tốt yêu cầu của một luận án Tiến sĩ ngành Kỹ thuật phần mềm. Tôi đồng ý để luận án được đưa ra đánh giá ở Hội đồng đánh giá luận án tiến sĩ.

7. Kết luận

   Luận án đáp ứng tốt yêu cầu của một luận án tiến sĩ theo quy định của ĐHQGHN. Bản tóm tắt phản ảnh trung thực nội dung của luận án.

Hà Nội, ngày 19 tháng 05 năm 2026

XÁC NHẬN CỦA CƠ QUAN CÔNG TÁC

NGƯỜI NHẬN XÉT
(Ký và ghi rõ họ tên)

Phạm Ngọc Hùng

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BẢN NHẬN XÉT LUẬN ÁN TIẾN SĨ

Họ và tên người nhận xét: Trần Hoàng Việt ..... Học hàm, học vị: Tiến sĩ.....

Chuyên ngành: Kỹ thuật phần mềm.....

Cơ quan công tác: Bộ môn ATTT, Khoa CNTT, Trường ĐHCN-ĐHQGHN.....

Họ và tên nghiên cứu sinh: Lê Văn Vinh .....

Tên đề tài luận án: Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền .....

Ngành đào tạo: Kỹ thuật phần mềm..... Mã số: 9480103 .....

Ý KIẾN NHẬN XÉT[1]

1. Tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài luận án

Luận án lựa chọn một vấn đề có tính cấp thiết cao trong bối cảnh các hệ thống phần mềm ngày càng phức tạp, quy mô lớn và đòi hỏi khả năng thích ứng nhanh, đó là thu hẹp khoảng cách giữa mô hình miền và phần mềm thực thi trong phương pháp Domain-Driven Design (DDD). Về ý nghĩa khoa học, luận án đã đề xuất cách tiếp cận có tính hệ thống nhằm biểu diễn đầy đủ các khía cạnh cấu trúc, hành vi, ràng buộc và bảo mật của mô hình miền, đồng thời xây dựng ngôn ngữ UDML với cú pháp và ngữ nghĩa hình thức, góp phần bổ sung nền tảng lý thuyết cho việc kiểm chứng và chuyển đổi mô hình trong DDD. Về ý nghĩa thực tiễn, các kỹ thuật và công cụ được đề xuất cho phép tự động hóa quá trình sinh mã và đảm bảo tính nhất quán giữa thiết kế và cài đặt, qua đó nâng cao chất lượng, giảm chi phí phát triển và bảo trì phần mềm, đặc biệt trong các miền ứng dụng chuyên biệt.

2. Sự không trùng lặp của đề tài nghiên cứu so với các công trình, luận văn, luận án đã công bố trong và ngoài nước; tính trung thực, rõ ràng và đầy đủ trong trích dẫn tài liệu tham khảo

Theo hiểu biết của người nhận xét, người nhận xét chưa phát hiện được sự trùng lặp trong luận án.

3. Mức độ tổng quan các vấn đề lý luận của đề tài

Luận án thể hiện mức độ tổng quan các vấn đề lý luận tương đối đầy đủ và có hệ thống, tập trung vào các hướng nghiên cứu cốt lõi liên quan đến Domain-Driven Design (DDD), ngôn ngữ chuyên biệt miền (DSL) và các kỹ thuật chuyển đổi mô hình. Tác giả đã nhận diện rõ các xu hướng tiếp cận hiện nay, đặc biệt là việc kết hợp DSL dựa trên chú thích với các phương pháp sinh mã tự động, đồng thời chỉ ra những hạn chế còn tồn tại như thiếu mô tả thống nhất về hành vi, khó khăn trong đặc tả ràng buộc phức tạp và thiếu cơ sở ngữ nghĩa

hình thức để kiểm chứng. Tuy nhiên, phần tổng quan chủ yếu tập trung vào khía cạnh kỹ thuật biểu diễn và chuyển đổi mô hình, trong khi có thể mở rộng hơn về các nền tảng lý thuyết liên quan như kiểm chứng hình thức, kỹ nghệ mô hình (MDE) hay các phương pháp đảm bảo chất lượng phần mềm, nhằm tăng cường chiều sâu học thuật và tính toàn diện của cơ sở lý luận.

4. Sự phù hợp giữa tên đề tài với nội dung, giữa nội dung với chuyên ngành và mã số ngành

Tên đề tài phù hợp với nội dung luận án, luận án có nội dung phù hợp với chuyên ngành và mã số chuyên ngành.

5. Độ tin cậy và tính hiện đại của phương pháp đã sử dụng để nghiên cứu

Phương pháp nghiên cứu được sử dụng trong luận án nhìn chung có độ tin cậy cao và mang tính hiện đại, khi kết hợp các hướng tiếp cận tiên tiến như thiết kế hướng miền (DDD), ngôn ngữ chuyên biệt miền (DSL), kỹ thuật chuyển đổi mô hình và kiểm chứng hình thức. Việc xây dựng ngôn ngữ UDML với cú pháp và ngữ nghĩa hình thức, cùng cơ chế ánh xạ sang các đặc tả hình thức để kiểm chứng, cho thấy tác giả đã tiếp cận theo hướng bài bản, có cơ sở khoa học rõ ràng và phù hợp với xu thế của kỹ nghệ phần mềm hiện nay. Bên cạnh đó, việc phát triển công cụ thực nghiệm và tiến hành đánh giá cũng góp phần củng cố tính tin cậy của kết quả nghiên cứu. Tuy nhiên, để tăng thêm tính thuyết phục, luận án có thể mở rộng phạm vi thực nghiệm trên nhiều miền ứng dụng đa dạng hơn hoặc so sánh sâu hơn với các phương pháp hiện có.

6. Kết quả nghiên cứu mới của tác giả; đóng góp mới cho sự phát triển khoa học chuyên ngành; đóng góp mới phục vụ sản xuất, kinh tế, xã hội, an ninh, quốc phòng và đời sống. Ý nghĩa khoa học, giá trị và độ tin cậy của những kết quả đó

Luận án đã đạt được các kết quả nghiên cứu mới có giá trị, thể hiện ở việc đề xuất cách tiếp cận tích hợp cho biểu diễn và chuyển đổi mô hình trong DDD. Các kỹ thuật biểu diễn mô hình miền cho phép đặc tả đồng thời cấu trúc, hành vi, ràng buộc và bảo mật trong một khuôn khổ thống nhất, góp phần khắc phục tính rời rạc của các nghiên cứu trước.

Một đóng góp quan trọng là ngôn ngữ UDML với cú pháp và ngữ nghĩa hình thức, hỗ trợ tích hợp các DSL và kiểm chứng mô hình. Điều này bổ sung cơ sở lý thuyết cho việc đảm bảo tính đúng đắn và nâng cao độ tin cậy trong phát triển phần mềm dựa trên mô hình.

Luận án cũng đề xuất các kỹ thuật chuyển đổi mô hình để sinh tự động phần mềm và phát triển công cụ thực nghiệm, qua đó chứng minh tính khả thi của phương pháp.

Về ứng dụng, các kết quả có ý nghĩa trong việc tăng tự động hóa, giảm chi phí phát triển và có tiềm năng áp dụng trong nhiều lĩnh vực, kể cả các hệ thống quan trọng như quản lý nhà nước, an ninh và quốc phòng.

7. Ưu điểm và nhược điểm về nội dung, kết cấu và hình thức của luận án

Nhìn chung, NCS đã thể hiện tinh thần cầu thị và có những cải tiến đáng kể trong cách trình bày luận án theo hướng hợp lý hơn, phù hợp với các góp ý từ vòng hội thảo và phản biện kín.

7.1. Về nội dung

Tuy nhiên, luận án vẫn còn một số hạn chế. Thứ nhất, phần đặt vấn đề chưa rõ ràng. Mục 1.1.3 nêu ba vấn đề cần giải quyết nhưng chưa xác định được bài toán chung của luận án. NCS cần điều chỉnh theo hướng bắt đầu từ bài toán chính, mô tả rõ đầu vào, đầu ra, sau đó phân rã thành các bài toán con tương ứng.

Phần trình bày các đóng góp chính của luận án còn thiếu nhất quán. Cụ thể, tại chương mở đầu (mục 1.4) nêu bốn đóng góp, trong khi chương kết luận chỉ liệt kê ba, đồng thời nội dung cũng không tương đồng. NCS cần rà soát và điều chỉnh để đảm bảo sự thống nhất giữa các phần.

NCS đã thể hiện tinh thần cầu thị khi bổ sung phần khảo sát các phương pháp AI cho bài toán nghiên cứu. Tuy nhiên, nội dung này còn ngắn và chưa nổi bật (chỉ khoảng một trang tại mục 2.5, trang 47). NCS cần mở rộng phạm vi khảo sát và tăng cường phân tích cho từng hướng tiếp cận; đồng thời, nếu khả thi, nên bổ sung thực nghiệm so sánh ở chương 6 để nâng cao tính thuyết phục của kết quả.

Chương 2 trình bày cả về cơ sở lý thuyết và tổng quan tình hình nghiên cứu. Theo người nhận xét thì nên chia thành hai chương hoặc chia thành hai mục con để làm rõ phần nào trình bày các khái niệm, phần nào trình bày tình hình nghiên cứu. Với cách trình bày như hiện tại, người đọc khó nhận biết rõ mục nào là đang trình bày khái niệm nền tảng, mục nào trình bày tình hình nghiên cứu.

Các định nghĩa được sử dụng trong luận án, được trình bày rải rác ở trong các chương. Theo tôi, NCS nên trình bày toàn bộ các khái niệm liên quan đến luận án trong Chương 2 để người đọc hiểu được chúng trước khi đi vào từng nội dung cụ thể.

Tên của Chương 4 chưa được rõ, mơ hồ. "Mối quan tâm" ở đây là gì? Nó là một phương pháp tích hợp mọi mối quan tâm vào (trong đó, cấu trúc, hành vi, bảo mật chỉ là ví dụ) hay chỉ tích hợp được cấu trúc, hành vi và bảo mật vào mô hình miền? NCS cần cân nhắc và trao đổi với thầy hướng dẫn để làm rõ hơn tiêu đề này.

Trong chương 6, phần trình bày đánh giá về các câu hỏi nghiên cứu, NCS trình bày khá đột ngột, không rõ ràng. NCS đang trình bày về công cụ, lại nói luôn về các câu hỏi nghiên cứu được trả lời như thế nào. Người đọc rất khó hiểu. → NCS nên tách biệt việc trình bày công cụ và việc trả lời, phân tích câu hỏi nghiên cứu. Ví dụ, nếu coi các câu hỏi nghiên cứu là tiêu chí đánh giá thì cần có một phần đánh giá các tiêu chí đó cho các kết quả được trình bày trong luận án hoặc công cụ.

7.2. Về hình thức

Một số hình vẽ như Hình 1.3 sử dụng ảnh có độ nét chưa cao cho các con số, Hình 1.4, NCS cần rà soát, chỉnh sửa thành ảnh vector để luận án đạt chất lượng tốt nhất về mặt trình bày. Một số hình vẽ vẫn còn để nội dung ở tiếng Anh như Hình 1.5. Nhìn chung, các ảnh, hình trong luận án có chất lượng không cao. NCS cần việt hóa và vẽ lại bằng ảnh vector để có chất lượng cao.

Một số câu chưa có trích dẫn cụ thể, ví dụ, trang 17, "Evans nhấn mạnh rằng DM không chỉ là tài liệu phân tích mà phải gắn chặt với cài đặt, chi phối việc phát triển phần mềm trong suốt vòng đời." nên có trích dẫn. Trích dẫn về tài liệu của Evans đã ở trước đó khá xa và thảo luận một số vấn đề khác nữa rồi. Nhiều chỗ khác cũng bị tương tự.

Một số chỗ liệt kê mã nguồn, nhưng không đánh số cùng với tiêu đề, khó để tham chiếu và theo dõi. Ví dụ, đoạn mã nguồn trên trang 23 trình bày về lớp Student, đoạn mã nguồn trên trang 57, 58, v.v. NCS cần bổ sung số thứ tự, mô tả và tham chiếu trong luận án.

Một số câu chưa rõ nghĩa. Ví dụ, trang 49, "Trong chương này, luận án đề xuất các kỹ thuật mở rộng và tích hợp mô hình miền (DM) nhằm đưa các khía cạnh hành vi và ràng buộc nghiệp vụ vào một mô hình miền hợp nhất có khả năng thực thi.". Cụm từ "tích hợp" mô hình miền (DM) là chưa rõ, tích hợp cái gì với cái gì? Như tôi hiểu, ý của NCS là tích hợp các khía cạnh hành vi và ràng buộc nghiệp vụ vào một mô hình miền hợp nhất có khả năng thực thi. → NCS cần rà soát luận án cẩn thận để các câu văn trong sáng, dễ hiểu hơn.

Một số từ viết tắt, nếu có thể thì nên bổ sung từ tiếng Anh tương ứng trong ngoặc đơn. Ví dụ, trang 49, "các DSL dựa trên chú thích (aDSL)" → bổ sung từ đầy đủ cho từ viết tắt aDSL.

Hình 3.3 rất mờ, Hình 3.5 rất nhỏ, v.v. NCS kiểm tra và điều chỉnh.

7.3. Một số câu hỏi

+ Các định nghĩa được sử dụng trong luận án, được trình bày rải rác ở trong các chương. Vì sao NCS không đưa vào chương 2 về các kiến thức nền tảng.
+ "Mối quan tâm" trong tiêu đề chương 4 là gì? Chương 4 trình bày một phương pháp tích hợp mọi mối quan tâm vào (trong đó, cấu trúc, hành vi, bảo mật chỉ là ví dụ) hay chỉ tích hợp được cấu trúc, hành vi và bảo mật vào mô hình miền?

8. Nội dung luận án đã được công bố trên tạp chí, tuyển tập công trình hội nghị khoa học và giá trị khoa học của những công trình đã công bố

Liên quan đến luận án, NCS đã công bố 1 bài báo trên tạp chí Information and Software Technology (Q1), 01 báo cáo tại hội nghị KSE2023 (WoS/Scopus), 01 báo cáo tại hội nghị RIVF 2024 (WoS/Scopus), 01 báo cáo tại hội nghị FAIR 2024, 01 báo cáo tại hội nghị SOICT 2025 (WoS/Scopus, accepted), 01 bài báo tại tạp chí e-Informatica Software Engineering Journal (WoS/Scopus Q3, chưa có kết quả), 01 bài báo tại tạp chí VNU Journal of science: Computer Science and Communication Engineering (2026).

Với số lượng công bố như vậy, luận án đảm bảo đủ khối lượng và chất lượng theo yêu cầu.

9. Luận án có đáp ứng các yêu cầu đối với luận án tiến sĩ không? Luận án có thể đưa ra Hội đồng đánh giá luận án tiến sĩ được hay không

Luận án đáp ứng các yêu cầu đối với luận án tiến sĩ và có thể đưa ra hội đồng đánh giá luận án tiến sĩ cấp ĐHQGHN.

10. Kết luận chung cần khẳng định mức độ đáp ứng yêu cầu đối với một luận án tiến sĩ theo quy định của Đại học Quốc gia Hà Nội; bản tóm tắt luận án phản ảnh trung thành nội dung cơ bản của luận án hay không

Luận án đáp ứng các yêu cầu đối với một luận án tiến sĩ theo quy định của ĐHQGHN. Bản tóm tắt luận án phản ánh trung thực với nội dung cơ bản của luận án.

11. Các ý kiến khác (nếu có).

Không.

Hà Nội, ngày 01 tháng 05 năm 2026

XÁC NHẬN CỦA CƠ QUAN CÔNG TÁC

NGƯỜI NHẬN XÉT

(Ký và ghi rõ họ tên)

TS. Trần Hoàng Việt

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BẢN NHẬN XÉT PHẢN BIỆN LUẬN ÁN TIẾN SĨ

Tên đề tài luận án: ***Nghiên cứu các kĩ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền***

Chuyên ngành: Kỹ thuật phần mềm
Mã số: 9480103.01
Nghiên cứu sinh: LÊ VĂN VINH
Người nhận xét: Nguyễn Mạnh Hùng
Học hàm, Học vị: PGS.TS.
Chuyên ngành: Công nghệ phần mềm
Cơ quan công tác: Học viện công nghệ bưu chính viễn thông

Ý KIẾN NHẬN XÉT

1. **Về các ý kiến, nội dung tiếp thu, sửa chữa, giải trình của NCS về nhận xét của phản biện độc lập**:

   – NCS đã có tiếp thu, sửa chữa và giải trình những phần không thay đổi để bảo vệ luận điểm của mình so với các nhận xét của phản biện độc lập. Nhìn chung, NCS đã thể hiện tinh thần cầu thị, tiếp thu ý kiến của các phản biện độc lập để hoàn thiện tốt hơn luận án của mình.

2. **Về tính cấp thiết, thời sự, ý nghĩa khoa học và thực tiễn của đề tài luận án**:

   – Chủ đề nghiên cứu có ý nghĩa khoa học và ý nghĩa thực tiễn.

3. **Kết luận về sự không trùng lặp (hay trùng lặp) của kết quả nghiên cứu của luận án với kết quả của các công trình khoa học của các tác giả khác; tính trung thực, rõ ràng và đầy đủ trong trích dẫn tài liệu tham khảo**:

   – Theo hiểu biết của phản biện thì chưa phát hiện có sự trùng lặp về đề tài hay kết quả nghiên cứu của luận án với các kết quả nghiên cứu đã được công bố khác.

4. **Mức độ tổng quan các vấn đề lí luận của đề tài**:

   – Đề tài có tính khái quát tương đối.

5. **Về sự phù hợp giữa tên đề tài với nội dung, giữa nội dung với chuyên ngành và mã số chuyên ngành**:

   – Đề tài có ý nghĩa khoa học và thực tiễn.

   – Tên đề tài là phù hợp với nội dung và đóng góp chính của luận án

6. **Sự hợp lý, độ tin cậy và tính hiện đại của các phương pháp đã sử dụng để nghiên cứu**:

– Phương pháp nghiên cứu được NCS sử dụng là có tính khoa học và hợp lí.

7. **Về những kết quả nghiên cứu mới của tác giả; nêu rõ những kết quả mới; đánh giá mức độ đóng góp cho sự phát triển khoa học chuyên ngành; đóng góp mới phục vụ sản xuất, kinh tế, xã hội, an ninh, quốc phòng và đời sống. Đánh giá ý nghĩa, giá trị khoa học và độ tin cậy của những kết quả đó:**

– Luận án có thể coi là có đóng góp trong ba nội dung chính: Một là đề xuất các kỹ thuật biểu diễn mô hình miền. Hai là đề xuất kỹ thuật tích hợp các mối quan tâm vào mô hình miền. Ba là đề xuất cải tiến chuyển đổi mô hình miền.

– Nhìn chung, các đóng góp này là đủ về lượng và chất của một luận án tiến sĩ ngành công nghệ phần mềm.

8. **Nhận xét về ưu điểm và nhược điểm về nội dung, kết cấu và hình thức của luận án:**

Nhìn chung, NCS đã có tinh thần cầu thị để cải tiến đáng kể cách trình bày luận án theo hướng hợp lí hơn như các góp ý của phản biện từ vòng hội thảo và phản biện kín.

Tuy nhiên, luận án còn một số điểm (có thể coi là) tồn tại như sau:

– Một là đặt vấn đề vẫn chưa rõ. Mục 1.1.3 (trang 7) có nêu lên 3 vấn đề con cần giải quyết, nhưng lại thiếu mất bài toán chung, bài toán chính của luận án cần giải quyết. Người đọc phải tự lắp ghép 3 vấn đề riêng lẻ được đề cập đến để xem bài toán chung nhất, chính nhất của luận án là gì, nhưng ghép lại thì 3 vấn đề đặt ra khá rời rạc, mối liện hệ chưa rõ ràng, chưa tạo nên một bức tranh chung, bài toán chung một cách rõ ràng. NCS nên trình bày lại mục đặt vấn đề này, nên bắt đầu bằng bài toán chung/bài toán chính của luận án, mô tả rõ đầu vào, đầu ra là gì, từ đó mới phân rã thành 3 bài toán con là 3 vấn đề đã được trình bày như phiên bản hiện tại

– Hai là, phần đóng góp chính của luận án được trình bày chưa nhất quán. Ở chương mở đầu, mục 1.4, trang 13, luận án liệt kê 4 đóng góp chính. Trong khi đó, ở chương kết luận, trang 176, luận văn lại liệt kê 3 đóng góp chính. Không nhất quán ở số lượng và nội dung các đóng góp chính so với mở đầu. Đề nghị NCS xem xét trình bày lại cho nhất quán.

– Ba là, mặc dù NCS đã có tinh thần cầu thị bổ sung phần khảo sát các phương pháp sử dụng AI cho bài toán của mình. Tuy nhiên, nội dung phần bổ sung này đang quá ngắn và mờ nhạt (vẻn vẹn 1 trang 47, mục 2.5). NCS nên có những khảo sát rộng hơn và những phân tích sâu hơn cho mỗi hướng tiếp cận theo định hướng này. Thậm chí, nếu có thể, nên bổ sung các thực nghiệm vào chương 6 để so sánh kết quả mô hình của luận án đề xuất với các mô hình theo hướng AI này thì kết quả luận án sẽ có tính thuyết phục cao hơn rất nhiều.

– Bốn là, luận án định nghĩa đến 18 định nghĩa hình thức nhưng không thấy rõ vai trò của các khái niệm này trong việc áp dụng vào tính toán trong các phương pháp đề xuất.

– Năm là, nội dung thực nghiệm và đánh giá ở chương 6 chưa đủ tính thuyết phục. Nội dung trình bày chủ yếu là tự đánh giá. Chưa thấy so sánh kết quả của mô hình đề xuất với các mô hình liên quan đã trình bày trong khảo sát ở chương 2.

– Sáu là, về mặt trình bày, bố cục luận án đã có thay đổi nhưng các phần nội dung liên quan chưa được cập nhật theo. Chẳng hạn, chương 6 trình bày các thực nghiệm và đánh giá nhưng kết luận chương (mục 6.4, trang 174) vẫn đnag kết luận là: chương này trình bày việc xây dựng và triển khai các công cụ hỗ trợ...

– Bảy là, tài liệu tham khảo còn thiếu tính cập nhật: mặc dù NCS đã có cố gắng cập nhật lại TLTK so với bản trước đây, tuy nhiên bản hiện tại vẫn chỉ có 28/135 tài liệu được công bố trong vòng 3 năm trở lại đây (2023 – nay); và chỉ có 41/135 tài liệu được công bố trong vòng 5 năm trở lại đây (2021 – nay). NCS nên tiếp tục bổ sung, cập nhật các tài liệu tham khảo mới hơn để tránh bị coi là hướng nghiên cứu không còn thu hút được giới nghiên cứu trong thời gian gần đây.

9. **Đánh giá về sự phù hợp của nội dung luận án với nội dung của các công trình đã được công bố trên tạp chí, tuyển tập công trình hội nghị khoa học; nhận xét về giá trị khoa học của các công trình đã công bố?**

– Luận án liệt kê 7 bài báo, trong đó, 1 bài thuộc tạp chí Q1, 1 bài tạp chí khác, 1 bài mới submit chưa có kết quả, 4 bài báo được công bố trong các hội nghị/hội thảo khoa học quốc tế có phản biện. 5/7 bài báo đã công bố là NCS đứng vị trí tác giả thứ nhất (không bao gồm bài Q1).

– Nhìn chung, nội dung các bài báo đã công bố có liên quan chặt chẽ với nội dung của luận án.

10. **Kết luận về sự đáp ứng các yêu cầu đối với luận án tiến sĩ hay? Luận án có thể đưa ra bảo vệ để nhận học vị tiến sĩ được hay không**?

– Luận án đạt yêu cầu của một luận án tiến sĩ.

– Đồng ý cho phép NCS bảo vệ luận án trước hội đồng.

11. **Kết luận chung cần khẳng định mức độ đáp ứng yêu cầu đối với một luận án tiến sĩ theo quy định của Đại học Quốc gia Hà Nội; Bản tóm tắt luận án có phản ánh trung thành nội dung cơ bản của luận án hay không?**

– Luận án đạt yêu cầu của một luận án tiến sĩ.

– Bản tóm tắt luận án phản ánh sát thực nội dung cơ bản của luận án.

12. **Các ý kiến khác (nếu có).**

- Không.

Hà Nội, ngày tháng năm 2026

XÁC NHẬN CỦA CƠ QUAN/ĐƠN VỊ
TL. GIÁM ĐỐC
KT. CHÁNH VĂN PHÒNG
PHÓ CHÁNH VĂN PHÒNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Lê Thị Cẩm Thuần

NGƯỜI NHẬN XÉT
(Ký và ghi rõ họ tên)

Nguyễn Mạnh Hùng

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BẢN NHẬN XÉT LUẬN ÁN TIẾN SĨ

Tên đề tài luận án: cho thiết kế hướng miền	Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình
Chuyên ngành:	Kỹ thuật phần mềm
Mã số:	9480103.01
Nghiên cứu sinh:	Lê Văn Vinh
Người hướng dẫn:	PGS.TS. Đặng Đức Hạnh
Người phản biện:	TS. Nguyễn Thanh Hùng
Cơ quan công tác:	Trường CNTT&TT - Đại học Bách Khoa Hà Nội.

Ý KIẾN NHẬN XÉT

1. **Tính cần thiết, thời sự, ý nghĩa khoa học và thực tiễn của đề tài luận án:**

   Đề tài luận án có tính cần thiết và thời sự cao trong bối cảnh các hệ thống phần mềm ngày càng gia tăng về quy mô, độ phức tạp và yêu cầu tự động hóa. Thiết kế hướng miền (DDD) là một phương pháp đã được thừa nhận rộng rãi trong thực tiễn, tuy nhiên vẫn tồn tại khoảng cách đáng kể giữa mô hình miền và phần mềm thực thi. Việc nghiên cứu các kỹ thuật biểu diễn mô hình miền giàu ngữ nghĩa và các bộ chuyển đổi mô hình nhằm thu hẹp khoảng cách này là một vấn đề có ý nghĩa khoa học và thực tiễn rõ ràng.

   Luận án đã lựa chọn một vấn đề có ý nghĩa khoa học và thực tiễn cao, phù hợp với xu hướng nghiên cứu hiện nay của Kỹ nghệ phần mềm hướng mô hình (MDSE), DDD và các kỹ thuật sinh tự động phần mềm. Kết quả nghiên cứu có khả năng ứng dụng trong xây dựng các công cụ hỗ trợ thiết kế, kiểm chứng và sinh mã nguồn tự động từ mô hình miền.

   Luận án không chỉ có ý nghĩa về mặt lý thuyết trong lĩnh vực kỹ nghệ phần mềm hướng mô hình và DDD, mà còn có tiềm năng ứng dụng thực tế trong việc nâng cao mức độ tự động hóa phát triển phần mềm, cải thiện chất lượng, khả năng bảo trì và tiến hóa của các hệ thống phần mềm nghiệp vụ phức tạp.

2. **Sự không trùng lặp của kết quả nghiên cứu của luận án với kết quả của các công trình khoa học của các tác giả khác; tính trung thực, rõ ràng và đầy đủ trong trích dẫn tài liệu tham khảo:**

   Các kết quả nghiên cứu trình bày trong luận án không trùng lặp với các công trình khoa học đã công bố trước đó. Tác giả đã khảo sát khá đầy đủ các nghiên cứu liên

quan trong và ngoài nước, chỉ ra rõ ràng những hạn chế, khoảng trống nghiên cứu và định vị đóng góp của mình một cách hợp lý.
Các tài liệu tham khảo được trích dẫn đầy đủ, chính xác, đúng chuẩn khoa học; nguồn gốc các ý tưởng, kết quả kế thừa được phân biệt rõ ràng với các kết quả mới do tác giả đề xuất. Luận án thể hiện tính trung thực khoa học và tuân thủ tốt các quy định về trích dẫn.

3. **Mức độ tổng quan các vấn đề lý luận của đề tài**

Luận án đã thực hiện tổng quan tương đối công phu và có hệ thống về:
Thiết kế hướng miền (DDD);
Ngôn ngữ chuyên biệt miền (DSL);
- Thiết kế hướng miền (DDD);
- Ngôn ngữ chuyên biệt miền (DSL);
- Kỹ nghệ phần mềm hướng mô hình (MDSE);
- UML/OCL;
- Event-B;
- Các phương pháp tích hợp mối quan tâm và chuyển đổi mô hình.

Các hạn chế của các nghiên cứu hiện có được phân tích khá rõ ràng, từ đó hình thành cơ sở cho các bài toán nghiên cứu và các mục tiêu của luận án.
Tuy nhiên, phần tổng quan về các hướng ứng dụng AI/LLM trong mô hình hóa miền và sinh mã nguồn còn tương đối ngắn, chưa phản ánh đầy đủ xu hướng nghiên cứu rất mạnh trong giai đoạn gần đây.

4. **Về sự phù hợp giữa tên đề tài với nội dung, giữa nội dung với chuyên ngành và mã số chuyên ngành**:

Tên đề tài luận án phù hợp chặt chẽ với nội dung nghiên cứu. Các vấn đề được trình bày trong luận án tập trung trực tiếp vào kỹ thuật biểu diễn mô hình, hợp nhất mô hình và chuyển đổi mô hình – đúng với bản chất của chuyên ngành Kỹ thuật phần mềm và mã số 9480103.01.
Nội dung luận án không vượt ra ngoài phạm vi chuyên ngành, đồng thời thể hiện rõ định hướng nghiên cứu chuyên sâu trong lĩnh vực thiết kế hướng miền và kỹ nghệ phần mềm hướng mô hình.

5. **Độ tin cậy và tính hiện đại của phương pháp đã sử dụng để nghiên cứu**

Luận án sử dụng các phương pháp nghiên cứu phù hợp, hiện đại và có độ tin cậy cao, bao gồm:

- Phân tích – tổng hợp tài liệu khoa học;
- Phương pháp siêu mô hình hóa và DSL;
- Kỹ thuật aDSL trong DDD;
- Phương pháp kiểm chứng hình thức (Event-B);
- Thực nghiệm và đánh giá trên các ca nghiên cứu điển hình.

Việc kết hợp các phương pháp mô hình hóa, chuyển đổi mô hình và kiểm chứng hình thức được thực hiện một cách có hệ thống, hợp lý, góp phần nâng cao độ tin cậy của các kết quả nghiên cứu.

Các đề xuất được hiện thực hóa bằng công cụ phần mềm và được đánh giá trên các hệ thống CourseMan, OrderMan, ProcessMan và OJS. Điều này giúp nâng cao độ tin cậy của các kết quả nghiên cứu.

6. **Kết quả nghiên cứu mới của tác giả; đóng góp mới cho sự phát triển khoa học chuyên ngành; ý nghĩa khoa học, giá trị và độ tin cậy của các kết quả**

Luận án đã đạt được những kết quả nghiên cứu mới có giá trị, nổi bật là:

- Đề xuất các kỹ thuật biểu diễn mô hình miền tích hợp cấu trúc, hành vi, ràng buộc và bảo mật trong cùng một khuôn khổ;
- Đề xuất ngôn ngữ mô hình miền hợp nhất UDML có ngữ nghĩa thực thi và khả năng kiểm chứng hình thức;
- Xây dựng các kỹ thuật và bộ chuyển đổi mô hình cho phép sinh tự động bản mẫu phần mềm từ mô hình miền;
- Phát triển công cụ hỗ trợ và đánh giá tính khả thi thông qua nhiều ca nghiên cứu thực tế.

Các kết quả này đóng góp thiết thực cho sự phát triển của chuyên ngành Kỹ thuật phần mềm, đặc biệt trong các hướng DDD, MDE và tự động hóa phát triển phần mềm. Các đóng góp có ý nghĩa khoa học rõ ràng và tiềm năng ứng dụng trong thực tiễn phát triển phần mềm nghiệp vụ.

7. **Ưu điểm và nhược điểm về nội dung, kết cấu và hình thức của luận án**:

Ưu điểm:

- Luận án tập trung giải quyết một bài toán xuyên suốt và có ý nghĩa lâu dài trong Kỹ thuật phần mềm, đó là thu hẹp khoảng cách giữa mô hình miền và phần mềm thực thi trong bối cảnh thiết kế hướng miền (DDD). Các vấn đề nghiên cứu được xác định rõ ràng, xuất phát từ thực tiễn và có cơ sở khoa học vững chắc.
- Nghiên cứu sinh đã khảo sát và phân tích một khối lượng lớn các công trình nghiên cứu trong và ngoài nước liên quan đến DDD, DSL, aDSL, MDE và kiểm chứng hình thức. Việc phân tích hiện trạng nghiên cứu được thực hiện có hệ thống, chỉ ra rõ các hạn chế và khoảng trống khoa học, từ đó làm cơ sở hợp lý cho các đề xuất của luận án.
- Các đóng góp của luận án được trình bày xuyên suốt và nhất quán, bao gồm:
    - Kỹ thuật biểu diễn mô hình miền tích hợp cấu trúc, hành vi, ràng buộc và bảo mật;
    - Ngôn ngữ mô hình miền hợp nhất UDML có ngữ nghĩa thực thi và hỗ trợ kiểm chứng hình thức;
    - Các kỹ thuật chuyển đổi mô hình nhằm sinh tự động bản mẫu phần mềm.

  Những đóng góp này có tính mới, không trùng lặp với các công trình đã công bố trước đó và đóng góp thiết thực cho chuyên ngành.

- Luận án không dừng ở mức đề xuất khái niệm hay mô hình lý thuyết, mà đã phát triển các công cụ hỗ trợ dựa trên các nền tảng hiện đại như JetBrains MPS, Acceleo và khung JDA. Việc hiện thực công cụ cho thấy tính khả thi của các phương pháp đề xuất và nâng cao giá trị thực tiễn của luận án.
- Các kỹ thuật đề xuất được đánh giá thông qua nhiều ca nghiên cứu điển hình thuộc các miền ứng dụng khác nhau như CourseMan, OrderMan, ProcessMan và OJS. Các kết quả thực nghiệm cho thấy phương pháp đề xuất có khả năng áp dụng trong thực tế và mang lại hiệu quả nhất định trong việc nâng cao mức độ tự động hóa phát triển phần mềm.

- Luận án được trình bày theo đúng quy định đối với luận án tiến sĩ, bố cục chặt chẽ, các chương được tổ chức logic, hình vẽ và bảng biểu minh họa rõ ràng, hỗ trợ tốt cho việc trình bày nội dung nghiên cứu.

Một số hạn chế:

- Các đề xuất trong luận án chủ yếu được hiện thực và đánh giá trong bối cảnh các hệ thống hướng đối tượng sử dụng Java, aDSL và các công cụ liên quan. Luận án chưa làm rõ khả năng mở rộng hoặc áp dụng phương pháp cho các ngôn ngữ, nền tảng và kiến trúc phần mềm hiện đại khác như microservices, kiến trúc hướng sự kiện hoặc các hệ thống phân tán quy mô lớn.
- Việc sử dụng đồng thời nhiều DSL (DCSL, AGL, RBACDom), các mẫu CAP, ngôn ngữ hợp nhất UDML và công cụ kiểm chứng hình thức Event-B có thể tạo ra độ phức tạp nhất định trong quá trình tiếp cận và triển khai. Điều này có thể làm tăng đường cong học tập, đặc biệt đối với các nhóm phát triển hoặc chuyên gia miền không có nền tảng sâu về mô hình hóa.
- Mặc dù luận án đã có các so sánh và đánh giá về mức độ biểu đạt và tự động hóa, nhưng các đánh giá định lượng về hiệu năng, chi phí phát triển, chi phí bảo trì hoặc khả năng mở rộng trong các dự án công nghiệp quy mô lớn còn chưa được trình bày đầy đủ.
- Luận án chưa phân tích kỹ các trường hợp biên, chẳng hạn như các hành vi miền có tính bất đồng bộ cao, các ràng buộc nghiệp vụ rất đặc thù hoặc các mô hình bảo mật phức tạp vượt ra ngoài RBAC. Việc làm rõ các giới hạn này sẽ giúp định vị rõ hơn phạm vi áp dụng của phương pháp đề xuất.
- Phần đánh giá thực nghiệm chủ yếu tập trung vào các ca nghiên cứu do nhóm nghiên cứu xây dựng; thiếu đánh giá trên các hệ thống công nghiệp quy mô lớn.

8. **Nội dung luận án với nội dung của các công trình đã được công bố trên tạp chí, tuyển tập công trình hội nghị khoa học; nhận xét về giá trị khoa học của các công trình đã công bố?**

   Nội dung luận án phù hợp và nhất quán với các công trình khoa học mà nghiên cứu sinh đã công bố trên các tạp chí và kỷ yếu hội nghị. Các công trình này có giá trị khoa học, phản ánh đúng hướng nghiên cứu và các kết quả chính của luận án, đồng thời góp phần khẳng định năng lực nghiên cứu độc lập của tác giả.

9. **Kết luận về sự đáp ứng các yêu cầu đối với luận án tiến sĩ hay không đáp ứng? Bản tóm tắt luận án có phản ánh trung thành nội dung cơ bản của luận án hay không? Luận án có thể đưa ra bảo vệ để nhận học vị tiến sĩ được hay không?**

   Luận án đã đề xuất được các kỹ thuật mới về biểu diễn mô hình miền, ngôn ngữ mô hình miền hợp nhất UDML, cơ chế kiểm chứng hình thức và các bộ chuyển đổi mô hình hướng tới sinh tự động phần mềm. Các kết quả nghiên cứu có tính mới, có cơ sở lý thuyết và đã được kiểm chứng bằng thực nghiệm.

   Luận án đáp ứng đầy đủ các yêu cầu đối với một luận án tiến sĩ theo quy định hiện hành. Bản tóm tắt luận án phản ánh trung thực các nội dung cơ bản và các kết quả chính của luận án.

   Luận án đủ điều kiện để đưa ra bảo vệ trước Hội đồng chấm luận án tiến sĩ và nghiên cứu sinh xứng đáng được xem xét cấp học vị Tiến sĩ Kỹ thuật phần mềm.

10. **Các ý kiến khác (nếu có).**

Nghiên cứu sinh có thể xem xét, làm rõ các phần:

- Bổ sung thêm phân tích so sánh định lượng với các nền tảng DDD/MDE hiện đại;
- Làm rõ khả năng tích hợp với các công cụ AI/LLM trong sinh mô hình và sinh mã nguồn;
- Nghiên cứu mở rộng UDML cho các kiến trúc microservices và hệ thống phân tán quy mô lớn trong các công trình tiếp theo.

Hà Nội, ngày 06 tháng 05 năm 2026

XÁC NHẬN CỦA CƠ QUAN CÔNG TÁC

NGƯỜI NHẬN XÉT
(*Ký và ghi rõ họ tên*)

TS. Nguyễn Thanh Hùng

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BẢN NHẬN XÉT LUẬN ÁN TIẾN SĨ

Họ và tên người nhận xét: Nguyễn Việt Hùng ..... Học hàm, học vị: PGS.TS.....

Chuyên ngành: Công nghệ thông tin .....

Cơ quan công tác: Học viện Kỹ thuật quân sự .....

Họ và tên nghiên cứu sinh: Lê Văn Vinh.....

Tên đề tài luận án: Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền.....

Ngành đào tạo: Kỹ thuật phần mềm ..... Mã số: 9480103.....

Ý KIẾN NHẬN XÉT[1]

1. Tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài luận án

Trong kỹ thuật phần mềm, việc thu hẹp khoảng cách giữa mô hình thiết kế và mã nguồn thực thi là một bài toán kinh điển nhưng vẫn đầy thách thức. Luận án đóng góp trực tiếp vào lý thuyết của hai lĩnh vực nền tảng: Thiết kế hướng miền (DDD) và kỹ nghệ phần mềm hướng mô hình (MDSE). NCS tập trung giải quyết thách thức trong việc hợp nhất các khía cạnh không đồng nhất như cấu trúc, hành vi và bảo mật vào một mô hình định nghĩa duy nhất. Điều này thúc đẩy lý thuyết kỹ nghệ hướng mô hình (MDSE) chuyển dịch từ vai trò mô tả tĩnh sang các thực thể có khả năng thực thi và tính toán. Việc thiết lập cơ sở ngữ nghĩa hình thức cho mô hình còn cho phép kiểm chứng toán học tính đúng đắn của hệ thống ngay từ giai đoạn thiết kế, giúp ngăn chặn các xung đột logic từ sớm. Những nghiên cứu này đều có ý nghĩa khoa học, đóng góp cho sự phát triển của ngành kỹ thuật phần mềm.

Nghiên cứu cung cấp công cụ để quản trị độ phức tạp và các quy tắc nghiệp vụ chồng chéo trong các hệ thống quy mô lớn. Kết quả của luận án hướng tới đảm bảo sự thống nhất về ngôn ngữ chung giữa chuyên gia nghiệp vụ và lập trình viên, loại bỏ tình trạng hiểu sai yêu cầu. Trọng tâm về chuyển đổi mô hình là tiền đề quan trọng cho việc sinh mã tự động, giúp tăng năng suất lao động và rút ngắn chu kỳ phát triển sản phẩm, có ý nghĩa thực tiễn lớn.

2. Sự không trùng lặp của đề tài nghiên cứu so với các công trình, luận văn, luận án đã công bố trong và ngoài nước; tính trung thực, rõ ràng và đầy đủ trong trích dẫn tài liệu tham khảo

Theo hiểu biết của người nhận xét, đề tài nghiên cứu không trùng lặp so với các công trình, luận văn, luận án đã công bố. NCS tham khảo 181 tài liệu, có trích dẫn đầy đủ cho

thấy sự trung thực và nghiêm túc trong nghiên cứu. Tuy nhiên, số lượng tài liệu tham khảo hơi nhiều.

3. Mức độ tổng quan các vấn đề lý luận của đề tài

Phần tổng quan các vấn đề lý luận của đề tài đã hệ thống hóa được các cơ sở lý luận liên quan, bao gồm thiết kế hướng miền (DDD), kỹ nghệ hướng mô hình (MDSE), ngôn ngữ chuyên biệt miền (DSL) và phương pháp hình thức (Event-B). Tác giả chỉ ra được các điểm hạn chế trong những nghiên cứu hiện hành, đặc biệt là sự thiếu hụt một cơ sở ngữ nghĩa thống nhất nhằm hợp nhất cấu trúc, hành vi, ràng buộc và bảo mật vào cùng một mô hình miền thực thi. Việc xác định được khoảng trống nghiên cứu này tạo cơ sở cần thiết để dẫn dắt đến các giải pháp kỹ thuật được đề xuất như AGL, CAP và UDML ở các chương tiếp theo. Tuy nhiên, do đề tài đề cập đến nhiều khía cạnh lý thuyết khác nhau, nội dung phần tổng quan đôi chỗ còn dàn trải và cần sự cô đọng hơn để bám sát trọng tâm cốt lõi. Bên cạnh đó, việc khảo sát xu hướng ứng dụng mô hình ngôn ngữ lớn (LLMs) trong sinh mô hình và mã nguồn (mục 2.5) mới dừng ở mức cơ bản, chưa thảo luận chi tiết sự tác động của công nghệ này đối với các phương pháp siêu mô hình truyền thống. Nhìn chung, phần tổng quan các vấn đề lý luận đã cung cấp đủ thông tin nền tảng và khung lập luận để hỗ trợ cho việc triển khai các hướng nghiên cứu của luận án.

4. Sự phù hợp giữa tên đề tài với nội dung, giữa nội dung với chuyên ngành và mã số ngành

Tên luận án phù hợp với nội dung thực hiện, và phù hợp vói chuyên ngành, mã số của ngành.

5. Độ tin cậy và tính hiện đại của phương pháp đã sử dụng để nghiên cứu

Phương pháp nghiên cứu của luận án dựa trên sự kết hợp giữa kỹ nghệ phần mềm hướng mô hình (MDSE) và thiết kế hướng miền (DDD), là những hướng tiếp cận mang tính hệ thống trong công nghệ phần mềm hiện nay. Độ tin cậy của các đề xuất được củng cố bằng việc sử dụng các tiêu chuẩn siêu mô hình hóa như MOF/Ecore để định nghĩa chính xác cú pháp và ngữ nghĩa cho ngôn ngữ UDML. Tác giả đã áp dụng phương pháp hình thức Event-B và nền tảng Rodin để kiểm chứng các nghĩa vụ chứng minh, giúp đảm bảo tính đúng đắn về mặt logic của các mô hình miền hợp nhất. Việc hiện thực hóa các bộ chuyển đổi mô hình thông qua các công cụ như ATL và Acceleo cho thấy tính thực tiễn và sự tương thích với các quy trình kỹ nghệ phần mềm hiện đại.

6. Kết quả nghiên cứu mới của tác giả; đóng góp mới cho sự phát triển khoa học chuyên ngành; đóng góp mới phục vụ sản xuất, kinh tế, xã hội, an ninh, quốc phòng và đời sống. Ý nghĩa khoa học, giá trị và độ tin cậy của những kết quả đó

NCS có các đóng góp mới:

(1) Đề xuất các kỹ thuật biểu diễn mở rộng (AGL, RBACDom, CAP);

(2) Ngôn ngữ UDML hợp nhất;

(3) Bộ chuyển đổi mô hình (RM2UDM);

Ngôn ngữ mô hình miền hợp nhất (UDML), cho phép tích hợp các khía cạnh không đồng nhất này vào một cây cú pháp trừu tượng (AST) và siêu mô hình duy nhất. Kết quả này giúp giải quyết sự phân mảnh trong Thiết kế hướng miền (DDD) truyền thống, đồng thời cung cấp cơ sở ngữ nghĩa hình thức để thu hẹp khoảng cách giữa kỹ nghệ hướng mô hình (MDSE) và mã nguồn thực thi.

Đối với ý nghĩa phục vụ sản xuất, kinh tế, xã hội: Phát triển bộ chuyển đổi mô hình (RM2UDM) hỗ trợ sinh tự động các bản mẫu phần mềm, giúp doanh nghiệp rút ngắn chu kỳ phát triển, giảm thiểu sai sót do lập trình thủ công và tối ưu hóa chi phí. Đối với lĩnh

vực an ninh và quốc phòng, việc tích hợp trực tiếp cơ chế kiểm soát truy cập (RBAC) và phân tách nhiệm vụ (SoD) vào ranh giới hành vi của mô hình thiết kế tạo ra cách tiếp cận "bảo mật từ gốc". Phương pháp này cung cấp một công cụ hữu ích để xây dựng và kiểm soát các hệ thống quản lý thông tin nhạy cảm, đòi hỏi độ an toàn và phân quyền khắt khe.

7. Ưu điểm và nhược điểm về nội dung, kết cấu và hình thức của luận án

Ưu điểm:

Luận án được tổ chức thành 7 chương chặt chẽ, đi từ lý thuyết nền tảng đến đề xuất kỹ thuật, thực nghiệm và kết luận. Luận án có bố cục tương đối hợp lý, các chương đề xuất kỹ thuật (chương 3, 4, 5) được phân bổ khối lượng kiến thức đồng đều.

Luận án đã giải quyết được vấn đề phân mảnh trong thiết kế phần mềm bằng cách tích hợp đồng thời bốn khía cạnh then chốt: cấu trúc, hành vi, ràng buộc nghiệp vụ và bảo mật vào một mô hình miền duy nhất. Việc sử dụng các phương pháp hình thức (Event-B) và siêu mô hình hóa (MOF/Ecore) giúp các đề xuất không dừng lại ở mức ý tưởng kỹ thuật mà có nền tảng toán học để kiểm chứng tính đúng đắn, thể hiện tính chặt chẽ của luận án.

Tính thực tiễn cao: Tác giả không chỉ dừng lại ở lý thuyết mà đã hiện thực hóa bằng bộ chuyển đổi mô hình (RM2UDM) và thực nghiệm trên các hệ thống thực tế như OJS hay CourseMan.

Luận án có văn phong khoa học,sử dụng thuật ngữ chuyên ngành thống nhất, nhất quán xuyên suốt luận án.

Các biểu đồ UML, siêu mô hình và đặc tả OCL được trình bày rõ ràng, hỗ trợ tốt cho việc diễn giải các khái niệm trừu tượng.

Nhược điểm:

- Nội dung tổng quan còn khá dàn trải, chương 2 bao hàm quá nhiều lĩnh vực từ DDD, MDE đến Event-B và RBAC. Việc cố gắng bao quát tất cả khiến một số phần thiếu đi sự đào sâu vào các nghiên cứu mới nhất (State-of-the-art) trong vòng 2 năm gần đây.
- Khoảng cách với chuyên gia nghiệp vụ: Mặc dù hướng tới "ngôn ngữ chung" (Ubiquitous Language) của DDD, nhưng các cú pháp của AGL hay CAP vẫn mang đậm tính kỹ thuật (Java/Annotation). Điều này đặt ra câu hỏi liệu các chuyên gia nghiệp vụ không biết lập trình có thực sự thẩm định được mô hình như triết lý DDD mong muốn hay không.
- Phạm vi kiến trúc giới hạn: Nghiên cứu tập trung mạnh vào kiến trúc hướng mô-đun (MOSA) và khung JDA. Tính thích ứng của phương pháp đối với các kiến trúc hiện đại khác như Microservices, Serverless hay các hệ thống phân tán đa ngôn ngữ (Polyglot) chưa được thảo luận sâu.
- Chưa có các khảo sát, đánh giá chi tiết về ứng dụng AI/LLM trong quy trình. Trong bối cảnh ứng dụng AI mạnh mẽ như hiện nay, nếu bổ sung được các đánh giá về cách tích hợp AI để tự động hóa việc sinh siêu mô hình hoặc kiểm chứng mô hình sẽ làm hoàn thiện hơn nữa luận án.
- Một số hình vẽ siêu mô hình (như Hình 4.2 hoặc 4.6) có mật độ chi tiết quá dày đặc, gây khó khăn cho việc nắm bắt các mối quan hệ cốt lõi nếu không phóng đại hình ảnh.
- Việc nhúng trực tiếp các đoạn mã aDSL (như trong Chương 3 và 4) đôi khi làm ngắt quãng luồng tư duy lý thuyết của người đọc; nên cân nhắc đưa một phần vào phụ lục để văn bản chính cô đọng hơn.

8. Nội dung luận án đã được công bố trên tạp chí, tuyển tập công trình hội nghị khoa học và giá trị khoa học của những công trình đã công bố

Luận án có danh mục công bố gồm 06 công trình khoa học chuyên ngành (01 công trình đã gửi và chưa có kết quả phản biện). Trong đó, có 01 công trình đăng trên tạp chí thuộc danh mục ISI/Scopus (Information and Software Technology). Mặc dù ở công trình này NCS không phải tác giả chính, nhưng các công trình còn lại trên tạp chí VNU Journal of Science và 4 hội nghị quốc tế và quốc gia uy tín (KSE, RIVF, SOICT, FAIR) mà NCS là tác giả chính đã phản ánh đầy đủ và trung thực các đóng góp mới của luận án, đáp ứng định mức công bố theo Quy chế 4555.

9. Luận án có đáp ứng các yêu cầu đối với luận án tiến sĩ không? Luận án có thể đưa ra Hội đồng đánh giá luận án tiến sĩ được hay không

Với các kết quả đạt được, luận án đáp ứng các yêu cầu đối với luận án tiến sĩ, có thể đưa ra Hội đồng đánh giá luận án tiến sĩ cấp ĐH Quốc gia Hà Nội.

10. Kết luận chung cần khẳng định mức độ đáp ứng yêu cầu đối với một luận án tiến sĩ theo quy định của Đại học Quốc gia Hà Nội; bản tóm tắt luận án phản ảnh trung thành nội dung cơ bản của luận án hay không

Luận án TS của NCS đáp ứng cơ bản các yêu cầu đối với một luận án tiến sĩ theo quy định của Đại học Quốc gia Hà Nội; bản tóm tắt luận án phản ánh trung thành nội dung cơ bản của luận án.

11. Các ý kiến khác (nếu có).

Hà Nội, ngày 5 tháng 5 năm 2026

NGƯỜI NHẬN XÉT

PGS.TS. Nguyễn Việt Hùng

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BẢN NHẬN XÉT LUẬN ÁN TIẾN SĨ

Họ và tên người nhận xét: Cao Tuấn Dũng Học hàm, học vị: PGS.TS .....

Chuyên ngành: Công nghệ thông tin .....

Cơ quan công tác: Đại học Bách Khoa Hà nội .....

Họ và tên nghiên cứu sinh: Lê Văn Vinh .....

Tên đề tài luận án: NGHIÊN CỨU CÁC KỸ THUẬT BIỂU DIỄN VÀ CHUYỂN ĐỔI MÔ HÌNH CHO THIẾT KẾ HƯỚNG MIỀN .....

Ngành đào tạo: Kỹ thuật phần mềm ..... Mã số: 9480103 .....

Ý KIẾN NHẬN XÉT[1]

1. Tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài luận án

Trong bối cảnh quy mô và độ phức tạp của các hệ thống phần mềm hiện đại ngày càng gia tăng, việc thu hẹp khoảng cách ngữ nghĩa giữa mô hình thiết kế và hệ thống thực thi trở thành một thách thức cốt lõi trong kỹ nghệ phần mềm. Các tiếp cận hiện tại theo nguyên lý thiết kế hướng miền vẫn gặp hạn chế trong việc biểu diễn thống nhất các khía cạnh hành vi và ràng buộc phức tạp, dẫn đến sự phân mảnh tri thức và thiếu cơ sở hình thức để kiểm chứng tính đúng đắn của mô hình. Do đó, việc nghiên cứu các kỹ thuật biểu diễn hợp nhất và chuyển đổi mô hình bảo toàn ngữ nghĩa là yêu cầu cấp thiết nhằm nâng cao hiệu quả và chất lượng trong phát triển phần mềm hướng miền. Luận án có ý nghĩa khoa học ở việc đề xuất một mô hình miền hợp nhất (UDML) với nền tảng ngữ nghĩa hình thức, cho phép biểu diễn nhất quán các khía cạnh cấu trúc, hành vi, ràng buộc và bảo mật trong thiết kế hướng miền. Đồng thời, nghiên cứu phát triển các kỹ thuật tích hợp DSL và chuyển đổi mô hình có bảo toàn ngữ nghĩa, góp phần hoàn thiện cơ sở lý thuyết cho phát triển phần mềm hướng mô hình và thu hẹp khoảng cách giữa mô hình và hệ thống thực thi. Nghiên cứu góp phần tối ưu hóa nguồn lực, giảm thiểu chi phí sản xuất và nâng cao chất lượng cũng như khả năng bảo trì cho các hệ thống phần mềm phức tạp trong thực tế.

2. Sự không trùng lặp của đề tài nghiên cứu so với các công trình, luận văn, luận án đã công bố trong và ngoài nước; tính trung thực, rõ ràng và đầy đủ trong trích dẫn tài liệu tham khảo

Theo sự hiểu biết của người nhận xét, các kết quả trong luận án không bị trùng lặp với các công trình đã công bố trong và ngoài nước. Các nghiên cứu liên quan được trích dẫn đầy đủ trong 135 tài liệu tham khảo.

3. Mức độ tổng quan các vấn đề lý luận của đề tài

Luận án đã thực hiện tổng quan khá đầy đủ và có hệ thống các vấn đề lý luận liên quan đến thiết kế hướng miền (DDD), và kỹ nghệ phần mềm hướng mô hình, thể hiện qua việc phân tích nhiều hướng tiếp cận khác nhau về biểu diễn và chuyển đổi mô hình. Tác giả không chỉ mô tả mà còn đánh giá rõ các hạn chế của từng hướng (UML/OCL, DSL..), đặc biệt là sự phân mảnh mô hình và thiếu cơ chế bảo toàn ngữ nghĩa. Trên cơ sở đó, luận án xác định được khoảng trống nghiên cứu một cách logic, làm rõ nhu cầu về một mô hình miền hợp nhất có khả năng thực thi. Nhìn chung, phần tổng quan có chiều sâu, bám sát mục tiêu nghiên cứu và tạo nền tảng lý thuyết vững chắc cho các đề xuất của luận án.

4. Sự phù hợp giữa tên đề tài với nội dung, giữa nội dung với chuyên ngành và mã số ngành

Tên đề tài phù hợp với nội dung nghiên cứu của luận án. Nội dung luận án hoàn toàn phù hợp với chuyên ngành Kỹ thuật phần mềm và mã số ngành.

5. Độ tin cậy và tính hiện đại của phương pháp đã sử dụng để nghiên cứu

Luận án sử dụng phương pháp nghiên cứu kết hợp giữa phân tích lý thuyết, mô hình hóa, xây dựng ngôn ngữ và kiểm chứng hình thức, kèm theo thực nghiệm trên các ca nghiên cứu điển hình, thuộc hướng nghiên cứu thiết kế – xây dựng trong kỹ nghệ phần mềm . Phương pháp này có căn cứ khoa học rõ ràng vì dựa trên các nền tảng hiện đại như DDD, DSL, kỹ nghệ hướng mô hình và kiểm chứng hình thức, đồng thời được kiểm nghiệm thông qua công cụ và thực nghiệm. Việc kết hợp giữa chứng minh lý thuyết và đánh giá thực nghiệm giúp tăng độ tin cậy của kết quả nghiên cứu. Nhìn chung, phương pháp nghiên cứu là hiện đại, phù hợp với lĩnh vực và đảm bảo tính khoa học cũng như khả năng áp dụng thực tế.

6. Kết quả nghiên cứu mới của tác giả; đóng góp mới cho sự phát triển khoa học chuyên ngành; đóng góp mới phục vụ sản xuất, kinh tế, xã hội, an ninh, quốc phòng và đời sống. Ý nghĩa khoa học, giá trị và độ tin cậy của những kết quả đó

Luận án đã đóng góp một số kết quả nghiên cứu vào lĩnh vực nghiên cứu kỹ thuật phần mềm, cụ thể là:

- Đề xuất các kỹ thuật biểu diễn nhằm xây dựng mô hình miền đầy đủ thông tin cho thiết kế hướng miền (DDD), tích hợp hành vi miền và các ràng buộc OCL, khắc phục tình trạng phân mảnh mô hình miền.
- Đề xuất phương pháp cho phép tích hợp đồng thời các mối quan tâm chuyên biệt (về cấu trúc, hành vi, bảo mật) vào một mô hình miền thống nhất, có khả năng thực thi.
- Phát triển các kỹ thuật chuyển đổi mô hình có bảo toàn ngữ nghĩa, hỗ trợ sinh tự động bản mẫu phần mềm từ mô hình miền .
- Xây dựng công cụ thực nghiệm để kiểm chứng và đánh giá các phương pháp đề xuất.

Các kết quả của luận án góp phần hoàn thiện cơ sở lý thuyết cho thiết kế hướng miền và kỹ nghệ phần mềm hướng mô hình, đặc biệt trong việc xây dựng mô hình miền hợp nhất có ngữ nghĩa rõ ràng và khả năng thực thi. Đồng thời, các kết quả này giúp thu hẹp khoảng cách giữa mô hình và triển khai, nâng cao tính nhất quán, tự động hóa và giá trị ứng dụng trong phát triển phần mềm.

7. Ưu điểm và nhược điểm về nội dung, kết cấu và hình thức của luận án

Luận án có ưu điểm là nội dung nghiên cứu có tính hệ thống, bám sát mục tiêu và giải quyết rõ khoảng trống khoa học đã đặt ra. Kết cấu luận án logic, các chương liên kết chặt chẽ từ cơ sở lý thuyết đến đề xuất và thực nghiệm

Một số điểm hạn chế:

- Kỹ thuật CAP hiện tại chủ yếu hỗ trợ các ràng buộc cấu trúc và định lượng (bất biến), nhưng chưa bao quát được các nhóm ràng buộc mang tính hành vi như tiền điều kiện/hậu điều kiện, kiểm soát truy cập phức tạp dựa trên ngữ cảnh, hay các phản ứng tự động của hệ thống. Do đó, phạm vi bao phủ của ràng buộc OCL còn giới hạn.
- Các ví dụ nghiên cứu minh họa (CourseMan, OJS, OrderMan) dù có tính thực tiễn nhưng vẫn ở quy mô trung bình. Khả năng mở rộng và hiệu năng của phương pháp trên các hệ thống công nghiệp quy mô lớn với hàng nghìn thực thể chưa được đánh giá một cách định lượng đầy đủ.

8. Nội dung luận án đã được công bố trên tạp chí, tuyển tập công trình hội nghị khoa học và giá trị khoa học của những công trình đã công bố

Các kết quả trong luận án đã được công bố qua 06 công trình trong đó có 05 công trình là tác giả đứng tên đầu, gồm 01 bài đăng tạp chí Wos – 01 bài đăng tạp chí trong nước, Q1 và 04 bài đăng tại hội nghị thuộc danh mục SCOPUS. Ngoài ra, một bài báo đang trong giai đoạn chờ kết quả bình duyệt. Các công trình đều được đăng trong các tạp chí, kỷ yếu hội nghị phù hợp với chuyên ngành nghiên cứu.

9. Luận án có đáp ứng các yêu cầu đối với luận án tiến sĩ không? Luận án có thể đưa ra Hội đồng đánh giá luận án tiến sĩ được hay không

Nội dung của luận án đáp ứng được yêu cầu về nội dung và hình thức của của một luận án Tiến sĩ và có thể đưa ra đánh giá tại hội đồng.

10. Kết luận chung cần khẳng định mức độ đáp ứng yêu cầu đối với một luận án tiến sĩ theo quy định của Đại học Quốc gia Hà Nội; bản tóm tắt luận án phản ảnh trung thành nội dung cơ bản của luận án hay không

Luận án ĐẠT yêu cầu của một luận án tiến sĩ theo quy định của Đại học Quốc gia Hà Nội; bản tóm tắt luận án phản ảnh trung thực nội dung cơ bản của luận án.

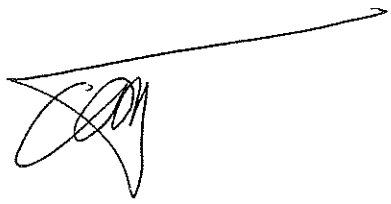
11. Các ý kiến khác (nếu có).

.....

Hà Nội, ngày 3 tháng 5 năm 2026...

XÁC NHẬN CỦA CƠ QUAN CÔNG TÁC

NGƯỜI NHẬN XÉT



Cao Tuấn Dũng

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BẢN NHẬN XÉT LUẬN ÁN TIẾN SĨ

Họ và tên người nhận xét: Trần Đăng Hưng Học hàm, học vị: PGS.TS
Chuyên ngành: Khoa học máy tính
Cơ quan công tác: Trường CNTT&TT, ĐH Công nghiệp Hà Nội
Họ và tên nghiên cứu sinh: Lê Văn Vinh
Tên đề tài luận án: Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền
Ngành đào tạo: Kỹ thuật phần mềm Mã số: 9480103

Ý KIẾN NHẬN XÉT

1. Tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài luận án

Sự phát triển của các hệ thống phần mềm lớn, đa nền tảng và thường xuyên thay đổi làm gia tăng nhu cầu về các phương pháp phát triển phần mềm có khả năng tái sử dụng và tự động hóa cao. Trong bối cảnh đó, thiết kế hướng miền và phát triển phần mềm hướng mô hình ngày càng được quan tâm nhằm hỗ trợ mô hình hóa tri thức miền và giảm công sức phát triển thủ công.

Tuy nhiên, việc biểu diễn và chuyển đổi mô hình giữa các mức trừu tượng, công cụ và nền tảng khác nhau vẫn còn nhiều khó khăn, ảnh hưởng đến tính nhất quán, khả năng tái sử dụng và bảo trì hệ thống. Vì vậy, nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền là cần thiết nhằm nâng cao hiệu quả phát triển phần mềm và hỗ trợ xây dựng các hệ thống phần mềm linh hoạt, dễ mở rộng và thích nghi tốt hơn với yêu cầu thay đổi. Do đó, chủ đề nghiên cứu của luận án có ý nghĩa khoa học và phù hợp với thực tiễn phát triển của chuyên ngành.

2. Sự không trùng lặp của đề tài nghiên cứu so với các công trình, luận văn, luận án đã công bố trong và ngoài nước; tính trung thực, rõ ràng và đầy đủ trong trích dẫn tài liệu tham khảo

Luận án trích dẫn 135 tài liệu tham khảo, các tài liệu tham khảo được trích dẫn đầy đủ và trung thực. Người đọc chưa phát hiện có sự trùng lặp so với các tà liệu đã công bố trong và ngoài nước.

3. **Sự phù hợp giữa tên đề tài với nội dung, giữa nội dung với chuyên ngành và mã số ngành**

Nội dung của đề tài luận án phù hợp với chuyên ngành và mã số chuyên ngành đào tạo.

4. **Kết quả nghiên cứu mới của tác giả; đóng góp mới cho sự phát triển khoa học chuyên ngành; đóng góp mới phục vụ sản xuất, kinh tế, xã hội, an ninh, quốc phòng và đời sống. Ý nghĩa khoa học, giá trị và độ tin cậy của những kết quả đó**

Luận án có các đóng góp chính sau đây:

- Đề xuất được kỹ thuật biểu diễn mô hình miền theo hướng tích hợp nhiều khía cạnh của hệ thống, bao gồm cấu trúc dữ liệu, hành vi, các ràng buộc nghiệp vụ và yêu cầu bảo mật. Đề xuất này cho phép mô hình miền phản ánh đầy đủ hơn tri thức nghiệp vụ, đồng thời tạo nền tảng cho việc tự động hóa các bước trong quá trình phát triển phần mềm hướng miền.
- Đề xuất một kỹ thuật hợp nhất các DSL theo các mối quan tâm chuyên biệt vào một mô hình miền hợp nhất có khả năng thực thi. Trên cơ sở đó, xây dựng nền tảng ngữ nghĩa phục vụ kiểm chứng hình thức nhằm đảm bảo tính nhất quán ngữ nghĩa của mô hình miền hợp nhất ngay từ giai đoạn thiết kế.
- Đề xuất các kỹ thuật chuyển đổi mô hình phục vụ phát triển phần mềm theo hướng tự động hóa. Các kỹ thuật này cho phép chuyển đổi từ mô hình miền sang các thành phần triển khai và bản mẫu phần mềm, qua đó giảm khối lượng lập trình thủ công, nâng cao tính tái sử dụng và đảm bảo sự nhất quán giữa các giai đoạn phân tích, thiết kế và cài đặt hệ thống.
- Xây dựng công cụ thực nghiệm và tiến hành đánh giá trên nhiều hệ thống ứng dụng khác nhau như CourseMan, OrderMan, ProcessMan và OJS. Các kết quả thực nghiệm cho thấy các phương pháp và kỹ thuật được đề xuất có tính khả thi, có khả năng áp dụng trong thực tiễn và hỗ trợ hiệu quả cho quá trình phát triển phần mềm hướng miền.

5. **Ưu điểm và nhược điểm về nội dung, kết cấu và hình thức của luận án**

Ưu điểm:

- Đề tài có tính thời sự và phù hợp với xu hướng phát triển phần mềm hiện đại theo hướng miền và hướng mô hình.
- Nội dung luận án được trình bày công phu, trong sáng và dễ hiểu;
- Luận án xây dựng được cách tiếp cận tương đối hệ thống, bao quát từ biểu diễn mô hình miền, hợp nhất các DSL, xây dựng nền tảng ngữ nghĩa cho kiểm chứng hình thức đến các kỹ thuật chuyển đổi mô hình và thực nghiệm.

Bên cạnh các ưu điểm, một vài điểm sau đây có thể cân nhắc thêm:

- Cần làm rõ hơn cơ chế xử lý xung đột ngữ nghĩa giữa các DSL chuyên biệt sau khi hợp nhất. Đây là vấn đề khá quan trọng vì các DSL thuộc các mối quan tâm khác nhau có thể phát sinh các ràng buộc không nhất quán.
- Nên bổ sung thêm các tình huống minh họa và cơ chế phát hiện, xử lý xung đột ngữ nghĩa nhằm tăng tính chặt chẽ và tính khả thi của mô hình đề xuất.
- Phần kiểm chứng hình thức dựa trên Event-B hiện mới tập trung vào cơ chế ánh xạ mô hình, trong khi chưa phân tích sâu tính bảo toàn ngữ nghĩa của phép chuyển đổi.
- Luận án đã đề xuất kỹ thuật sinh mã từ mô hình, tuy nhiên chưa đánh giá đầy đủ chất lượng của mã nguồn được sinh tự động; nên bổ sung các tiêu chí đánh giá như khả năng bảo trì, mở rộng và tối ưu mã nguồn nhằm tăng tính thuyết phục cho giải pháp đề xuất.

6. Nội dung luận án đã được công bố trên tạp chí, tuyển tập công trình hội nghị khoa học và giá trị khoa học của những công trình đã công bố

Nội dung của luận án đã được công bố trong 6 công trình, trong đó có 01 bài ISI Q1; 3 bài hội nghị quốc tế có phản biện; 2 bài tạp chí và hội nghị trong nước; ngoài ra còn có 01 bản thảo đang gửi tạp chí quốc tế. Nội dung của các bài báo phản ánh nội dung luận án. Về xuất bản luận án đạt mức khá.

7. Kết luận

a. Luận án đáp ứng đầy đủ yêu cầu về nội dung và hình thức đối với một luận án TS theo quy định của ĐHQGHN.

b. Bản tóm tắt phản ánh trung thực nội dung luận án.

c. Đồng ý cho NCS đưa luận án ra bảo vệ tại HĐ cấp ĐHQG để nhận học vị TS.

Hà Nội, ngày 4 tháng 5 năm 2026

Người nhận xét

Trần Đăng Hưng

Ngày ký: 07/05/2026 11:18
Người ký: Trần Đăng Hưng
Nơi ký: Đại học Công nghiệp Hà Nội

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Hà Nội, ngày 16 tháng 12 năm 2025

QUYẾT NGHỊ
CỦA HỘI ĐỒNG ĐÁNH GIÁ LUẬN ÁN TIẾN SĨ

Căn cứ Quyết định số 721/QĐ-ĐHCN ngày 09/4/2026 của Hiệu trưởng Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội về việc thành lập Hội đồng đánh giá luận án tiến sĩ, Hội đồng đánh giá luận án tiến sĩ của nghiên cứu sinh Lê Văn Vinh, sinh ngày 05/5/1982 đã họp vào hồi 08h30 ngày 20/6/2026 tại Trường Đại học Công nghệ.

Tên đề tài luận án: Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền

Ngành đào tạo: Kỹ thuật phần mềm Mã số: 9480103

Sau khi nghe nghiên cứu sinh (NCS) trình bày kết quả phản biện độc lập luận án, trình bày tóm tắt luận án tiến sĩ; các phản biện đọc nhận xét; Thư ký Hội đồng đọc bảng tổng hợp các nhận xét khác; NCS trả lời các câu hỏi của Hội đồng và đại biểu tham dự; đại diện tập thể hướng dẫn đọc nhận xét, Hội đồng đã họp, trao đổi ý kiến và thống nhất kết luận như sau:

1. Về ý nghĩa khoa học của luận án, các kết quả của luận án:

Luận án đề xuất pp hợp biến diễn và chuyển đổi mô hình trong DDD nhằm thu hẹp khoảng cách giữa thiết kế và thi công thực thi. NCS đề xuất mô hình miền để tích hợp toàn diện các ý tưởng thiết kế và xây dựng hoặc OCL. Nghiên cứu xây dựng ngôn ngữ mô hình miền hợp nhất hợp có hệ thống các DSL. Từ đó phát triển bộ chuyển đổi mô hình để tự động hóa trong phát triển phần mềm. Đây là những đóng góp có ý nghĩa khoa học và thực tế cao trong ngành CNPM

2. Về tính hiện đại, hợp lý và độ tin cậy của phương pháp nghiên cứu mà NCS đã sử dụng:

Mục tiêu của luận án được phát biểu rõ ràng, khoa học. NCS đã khảo sát các IVC liên quan, tìm ra khoảng trống nghiên cứu để xuất mô hình, tiến hành thực nghiệm để đánh giá hiệu quả của các mô hình đề xuất. Phương pháp nghiên cứu của NCS là phù hợp với nội dung nghiên cứu, có độ tin cậy cao, có tính hiện đại.

3. Về các kết quả mới của luận án. Giá trị của các kết quả này trong lĩnh vực khoa học chuyên ngành về mặt lý thuyết và ứng dụng:

Luận án đạt được các kết quả chính như sau:

- Đề xuất phương pháp biểu diễn mô hình miền có khả năng thực thi, cho phép tích hợp các khía cạnh cấu trúc, hành vi và tăng cường bảo mật, đồng thời xây dựng nền tảng ngữ nghĩa và có thể kiểm chứng hình thức cho mô hình
- Đề xuất ngôn ngữ mô hình miền bậc nhất (UML), được xây dựng dựa trên tiếp cận siêu mô hình và cây cú pháp trừu tượng, được kiểm chứng bằng Event-B
- Đề xuất các kỹ thuật chuyển đổi mô hình cho phép sinh tự động các bản mẫu phần mềm từ mô hình miền, bảo toàn ngữ nghĩa
- Các kết quả trên là có ý nghĩa khoa học, đóng góp cho CNPM

4. Những thiếu sót về nội dung và hình thức của luận án:

- Luận án vẫn còn tồn tại một số lỗi về văn phong khoa học
- Bổ sung các khảo sát, thảo luận về xu hướng sử dụng AI trong bài toán liên quan, đặc biệt về AI gần đây
- Các đóng góp của luận án cần được trình bày nhất quán về nội dung hình thức, văn phong
- Bổ sung thêm các khả năng mở rộng theo góp ý của hội đồng

5. Kết quả kiểm phiếu

- Số phiếu hợp lệ: ..06.. phiếu
- Số phiếu không hợp lệ: ..0.. phiếu
- Số phiếu đánh giá ĐẠT: ..06.. phiếu
- Số phiếu đánh giá KHÔNG ĐẠT: ..0.. phiếu

6. Kết luận của Hội đồng:

- Luận án đáp ứng yêu cầu của luận án tiến sĩ chuyên ngành kỹ thuật phần mềm
- NCS cần chỉnh sửa theo các góp ý chi tiết của hội đồng
- Bổ sung các nghiên cứu liên quan trong 5 năm gần đây
- chỉnh sửa luận án để đảm bảo tính nhất quán
- Bổ sung thảo luận về khả năng mở rộng của LA.

7. Đề nghị khen thưởng (nếu luận án đặc biệt xuất sắc)

.....

.....

.....

.....

Quyết nghị này được 06/06 / 07 thành viên của Hội đồng nhất trí thông qua./.

THƯ KÝ HỘI ĐỒNG	CHỦ TỊCH HỘI ĐỒNG
Trần Hoàng Việt	Phan Ngọc Hưng

XÁC NHẬN CỦA ĐƠN VỊ ĐÀO TẠO
TL. HIỆU TRƯỞNG

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ – ĐẠI HỌC QUỐC GIA HÀ NỘI

KT. TRƯỞNG PHÒNG CÔNG TÁC SINH VIÊN
PHÓ TRƯỞNG PHÒNG

Nguyễn Huy Tiệp

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

THÔNG TIN VỀ LUẬN ÁN TIẾN SĨ

1. Họ và tên nghiên cứu sinh: Lê Văn Vinh 2. Giới tính: Nam

3. Ngày sinh: 05/05/1982 4. Nơi sinh: Nghệ An

5. Quyết định công nhận nghiên cứu sinh số: 25/QĐ-CTSV, ngày 14 tháng 01 năm 2022 của Hiệu trưởng Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội.

6. Các thay đổi trong quá trình đào tạo:

- Quyết định số 2833/QĐ-ĐHCN ngày 03/12/2024 của Hiệu trưởng Trường Đại học Công nghệ về việc gia hạn thời gian học tập cho NCS khóa QHI-2021 đợt 2.
- Quyết định số 3088/QĐ-ĐHCN ngày 31 tháng 12 năm 2024 của Hiệu trưởng Trường Đại học Công nghệ về việc điều chỉnh cán bộ hướng dẫn cho NCS Lê Văn Vinh khóa QH-2021.

7. Tên đề tài luận án: Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền

8. Ngành đào tạo: Kỹ thuật phần mềm 9. Mã số: 9480103.01

10. Cán bộ hướng dẫn khoa học: PGS. TS. Đặng Đức Hạnh

11. Tóm tắt các kết quả mới của luận án:

Luận án tập trung nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền (DDD), với mục tiêu thu hẹp khoảng cách giữa mô hình miền ở mức khái niệm và mô hình miền có khả năng thực thi. Đối tượng nghiên cứu của luận án là mô hình miền trong DDD và các kỹ thuật dựa trên ngôn ngữ chuyên biệt miền (DSL) nhằm biểu diễn, tích hợp và chuyển đổi mô hình miền theo hướng hỗ trợ thực thi, kiểm chứng hình thức và sinh tự động phần mềm.

Về phương pháp nghiên cứu, luận án áp dụng cách tiếp cận mô hình hóa dựa trên DSL kết hợp với thiết kế ngôn ngữ, kiểm chứng hình thức và phát triển lặp dựa trên mô hình miền. Các giải pháp đề xuất được hiện thực hóa thông qua công cụ hỗ trợ và được đánh giá bằng thực nghiệm trên các ca nghiên cứu điển hình, qua đó kiểm chứng tính khả thi và hiệu quả của các kỹ thuật đề xuất.

Các đóng góp chính và kết quả mới của luận án bao gồm:

và nâng cao chất lượng hệ thống. Nhờ đó, các kết quả góp phần gia tăng mức độ tự động hóa, rút ngắn thời gian phát triển, giảm chi phí và hỗ trợ kiểm tra tính tuân thủ giữa thiết kế và cài đặt trong thực tiễn.

13. Những hướng nghiên cứu tiếp theo:

- Hoàn thiện công cụ UDML dưới dạng plugin, đồng thời xây dựng các cú pháp biểu diễn đồ họa cho đặc tả hành vi, bảo mật và các mẫu chú thích ràng buộc (CAP), qua đó nâng cao khả năng sử dụng và hỗ trợ người dùng.
- Tiếp tục hoàn thiện kỹ thuật biểu diễn mô hình miền bằng cách mở rộng CAP cho các miền ứng dụng thực tế, thông qua việc xây dựng thư viện các mẫu ràng buộc hỗ trợ biểu thức OCL phức tạp, góp phần nâng cao khả năng áp dụng, tính tích hợp, hiệu năng thực thi và khả năng bảo trì của mô hình miền.
- Mở rộng khung phương pháp cho các DSL chuyên biệt theo từng mối quan tâm, đồng thời hỗ trợ các cơ chế phân quyền động và phụ thuộc ngữ cảnh đa dạng hơn.
- Tích hợp các kỹ thuật AI/LLM vào quy trình mô hình hóa miền nhằm hỗ trợ giai đoạn khởi tạo mô hình, đặc biệt trong việc sinh nháp đặc tả và gợi ý các thành phần từ mô tả nghiệp vụ; các đặc tả này cần được chuẩn hóa trong các DSL và kiểm chứng bằng các cơ chế hình thức để đảm bảo tính đúng đắn và nhất quán.

14. Các công trình đã công bố có liên quan đến luận án:

1. Duc-Hanh Dang, Duc Minh Le, Van-Vinh Le. AGL: Incorporating behavioral aspects into domain-driven design. *Information and Software Technology*, 163, 107284, 2023. DOI: 10.1016/j.infsof.2023.107284. (WoS/Scopus, Q1)
2. Van-Vinh Le, Nghia-Trong Be, Duc-Hanh Dang. On Automatic Generation of Executable Domain Models for Domain-Driven Design. *Proc. 15th. Int. Conf. Knowledge and Systems Engineering (KSE)*, IEEE, 2023. DOI: 10.1109/kse59128.2023.10299453. (WoS/Scopus)
3. Van-Vinh Le, Duc-Hanh Dang. An Approach to Composing Concerns for an Executable Unified Domain Model. *Proc. 18th. Int. Conf. Computing and Communication Technologies (RIVF)*, pages 415–419, IEEE, 2024. DOI: 10.1109/RIVF64335.2024.11009109. (WoS/Scopus)
4. Le Van Vinh, Dang Duc Hanh. RM2UDM: A method for automatically generating functional prototypes from requirement models. *Proc 17th. Nat. Conf Fundamental and Applied Information Technology Research (FAIR)*, pages 734–741, 2024. ISBN: 978-604-357-304-6, DOI: 10.15625/vap.2024.0271.
5. Van-Vinh Le, Nhat-Hoang Nguyen, Duc-Quyen Nguyen, Duc-Hanh Dang. A Method for Composing Concerns into a Unified Domain Model in Domain-Driven Design. *Proc. 14th Int. Symp. on Information and Communication Technology (SOICT)*, Springer, 2025. ISSN 1865-0929. (WoS/Scopus).

VIETNAM NATIONAL UNIVERSITY, HANOI
VNU UNIVERSITY OF ENGINEERING AND TECHNOLOGY

SOCIALIST REPUBLIC OF VIETNAM
Independence – Freedom – Happiness

INFORMATION ON DOCTORAL THESIS

1. Full name : Le Van Vinh
2. Sex: Male
3. Date of birth: May 5, 1982
4. Place of birth: Nghe An

5. Admission decision number: 25/QĐ-CTSV, dated January 14, 2022, issued by the Rector of the University of Engineering and Technology, Vietnam National University, Hanoi.

6. Changes in academic process:

- Decision No. 2833/QĐ-ĐHCN dated December 3, 2024, issued by the Rector of the University of Engineering and Technology on the extension of the study period for PhD candidates of cohort QHI-2021 (phase 2).
- Decision No. 3088/QĐ-ĐHCN dated December 31, 2024, issued by the Rector of the University of Engineering and Technology on the adjustment of supervisors for PhD candidate Lê Văn Vinh, cohort QH-2021.

7. Official thesis title: An Investigation of Model Representation and Transformation Techniques for Domain-Driven Design

8. Major: Software Engineering

9. Code: 9480103.01

10. Supervisors: Prof. Dr. Đặng Đức Hạnh

11. Summary of the new findings of the thesis:

The dissertation focuses on model representation and transformation techniques within Domain-Driven Design (DDD), with the objective of narrowing the gap between conceptual domain models and executable domain models. The research investigates domain models in DDD and techniques based on Domain-Specific Languages (DSLs) for representing, integrating, and transforming domain models to support executability, formal verification, and automated software generation.

and security. Furthermore, the proposed solutions contribute to improving development efficiency, reducing costs and development time, and meeting both economic and technical objectives in modern software engineering.

12. Practical applicability, if any:

The results of the dissertation can be directly applied to software development within DDD, particularly for systems with complex business domains. Domain model representation techniques enable the complete and consistent specification of structural, behavioral, and security aspects, thereby effectively supporting the analysis and design process. UDML and DSL integration techniques enable the construction of an executable unified domain model, while model transformation techniques support automated software generation, ensuring the preservation of domain semantics and improving system quality. As a result, these contributions enhance automation, reduce development time and costs, and support conformance checking between design and implementation in practice.

13. Further research directions, if any:

- Enhance the UDML tool as a plugin, while developing graphical syntax for representing behavioral specifications, security aspects, and CAP, thereby improving usability and user support.
- Further advance domain model representation techniques by extending CAP to real-world application domains through the construction of a library of constraint patterns supporting complex OCL expressions, thereby improving applicability, integration capability, execution performance, and maintainability of domain models.
- Extend the methodological framework to to support concern-specific DSLs, while enabling more expressive, dynamic, and context-aware access control mechanisms.
- Integrate AI/LLM techniques into the domain modeling process to support the model initialization phase, particularly in generating draft specifications and suggesting model elements from business descriptions; these specifications should be formalized within DSLs and validated using formal verification mechanisms to ensure correctness and consistency.

14. Thesis-related publications:

1. Duc-Hanh Dang, Duc Minh Le, Van-Vinh Le. AGL: Incorporating behavioral aspects into domain-driven design. *Information and Software Technology*, 163, 107284, 2023. DOI: 10.1016/j.infsof.2023.107284. (WoS/Scopus, Q1)

TRÍCH YẾU LUẬN ÁN

1. Thông tin chung về luận án

- Tên tác giả: Lê Văn Vinh
- Tên luận án: Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền
- Ngành khoa học của luận án: Kỹ thuật phần mềm Mã số: 9480103
- Tên đơn vị đào tạo: Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội

2. Nội dung bản trích yếu

2.1. Mục đích nghiên cứu

Luận án nghiên cứu đề xuất các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền, nhằm thu hẹp khoảng cách giữa mô hình miền ở mức khái niệm và mô hình miền có khả năng thực thi. Luận án hướng đến việc biểu diễn đầy đủ các khía cạnh của miền, bao gồm các khía cạnh như về cấu trúc, hành vi, và chính sách bảo mật; Đồng thời nghiên cứu các kỹ thuật sinh tự động phần mềm đúng yêu cầu, đảm bảo chất lượng và tuân thủ nguyên lý thiết kế hướng miền (DDD). Các nội dung chính của luận án bao gồm:

- Mở rộng mô hình miền để tích hợp hành vi, bảo mật miền và các ràng buộc OCL, tạo ra mô hình miền giàu thông tin và có khả năng thực thi.
- Xây dựng ngôn ngữ mô hình miền hợp nhất (UDML) nhằm tích hợp có hệ thống các ngôn ngữ chuyên biệt miền (DSL) theo mối quan tâm vào một mô hình miền hợp nhất.
- Nghiên cứu bộ chuyển đổi mô hình nhằm gia tăng tự động hóa phát triển phần mềm từ mô hình miền trong bối cảnh DDD.

2.2. Đối tượng nghiên cứu

Đối tượng nghiên cứu của luận án là mô hình miền trong DDD, cùng với các kỹ thuật dựa trên DSL để biểu diễn, tích hợp và chuyển đổi mô hình miền, nhằm hỗ trợ khả năng thực thi, kiểm chứng hình thức và sinh phần mềm tự động.

2.3. Phương pháp nghiên cứu

Luận án đề xuất áp dụng phương pháp mô hình hóa ngôn ngữ chuyên biệt miền, thiết kế, kiểm chứng hình thức, phát triển lặp lại dựa trên một mô hình miền để xây dựng ngôn ngữ và chuyển đổi mô hình, kết hợp thực nghiệm và đánh giá trên các ca nghiên cứu điển hình. Tiếp đó, Luận án tập trung giải quyết bài toán nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền.

Proc. 14th Int. Symp. on Information and Communication Technology (SOICT), Springer, 2025. ISSN 1865-0929. (WoS/Scopus).

6. Van-Vinh Le, Nhat-Hoang Nguyen, Duc-Quyen Nguyen, Duc-Hanh Dang. A Semantic Framework and Tool Support for Unified Executable Domain Models in UDML: A Case Study on the RBAC Concern. VNU *Journal of Science: Computer Science and Communication Engineering*, Vol 42 No 1, pages 79-114, 2026. ISSN: 2615-9260, DOI: 10.25073/2588-1086/vnucsce.6743.

Hà Nội, ngày 02 tháng 07 năm 2026

NGHIÊN CỨU SINH	**CÁN BỘ HƯỚNG DẪN**
(*Ký và ghi rõ họ, tên*)	(*Ký và ghi rõ họ, tên*)

XÁC NHẬN CỦA CƠ SỞ ĐÀO TẠO

THE ABSTRACT OF DOCTORAL THESIS

1. General information about the thesis

Author: Le Van Vinh

Title of the doctoral thesis: An Investigation of Model Representation and Transformation Techniques for Domain-Driven Design

Field of study: Software Engineering Code: 9480103

Training institution: University of Engineering and Technology, Vietnam National University

2. Abstract content

2.1. Research Purpose

Proposing techniques for representing domain models in the context of Domain-Driven Design (DDD), enabling the explicit and consistent specification of structural, behavioral, and business-constraint aspects of the business domain. These techniques extend the expressive power of domain models and provide a foundation for subsequent model integration, formal verification, and model transformation.

Proposing a technique for unifying domain-specific languages (DSLs) associated with heterogeneous concerns into a unified, executable domain model. On this basis, the dissertation establishes a semantic foundation for formal verification, aiming to ensure the semantic consistency of the unified domain model at the design stage.

Proposing model transformation techniques based on the unified domain model, enabling the automatic generation of executable software prototypes while preserving domain semantics and transformation quality, and supporting domain-specific usage scenarios in specialized application domains.

2.2. Research Object

The research object of this dissertation is the domain model in DDD, together with DSL-based techniques for representing, integrating, and transforming domain models, with the aim of supporting executability, formal verification, and automatic software generation.

2.3. Research methods

For achieving the research purpose, methods for domain modeling, DSL design, semantic integration of heterogeneous concerns, model transformation, and formal verification are employed. In addition, a design-and-implementation–based research methodology is

The framework supports both model-to-model (M2M) and model-to-text (M2T) transformations and is implemented on top of a modular software architecture. Experimental results from multiple case studies including: CourseMan, OrderMan, ProcessMan, and OJS demonstrate that the proposed transformations can automatically generate executable software prototypes that conform to domain requirements and DDD principles.

The dissertation implements a prototype toolchain to validate the feasibility and applicability of the proposed approaches. Through empirical evaluation on representative domain scenarios, the results show that the proposed techniques effectively support executable domain modeling, semantic integration, and automated software generation. The findings confirm that the approach significantly reduces manual effort, improves semantic consistency between models and implementations, and enhances the maintainability and evolvability of DDD-based software systems.

In conclusion, this dissertation contributes a coherent and systematic framework for domain modeling, semantic integration, and model transformation in DDD. By placing the domain model at the center of both semantics and execution, the proposed approach bridges the long-standing gap between domain specification and software realization, and provides a solid foundation for future research on executable domain models and model-driven DDD.

These results have been published in conference proceedings and ISI/Scopus-indexed journal papers.

1. Duc-Hanh Dang, Duc Minh Le, Van-Vinh Le. AGL: Incorporating behavioral aspects into domain-driven design. *Information and Software Technology*, 163, 107284, 2023. DOI: 10.1016/j.infsof.2023.107284. (WoS/Scopus, Q1)
2. Van-Vinh Le, Nghia-Trong Be, Duc-Hanh Dang. On Automatic Generation of Executable Domain Models for Domain-Driven Design. *Proc. 15th. Int. Conf. Knowledge and Systems Engineering (KSE)*, IEEE, 2023. DOI: 10.1109/kse59128.2023.10299453. (WoS/Scopus)
3. Van-Vinh Le, Duc-Hanh Dang. An Approach to Composing Concerns for an Executable Unified Domain Model. *Proc. 18th. Int. Conf. Computing and Communication Technologies (RIVF)*, pages 415–419, IEEE, 2024. DOI: 10.1109/RIVF64335.2024.11009109. (WoS/Scopus)
4. Le Van Vinh, Dang Duc Hanh. RM2UDM: A method for automatically generating functional prototypes from requirement models. *Proc 17th. Nat. Conf Fundamental and Applied Information Technology Research (FAIR)*, pages 734–741, 2024. ISBN: 978-604-357-304-6, DOI: 10.15625/vap.2024.0271.
5. Van-Vinh Le, Nhat-Hoang Nguyen, Duc-Quyen Nguyen, Duc-Hanh Dang. A Method for Composing Concerns into a Unified Domain Model in Domain-Driven Design.

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Hà Nội, ngày 02 tháng 7 năm 2026

BẢN XÁC NHẬN ĐÃ SỬA CHỮA CÁC THIẾU SÓT CỦA LUẬN ÁN

Ông/bà Lê Văn Vinh được công nhận là nghiên cứu sinh (NCS) theo Quyết định công nhận số 25/QĐ-CTSV ngày 14/01/2022 của Hiệu trưởng Trường Đại học Công nghệ, thời gian đào tạo từ ngày 01/01/2022 đến ngày 31/12/2024; tôi được gia hạn học tập 18 tháng (1,5 năm) theo Quyết định số 2833/QĐ-ĐHCN ngày 03/12/2024 của Hiệu trưởng Trường Đại học Công nghệ. Thực hiện Quyết định số 721/QĐ-ĐHCN ngày 09/4/2026 của Hiệu trưởng trường Đại học Công nghệ về việc thành lập Hội đồng cấp ĐHQG đánh giá luận án tiến sĩ của NCS Lê Văn Vinh Hội đồng đã họp ngày 20 tháng 6 năm 2026, (có Biên bản kèm theo). Theo Quyết nghị của Hội đồng cấp ĐHQG đánh giá luận án tiến sĩ, NCS phải bổ sung và sửa chữa các điểm sau đây trong luận án:

1. *Yêu cầu chỉnh sửa 01*

   Bổ sung khảo sát các nghiên cứu liên quan tích hợp LLM vào luận án.

2. *Yêu cầu chỉnh sửa 02*

   Rà soát trình bày, văn phong đảm bảo nhất quán giữa mô tả, nội dung, đánh giá.

3. *Yêu cầu chỉnh sửa 03*

   Bổ sung thảo luận khả năng mở rộng và ứng dụng của luận án (bổ sung vào phần chương Kết luận).

4. *Yêu cầu chỉnh sửa 04*

   Rà soát lỗi trình bày, lỗi soạn thảo toàn bộ luận án

Ngày 02/7/2026, NCS Lê Văn Vinh đã nộp bản luận án và tóm tắt luận án có chỉnh sửa. Chúng tôi nhận thấy rằng nội dung, hình thức của luận án và tóm tắt luận án đã được sửa chữa, bổ sung theo các điểm trên của Quyết nghị (Bản giải trình sửa chữa của NCS kèm theo).

Đề nghị Trường Đại học Công nghệ, cho phép NCS Lê Văn Vinh được làm các thủ tục khác để xem xét công nhận học vị cho NCS.

Xin trân trọng cảm ơn./.

THƯ KÝ HỘI ĐỒNG | **CHỦ TỊCH HỘI ĐỒNG**

Phạm Ngọc Hưng

NGHIÊN CỨU SINH | **CÁN BỘ HƯỚNG DẪN** | **XÁC NHẬN CỦA KHOA CNTT**

Lê Văn Vinh | Đặng Đức Hạnh | Phạm Ngọc Hưng

XÁC NHẬN CỦA TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

T/L HIỆU TRƯỞNG

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

KT. TRƯỞNG PHÒNG CÔNG TÁC SINH VIÊN

PHÓ TRƯỞNG PHÒNG

Nguyễn Huy Tiệp

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Hà Nội, ngày 02 tháng 7 năm 2026

BẢN GIẢI TRÌNH
CỦA NGHIÊN CỨU SINH VỀ VIỆC SỬA CHỮA, BỔ SUNG LUẬN ÁN

Họ và tên NCS: Lê Văn Vinh

Tên đề tài luận án: Nghiên cứu các kỹ thuật biểu diễn và chuyển đổi mô hình cho thiết kế hướng miền

Chuyên ngành: Kỹ thuật phần mềm Mã số: 9480103

Sau khi họp Hội đồng cấp ĐHQG đánh giá luận án tiến sĩ, Nghiên cứu sinh đã nghiêm túc thực hiện việc nghiên cứu, tiếp thu, sửa chữa, bổ sung luận án theo các ý kiến đóng góp của các Phản biện cũng như các ý kiến của các thành viên Hội đồng và có các giải trình thông qua bảng chỉnh sửa như sau:

1. Quyết nghị của Hội đồng cấp ĐHQG đánh giá luận án tiến sĩ

STT	Ý kiến của Hội đồng	Kết quả chỉnh sửa	Vị trí chỉnh sửa trong luận án
1	Bổ sung khảo sát các nghiên cứu liên quan tích hợp LLM vào luận án	NCS tiếp thu và bổ sung: - Đã bổ sung Bổ sung khảo sát các nghiên cứu liên quan tích hợp LLM vào luận án - Đã bổ sung thảo luận, đánh giá khả năng ứng dụng LLM vào các phương pháp đề xuất của luận án	- Bổ sung khảo sát các nghiên cứu liên quan tích hợp LLM vào luận án tại mục 2.5 từ trang 46 đến trang 50 - Bổ sung thảo luận khả năng ứng dụng LLM tại mục 7.2 từ tang 183 đến trang 184 Đã bổ sung khảo sát thêm 3 tài liệu:
2	Rà soát trình bày, văn phong đảm bảo nhất quán giữa mô tả, nội dung, đánh giá.	NCS tiếp thu và rà soát toàn bộ luận án để đảm bảo về mặt thể thức trình bày, văn phong nhất quán	Toàn bộ luận án

3	Bổ sung thảo luận khả năng mở rộng và ứng dụng của luận án (bổ sung vào phần chương Kết luận).	NCS tiếp thu và đã viết lại, bổ sung đầy đủ trong chương 7 (mục 7.1 và 7.2) thảo luận khả năng mở rộng và ứng dụng của luận án	Từ trang 178 đến trang 184
4	Rà soát lỗi trình bày, lỗi soạn thảo toàn bộ luận án	NCS tiếp thu và đã rà soát lại toàn bộ luận án	Toàn bộ luận án

2. Ý kiến của từng thành viên hội đồng

STT	Yêu cầu chỉnh sửa (nội dung trước khi chỉnh sửa)	Nội dung sau khi chỉnh sửa	Chỉnh sửa tại trang
1. PGS. TS. Nguyễn Mạnh Hùng (phản biện)			
1.1	Đặt vấn đề chưa rõ (Chưa nêu rõ bài toán chung của luận án)	NCS tiếp thu và trình bày lại mục 1.1.3 bắt đầu từ bài toán chung sau đó mới tách ra làm 3 vấn đề: Biểu diễn mô hình miền. Hợp nhất các mối quan tâm. Chuyển đổi mô hình. Bài toán chung mà luận án hướng tới là: Thu hẹp khoảng cách giữa Domain Model và phần mềm thực thi trong DDD thông qua các kỹ thuật biểu diễn, hợp nhất và chuyển đổi mô hình. Ba vấn đề nghiên cứu được xem là ba bài toán con để giải quyết bài toán tổng thể này.	Trình bày lại Mục 1; 1.1.3 của chương 1 (trang 7)
1.2	Đóng góp chính không nhất quán	NCS tiếp thu ý kiến này và viết lại để nhất quán. Luận án có 3 nhóm đóng góp khoa học chính: 1. Kỹ thuật biểu diễn mô hình miền. 2. Kỹ thuật hợp nhất các mối quan tâm vào mô hình miền. 3. Kỹ thuật chuyển đổi mô hình và sinh bản mẫu phần mềm.	Trình bạy lại mục 1.4 Trang 13 Trong Chương 7 NCS gom công cụ thực nghiệm vào phần kiểm chứng các đóng góp nên còn 3 mục.

1.3	Phần AI/LLM còn ngắn và mờ nhạt	NCS tiếp thu và hoàn thiện: Mục AI/LLM được bổ sung nhằm cập nhật bối cảnh nghiên cứu gần đây. Tuy nhiên AI/LLM không phải trọng tâm nghiên cứu của luận án. Luận án tập trung vào: DSL, DDD, MDSE, Model Transformation, Formal Verification Do đó phần AI chỉ đóng vai trò khảo sát xu hướng và định vị nghiên cứu. Việc tích hợp AI/LLM vào các bộ chuyển đổi mô hình là một hướng nghiên cứu tiếp theo rất tiềm năng.	NCS thực hiện cập nhật, bổ sung đầy đủ hơn về phần AI/LLM. Mục 2.5 trang 46 đến trang 50.
1.4	Các định nghĩa được trình bày nhưng không thấy vai trò của các khái niệm được sử dụng	NCS tiếp thu và rà soát lại toàn bộ: Các định nghĩa hình thức được sử dụng để xây dựng nền tảng ngữ nghĩa của UDML. Vai trò chính gồm: Định nghĩa cú pháp trừu tượng của UDML. Định nghĩa các phần tử hành vi, ràng buộc và bảo mật. Làm cơ sở cho ánh xạ UDML sang Event-B. Hỗ trợ kiểm chứng hình thức.	Chủ yếu tập trung tại chương 4
1.5	Thực nghiệm và đánh giá ở chương 6 chưa đủ tính thuyết phục.	NCS tiếp thu và trình bày tốt hơn để có tính thuyết phục và cho thấy so sánh kết quả của mô hình đề xuất với các mô hình liên quan đã trình bày trong chương 2. Trong Chương 6, luận án tập trung đánh giá: Tính khả thi; Tính biểu đạt; Khả năng tích hợp; Khả năng sinh bản mẫu trên các ca nghiên cứu thực tế. Tuy nhiên việc so sánh định lượng trực tiếp với các công cụ khác còn hạn chế do: - Các nghiên cứu sử dụng DSL khác nhau.	Bổ sung thêm phần đánh giá thực nghiệp ở chương 6. Bổ sung trong tương lai luận án dự định mở rộng theo hướng benchmark và đối sánh thực nghiệm với các nền tảng tương đương.

		- Mục tiêu khác nhau. - Không công bố đầy đủ tạo tác để tái lập thực nghiệm.	
1.6	Kết luận Chương 6 chưa cập nhật	NCS tiếp thu đây là lỗi biên tập sau khi tái cấu trúc nội dung chương. Nội dung kết luận chương sẽ được chỉnh sửa để phản ánh đầy đủ cả: Công cụ hỗ trợ. Thực nghiệm. Đánh giá.	Bổ sung phần kết luận của chương 6 trang 174
1.7	Tài liệu còn thiếu tính cập nhật	NCS tiếp thu và cập nhật. Tuy nhiên cần lưu ý rằng luận án nghiên cứu trên các nền tảng: DDD (Evans); DSL; OCL; Event-B; MDSE Đây là các hướng nghiên cứu có tính nền tảng và nhiều công trình kinh điển vẫn được cộng đồng sử dụng rộng rãi. Luận án được bổ sung thêm 4 tài liệu được công bố từ năm 2023 trở lại đây về LLM	Toàn bộ luận án
2. TS. Nguyễn Thanh Hùng (phản biện)			
2.1	Phần tổng quan về AI/LLM trong mô hình hóa miền và sinh mã nguồn còn ngắn.	NCS tiếp thu bổ sung để Luận án làm rõ AI/LLM là hướng nghiên cứu liên quan và tiềm năng, nhưng không phải trọng tâm chính. Bổ sung phân tích vai trò AI/LLM trong hỗ trợ sinh mô hình, sinh mã và đối sánh với hướng DSL/MDSE của luận án.	Mục 2.5 – Hướng tiếp cận sử dụng AI/LLM; bổ sung thảo luận tại 7.2 – Hướng phát triển tiếp theo.
2.2	Phương pháp chủ yếu đánh giá trong bối cảnh Java, aDSL và công cụ liên quan; chưa làm rõ khả năng mở rộng sang microservices, event-driven, hệ phân tán.	NCS tiếp thu một phần. Luận án làm rõ phạm vi hiện tại tập trung vào DDD, aDSL, Java/JDA để kiểm chứng tính khả thi của phương pháp. Khả năng mở rộng sang microservices, event-driven và hệ phân tán được xác định là không thuộc phạm vi nghiên cứu và là hướng phát triển tiếp theo.	Mục 1.2 – Phạm vi nghiên cứu; mục 7.2 – Hướng phát triển tiếp theo.

2.3	Việc sử dụng nhiều DSL, CAP, UDML và Event-B có thể làm tăng độ phức tạp triển khai.	NCS tiếp thu Luận án bổ sung giải thích rằng các DSL được tổ chức theo từng mối quan tâm: DCSL cho cấu trúc, CAP cho ràng buộc, AGL cho hành vi, RBACDom cho bảo mật, sau đó hợp nhất bằng UDML. Mục tiêu là quản lý phức tạp bằng mô-đun hóa thay vì trộn lẫn trong một đặc tả duy nhất.	Mục 4.2, 4.3 – Phương pháp tích hợp các mối quan tâm vào mô hình miền; mục 4.5 – Tổng kết chương.
2.4	Chưa có đánh giá định lượng đầy đủ về hiệu năng, chi phí phát triển, chi phí bảo trì, khả năng mở rộng trong dự án công nghiệp lớn.	NCS tiếp thu và cập nhật để Luận án làm rõ thực nghiệm hiện tại tập trung vào tính khả thi, tính biểu đạt, khả năng tích hợp và khả năng sinh bản mẫu. Các đánh giá định lượng quy mô công nghiệp được xác định là giới hạn và hướng nghiên cứu tiếp theo.	Mục 6.3 – Thực nghiệm và đánh giá; bổ sung hạn chế tại 7.2 – Hướng phát triển tiếp theo.
2.5	Chưa phân tích kỹ các trường hợp biên: hành vi bất đồng bộ cao, ràng buộc rất đặc thù, bảo mật phức tạp ngoài RBAC.	NCS đã bổ sung thêm để có sự phân tích các trường hợp biên. Tuy nhiên, Luận án làm rõ phạm vi hiện tại tập trung vào hành vi miền dựa trên AGL và chính sách bảo mật dạng RBACDom. Các trường hợp vượt ngoài RBAC, hành vi bất đồng bộ và ràng buộc đặc thù được xem là giới hạn của nghiên cứu hiện tại.	Mục 1.2 – Phạm vi nghiên cứu; mục 4.2.2 – RBACDom; mục 7.2 – Hướng phát triển tiếp theo.
2.6	Thực nghiệm chủ yếu trên các ca nghiên cứu do nhóm nghiên cứu xây dựng; thiếu hệ thống công nghiệp quy mô lớn.	NCS tiếp thu và bổ sung nhận định rằng đánh giá trên hệ thống công nghiệp lớn là hướng mở rộng cần thực hiện tiếp. Luận án nhấn mạnh các ca CourseMan, OrderMan, ProcessMan, OJS được dùng để kiểm chứng tính khả thi trên nhiều miền khác nhau.	Mục 6.3 – Thực nghiệm và đánh giá; mục 7.2 – Hướng phát triển tiếp theo.
2.7	Cần bổ sung so sánh định lượng với các nền tảng DDD/MDE hiện đại.	NCS tiếp thu một phần. Luận án đã có so sánh theo mức độ biểu đạt và khả năng tự động hóa; tuy nhiên so sánh định lượng sâu với nền tảng công nghiệp hiện đại chưa phải trọng tâm.	Mục 6.3.1, 6.3.2, 6.3.3; mục 7.2.

		Nội dung này được bổ sung vào phần hạn chế và hướng phát triển.	
2.8	Cần làm rõ khả năng tích hợp với AI/LLM trong sinh mô hình và sinh mã nguồn.	NCS tiếp thu, Luận án bổ sung định vị AI/LLM như hướng hỗ trợ trong tương lai, có thể kết hợp với UDML để gợi ý mô hình, sinh đặc tả DSL hoặc hỗ trợ sinh mã.	Mục 2.5; mục 7.2.
2.9	Cần nghiên cứu mở rộng UDML cho microservices và hệ phân tán quy mô lớn.	NCS tiếp thu, Luận án xác định đây là hướng phát triển tiếp theo, đặc biệt trong việc mở rộng UDML để biểu diễn các bounded contexts, giao tiếp dịch vụ và ràng buộc phân tán.	Mục 7.2 – Hướng phát triển tiếp theo.
PGS.TS. Nguyễn Việt Hùng (Phản biện)			
3.1	Cương 2 còn dàn trải, bao hàm nhiều nội dung: DDD, MDE, Event-B, RBAC.	NCS tiếp thu, rà soát và cô đọng lại phần tổng quan theo đúng trục chính của luận án: biểu diễn mô hình miền → tích hợp mối quan tâm → chuyển đổi mô hình.	Chương 2, đặc biệt các mục 2.1–2.4 và 2.6.
3.2	Một số nội dung chưa đào sâu các nghiên cứu mới nhất trong 2 năm gần đây.	NCS tiếp thu bổ sung và cập nhật thêm các tài liệu gần đây liên quan đến DDD, DSL, MDSE, model transformation, AI/LLM. Đã bổ sung thêm 4 tài liệu về LLM	Chương 2 và Tài liệu tham khảo.
3.3	Mục 2.5 về AI/LLM còn cơ bản, chưa phân tích tác động tới phương pháp siêu mô hình truyền thống.	NCS tiếp thu. Phần này sẽ bổ sung và làm rõ AI/LLM là hướng tiếp cận mới, có tiềm năng hỗ trợ sinh mô hình và mã nguồn, nhưng luận án vẫn tập trung vào phương pháp có cơ sở ngữ nghĩa hình thức dựa trên DSL/MDSE.	Mục 2.5 và 7.2 – Hướng phát triển tiếp theo.
3.4	AGL và CAP dùng Java/Annotation nên có thể khó tiếp cận với chuyên gia nghiệp vụ không biết lập trình.	NCS tiếp thu một phần. Luận án làm rõ vai trò của aDSL là phục vụ mô hình miền hướng thực thi; chuyên gia nghiệp vụ tham gia ở mức xác nhận mô hình, còn nhà phát triển/thiết kế ngôn ngữ hiện thực hóa bằng annotation.	Mục 1.2, 2.1.4, 3.2, 3.3.

3.5	Phạm vi kiến trúc còn giới hạn ở MOSA và JDA; chưa thảo luận sâu microservices, serverless, polyglot.	NCS tiếp thu thực hiện xác định rõ phạm vi luận án tập trung vào JDA/MOSA để kiểm chứng tính khả thi; mở rộng sang microservices, event-driven, serverless và hệ phân tán là hướng phát triển tiếp theo.	Mục 1.2 – Phạm vi nghiên cứu và 7.2.
3.6	Chưa có đánh giá chi tiết về tích hợp AI để sinh siêu mô hình hoặc kiểm chứng mô hình.	NCS tiếp thu. Luận án bổ sung định hướng kết hợp AI/LLM trong tương lai, nhưng không đưa vào thực nghiệm chính vì trọng tâm hiện tại là kỹ thuật biểu diễn, hợp nhất và chuyển đổi có kiểm chứng hình thức.	Mục 2.5 và 7.2.
3.7	Một số hình siêu mô hình như Hình 4.2, 4.6 quá dày đặc.	NCS tiếp thu. Thực hiện rà soát lại hình vẽ, bổ sung diễn giải để người đọc nắm được các thành phần lõi trước khi đi vào chi tiết kỹ thuật.	Mục 4.2.2, các hình 4.2, 4.6.
3.8	Các đoạn mã aDSL trong Chương 3, 4 làm ngắt mạch đọc lý thuyết.	NCS tiếp thu. Thực hiện cân nhắc giữ lại các đoạn mã cần thiết để minh họa cú pháp cụ thể; các phần chi tiết triển khai có thể được rút gọn hoặc chuyển sang phụ lục nếu cần.	Chương 3, Chương 4, phần mô tả cú pháp cụ thể của CAP, AGL, RBACDom, UDML.
3.9	Chưa làm rõ khả năng mở rộng đối với các hành vi bất đồng bộ, ràng buộc đặc thù, mô hình bảo mật ngoài RBAC.	NCS tiếp thu. Luận án làm rõ phạm vi hiện tại tập trung vào hành vi miền theo AGL và bảo mật dựa trên RBACDom; các trường hợp ngoài RBAC và hành vi bất đồng bộ là hướng nghiên cứu tiếp theo.	Mục 1.2, 4.2.2, 7.2.
3.10	Thực nghiệm chủ yếu trên CourseMan, OrderMan, ProcessMan, OJS; chưa có hệ thống công nghiệp quy mô lớn.	NCS tiếp thu. Luận án xác định các ca nghiên cứu hiện tại dùng để đánh giá tính khả thi và tính biểu đạt; đánh giá trên hệ thống công nghiệp lớn là hướng mở rộng tiếp theo. Luận án đã xem đây là một nguy cơ ảnh hưởng đến sự đúng đắn (validity)	Chương 6, đặc biệt 6.3 và 7.2.

		của luận án và nhấn mạnh ở các phần thảo luận cụ thể trong chương 6	
3.11	Cần bổ sung so sánh định lượng với các nền tảng DDD/MDE hiện đại.	NCS tiếp thu một phần. Luận án đã có so sánh theo tính biểu đạt và mức độ tự động hóa; đánh giá định lượng sâu về chi phí, hiệu năng, bảo trì sẽ được bổ sung trong nghiên cứu tiếp theo.	Mục 6.3.1–6.3.3 và 7.2.
4. PGS.TS. Cao Tuấn Dũng (ủy viên)			
4.1	CAP chủ yếu hỗ trợ ràng buộc cấu trúc và định lượng; chưa bao quát tiền điều kiện/hậu điều kiện, kiểm soát truy cập theo ngữ cảnh, phản ứng tự động của hệ thống.	NCS tiếp thu. Luận án làm rõ phạm vi của CAP là tích hợp các nhóm ràng buộc OCL thiết yếu vào mô hình miền, đặc biệt các ràng buộc cấu trúc, bất biến và ràng buộc nghiệp vụ thường gặp. Các ràng buộc hành vi như tiền điều kiện/hậu điều kiện và phản ứng tự động không phải trọng tâm của CAP mà được xử lý ở hướng AGL và UDML. Kiểm soát truy cập được tách thành mối quan tâm bảo mật qua RBACDom.	Mục 3.2 – Kỹ thuật tích hợp ràng buộc OCL vào mô hình miền; mục 3.3 – Kỹ thuật tích hợp hành vi; mục 4.2.2 – RBACDom; mục 7.2 – Hướng phát triển tiếp theo.
4.2	Phạm vi bao phủ ràng buộc OCL còn giới hạn.	NCS tiếp thu một phần. Luận án không đặt mục tiêu bao phủ toàn bộ OCL, mà tập trung vào các nhóm ràng buộc có tính lặp lại và có khả năng đóng gói thành Constraint Annotation Pattern. Đây là lựa chọn có chủ đích để phục vụ tái sử dụng và sinh bản mẫu phần mềm. Các nhóm OCL phức tạp hơn được xác định là hướng mở rộng tiếp theo.	Mục 3.2.2 – Tích hợp mẫu CAP vào mô hình miền; mục 6.3.1 – Đánh giá kỹ thuật tích hợp ràng buộc; mục 7.2.
4.3	Các ví dụ CourseMan, OJS, OrderMan có tính thực tiễn nhưng ở quy mô trung bình.	NCS tiếp thu. Luận án làm rõ các ca nghiên cứu được chọn nhằm đánh giá tính khả thi, tính biểu đạt và khả năng áp dụng trên nhiều miền khác nhau: giáo dục, đơn hàng/quy trình và hệ thống tạp chí mở. Mục tiêu của thực nghiệm hiện tại là chứng minh phương pháp hoạt động	Mục 1.2 – Phạm vi nghiên cứu; mục 6.3 – Thực nghiệm và đánh giá.

		được trên các ca đại diện, chưa phải đánh giá công nghiệp quy mô lớn.	
4.4	Chưa đánh giá định lượng đầy đủ khả năng mở rộng và hiệu năng trên hệ thống công nghiệp lớn với hàng nghìn thực thể.	NCS tiếp thu, đây là hạn chế của luận án. Chương 6 tập trung vào đánh giá khả năng biểu diễn, tích hợp, kiểm chứng và sinh bản mẫu. Các chỉ số hiệu năng, chi phí phát triển, chi phí bảo trì và khả năng mở rộng trên hệ thống công nghiệp lớn sẽ được bổ sung trong nghiên cứu tiếp theo.	Mục 6.3 – Thực nghiệm và đánh giá; mục 7.2 – Hướng phát triển tiếp theo.
4.5	Cần làm rõ hơn khả năng áp dụng thực tế của phương pháp.	NCS tiếp thu. Luận án đã bổ sung/nhấn mạnh vai trò của công cụ thực nghiệm: CAP/UDML tool, công cụ AGL, môi trường UDML, ATL/Acceleo và JDA. Các công cụ này chứng minh chuỗi đề xuất không chỉ là mô hình lý thuyết mà có thể hiện thực hóa thành quy trình sinh bản mẫu phần mềm.	Mục 6.2 – Công cụ hỗ trợ; mục 6.3 – Thực nghiệm và đánh giá.
4.6	Cần định vị rõ giới hạn nghiên cứu.	NCS tiếp thu. Luận án xác định rõ phạm vi tập trung vào DDD, DSL/aDSL, UDML, Event-B, chuyển đổi mô hình và sinh bản mẫu phần mềm. Các vấn đề như bao phủ toàn bộ OCL, hệ thống công nghiệp rất lớn, ràng buộc hành vi phức tạp và đánh giá hiệu năng quy mô lớn được đưa vào hướng phát triển tiếp theo.	Mục 1.2 – Mục tiêu và phạm vi nghiên cứu; mục 7.2 – Hướng phát triển tiếp theo.
5. PGS. TS. Trần Đăng Hưng (ủy viên)			
5.1	Cần làm rõ hơn cơ chế xử lý xung đột ngữ nghĩa giữa các DSL chuyên biệt sau khi hợp nhất.	NCS tiếp thu. Luận án đã xác định UDML là mô hình trung tâm để hợp nhất các DSL theo mối quan tâm. Cơ chế tích hợp không chỉ gộp cú pháp mà còn tích hợp ngữ nghĩa của từng DSL vào ngữ nghĩa chung của UDML. Khi phát sinh xung đột trong quan hệ hoặc ràng buộc, thuật toán tích hợp áp dụng quy	Mục 4.3.2 – Biểu diễn và tích hợp các mối quan tâm; thuật toán tích hợp DSL vào UDML, đặc biệt phần xử lý cạnh/ràng buộc và tích hợp ngữ nghĩa.

		tắc ánh xạ hoặc quy tắc ưu tiên, đồng thời kiểm tra tính nhất quán giữa ngữ nghĩa của DSL thành phần và UDML.	
5.2	Các DSL thuộc các mối quan tâm khác nhau có thể phát sinh ràng buộc không nhất quán.	NCS tiếp thu. Luận án giải quyết vấn đề này bằng cách đặt các DSL trên một trạng thái hệ thống chung của UDML. DCSL, AGL và RBACDom không được hiểu như các mô hình tách rời, mà được đồng bộ hóa trên mô hình trạng thái hợp nhất; hành vi AGL xác định chuyển trạng thái hợp lệ, còn RBACDom được diễn giải như điều kiện bảo vệ ràng buộc các chuyển trạng thái đó.	Mục 4.1–4.4, đặc biệt phần xác định UDML, AGL và RBACDom trên cùng trạng thái hệ thống hợp nhất.
5.3	Nên bổ sung tình huống minh họa và cơ chế phát hiện, xử lý xung đột ngữ nghĩa.	NCS tiếp thu một phần. Luận án đã minh họa tích hợp trên các ca CourseMan, OJS và công cụ UDML/MPS; tuy nhiên phản biện đúng khi cho rằng cần làm nổi bật hơn các tình huống xung đột. Phần giải trình nên nhấn mạnh rằng xung đột được phát hiện thông qua kiểm tra ràng buộc, hệ thống kiểu, kiểm tra nhất quán trong UDML và kiểm chứng Event-B; phần minh họa có thể được bổ sung trong thuyết trình hoặc phụ lục.	Mục 6.2.3 – Công cụ hỗ trợ tích hợp các mối quan tâm, Hình 6.10–6.12; mục 4.3.2 và 4.4.
5.4	Phần kiểm chứng hình thức dựa trên Event-B mới tập trung vào ánh xạ mô hình, chưa phân tích sâu tính bảo toàn ngữ nghĩa của phép chuyển đổi.	NCS tiếp thu. Luận án đã có cơ sở bảo toàn ngữ nghĩa ở mức UDML–AGL–Event-B: mỗi trạng thái UDML chứa thành phần hành vi AGL, hành vi AGL là phép chiếu của trạng thái hợp nhất UDML qua πAGL; khi ánh xạ sang Event-B, mỗi trạng thái Event-B tương ứng với một hình chụp hợp lệ của UDML và mỗi event Event-B biểu diễn một chuyển trạng thái khả dĩ. Có thể bổ sung diễn giải rõ hơn về bảo toàn trạng thái, bất	Mục 4.4.2 – Ánh xạ thực thi trong AGL sang UDML và 4.4.3 – Định nghĩa ngữ nghĩa UDML sử dụng Event-B.

		biến và chuyển trạng thái trong phần kiểm chứng.	
5.5	Cần đánh giá đầy đủ hơn chất lượng mã nguồn được sinh tự động.	NCS tiếp thu. Luận án hiện tập trung đánh giá tính khả thi của chuỗi chuyển đổi và khả năng sinh bản mẫu phần mềm dựa trên JDA. Các tiêu chí như khả năng bảo trì, khả năng mở rộng và tối ưu mã nguồn chưa phải trọng tâm đánh giá định lượng. Đây là hạn chế hợp lý và được xác định là hướng mở rộng trong nghiên cứu tiếp theo.	Mục 5.4 – Sinh mã nguồn bản mẫu phần mềm; mục 6.2.4 – Công cụ hỗ trợ chuyển đổi mô hình; mục 7.2 – Hướng phát triển tiếp theo.
5.6	Nên bổ sung tiêu chí đánh giá như khả năng bảo trì, mở rộng, tối ưu mã nguồn.	NCS tiếp thu. Luận án đã đánh giá ở mức bản mẫu: sinh mô hình, sinh đặc tả, sinh mã và chạy được trên JDA. Các tiêu chí chất lượng mã nguồn công nghiệp như khả năng bảo trì, mở rộng và tối ưu mã nguồn cần một nghiên cứu thực nghiệm riêng trên hệ thống lớn và sẽ là hướng nghiên cứu tiếp theo	Mục 6.3 – Thực nghiệm và đánh giá; các công cụ sinh bản mẫu trong Hình 6.13–6.18.
6. TS. Trần Hoàng Việt (Thư ký)			
6.1	Phần đặt vấn đề chưa nêu rõ bài toán chung; mục 1.1.3 mới nêu 3 vấn đề con.	NCS tiếp thu. Bài toán chung của luận án được xác định lại là: thu hẹp khoảng cách giữa mô hình miền và phần mềm thực thi trong DDD thông qua biểu diễn, hợp nhất và chuyển đổi mô hình. Ba vấn đề trong mục 1.1.3 là ba bài toán con: biểu diễn DM, hợp nhất mối quan tâm, và chuyển đổi mô hình.	Mục 1.1.3, 1.2. Thesis đã nêu rõ mục tiêu thu hẹp khoảng cách giữa DM và hiện thực phần mềm.
6.2	Đóng góp chính chưa nhất quán: Chương 1 có 4 đóng góp, Chương 7 có 3 đóng góp.	NCS tiếp thu. Chỉnh theo hướng thống nhất thành 3 nhóm đóng góp khoa học chính: 1) biểu diễn DM; 2) hợp nhất mối quan tâm bằng UDML; 3) chuyển đổi mô hình và sinh bản mẫu. Phần công cụ thực nghiệm không tách thành đóng góp khoa học độc lập mà là phần hiện thực và đánh giá.	Mục 1.4 và 7.1.

6.3	Mục AI/LLM còn ngắn, chưa nổi bật.	NCS tiếp thu. Bổ sung phân tích AI/LLM như một hướng liên quan mới, nhưng làm rõ đây không phải trọng tâm của luận án. Thesis tập trung vào DSL, UDML, Event-B và chuyển đổi mô hình có cơ sở ngữ nghĩa. AI/LLM được đưa vào hướng phát triển tiếp theo.	Mục 2.5, 7.2.
6.4	Chương 2 gộp cơ sở lý thuyết và tổng quan nghiên cứu nên khó theo dõi.	NCS tiếp thu. Có thể chỉnh lại cấu trúc Chương 2 theo hai nhóm rõ ràng: cơ sở lý thuyết gồm DDD, DSL, MDSE, OCL, Event-B; và tổng quan nghiên cứu gồm biểu diễn DM, tích hợp mối quan tâm, chuyển đổi mô hình.	Chương 2, đặc biệt 2.1–2.6.
6.5	Các định nghĩa hình thức nằm rải rác, nên đưa về Chương 2.	NCS tiếp thu và giải trình làm rõ: Các khái niệm nền tảng nên đặt ở Chương 2. Tuy nhiên, các định nghĩa hình thức như UDML, trạng thái hợp lệ, ánh xạ AGL–UDML, Event-B được đặt ở Chương 4 vì chúng là kết quả đề xuất, không phải kiến thức nền. Việc đặt tại Chương 4 giúp định nghĩa gắn trực tiếp với phương pháp UDML.	Giữ ở 4.4.1–4.4.3; bổ sung dẫn nhập ở Chương 2 nếu cần.
6.6	Tên Chương 4 "mối quan tâm" chưa rõ.	NCS tiếp thu. Làm rõ "mối quan tâm" trong luận án là các khía cạnh chuyên biệt của mô hình miền, trước hết gồm cấu trúc, hành vi, bảo mật, và có thể mở rộng theo cơ chế DSL chuyên biệt theo mối quan tâm. Luận án đã mô tả UDML tích hợp các DSL theo mối quan tâm vào mô hình miền hợp nhất.	Chương 4, đặc biệt 4.1, 4.2, 4.3.
6.7	Chương 6 trình bày công cụ rồi chuyển sang trả lời câu hỏi nghiên cứu còn đột ngột.	NCS tiếp thu. Chỉnh Chương 6 theo mạch rõ hơn: công cụ hỗ trợ → ca nghiên cứu → tiêu chí đánh giá → kết quả theo từng kỹ thuật → tổng hợp trả lời câu hỏi nghiên cứu.	Chương 6, đặc biệt 6.2, 6.3, 6.4.

6.8	Hình vẽ chất lượng chưa cao, nên vẽ lại vector và Việt hóa.	NCS tiếp thu. Rà soát và thay thế các hình có độ phân giải thấp bằng bản vector; Việt hóa nhãn tiếng Anh trong các hình chính nếu không cần giữ nguyên thuật ngữ kỹ thuật.	Hình 1.3, 1.4, 3.3, 3.5 và các hình liên quan.
6.9	Một số câu thiếu trích dẫn gần vị trí phát biểu, ví dụ nhận định về Evans.	NCS tiếp thu. Bổ sung trích dẫn trực tiếp tại các câu khẳng định quan trọng, đặc biệt các phát biểu liên quan đến DDD, DM, Evans, DSL, MDSE, Event-B.	Chương 1, 2.
6.10	Các đoạn mã nguồn chưa được đánh số và đặt tiêu đề.	NCS tiếp thu. Bổ sung nhãn dạng Listing cho các đoạn mã, kèm tiêu đề và tham chiếu trong nội dung.	Các đoạn mã ở Chương 2, 3, 4.
6.11	Một số câu chưa rõ nghĩa, ví dụ "tích hợp mô hình miền".	NCS tiếp thu. Sửa thành diễn đạt rõ hơn: tích hợp các khía cạnh hành vi và ràng buộc nghiệp vụ vào mô hình miền hoặc tích hợp các DSL theo mối quan tâm vào UDML.	Chương 3, 4, đặc biệt đoạn mở đầu Chương 3.
6.12	Một số từ viết tắt chưa có tên tiếng Anh đầy đủ.	NCS tiếp thu. Bổ sung tên đầy đủ cho các từ viết tắt khi xuất hiện lần đầu, ví dụ aDSL – annotation-based Domain-Specific Language.	Danh mục từ viết tắt; các chương 1–4.
7. PGS. TS. Phạm Ngọc Hùng (Chủ tịch)			
7.1	Quy mô thực nghiệm còn hạn chế.	NCS tiếp thu. Trong luận án, thực nghiệm hiện tập trung vào các ca nghiên cứu CourseMan, OrderMan, ProcessMan và OJS để đánh giá tính khả thi, tính biểu đạt và khả năng sinh bản mẫu. Các ca này đại diện cho nhiều miền khác nhau nhưng chưa phải hệ thống công nghiệp quy mô lớn. Đây là hạn chế của luận án.	Phần này được phân tích làm rõ và đưa vào hướng nghiên cứu trong tương lai tại Chương 7, mục 7.2
7.2	Nên bổ sung thực nghiệm quy mô lớn hơn để minh chứng khả năng áp dụng thực tiễn.	NCS tiếp thu. Luận án đã chứng minh được chuỗi kỹ thuật từ biểu diễn mô hình miền → hợp nhất UDML → chuyển đổi mô hình → sinh bản mẫu phần mềm. Tuy nhiên, đánh giá trên hệ thống công nghiệp lớn với nhiều	Phần này được phân tích làm rõ và đưa vào hướng nghiên cứu trong tương lai tại

		mô-đun, nhiều thực thể và nhiều quy tắc nghiệp vụ sẽ được xác định là hướng nghiên cứu tiếp theo.	Chương 7, mục 7.2
7.3	Xu hướng sử dụng AI trong bài toán này đang được quan tâm lớn.	NCS tiếp thu. Luận án đã bổ sung nội dung cho mục 2.5 – Hướng tiếp cận sử dụng AI/LLM, tuy nhiên nội dung này chỉ đóng vai trò khảo sát xu hướng. Trọng tâm của luận án là hướng DSL/MDSE có cơ sở ngữ nghĩa hình thức, không phải phương pháp AI.	NCS thực hiện cập nhật, bổ sung đầy đủ hơn về phần AI/LLM. Mục 2.5 trang 46 đến trang 50.
7.4	Nên bổ sung thảo luận, so sánh với hướng AI.	NCS tiếp thu. Luận án xem xét bổ sung thảo luận rằng AI/LLM có tiềm năng hỗ trợ sinh mô hình, sinh DSL hoặc sinh mã, nhưng khó đảm bảo tính đúng đắn hình thức nếu không có mô hình trung gian và cơ chế kiểm chứng. Trong luận án, UDML + Event-B cung cấp nền tảng kiểm chứng mà hướng AI thuần túy chưa bảo đảm.	NCS bổ sung thảo luận tại mục 7.2 chương 7 trang 181 đến trang 184
7.5	Cần chỉnh sửa văn phong để tăng tính phân tích đánh giá.	NCS tiếp thu. Thực hiện rà soát lại các đoạn mô tả còn liệt kê, bổ sung phân tích về lý do chọn phương pháp, giới hạn áp dụng, và liên hệ giữa kết quả thực nghiệm với các câu hỏi nghiên cứu.	Đã chỉnh sửa văn phong toàn bộ luận án
7.6	Rà soát lỗi trình bày.	NCS tiếp thu. Rà soát toàn bộ luận án về chính tả, thuật ngữ, đánh số hình/bảng/mã nguồn, chất lượng hình ảnh và sự nhất quán giữa phần mở đầu, kết luận và các chương.	Đã rà soát lỗi trình bày toàn bộ luận án

Trả lời các câu hỏi của Hội đồng:

- PGS.TS. Cao Tuấn Dũng

Câu hỏi 1:

Nếu có nhiều DSL, luận án có chứng minh tính bảo toàn ngữ nghĩa không?

Trả lời:

Hiện tại với tiếp cận là tích hợp nhiều DSL vào mô hình miền hợp nhất UDML các đề xuất được trình bày theo mạch này, và luận án đã giải quyết và chứng minh tính bảo toàn ngữ nghĩa.

Đối với nhiều DSL được tích hợp thì luận án đã giải quyết tính bảo toàn ngữ nghĩa bằng cách định nghĩa hình thức sử dụng các phép chuyển hình thức để kiểm chứng bằng công cụ toán học (Event-B).

Câu hỏi 2:

Xây dựng kho tri thức miền, UDML có đáp ứng yêu cầu cho kho này chưa?

Trả lời:

Luận án đã áp dụng thành công với các tri thức miền: CourseMan, OrderMan, ProcessMan và nỗ lực mở rộng kho này cho các miền phức tạp hơn. Luận án cũng đã bổ sung vào hướng phát triển trong tương lai để hoàn thiện kho tri thức miền cho UDML

- TS. Trần Việt Hoàng

Câu hỏi 1:

Vì sao không đưa toàn bộ định nghĩa vào Chương 2?

Trả lời:

Các khái niệm nền tảng đã được trình bày ở Chương 2. Các định nghĩa hình thức về UDML, trạng thái hợp lệ, ánh xạ AGL–UDML và Event-B là kết quả đề xuất của luận án, nên được đặt ở Chương 4 để gắn trực tiếp với phương pháp hợp nhất và kiểm chứng mô hình miền.

Câu hỏi 2:

"Mối quan tâm" trong Chương 4 là gì?

Trả lời:

"Mối quan tâm" là từng khía cạnh chuyên biệt của mô hình miền được biểu diễn bằng DSL riêng. Trong Thesis, các mối quan tâm chính được xử lý là cấu trúc, hành vi và bảo mật; ràng buộc được tích hợp thông qua CAP. Chương 4 không khẳng định đã xử lý mọi mối quan tâm có thể có, mà đề xuất một cơ chế UDML có khả năng tích hợp các DSL theo mối quan tâm.

Trên đây là giải trình chi tiết của nghiên cứu sinh về việc tiếp thu, chỉnh sửa, bổ sung luận án theo các ý kiến đóng góp của các thành viên tham gia cũng như các nhà khoa học tham dự.

Xin trân trọng cảm ơn./.

CÁN BỘ HƯỚNG DẪN	NGHIÊN CỨU SINH
Đỗ Đức Thái	Lê Văn Vinh