ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

---------- ---------

PHẠM VĂN TRƯỜNG

NGHIÊN CỨU KỸ THUẬT CHUYỂN ĐỔI MÔ HÌNH SANG VĂN BẢN VÀ ỨNG DỤNG VÀO SINH MÃ NGUỒN JAVA

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2020

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

---------- ---------

PHẠM VĂN TRƯỜNG

NGHIÊN CỨU KỸ THUẬT CHUYỂN ĐỔI MÔ HÌNH SANG VĂN BẢN VÀ ỨNG DỤNG VÀO SINH MÃ NGUỒN JAVA

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm
Mã số học viên: 17025008

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2020

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của cá nhân tôi, được thực hiện qua sự hướng dẫn tận tình của thầy ***TS. Đặng Đức Hạnh.***

Các nội dung nghiên cứu và kết quả thực nghiệm được trình bày trong luận văn là hoàn toàn trung thực, do cá nhân tôi cài đặt, cấu hình và lên kịch bản. Các kiến thức hàn lâm được tôi chắt lọc từ các tài liệu tham khảo trên mạng, sách và các bài báo khoa học của các tác giả uy tín trong cùng lĩnh vực nghiên cứu.

Hà Nội, tháng 12 năm 2020

Người thực hiện

Phạm Văn Trường

LỜI CẢM ƠN

Lời đầu tiên, tôi xin dành lời cảm ơn sâu sắc nhất tới giảng viên hướng dẫn của tôi, ***TS. Đặng Đức Hạnh*** – Giảng viên Bộ môn ***Công nghệ Phần mềm*** – Khoa ***Công nghệ Thông tin*** – Trường ***Đại học Công nghệ - ĐHQGHN***, là người đã trực tiếp định hướng và hướng dẫn tôi hoàn thành luận văn này. Tôi cũng xin được cảm ơn sự hỗ trợ của đề tài nghiên cứu khoa học cấp Đại học Quốc gia Hà Nội, mã số ***QG.20.54***.

Với cá nhân tôi, lĩnh vực này là một lĩnh vực hoàn toàn mới và vô cùng trừu tượng, thời điểm ban đầu còn gặp nhiều khó khăn về việc nghiên cứu cũng như cách tiếp cận, nhưng qua sự định hướng cả về kĩ năng chuyên môn và phương pháp nghiên cứu của Thầy, tôi đã thu được những kiến thức nhất định sau khi thực hiện luận văn này. Bên cạnh đó, trong khoảng thời gian học tập và tham gia nghiên cứu tại Trường Đại học Công nghệ – ĐHQGHN, với sự giảng dạy và giúp đỡ của các Thầy/Cô cùng các bạn học viên, tôi đã học được rất nhiều kiến thức bổ ích và có nhiều tính thực tiễn. Những kiến thức gặt hái được giúp tôi có khả năng tư duy, phân tích, tổng hợp các vấn đề một cách khoa học, và thậm chí áp dụng được khá nhiều vào công việc tôi đang làm.

Một lần nữa, tôi xin gửi lời cảm ơn chân thành nhất tới các Thầy/Cô và các bạn. Tôi cũng xin gửi lời cảm ơn tới gia đình đã luôn luôn động viên tôi vượt qua khó khăn và hoàn thành tốt công việc học tập và nghiên cứu tại đây. Do lĩnh vực nghiên cứu được đề cập trong luận văn còn mới lạ, chưa được áp dụng nhiều, và vẫn còn đang trong giai đoạn phát triển, cho nên tôi đã gặp không ít khó khăn trong việc nghiên cứu. Hạn chế về mặt thời gian và phát sinh từ công việc hiện khiến tôi chưa tập trung hết khả năng và sự sáng tạo để khai thác các vấn đề một cách kỹ càng và đầy đủ hơn nữa. Do vậy luận văn sẽ còn nhiều hạn chế, rất mong nhận được ý kiến đóng góp của các Thầy/Cô và bạn đọc quan tâm.

Xin chân thành cảm ơn.

Hà Nội, tháng 12 năm 2020

Người thực hiện

Phạm Văn Trường

MỤC LỤC

DANH MỤC KÝ HIỆU CÁC CHỮ VIẾT TẮT

Chữ viết tắt	Chú giải
MDSE	***Model-Driven Software Enginerring***
IDE	***Integrated Development Environment***
MDD	***Model-Driven Development***
MDA	***Model-Driven Architecture***
UML	***Unified Modeling Language***
XMI	***XML Metadata Interchange***
M2M	***Model To Model***
M2T	***Model To Text***
ATL	***ATLAS Transformation Language***
OCL	***Object Constraint Language***
PIM	***Platform-Independent Model***
PSM	***Platform-Specific Models***
JSP	***Java Server Pages***
MOF	***Meta-Object Facility***
XSTL	***eXtensible Stylesheet Language Transformations***

DANH MỤC HÌNH ẢNH VÀ ĐỒ THỊ

DANH MỤC BẢNG BIỂU

MỞ ĐẦU

Hiện nay ngành công nghệ thông tin nói chung và phát triển phần mềm nói riêng đang ngày một phát triển và là xu thế tất yếu để vươn lên của mỗi quốc gia. Các hệ thống công nghệ thông tin cũng ngày một đa dạng, phức tạp và cồng kềnh hơn trước rất nhiều, do nhu cầu phát triển và tiến hóa về mặt kiến trúc. Các hệ thống không chỉ dừng lại ở một phiên bản phát triển mà còn phát sinh nhiều bản cập nhật. Khi trải nghiệm người dùng và các chức năng mới được thêm vào cùng với các yêu cầu mới về hệ thống được đưa ra, và các phiên bản phần mềm mới lại ra đời sao cho có thể hoạt động tốt được trên nền tảng mới. Điều này tất nhiên nhằm đáp ứng nhu cầu người dùng thực tế, mặt khác lại gây ra rất nhiều phiền phức cho các nhà phát triển, và lúng túng trong các khâu quản lý, duy trì phần mềm một cách thủ công nếu họ vẫn áp dụng phương pháp phát triển cũ. Công sức và thời gian tiêu tốn cho việc duy trì đó là rất lớn. Vậy bài toán đặt ra đó là làm thế nào để tăng tính tự động, giảm tải các chi phí phát sinh và bớt các khâu thủ công trong quá trình phát triển phần mềm.

Để giải quyết vấn đề này, các nhà phát triển đã chuẩn bị các nguồn lực và các đầu vào cần thiết bằng cách mô hình hóa một cách bao quát các thành phần của hệ thống, các bộ phận có liên quan thông qua các quy tắc và cú pháp tương ứng cho từng phương pháp. Việc này được thực hiện thông qua ngôn ngữ mô hình hóa. Kết quả của việc mô hình hóa sẽ giúp việc phát triển phần mềm đi đúng hướng, giải quyết vấn đề quản lý các phiên bản, tăng tính tự động hóa, qua đó tăng hiệu suất, giảm chi phí phát triển phần mềm. Từ yêu cầu phát triển đó, phương pháp phát triển phần mềm hướng mô hình ra đời, tập trung mô hình hóa phần mềm, từ đó chuyển đổi sang các mô đun, mã nguồn có thể thực thi bằng các công cụ chuyển đổi mô hình.

Từ những yêu cầu thực tế và bức thiết trong phát triển phần mềm, tôi đã lựa chọn đề tài "***Nghiên cứu kỹ thuật chuyển đổi mô hình sang văn bản và ứng dụng vào sinh mã nguồn Java***" nhằm đáp ứng việc nghiên cứu phương pháp chuyển đổi hướng mô hình. Để thực hiện giai đoạn mô hình hóa, các mô hình cũng cần được bổ sung thêm các thông tin thực hiện chương trình, ví dụ như các yêu cầu về thành phần của hệ thống, các ràng buộc trong nền tảng và các chức năng,... Ngoài ra, các hành vi của ứng dụng cũng cần được mô hình hóa, như là sự tương tác giữa các đối tượng, các phương thức sử dụng, khai báo và xử lý kết quả trả về...bên cạnh việc tập trung vào tìm hiểu và nghiên cứu các kỹ thuật chuyển đổi mô hình dựa theo giải pháp ModelToText, luận văn hướng tới việc xây dựng quy tắc sinh mã nguồn Java một cách tự động và áp dụng cho bài toán cụ thể. Kết quả đạt được sẽ gồm các tệp mã nguồn Java kết hợp với các cấu hình cần thiết để có thể thực thi và kiểm tra chương trình sau khi sinh tự động.

Cấu trúc luận văn được chia thành các mục như sau:

Chương 1: Tổng quan về phương pháp phát triển hướng mô hình trong phát triển phần mềm (MDSD/MDSE). Chương đầu tiên tập trung đề cập đến phương pháp phát triển hướng mô hình trong phát triển phần mềm, tìm hiểu các thuật ngữ và các thành phần thiết yếu. Sau đó trình bày cụ thể hơn về phạm vi chuyển đổi mô hình, gồm chuyển đổi mô hình sang mô hình và mô hình sang văn bản.

Chương 2: Hướng tới khía cạnh cụ thể của chuyển đổi mô hình sang văn bản, với ứng dụng phổ biến nhất là sinh mã nguồn tự động (Code generation). Khi áp dụng phương pháp này, ngoài các chế tác có thể sinh ra từ các mô hình thì chương này tập trung vào chế tác sinh mã nguồn tự động, sử dụng công cụ chuyển đổi, ngôn ngôn ngữ chuyển đổi, cụ thể sẽ áp dụng ngôn ngữ Acceleo. Qua đó thống kê đánh giá các mặt tích cực, hạn chế của phương pháp được áp dụng.

Chương 3: Với khía cạnh sinh mã nguồn đã trình bày trong chương trước, chương 3 tập trung nghiên cứu bài toán cụ thể sinh mã nguồn Java từ biểu đồ lớp với input/output cần thiết. Ứng với bài toán này, chương 3 mô tả các quy tắc chuyển đổi, ví dụ áp dụng, template, dữ liệu mẫu…

Chương 4: Cài đặt ví dụ cụ thể của phương pháp áp dụng chuyển đổi mô hình sang mã nguồn Java và kết quả thực nghiệm, đánh giá kết quả đạt được (kết quả, hạn chế) và hướng nghiên cứu, phát triển mới trong tương lai.

CHƯƠNG 1. KIẾN THỨC NỀN TẢNG

Chương đầu tiên sẽ khái quát các kiến thức nền tảng xoay quanh đề tài nghiên cứu, các thuật ngữ chính và các công cụ hỗ trợ. Chương gồm các phần chính: tổng quan phát triển phần mềm hướng mô hình và các khái niệm cốt lõi, chuyển đổi từ mô hình sang mô hình hoặc từ mô hình sang văn bản, và phần cuối là tổng kết chương.

1.1. Phát triển phần mềm hướng mô hình

Mô hình (***model***) là một sự trừu tượng của hệ thống thường được sử dụng để thay thế cho hệ thống đang được nghiên cứu. Nhìn chung, một mô hình đại diện cho một cái nhìn đơn giản và phân mảnh từng phần của một hệ thống, do vậy, việc tạo ra nhiều mô hình là khá cần thiết để thể hiện và hiểu rõ hơn về hệ thống đang được tiếp cận nghiên cứu. Mô hình hóa là một phương pháp kỹ thuật nổi tiếng được áp dụng bởi các lĩnh vực kỹ thuật cũng như các lĩnh vực khác như Vật lý, Toán học, Sinh học, Kinh tế, Chính trị và Triết học… Tuy nhiên, luận văn sẽ chỉ tập trung vào các mô hình trong bối cảnh cụ thể đó là lĩnh vực kỹ thuật phần mềm. Điều đó có nghĩa là các mô hình có bản chất dựa trên ngôn ngữ và có xu hướng mô tả hoặc quy định một hệ thống nào đó, ví dụ với các mô hình trong Toán học vốn được hiểu là diễn giải một lý thuyết.

Các mô hình cho phép chia sẻ tầm nhìn chung giữa các bên liên quan về cả mặt kỹ thuật và phi kỹ thuật, tạo điều kiện và thúc đẩy giao tiếp giữa các bên. Hơn nữa, các mô hình còn làm cho việc lập kế hoạch dự án hiệu quả hơn đồng thời cung cấp một cái nhìn phù hợp hơn về hệ thống được phát triển qua đó cho phép kiểm soát dự án theo các tiêu chí đã đặt ra từ ban đầu. Ngày nay, một xu hướng tiếp cận mới đã xuất hiện, với góc nhìn các mô hình không chỉ là các chế tác về mặt tài liệu, mà còn là chế tác về việc triển khai thực hiện trong lĩnh vực kỹ thuật phần mềm, cho phép tạo hoặc thực thi tự động các hệ thống phần mềm. Các đề xuất xoay quanh mô hình như vậy được gọi chung là "Kỹ thuật phát triển phần mềm theo hướng mô hình (***Model-Driven Software Enginerring – MDSE***)". Đây không phải là một kỹ thuật cụ thể, mà là một cách tiếp cận, nhằm mục đích định hướng vấn đề theo hướng mô hình hóa.

Kỹ thuật phát triển phần mềm theo hướng mô hình (***MDSE***) có thể được hiểu như một phương pháp luận nhằm áp dụng các ưu điểm của mô hình hóa vào các kỹ thuật phần mềm. Trong ngữ cảnh của MDSE, các khái niệm cốt lõi là: mô hình và phép chuyển đổi (***transformation***) tức là các hoạt động thao tác trên mô hình. Dưới đây là phương trình nổi tiếng của tác giả Niklaus Wirth [8] cho thấy các chương trình được tạo ra từ các thành phần cốt lõi:

Algorithms + Data Structures = Programs

Điều này có nghĩa là các chương trình được tạo nên từ 2 thành phần chính gồm: Thuật toán (***Algorithms***) và Cấu trúc dữ liệu (***Data Structures***). Tuy nhiên trong ngữ cảnh MDSE, điều này sẽ được nhìn nhận dưới góc nhìn khác, chương trình (có thể hiểu là một sản phẩm phần mềm) được tạo thành từ 2 thành phần chính: mô hình và các phép chuyển đổi:

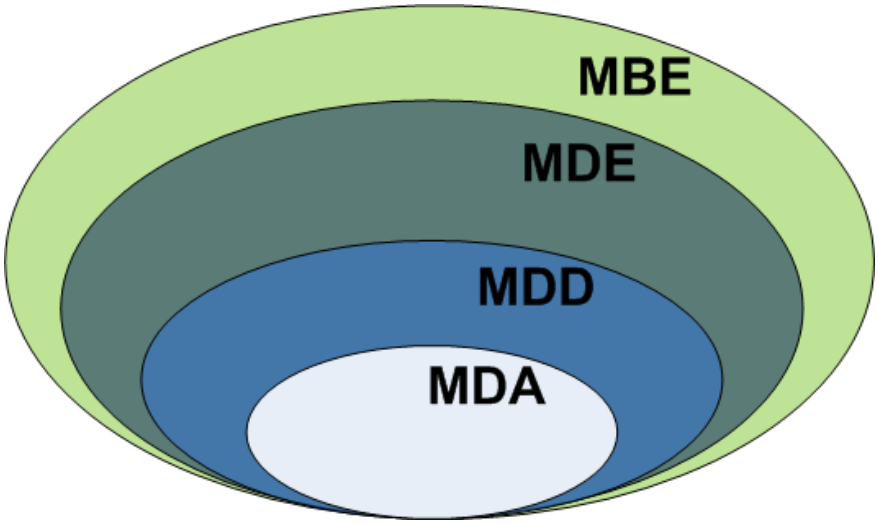
Models + Transformations = Software

Rõ ràng, cả mô hình và phép chuyển đổi đều cần được thể hiện dưới một dạng ký hiệu nào đó hoặc ngôn ngữ nào đó, mà trong MDSE gọi là ngôn ngữ mô hình hóa (theo cách tương tự như trong phương trình *Wirth*, các thuật toán và cấu trúc dữ liệu cần được định nghĩa trong một số ngôn ngữ lập trình). Tuy nhiên, điều này vẫn chưa đủ. Phương trình không cho chúng ta biết loại mô hình nào và theo thứ tự nào, ở mức trừu tượng nào… cần được xác định tùy thuộc vào loại phần mềm chúng ta muốn tạo ra. Đó là lúc quá trình lựa chọn theo mô hình phát huy tác dụng. Cuối cùng cần một bộ công cụ thích hợp để làm cho MDSE khả thi và hữu ích hơn trong thực tế. Đối với lập trình, chúng ta cần các IDE (***Integrated Development Environment*** – Môi trường tích hợp phát triển) cho phép chúng ta xác định các mô hình và phép chuyển đổi cũng như trình biên dịch hoặc trình thông dịch để thực thi chúng nhằm tạo ra các tạo tác phần mềm cuối cùng.

Dưới góc nhìn của MDSE giống như nguyên tắc cơ bản "mọi thứ đều là một đối tượng", thì MDSE coi "mọi thứ đều là mô hình", hữu ích trong thúc đẩy công nghệ theo hướng đơn giản, tổng quát và sức mạnh tích hợp cho các ngôn ngữ và công nghệ hướng đối tượng trong những năm 1980 [5]. Trên thực tế trong bối cảnh này, người ta có thể nghĩ ngay đến tất cả các thành phần được mô tả ở trên như một thứ có thể được mô hình hóa. Đặc biệt, người ta có thể xem các phép chuyển đổi là các mô hình hoạt động cụ thể trên các mô hình. Bản thân định nghĩa của ngôn ngữ mô hình hóa có thể được xem như là một mô hình: MDSE đề cập đến quy trình này là siêu mô hình hóa (tức là mô hình hóa một mô hình, hoặc hơn thế…). Theo cách này, người ta có thể định nghĩa các mô hình cũng như các quy trình, công cụ phát triển và chương trình kết quả. Phần tiếp theo, luận văn sẽ đi lướt qua các khái niệm, thuật ngữ chính nhằm tăng sự hiểu biết cần thiết đối với phương pháp luận MDSE trước khi áp dụng vào bài toán trong ngữ cảnh cụ thể.

1.1.1. Các thuật ngữ chính

Hình 1.1 dưới đây cho thấy một cách trực quan về mối quan hệ giữa các thuật ngữ mô tả phương pháp mô hình hóa [8].



Hình 1. 1. Mối quan hệ giữa các thuật ngữ viết tắt MD.*

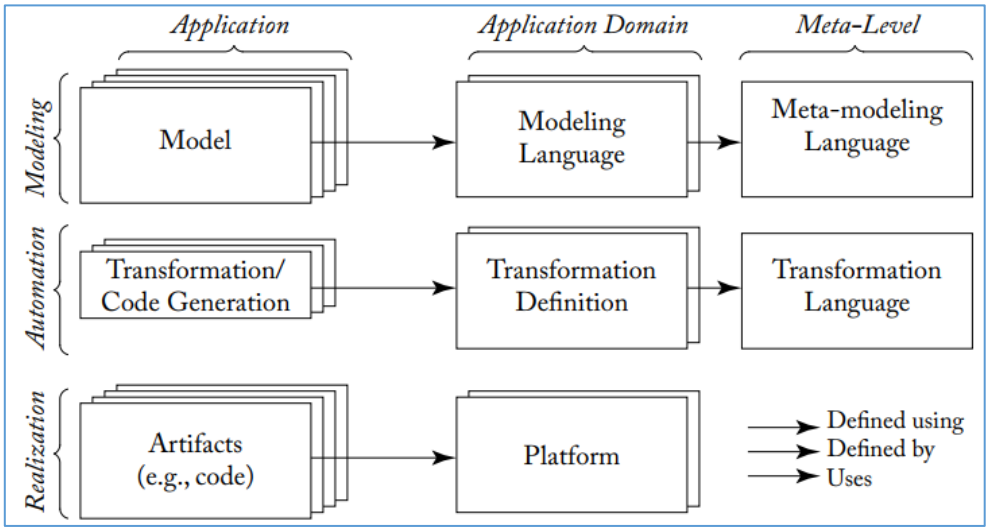
Trong đó:

- Phát triển hướng mô hình (***Model-Driven Development – MDD***) là mô hình phát triển sử dụng các mô hình làm thành phần chính của quá trình phát triển. Thông thường, trong MDD, việc triển khai được tạo tự động (bán tự động) từ các mô hình.
- Kiến trúc hướng mô hình (***Model-Driven Architecture – MDA***) là một kiến trúc phát triển của MDD do "Nhóm quản lý đối tượng" (***Object Management Group – OMG***) đề xuất và do đó dựa trên việc sử dụng các tiêu chuẩn OMG. Do đó, MDA có thể được coi là một tập con của MDD, trong đó các ngôn ngữ mô hình hóa và chuyển đổi được tiêu chuẩn hóa bởi OMG.
- Kỹ thuật dựa trên mô hình hoặc "phát triển dựa trên mô hình" (***Model-based engineering – MBE***) để chỉ phiên bản MDE mềm hơn. Có nghĩa là quy trình MBE là một quy trình trong đó các mô hình phần mềm đóng một vai trò quan trọng mặc dù chúng không nhất thiết phải là tạo tác chính của sự phát triển (tức là các mô hình không "điều khiển" quy trình như trong MDE). Một ví dụ sẽ là một quá trình phát triển trong đó, trong giai đoạn phân tích, các nhà thiết kế chỉ định các mô hình miền của hệ thống nhưng sau đó các mô hình này được giao trực tiếp cho các lập trình viên dưới dạng bản thiết kế để lập trình theo cách thủ công (không liên quan đến việc tạo mã tự động và không có định nghĩa rõ ràng của bất kỳ mô hình nền tảng cụ thể nào). Trong quá trình này, các mô hình vẫn đóng một vai trò quan trọng nhưng không phải là tạo tác trung tâm của quá trình phát triển và có thể kém hoàn thiện hơn (tức là chúng có thể được sử dụng nhiều hơn như các bản thiết kế

hoặc bản phác thảo của hệ thống) so với các mô hình trong cách tiếp cận MDD. MBE là một tập con của MDE. Tất cả các quy trình dựa theo hướng mô hình hóa đều dựa trên mô hình, nhưng không phải ngược lại.

1.1.2. Các cấp độ của MDSE

Phần này sẽ đi vào chi tiết về các thành phần của MDSE, làm rõ rằng mô hình hóa có thể được áp dụng ở các mức độ trừu tượng khác nhau. Bên cạnh đó sẽ mô tả vai trò và bản chất của các phép chuyển đổi mô hình. MDSE cung cấp một tầm nhìn toàn diện để phát triển hệ thống. Hình 1.2 dưới đây cho thấy tổng quan về các khía cạnh chính được xem xét trong MDSE và tóm tắt cách giải quyết các vấn đề khác nhau. MDSE tìm kiếm các giải pháp theo các khía cạnh: khái niệm hóa (các cột trong hình) và thực hiện hóa (các hàng trong hình).



Hình 1. 2. Tổng quan về phương pháp MDSE (quy trình từ trên xuống).

Vấn đề thực hiện hóa (***implement***) [8] liên quan đến việc ánh xạ các mô hình tới một số hệ thống đang thực thi hiện tại hoặc trong tương lai. Do đó sẽ bao gồm việc xác định ba khía cạnh cốt lõi.

- Cấp độ mô hình (***Model-level***): nơi các mô hình được xác định.
- Cấp độ hiện thực hóa (***Realization-level***): nơi các giải pháp được thực hiện thông qua các mã nguồn thực sự được sử dụng trong hệ thống đang thực thi.
- Cấp độ tự động hóa (***Automation-level***): nơi đặt các ánh xạ từ cấp độ mô hình hóa đến cấp độ hiện thực hóa.

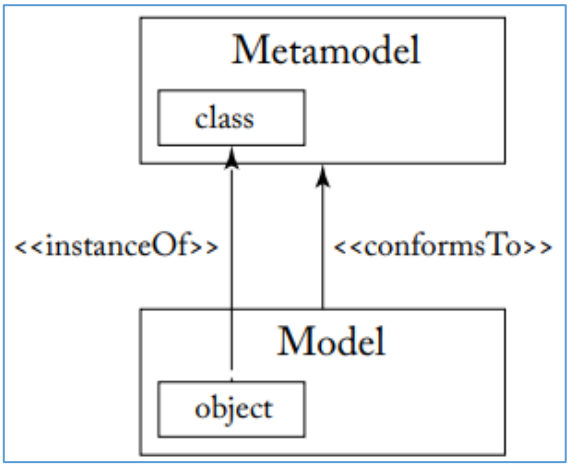
Vấn đề khái niệm hóa (***conceptualization***) được định hướng để xác định các mô hình khái niệm để mô tả hệ thống một cách thực tế. Điều này có thể được áp dụng ở ba mức chính sau đây:

- ***Mức ứng dụng***: nơi mô hình của các ứng dụng được xác định, các quy tắc chuyển đổi được thực hiện và các thành phần đang chạy thực tế được tạo ra.
- ***Mức miền ứng dụng***: nơi xác định định nghĩa của ngôn ngữ mô hình hóa, phép chuyển đổi và nền tảng triển khai cho một miền cụ thể.
- ***Mức meta-level***: nơi xác định khái niệm về mô hình và các phép chuyển đổi.

Luồng cốt lõi của MDSE là từ các mô hình ứng dụng xuống quá trình chạy thực, thông qua các chuyển đổi mô hình ở mức độ tiếp theo. Điều này cho phép tái sử dụng các mô hình và thực thi hệ thống trên các nền tảng khác nhau. Ở cấp độ hiện thực, phần mềm đang chạy dựa trên một nền tảng cụ thể (được xác định cho một miền ứng dụng cụ thể) để thực thi nó. Để thực hiện điều này, các mô hình được chỉ định theo ngôn ngữ mô hình hóa, lần lượt được xác định theo ngôn ngữ siêu mô hình hóa. Việc thực hiện chuyển đổi được xác định dựa trên một tập hợp các quy tắc chuyển đổi, được xác định bằng cách sử dụng một ngôn ngữ chuyển đổi cụ thể. Trong hình này, việc xây dựng hệ thống được kích hoạt bởi một quy trình từ trên xuống từ các mô hình quy định nhằm xác định phạm vi bị giới hạn như thế nào và mục tiêu nên được thực hiện như thế nào. Mặt khác, tính trừu tượng được sử dụng từ dưới lên để tạo ra các mô hình mô tả của hệ thống.

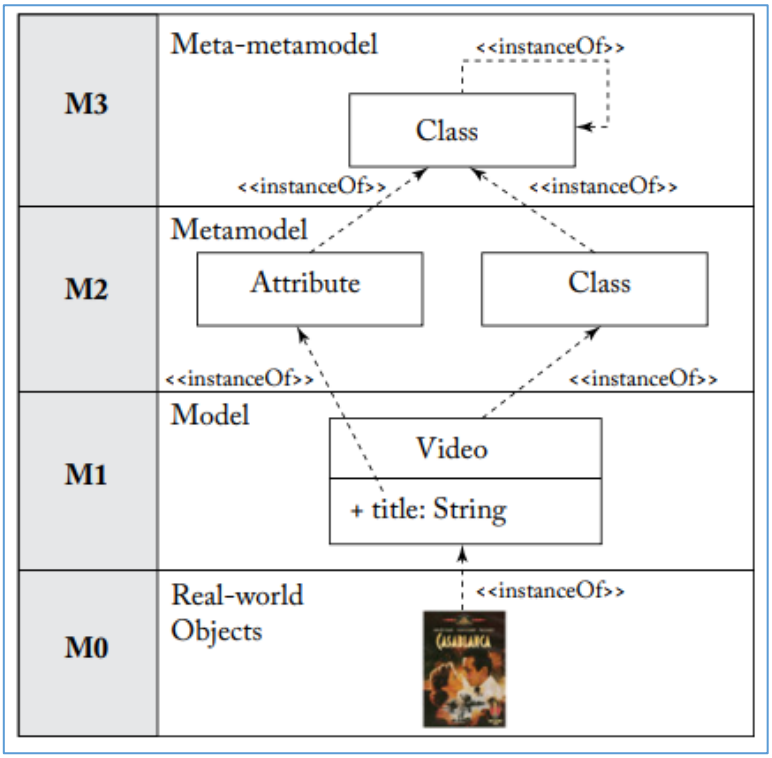
1.1.3. Meta-model

Khi các mô hình đóng một vai trò quan trọng và phổ biến trong MDSE, một bước tiếp theo để định nghĩa các mô hình là thể hiện bản thân chính các mô hình đó giống như là "các thể hiện" của một số mô hình trừu tượng hơn. Do đó, giống hệt như cách chúng ta định nghĩa một mô hình giống như một sự trừu tượng nào đó của các hiện tượng trong thế giới thực, chúng ta có thể định nghĩa một ***meta-model*** (siêu mô hình) [8] như một sự trừu tượng, làm nổi bật các thuộc tính của chính mô hình đó. Thực tế, meta-model về cơ bản cấu thành định nghĩa của một ngôn ngữ mô hình hóa, vì chúng cung cấp cách mô tả toàn bộ lớp mô hình có thể được biểu diễn bằng ngôn ngữ đó. Do đó, người ta có thể định nghĩa các mô hình thực tế, và sau đó là các mô hình mô tả mô hình (gọi là meta-model). Một cách cụ thể, một mô hình tuân theo meta-model của nó khi tất cả các phần tử của nó có thể được biểu thị dưới dạng các thể hiện của các lớp meta-model.



Hình 1. 3. Mối quan hệ giữa conformsTo và instanceOf.

Hình 1.4 cho thấy một ví dụ về meta-model: các đối tượng trong thế giới thực được hiển thị ở mức M0 (trong ví dụ này là một bộ phim), đại diện được mô hình hóa của chúng được hiển thị ở cấp độ M1, nơi mô hình mô tả khái niệm Video với các thuộc tính. Meta-model của mô hình này được hiển thị ở mức M2 và mô tả các khái niệm được sử dụng ở M1 để xác định mô hình, đó là: Lớp (Class), Thuộc tính (Attribute) và Sự thể hiện (Instance). Cuối cùng, cấp độ M3 xác định các khái niệm được sử dụng tại M2: tập hợp này thu gọn trong khái niệm Lớp duy nhất. Rõ ràng là không cần thêm các cấp meta-model vượt quá M3, vì chúng sẽ luôn chỉ bao gồm khái niệm Lớp.

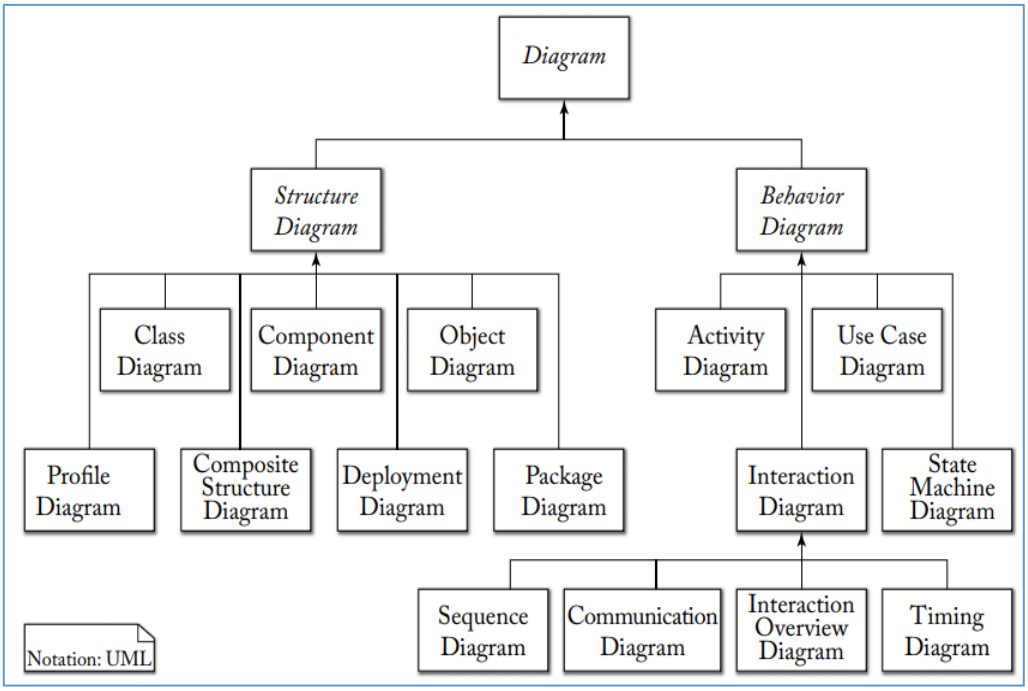


Hình 1. 4. Sự phân cấp model, metamodel và meta-metamodel.

Meta-model có thể được sử dụng cho việc: xác định ngôn ngữ mới để mô hình hóa hoặc lập trình, xác định ngôn ngữ mô hình hóa mới để trao đổi và lưu trữ thông tin, và cuối cùng nhằm xác định các thuộc tính hoặc tính năng mới được liên kết với thông tin hiện có (***meta-data*** – siêu dữ liệu).

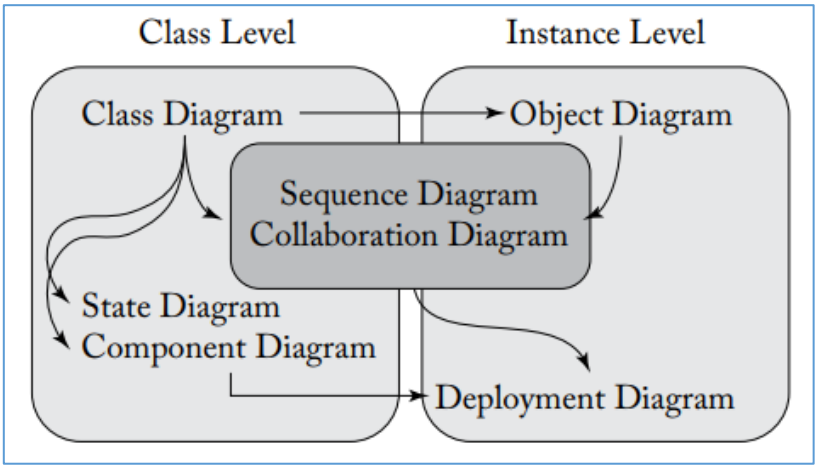
1.1.4. Unified Modeling Language

Phần này nhằm cung cấp một cái nhìn tổng quan về ngôn ngữ UML (***Unified Modeling Language*** – Ngôn ngữ mô hình hóa đồng nhất) [8]. Bên cạnh việc được biết đến và chấp nhận rộng rãi, UML như một ngôn ngữ khá thú vị để thảo luận về các tính năng và khía cạnh được trình bày với các đặc điểm chung cho các ngôn ngữ mô hình hóa. Nhìn chung UML là một bộ ngôn ngữ chính thức, vì nó bao gồm một tập hợp các biểu đồ khác nhau để mô tả một hệ thống từ các khía cạnh khác nhau. Hình 1.5 là sự phân loại của biểu đồ UML. Có 7 biểu đồ khác nhau có thể được sử dụng để mô tả các khía cạnh tĩnh (cấu trúc) của hệ thống, trong khi 7 biểu đồ khác có thể được sử dụng để mô tả các khía cạnh động (hành vi).



Hình 1. 5. Phâm loại biểu đồ UML.

Như thể hiện trong Hình 1.6 dưới đây, một số trong số các sơ đồ này mô tả đặc điểm của các lớp (tức là các khái niệm trừu tượng), trong khi các sơ đồ khác được sử dụng để mô tả các tính năng và hành vi của các mục riêng lẻ (tức là các cá thể). Một số sơ đồ có thể được sử dụng để mô tả cả hai cấp độ.



Hình 1. 6. Phân tách giữa các mô hình dựa trên lớp.

Luận văn sẽ áp dụng một trong số những biểu đồ thể hiện bởi ngôn ngữ UML, nhằm xác định meta-model cụ thể của một model.

1.1.5. Biểu đồ lớp

1.1.5.1. Định nghĩa

Biểu đồ lớp là một loại biểu đồ UML mô tả cấu trúc của một hệ thống theo các lớp, thuộc tính của hệ thống đó và mối quan hệ giữa các lớp trong hệ thống. Biểu đồ lớp phổ biến nhất trong mô hình hóa hệ thống hướng đối tượng [11], được sử dụng để mô hình hóa khung thiết kế tĩnh của hệ thống và các lớp trong hệ thống. Các lớp là đại diện cho các "đối tượng" được xử lý trong hệ thống. Các lớp có thể quan hệ với nhau trong nhiều dạng thức:

- Liên kết (***associated***): được nối kết với nhau.
- Phụ thuộc (***dependent***): một lớp này phụ thuộc vào lớp khác.
- Chuyên biệt hóa (***specialized***): một lớp này là một kết quả chuyên biệt hóa của lớp khác.
- Đóng gói (***packaged***): hợp với nhau thành một đơn vị.

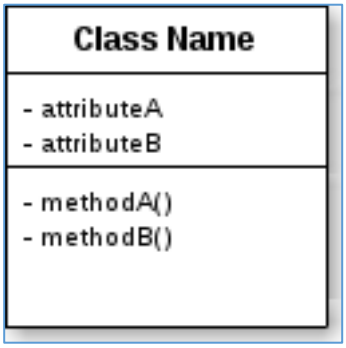
Tất cả các mối quan hệ đó đều được thể hiện trong biểu đồ lớp, đi kèm với cấu trúc bên trong của các lớp theo khái niệm thuộc tính (***attribute***) và phương thức (***operation***). Biểu đồ được coi là biểu đồ tĩnh theo phương diện cấu trúc được miêu tả ở đây có hiệu lực tại bất kỳ thời điểm nào trong toàn bộ vòng đời hệ thống.

Một hệ thống thường sẽ có một loạt các biểu đồ lớp – không phải bao giờ tất cả các biểu đồ lớp này cũng được nhập vào một biểu đồ lớp tổng thể duy nhất – và một lớp có thể tham gia vào nhiều biểu đồ lớp.

1.1.5.2. Các thành phần

Tùy thuộc vào ngữ cảnh, các lớp trong biểu đồ lớp có thể đại diện cho các đối tượng chính, các tương tác trong ứng dụng hoặc các lớp được lập trình. Luận văn áp dụng mô tả các lớp trong lập trình hướng đối tượng. Biểu đồ lớp tiêu chuẩn bao gồm ba thành phần chính sau đây:

- ***Phần trên***: Chứa tên của lớp. Phần này luôn là bắt buộc.
- ***Phần giữa***: Chứa các thuộc tính của lớp. Sử dụng phần này để mô tả các phẩm chất của lớp. Điều này chỉ được yêu cầu khi mô tả một thể hiện cụ thể của một lớp.
- ***Phần dưới cùng***: Bao gồm các thao tác (phương thức) của lớp. Được hiển thị ở định dạng danh sách, mỗi thao tác chiếm một dòng riêng. Các hoạt động mô tả cách một lớp tương tác với dữ liệu.

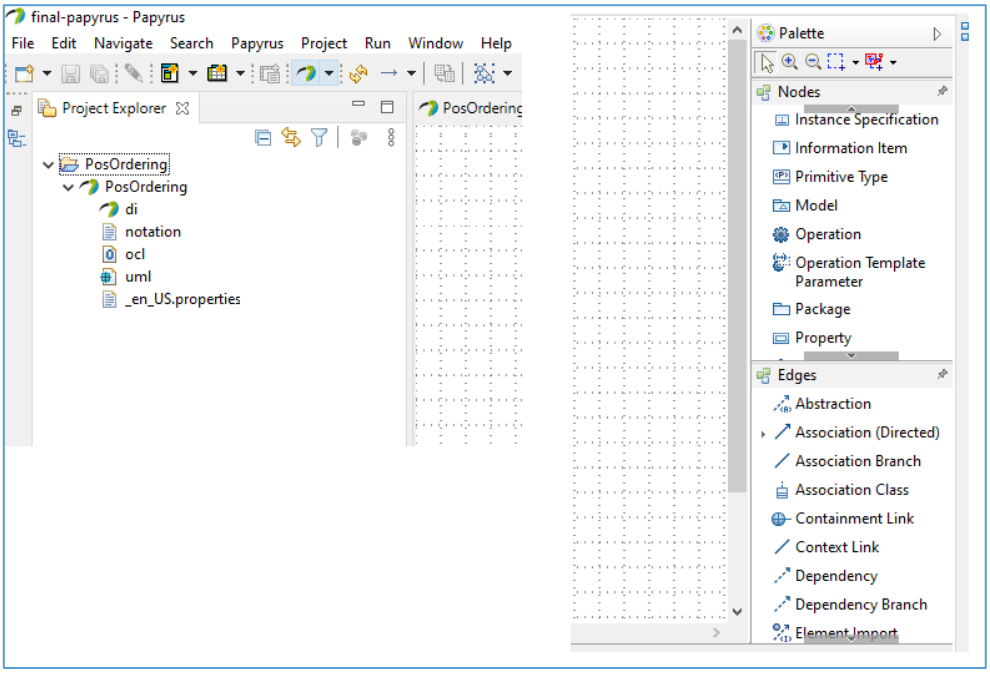


Hình 1. 7. Các thành phần của một lớp trong biểu đồ lớp.

Cũng giống với phạm vi truy cập các thuộc tính, phương thức…tất cả các lớp đều có các phạm vi truy cập khác nhau. Các cấp độ truy cập với các ký hiệu tương ứng gồm: Public (+), Private (-), Protected (#), Packgage (~), Derived (/), Static (gạch chân).

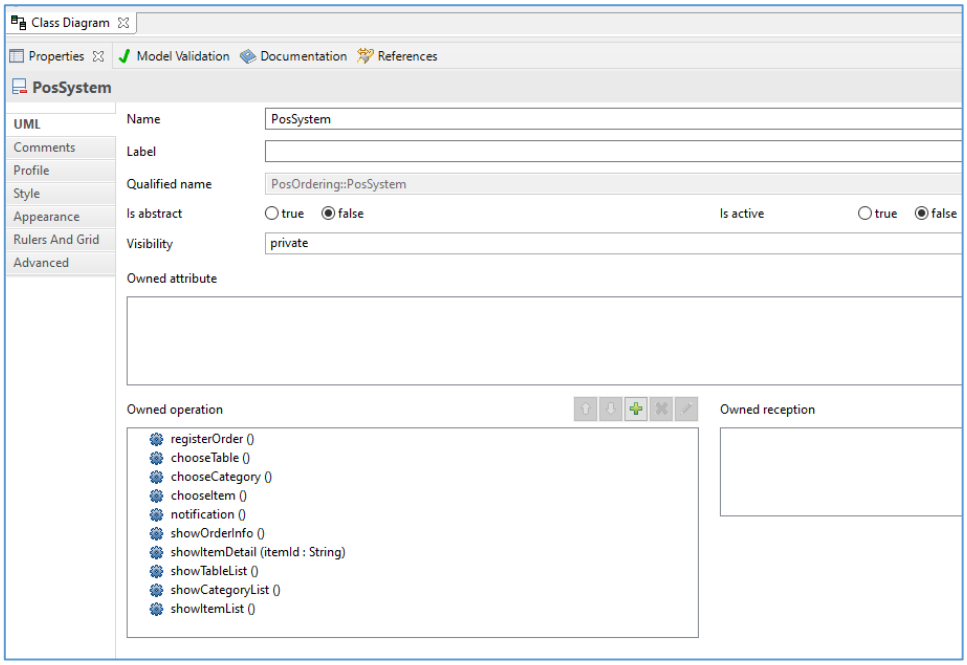
1.1.6. Công cụ

Papyrus là một công cụ bao gồm một số trình chỉnh sửa, chủ yếu là trình chỉnh sửa đồ họa nhưng cũng được hoàn thiện với các trình chỉnh sửa khác như trình soạn thảo dựa trên văn bản và dựa trên cây thư mục. Tất cả các trình soạn thảo này cho phép xem đồng thời nhiều sơ đồ của một mô hình UML nhất định. ***Papyrus*** có khả năng tùy biến cao và cho phép thêm các kiểu sơ đồ mới được phát triển bằng bất kỳ công nghệ nào tương thích với Eclipse (GEF, GMF, EMF Tree Editors, ...) [12]. Điều này đạt được thông qua cơ chế plug-in sơ đồ.



Hình 1. 8. Công cụ papyrus.

Với việc thiết kế biểu đồ lớp bằng Papyrus, chúng ta có thể thấy rõ các thành phần của một lớp.



Hình 1. 9. Chế độ view các thuộc tính của Papyrus.

1.2. Chuyển đổi mô hình

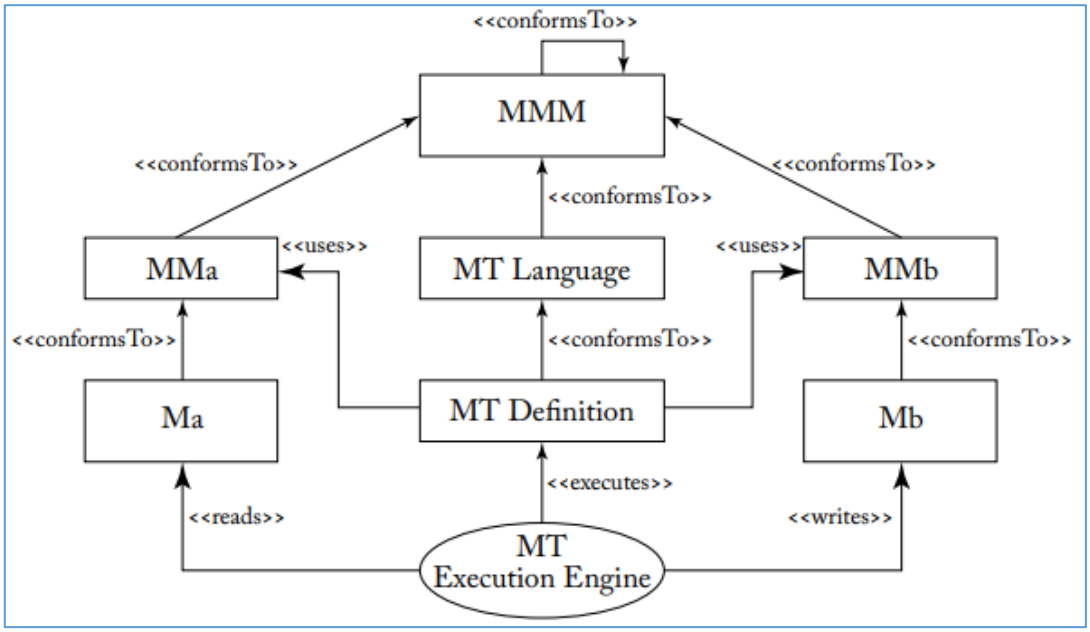
Phần tiếp theo đây sẽ là phần quan trọng nhất mà luận văn muốn đề cập tới, đó là chuyển đổi mô hình (***transformation***). Bên cạnh các mô hình, các phép chuyển đổi mô hình đại diện cho thành phần quan trọng khác của MDSE và cho phép xác định ánh xạ giữa các

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

MDSE cung cấp các ngôn ngữ thích hợp để xác định các phép chuyển đổi mô hình nhằm cung cấp cho các nhà thiết kế các giải pháp tối ưu hóa để chỉ định các quy tắc chuyển đổi. Các ngôn ngữ này có thể được sử dụng để xác định các phép chuyển đổi mô hình về các mẫu chuyển đổi thường được áp dụng cho các mô hình theo một số quy tắc đối sánh được kiểm tra trên các phần tử mô hình. Các quy tắc chuyển đổi như vậy có thể được định nghĩa theo các cách tiếp cận khác nhau: việc chuyển đổi có thể được viết thủ công từ đầu bởi nhà phát triển hoặc có thể được định nghĩa như một đặc tả tinh chỉnh của một quy tắc hiện có. Ngoài ra, bản thân các phép chuyển đổi có thể được tạo ra tự động theo một số quy tắc ánh xạ mức cao hơn giữa các mô hình. Kỹ thuật này dựa trên hai giai đoạn [8]:

- **Giai đoạn 1**: Xác định ánh xạ giữa các phần tử của một mô hình với các phần tử của một mô hình khác (ánh xạ mô hình hoặc dệt mô hình).
- **Giai đoạn 2**: Tự động hóa việc tạo ra các quy tắc chuyển đổi (hay luật chuyển đổi) thực tế thông qua một hệ thống với đầu vào là hai định nghĩa mô hình, ánh xạ giữa chúng và tạo ra các phép biến đổi.

Chính điều này cho phép các nhà phát triển tập trung vào các khía cạnh khái niệm của mối quan hệ giữa các mô hình và sau đó ủy thác việc định nghĩa các quy tắc chuyển đổi (có thể được thực hiện trong các không gian kỹ thuật khác nhau). Một khía cạnh thú vị khác liên quan đến tầm nhìn "mọi thứ đều là mô hình" là thực tế bản thân các phép biến đổi có thể được xem như là các mô hình và được quản lý như vậy, bao gồm cả meta-model. Phần dưới của Hình 1.10 cho thấy hai mô hình (Ma và Mb), và một phép chuyển đổi Mt biến Ma thành Mb. Trong cấp độ trên, các meta-model tương ứng được xác định (MMa, MMb và MMt), mà ba mô hình (Ma, Mb và Mt) tương ứng tuân theo. Đổi lại, tất cả chúng đều tuân theo cùng một Meta-metamodel – MMM.



Hình 1. 10. Vai trò và định nghĩa của phép biến hình giữa các mô hình.

Có hai dạng chuyển đổi mô hình chính: chuyển đổi mô hình sang mô hình và chuyển đổi mô hình sang văn bản, phần tiếp theo sẽ trình bày cụ thể về hai dạng chuyển đổi này.

1.2.1. Chuyển đổi mô hình sang mô hình

Mô hình không phải là thực thể cô lập cũng không phải là thực thể tĩnh. Là một phần của quy trình MDE, các mô hình được hợp nhất (nghĩa là để đồng nhất các phiên bản khác nhau của một hệ thống), được căn chỉnh (nghĩa là để tạo ra một biểu diễn toàn cầu của hệ thống từ các góc nhìn khác nhau đến lý do về tính nhất quán), được cấu trúc lại (nghĩa là để cải thiện cấu trúc bên trong của chúng mà không thay đổi hành vi), được tinh chỉnh (nghĩa là để chi tiết hóa các mô hình cấp cao) và được dịch (sang các ngôn ngữ / cách biểu diễn khác). Tất cả các hoạt động này trên các mô hình được thực hiện dưới dạng chuyển đổi mô hình [13], hoặc dưới dạng biến đổi mô hình sang mô hình (Model To Model – M2M) hoặc mô hình sang văn bản (Model To Text – M2T) (sẽ được trình bày trong mục tiếp theo). Trước đây, các tham số đầu vào và đầu ra của phép biến đổi là các mô hình, trong khi ở phần sau, đầu ra là một chuỗi văn bản. Phần này sẽ tập trung nghiên tìm hiểu và đề cập tới chuyển đổi mô hình sang mô hình.

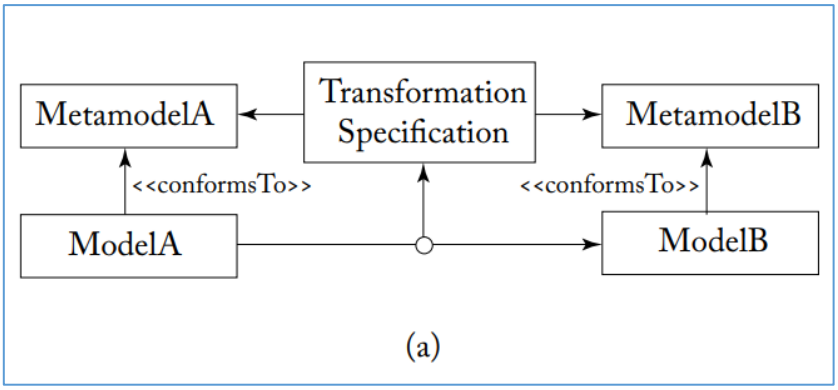
1.2.1.1. Chuyển đổi mô hình và sự phân loại

Kể từ khi ra đời các ngôn ngữ lập trình cấp cao đầu tiên được biên dịch sang trình hợp dịch, phép biến đổi là một trong những kỹ thuật quan trọng trong kỹ thuật phần mềm. Không có gì đáng ngạc nhiên, các phép biến đổi mô hình cũng rất quan trọng trong MDE và có nhiều phương thức khác nhau để giải quyết các nhiệm vụ khác nhau [7] [15]. Theo nghĩa chung, phép biến đổi M2M là một chương trình lấy một hoặc nhiều mô hình làm vào (input)

để tạo ra một hoặc nhiều mô hình đầu ra (output). Trong hầu hết các trường hợp, các phép biến đổi một – một, có một mô hình đầu vào và một mô hình đầu ra, như vậy là đủ. Ví dụ, hãy xem xét việc chuyển đổi một sơ đồ lớp thành một mô hình quan hệ. Tuy nhiên, cũng có những trường hợp yêu cầu các phép biến đổi một – nhiều, nhiều – một hoặc thậm chí nhiều – nhiều, ví dụ, một kịch bản hợp nhất mô hình trong đó mục tiêu là hợp nhất một số sơ đồ lớp thành một dạng biểu đồ tích hợp.

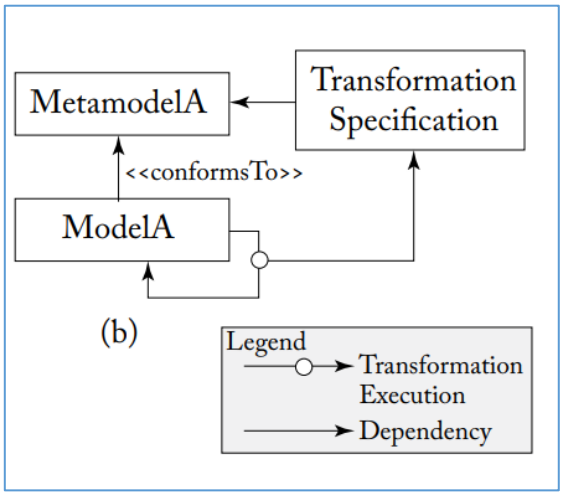
Bên cạnh việc phân loại các phép biến đổi mô hình dựa trên số lượng mô hình đầu vào và đầu ra, một khía cạnh khác là liệu sự chuyển đổi có giữa các mô hình từ hai ngôn ngữ khác nhau, được gọi là phép biến đổi ngoại sinh hay trong các mô hình được viết bằng cùng một ngôn ngữ mà sau đó được gọi là phép biến đổi nội sinh. Ví dụ: mô hình UML, được chuyển đổi thành mô hình nền tảng cụ thể, ví dụ: mô hình Java – Java model. Một ví dụ nổi tiếng cho các phép biến đổi nội sinh là tái cấu trúc mô hình. Tương tự, một mô hình có thể được cải tiến chất lượng, và việc đó có thể đạt được bằng cách tái cấu trúc các mô hình bằng cách sử dụng một phép chuyển đổi. Hơn nữa, các phép biến đổi mô hình ngoại sinh không chỉ có thể sử dụng được cho các phép biến đổi dọc như kịch bản UML sang Java đã nói ở trên, trong đó mức độ trừu tượng của các mô hình đầu vào và đầu ra là khác nhau. Một cách sử dụng khác là cho các phép biến đổi ngang trong đó các mô hình đầu vào và đầu ra ít nhiều vẫn ở cùng một mức trừu tượng. Ví dụ: các phép biến đổi ngoại sinh theo chiều ngang đang được sử dụng để thực hiện trao đổi mô hình giữa các công cụ mô hình hóa khác nhau, ví dụ: dịch sơ đồ lớp UML sang sơ đồ ER (Entity–relationship model – Mô hình mối quan hệ và thực thể) [8].

Trong thập kỷ qua, hai mô hình thực thi trung tâm cho các phép biến đổi mô hình đã xuất hiện và được so sánh trong hình dưới đây. Đầu tiên, có các phép biến đổi ngoài vị trí để tạo mô hình đầu ra từ đầu (xem Hình 1.11). Các phép biến đổi như vậy đặc biệt phù hợp với các phép biến đổi ngoại sinh.



Hình 1. 11. Các loại chuyển đổi mô hình sang mô hình ngoại sinh.

Thứ hai, có các phép biến đổi tại chỗ để viết lại một mô hình bằng cách tạo, xóa và cập nhật các phần tử trong mô hình đầu vào (xem Hình 1.12). Tất nhiên, mô hình này hoàn toàn phù hợp với các biến đổi nội sinh như tái cấu trúc.



Hình 1. 12. Các loại chuyển đổi mô hình sang mô hình nội sinh.

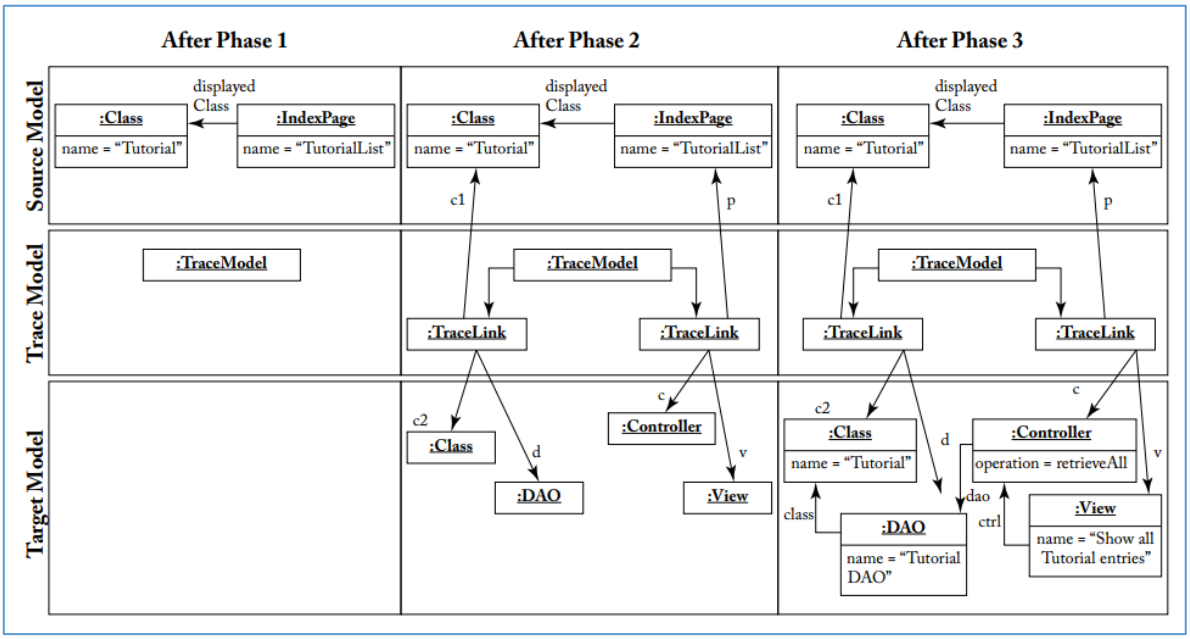
Trong các phần tiếp theo, luận văn sẽ trình bày cách xác định các phép biến đổi ngoại sinh giống như phép biến đổi bên ngoài sử dụng ATL và các phép biến đổi nội sinh giống như các phép biến đổi tại chỗ sử dụng ngôn ngữ biến đổi đồ thị.

1.2.1.2. Ngoại sinh và sự chuyển đổi bên ngoài

Nhiều cách tiếp cận chuyển đổi mô hình khác nhau đã được đề xuất trong thập kỷ qua, chủ yếu dựa trên sự kết hợp của các khái niệm khai báo và mệnh lệnh, chẳng hạn như ATL [3], ETL [6] và RubyTL [2], hoặc trên các phép biến đổi đồ thị, chẳng hạn như AGG [14] và Fujaba [10]. Tuy vậy ngôn ngữ chuyển đổi ATL (ATLAS Transformation Language) đã được chọn làm ngôn ngữ chuyển đổi minh họa để phát triển các phép biến đổi ngoại sinh, bởi vì nó là một trong những ngôn ngữ chuyển đổi được sử dụng rộng rãi nhất [8], cả trong học thuật và công nghiệp, và có công cụ hoàn thiện, có sẵn. ATL là một ngôn ngữ dựa trên quy tắc được xây dựng chủ yếu dựa trên ngôn ngữ ràng buộc đối tượng (OCL – Object Constraint Language), nhưng cung cấp các tính năng ngôn ngữ dành riêng cho các phép biến đổi mô hình bị thiếu trong OCL, ví dụ như việc tạo các phần tử mô hình.

ATL được thiết kế như một ngôn ngữ chuyển đổi mô hình kết hợp chứa hỗn hợp các cấu trúc khai báo và mệnh lệnh. Các phép biến đổi ATL là đơn hướng, có nghĩa là nếu cần chuyển đổi từ ngôn ngữ A sang ngôn ngữ B và ngược lại thì phải phát triển hai phép biến đổi. Các phép biến đổi ATL đang hoạt động trên các mô hình nguồn chỉ đọc và tạo ra các mô hình đích chỉ ghi. Một lưu ý rằng đối với các phép biến đổi bên ngoài, thay vì mô hình đầu vào, mô hình nguồn (source model) là thuật ngữ thường được sử dụng và thay vì mô hình đầu ra, mô hình mục tiêu (target model) là thuật ngữ thường được sử dụng.

Quay trở lại chế độ hoạt động của phép biến đổi ATL, trong quá trình thực hiện phép biến đổi, các mô hình nguồn được truy vấn nhưng không cho phép thay đổi chúng. Ngược lại, các phần tử mô hình đích được tạo, nhưng không được truy vấn trực tiếp trong quá trình chuyển đổi. ATL là một ngôn ngữ mạnh mẽ cho phép xác định phép chuyển đổi trong một cú pháp ngắn gọn. Để giải thích các chi tiết ẩn đằng sau cú pháp ATL, bây giờ chúng ta hãy xem xét quá trình thực thi chuyển đổi thực tế mà máy ảo ATL đạt được. Việc thực hiện một phép biến đổi ATL được cấu trúc thành ba giai đoạn tuần tự được giải thích ở phần sau và được minh họa bằng cách sử dụng một đoạn trích nhỏ của một lần chạy chuyển đổi cụ thể (xem hình 1.13) của phép biến đổi ATL.



Hình 1. 13. Các giai đoạn thực thi chuyển đổi ATL.

Giai đoạn 1: Khởi tạo mô-đun. Trong giai đoạn đầu, trong số những thứ khác, mô hình theo dõi để lưu trữ các liên kết theo dõi giữa các phần tử nguồn và đích được khởi tạo. Trong giai đoạn 2 tiếp theo, mỗi lần thực thi một quy tắc đã so khớp sẽ được lưu trữ trong mô hình theo dõi bằng cách tạo ra một liên kết theo dõi trỏ đến các phần tử đầu vào phù hợp và đến các phần tử đầu ra đã tạo. Như chúng ta sẽ thấy ở phần sau, mô hình vết (trace model) là một khái niệm quan trọng đối với các phép biến đổi ngoại sinh: để dừng thực hiện một phép biến đổi và để gán các đặc trưng của các phần tử đích dựa trên các giá trị của các phần tử nguồn.

Giai đoạn 2: Phân bổ các yếu tố mục tiêu. Trong giai đoạn thứ hai, công cụ biến đổi ATL tìm kiếm các kết quả phù hợp cho mẫu nguồn của các quy tắc đã so khớp bằng cách tìm các cấu hình hợp lệ của các phần tử mô hình nguồn. Khi điều kiện khớp của một quy tắc đã so khớp (tất cả các phần tử mẫu đầu vào đều bị ràng buộc và điều kiện bộ lọc hợp lệ) được đáp ứng bởi cấu hình các phần tử mô hình nguồn, công cụ chuyển đổi ATL sẽ phân

bổ tập hợp các phần tử mô hình đích tương ứng dựa trên mẫu đích đã khai báo các yếu tố. Xin lưu ý rằng chỉ có các phần tử được tạo, nhưng các tính năng của chúng chưa được thiết lập. Hơn nữa, đối với mỗi trận đấu, có một liên kết theo dõi được tạo ra để kết nối các phần tử nguồn phù hợp và các phần tử đích đã tạo.

Giai đoạn 3: Khởi tạo phần tử mục tiêu. Trong giai đoạn thứ ba, mỗi phần tử mô hình mục tiêu được phân bổ được khởi tạo bằng cách thực thi các ràng buộc được xác định cho phần tử mô hình đích. Trong các ràng buộc, các lệnh gọi của hoạt động giải quyết khá phổ biến. Thao tác này cho phép tham chiếu đến bất kỳ phần tử nào của mô hình đích đã được tạo trong giai đoạn thực thi thứ hai cho một phần tử mô hình nguồn nhất định. Thao tác này được đặc tả theo dạng sau:

ResolutionTemp(srcObj: OclAny, targetPatternElementVar: String) [8].

Tham số đầu tiên đại diện cho phần tử mô hình nguồn mà các phần tử mô hình đích phải được giải quyết. Tham số thứ hai là tên biến của phần tử mẫu đích cần được truy xuất. Tham số thứ hai là bắt buộc, vì có thể tạo một số phần tử đích cho một phần tử nguồn bằng cách sử dụng nhiều phần tử mẫu đích trong một quy tắc như trường hợp trong ví dụ của chúng tôi. Do đó, khi một phần tử mô hình nguồn phải được giải quyết, biến của phần tử mẫu mục tiêu đã tạo ra phần tử mục tiêu được yêu cầu phải được đưa ra.

1.2.1.3. Nội sinh và sự chuyển đổi nội tại

Theo những gì đã được thảo luận, các mô hình đã được sửa đổi thủ công bằng cách áp dụng các hành động trong công cụ chỉnh sửa mô hình như thêm và xóa các phần tử mô hình hoặc cập nhật giá trị của chúng. Tuy nhiên, có rất nhiều tình huống mà việc sửa đổi mô hình nên hoặc phải được tự động hóa. Nhớ lại rằng các phép biến đổi ngoại sinh đòi hỏi chúng ta phải xây dựng hoàn toàn mô hình đầu ra từ đầu. Nếu các phép biến đổi ngoại sinh được sử dụng để giới thiệu các sửa đổi cho một mô hình, thì điều này sẽ yêu cầu sao chép mô hình nguồn hoàn chỉnh sang mô hình đích, ngoại trừ các phần tử đó bị xóa hoặc sửa đổi. Do đó, các chế độ thực thi thay thế cho phép biến đổi mô hình có sẵn phù hợp hơn cho kịch bản chuyển đổi này, chỉ các quy tắc chuyển đổi quan tâm đến phần động, tức là các thay đổi (nhưng không có quy tắc nào để chỉ sao chép các phần tử chưa được sửa đổi) là bắt buộc.

Phép biến đổi đồ thị (Graph transformation) [4] là một cách tiếp cận rất đơn giản để thực hiện các phép biến đổi mô hình tại chỗ. Do đó, chúng đã được chọn để chứng minh sự phát triển và sử dụng các phép biến đổi mô hình nội tại. Biến đổi đồ thị là một kỹ thuật dựa trên quy tắc, khai báo để thể hiện các phép biến đổi mô hình nội tại dựa trên thực tế là các mô hình và siêu mô hình có thể được biểu diễn dưới dạng đồ thị (với các nút và cạnh được đánh máy, quy ước). Do đó, các mô hình có thể được thao tác bằng các kỹ thuật biến đổi

đồ thị. Các phép biến đổi đồ thị đặc biệt hữu ích để xác định các phép biến đổi tại chỗ để hỗ trợ, ví dụ: mô phỏng mô hình, tối ưu hóa, thực thi, tiến hóa và tái cấu trúc.

1.2.1.4. Chuỗi chuyển đổi mô hình

Các phép biến đổi có thể là các quá trình phức tạp cần được tự mô hình hóa và cấu trúc thành các bước khác nhau để tránh có một phép biến đổi đơn lẻ. Chuỗi chuyển đổi là kỹ thuật được lựa chọn để lập mô hình điều phối các phép biến đổi mô hình khác nhau. Chuỗi chuyển đổi được xác định bằng các ngôn ngữ điều phối cho phép chúng ta lập mô hình các bước tuần tự của phép biến đổi. Điều này có nghĩa là các mô hình đầu vào cho chuyển đổi đầu tiên được xác định, các mô hình đầu ra của chuyển đổi trở thành mô hình đầu vào cho chuyển đổi thứ hai tiếp theo, v.v. Các chuỗi biến đổi phức tạp hơn cũng có thể kết hợp các nhánh, vòng lặp có điều kiện và các cấu trúc điều khiển khác.

Chuỗi chuyển đổi không chỉ là phương tiện để tách phép biến đổi được phát triển thành một số mô-đun, mà còn cho phép xây dựng các phép biến đổi mô hình phức tạp từ các phép biến đổi đã được xác định. Hơn nữa, việc có các phép biến đổi nhỏ hơn tập trung vào các khía cạnh nhất định cũng có thể cho phép khả năng tái sử dụng cao hơn. Chuỗi chuyển đổi mô hình có thể được xác định bằng các ngôn ngữ xây dựng dạng văn bản nổi tiếng như ANT. Tuy nhiên, cũng có các ngôn ngữ chuyên dụng để mô hình hóa chuỗi chuyển đổi bằng cách sử dụng một tập hợp con của ngôn ngữ sơ đồ hoạt động UML.

Cho đến nay, chúng ta đã xem các phép biến đổi là các phép toán để thao tác các mô hình, nhưng trên thực tế, bản thân các phép biến đổi có thể được coi là các mô hình, vì chúng là các thể hiện của một siêu mô hình chuyển đổi, tức là, một phép biến đổi ATL có thể được biểu thị như một thể hiện của siêu mô hình ATL (xác định cú pháp trừu tượng của ngôn ngữ ATL), và do đó, có thể được xem như một mô hình.

Tính đồng nhất này cho phép chúng ta sử dụng lại các công cụ và phương pháp, tức là các công cụ tương tự có thể được sử dụng để tạo mô hình có thể được sử dụng để tạo ra các mô hình biến đổi và nó tạo ra một khuôn khổ về lý thuyết có thể được áp dụng đệ quy vì các phép biến hình có thể tự biến đổi [8]. Hơn nữa, điều quan trọng là phải tạo điều kiện thuận lợi cho việc thực hiện các phép biến hình. Cũng giống như một mô hình bình thường có thể được tạo ra, sửa đổi và tăng cường thông qua phép biến đổi, một mô hình biến đổi có thể được tạo ra, sửa đổi, v.v. bằng chuyển đổi bậc cao (Higher Order Transformation – HOT). Điều này có nghĩa là chúng ta có thể viết các phép biến đổi lấy đầu vào là phép biến đổi mô hình và / hoặc tạo ra một phép biến đổi mô hình làm đầu ra.

1.2.2. Chuyển đổi mô hình sang văn bản

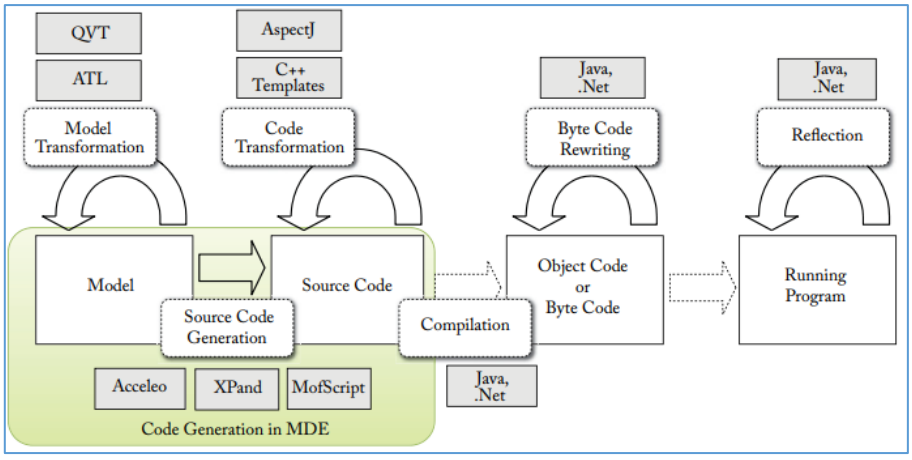
Một số khái niệm, ngôn ngữ và công cụ đã được đề xuất để tự động hóa việc lấy văn bản từ các mô hình bằng cách sử dụng các phép chuyển đổi mô hình sang văn bản (Model

to Text – M2T). Các phép biến đổi như vậy đã được sử dụng để tự động hóa một số nhiệm vụ kỹ thuật phần mềm như tạo tài liệu, danh sách tác vụ, v.v [8]. Tất nhiên, ứng dụng hàng đầu của phép biến đổi M2T là sinh mã nguồn. Trên thực tế, các phép biến đổi M2T chủ yếu quan tâm đến việc sinh mã để đạt được sự chuyển đổi từ mức mô hình sang mức mã nguồn của một ngôn ngữ lập trinh cụ thể… Lưu ý rằng không chỉ mã đại diện cho hệ thống cần phát triển có thể được bắt nguồn từ các mô hình, mà còn có các tạo tác khác liên quan đến mã như các test cases, tập lệnh triển khai, v.v. Các phép biến đổi M2T hiện là cầu nối cho các nền tảng thực thi và các công cụ phân tích. Trong chương này, luận văn sẽ bắt đầu với những kiến thức cơ bản về tạo mã dựa trên mô hình bằng cách sử dụng các phép biến đổi M2T, thảo luận về các cách tiếp cận khác nhau để thực hiện các phép biến đổi M2T và cuối cùng, báo cáo về các kỹ thuật để xác định độ phức tạp của việc tạo mã cũng như áp dụng vào bài toán.

1.2.2.1. Mô hình và định nghĩa mã nguồn

Với kiến trúc hướng mô hình MDE thì việc sinh mã nguồn coi mô hinh như một chủ thể, và tất nhiên mã nguồn là một đầu ra cần có. Trong đó mô hình là một hiện vật phải được chuyển đổi thành mã nguồn cụ thể hơn để được thực thi, và mã nguồn là thứ có thể được biên dịch và thực thi trực tiếp. Định nghĩa này rất quan trọng, vì nó nói rằng mô hình UML không được coi là mã và giao diện ngôn ngữ lập trình không được coi là mô hình. Đối với việc sinh mã nguồn nhiều bước, có nhiều chuyển đổi mô hình sang mô hình và một bước tạo mã nguồn cuối cùng. Mã được tạo sau đó được biên dịch và thực thi. Một mô hình được coi là tương đương với một đặc tả trong ngôn ngữ dành riêng cho miền văn bản (Domain-Specific Language – DSL). Do đó, ngữ pháp của DSL giống như meta-model của mô hình.

Hình 1.14 dưới đây cho thấy nơi tạo mã có thể áp dụng trong vòng đời phát triển ứng dụng, các đoạn sau đưa ra một số giải thích bổ sung.



Hình 1. 14. Vòng đời phát triển ứng dụng và áp dụng sinh mã nguồn [9].

1.2.2.2. Sinh mã nguồn tự động

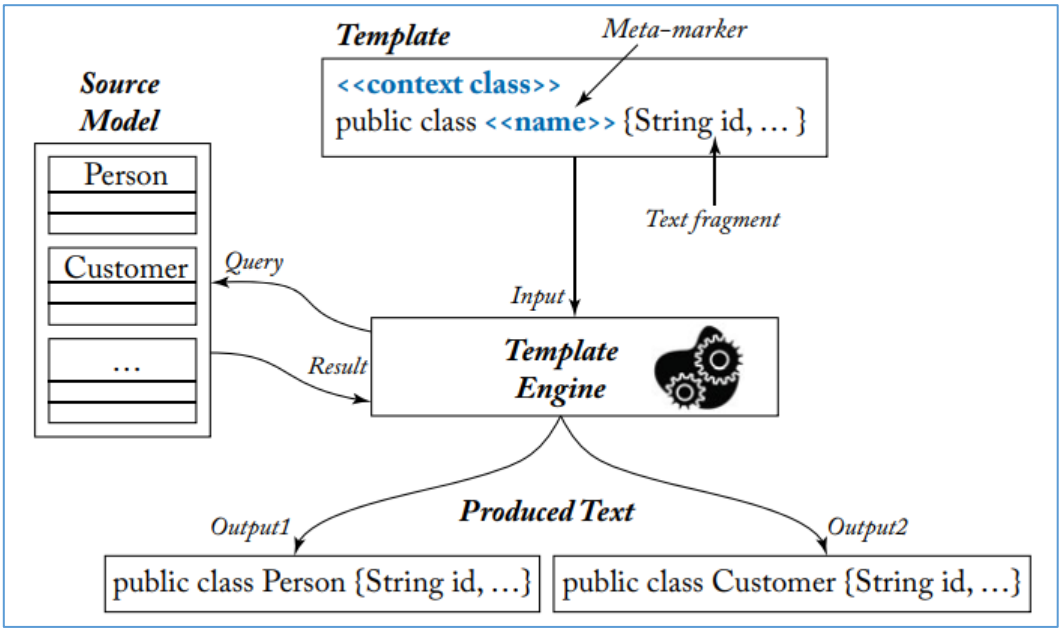
Tạo mã có một truyền thống lâu đời trong kỹ thuật phần mềm bắt nguồn từ những ngày đầu của các ngôn ngữ lập trình cấp cao, nơi các trình biên dịch đầu tiên phát triển. Sinh mã nguồn tự động là tạo tác quan trọng nhất của phép chuyển đổi mô hình sang văn bản, nhằm mục đích chính tạo mã thực thi. Điều này thường liên quan đến một số loại trừu tượng hóa hoặc cụ thể hóa của mô hình. Nguồn được tạo thường yêu cầu biên dịch trước khi nó có thể được thực thi. Tuy nhiên, đôi khi tạo mã byte hoặc mã máy trực tiếp.

Ngoài các kỹ thuật tạo mã được đề cập ở trên, có một số môi trường meta-model tích hợp có sẵn (chẳng hạn như GME [ISIS, web] hoặc MetaEdit + [Metacase, web]) [8]. Các công cụ này cung cấp môi trường chung, tích hợp để xây dựng, xác thực và quản lý các mô hình dựa trên meta-model tùy chỉnh. Chúng cũng có thể được sử dụng để tạo mã nguồn từ các mô hình này. Tuy nhiên, những công cụ này nằm ngoài phạm vi của luận văn. Điều này cũng đúng với các công cụ UML có thể tạo mã như Rose, XDE hoặc Together / J. Có sự khác biệt về mục tiêu tạo mã trong trình biên dịch [1] và trong MDE Mặc dù tất cả các công cụ này đều có thể tạo mã nguồn (một số thậm chí theo cách có thể tùy chỉnh) nhưng tất cả chúng đều sử dụng một số kỹ thuật mà khó có thể tùy chỉnh quy tắc chuyển đổi.

1.2.2.3. Những lợi ích của ngôn ngữ chuyển đổi mô hình sang văn bản M2T

Trong phần trước, bộ tạo mã dựa trên Java đã được trình bày để chỉ ra cách các tính năng cần thiết phải được triển khai trong GPL. Các ngôn ngữ chuyển đổi M2T nhằm mục đích cải thiện sự phát triển của trình tạo mã bằng cách giải quyết các nhược điểm đã nói ở trên của các trình tạo mã dựa trên GPL.

Mã tĩnh / động được tách biệt: các ngôn ngữ chuyển đổi M2T tách mã tĩnh và mã động bằng cách sử dụng phương pháp dựa trên khuôn mẫu (template) để phát triển các phép biến đổi M2T. Mẫu có thể được coi là một loại kế hoạch chi tiết xác định các phần tử văn bản tĩnh được chia sẻ bởi tất cả các phần tạo tác cũng như các phần động phải chứa đầy thông tin cụ thể cho từng trường hợp cụ thể. Do đó, một mẫu chứa các đoạn văn bản đơn giản cho phần tĩnh và cái gọi là điểm đánh dấu (meta-makers) cho phần động. Meta-markers là bộ giữ chỗ và phải được giải thích bởi một công cụ mẫu xử lý các mẫu và truy vấn các nguồn dữ liệu bổ sung để tạo ra các phần động. Rõ ràng, trong các phép biến đổi M2T, các nguồn dữ liệu bổ sung là các mô hình. Hình 1.15 dưới đây tóm tắt ý tưởng chính của việc tạo văn bản dựa trên khuôn mẫu.



Hình 1. 15. Mẫu, công cụ mẫu và mô hình nguồn để tạo văn bản.

Cấu trúc đầu ra rõ ràng: sử dụng các mẫu cho phép chúng ta biểu diễn rõ ràng cấu trúc của văn bản đầu ra trong khuôn mẫu. Điều này đạt được bằng cách nhúng mã để tạo các phần động của đầu ra trong văn bản đại diện cho phần tĩnh — chính xác là nghịch đảo của trình tạo mã dựa trên Java trước đó. Ở đây, cơ chế tương tự cũng được áp dụng như đối với Java Server Pages (JSP), cho phép trình bày rõ ràng cấu trúc của các trang HTML và nhúng mã Java – ngược lại với Java Servlet. Việc có cấu trúc của đầu ra được thể hiện rõ ràng trong các mẫu dẫn đến các đặc tả tạo mã dễ đọc và dễ hiểu hơn là chỉ sử dụng các biến String để lưu trữ văn bản đầu ra. Các mẫu cũng dễ dàng phát triển các bộ tạo mã. Ví dụ: một mẫu có thể được phát triển bằng cách chỉ thêm mã mẫu vào một mẫu và nhà phát triển thay thế các phần mã động bằng các điểm đánh dấu. Bằng cách này: (i) quá trình trừu tượng hóa từ một ví dụ mã cụ thể đến một đặc tả mẫu là đơn giản và (ii) mẫu có cấu trúc và định dạng tương tự như mã được tạo ra, cho phép theo dõi ảnh hưởng của các mẫu đối với mã.

Ngôn ngữ truy vấn khai báo: trong các điểm đánh dấu, chúng ta cần truy cập thông tin được lưu trữ trong các mô hình. OCL là sự lựa chọn để thực hiện công việc này trong hầu hết các ngôn ngữ chuyển đổi M2M. Do đó, các ngôn ngữ chuyển đổi M2T hiện tại cũng cho phép chúng ta sử dụng OCL (hoặc một phương ngữ của OCL) để chỉ định các dấu. Các ngôn ngữ mẫu khác không được thiết kế riêng cho các mô hình nhưng hỗ trợ bất kỳ loại nguồn nào đều sử dụng các ngôn ngữ lập trình tiêu chuẩn như Java để chỉ định các dấu.

Tái sử dụng: các ngôn ngữ chuyển đổi M2T hiện tại đi kèm với sự hỗ trợ của công cụ cho phép chúng ta đọc trực tiếp trong các mô hình và tuần tự hóa văn bản thành các tệp chỉ bằng cách xác định các tệp cấu hình. Do đó, không cần phải phát triển việc định nghĩa lại mô hình cũ kĩ lỗi thời và tuần tự hóa văn bản theo cách thủ công.

1.3. Tổng kết chương

Phát triển phần mềm hướng mô hình có thể hiểu đơn giản là một phương pháp phát triển dựa trên nền tảng mô hình hóa các khía cạnh của một hệ thống, sau đó áp dụng chuyển đổi mô hình thành các tạo tác phần mềm hữu dụng. Dưới góc nhìn MDSE, việc phát triển phần mềm xoay quanh mô hình hóa và phép chuyển đổi mô hình. Chuyển đổi mô hình gồm hai dạng chính: từ mô hình sang mô hình và từ mô hình sang văn bản. Chuyển đổi từ mô hình sang mô hình chính là đưa một mô hình về dạng mô hình khác, ví dụ như chuyển đổi một sơ đồ lớp thành một mô hình quan hệ, và các phép biến đổi có thể là 1 – 1, 1 – nhiều hoặc nhiều – nhiều. Dạng còn lại, chuyển đổi từ mô hình sang văn bản, có thể chuyển đổi từ mô hình sang một số tạo tác phần mềm như các test cases, tập lệnh triển khai… và khía cạnh quan trọng nhất là sinh mã nguồn tự động.

CHƯƠNG 2. TỔNG QUAN KỸ THUẬT SINH MÃ NGUỒN

Chương này sẽ giới thiệu tổng quan các kỹ thuật sinh mã nguồn chính, đặc điểm của từng kỹ thuật sinh mã nguồn, đặc biệt là kỹ thuật sinh mã nguồn bằng ngôn ngữ chuyển đổi, các ngôn ngữ chuyển đổi. Tiếp đó sẽ tập trung khai thác kỹ thuật sinh mã nguồn sử dụng ngôn ngữ chuyển đổi Acceleo, ví dụ đơn giản cho ngôn ngữ này và phần cuối sẽ tổng kết nội dung đã nêu trong chương.

2.1. Giới thiệu

Trong thực tế, mỗi hệ thống là một miền ứng dụng và có thể có nhiều hệ thống với những chức năng tương tự hoặc gần tương tự nhau, thậm chí có các tác vụ lặp lại trong một hệ thống. Vấn đề đặt ra đó là thật khó để triển khai mã nguồn từ hệ thống này sang hệ thống khác mà cần phải lập trình lại hoàn toàn, có rất nhiều lý do cho việc này: vấn đề bảo mật, bản quyền… Hơn nữa, dù trong một hệ thống có những tính năng tương tự nhau, có thể sử dụng lại một phần mã nguồn, tuy nhiên việc viết lại các đoạn mã nguồn có thể tăng khả năng xuất hiện lỗi [16]. Khả năng xuất hiện lỗi không nằm trong mã nguồn mà nằm trong phương pháp và thao tác sử dụng lại, cần thay đổi môi trường, biến, file mã nguồn để phù hợp với những điều khiện khác nhau. Hiện tại có thể xử lý vấn đề này bằng cách tìm kiếm mã nguồn mở hoặc mẫu phát triển có sẵn, tuy nhiên phương pháp này khá rủi ro và nếu tốt thì hầu hết phải trả phí. Chính vì vậy cần phải có một giải pháp tối ưu nhằm tự động hóa các tác vụ lặp lại hoặc các chức năng giống nhau, nhằm giảm thiểu chi phí về cả vật chất và công sức phát triển.

Như đã trình bày trong chương trước, sinh mã nguồn chính là khía cảnh nổi bật và được quan tâm nhất của chuyển đổi mô hình sang văn bản. Khi làm việc với các mô hình, việc tự động hóa các tác vụ lặp lại thường có thể đạt được bằng cách sinh mã tự động từ các mô hình. Điều này giúp tạo mã nguồn chung thay vì phải viết lại, hoặc thậm chí tách phần logic ra khỏi phần nền tảng, giúp ứng dụng cho thể dễ dàng triển khai trên các nền tảng khác nhau. Chương này xoay quanh hai phương pháp sinh mã nguồn tự động gồm phương pháp: sinh mã nguồn bằng ngôn ngữ lập trình và ngôn ngữ chuyển, phần cuối là tổng kết chương.

2.2. Sinh mã nguồn bằng ngôn ngữ lập trình

Nhìn chung, việc triển khai bộ tạo mã có thể dựa trên nguyên tắc MDE hoặc chỉ sử dụng phương pháp lập trình truyền thống. Theo cách tiếp cận thứ hai, trình tạo mã có thể được triển khai dưới dạng chương trình sử dụng API mô hình được tạo tự động từ meta-model để xử lý các mô hình đầu vào và in ra các câu lệnh mã ra tệp bằng cách sử dụng bộ ghi file tiêu chuẩn cung cấp bởi API của ngôn ngữ lập trình được sử dụng để triển khai bộ tạo mã.

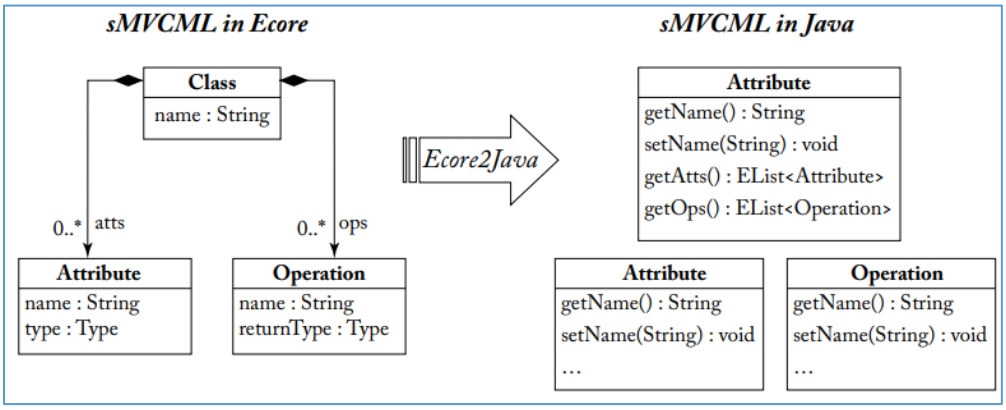
API model được hiện thực hóa trong EMF bằng cách sử dụng chính một phép chuyển đổi M2T đọc metamodel dựa trên Ecore và tạo ra một lớp Java cho mỗi lớp Ecore [8]. Trong hình dưới đây có một đoạn trích từ meta-model sMVCML (simple MVC modeling language – ngôn ngữ mô hình hóa cấu trúc MVC thuần) được hiển thị ở phía bên trái và các lớp Java tương ứng ở phía bên phải. Ánh xạ từ Ecore sang Java hầu như rất đơn giản — đây là một trong những mục tiêu thiết kế của Ecore. Đối với mỗi tính năng của metaclasses, các phương thức getter và setter tương ứng được tạo ở phía Java. Điều này có nghĩa là, một mô hình có thể được đọc, sửa đổi và tạo hoàn toàn từ đầu bằng cách sử dụng mã Java được tạo như vậy thay vì sử dụng các bộ chỉnh sửa mô hình.

Trước khi đi sâu vào các ngôn ngữ chuyển đổi M2T cụ thể trong phần tiếp theo, luận văn sẽ trình bày cách GPL có thể được sử dụng để phát triển bộ tạo mã. Ý tưởng cơ bản của phương pháp sinh mã nguồn sử dụng ngôn ngữ lập trình này là:

- ***Bước 1***: Sử dụng API model được tạo ra từ meta-model nhằm xử lý các mô hình.
- ***Bước 2***: Sử dụng trình tạo mã nhằm thực hiện việc chuyển đổi.

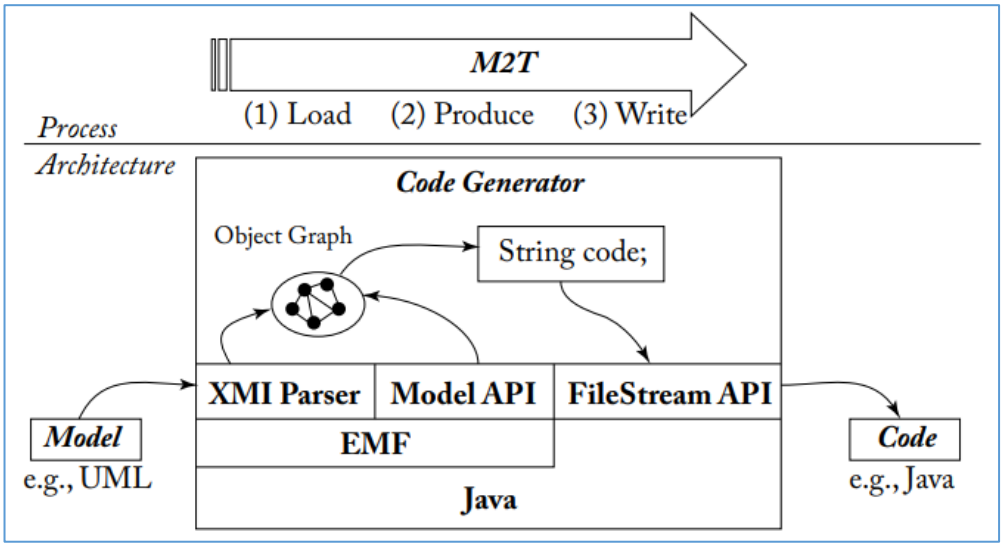
Hình dưới đây minh họa các giai đoạn phải được hỗ trợ bởi bộ tạo mã. Bao gồm:

- Load model (tải/nạp mô hình): Mô hình phải được giải mã hóa từ biểu diễn XMI sang biểu đồ đối tượng được tải/nạp trong bộ nhớ. Đối với điều này, các framework API siêu mô hình hóa hiện tại cung cấp các hoạt động cụ thể.



Hình 2. 1. API model được tạo từ một đoạn trích của sMVCML metamodel.

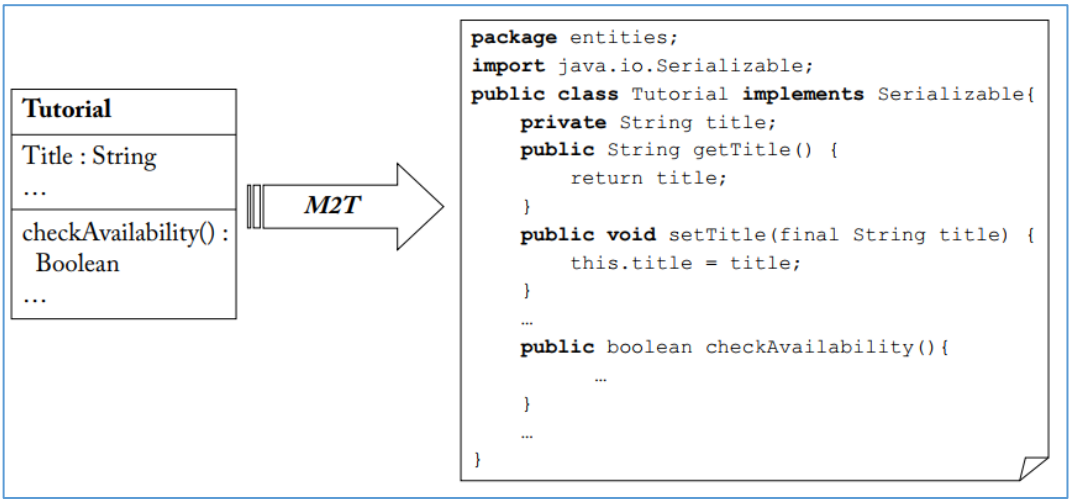
- Produce code (sản xuất mã): Thu thập thông tin mô hình cần thiết để tạo mã bằng cách sử dụng API mô hình để xử lý mô hình. Thông thường, biểu đồ đối tượng được duyệt bắt đầu từ phần tử gốc của một mô hình xuống các phần tử lá của nó.



Hình 2. 2. Tạo mã thông qua ngôn ngữ lập trình – mã Java tạo mã Java.

- Write code (viết mã): Ví dụ, mã được lưu trong các biến String và cuối cùng, được lưu vào các tệp qua các luồng.

Sau đây luận văn sẽ trình bày ví dụ cụ thể về việc tạo mã. Hình 15 cho thấy bản dịch mẫu của lớp sMVCML (xin lưu ý rằng đối với sMVCML, ký hiệu sơ đồ lớp UML được sử dụng lại) của ví dụ đang chạy (xem phía bên trái) sang mã Java tương ứng (xem phía bên phải). Như có thể thấy trong hình này, bản dịch đơn giản, nhưng đủ để chỉ ra các khái niệm chính cần thiết để triển khai trình tạo mã. Lớp sMVCML được dịch thành một lớp Java thực hiện giao diện Serializable, thuộc tính sMCVML thành một biến riêng với các phương thức getter / setter để truy cập / sửa đổi biến và thao tác sMVCML thành một phương thức Java. Tuy nhiên, liên quan đến cái sau, chỉ chữ ký phương thức có thể được lấy từ mô hình sMVCML. Việc triển khai các phương pháp được hoãn lại ở cấp mã. Do đó, một cách tiếp cận tạo mã một phần được sử dụng. Một yêu cầu chính là quan tâm đến mã được thêm thủ công trong mã được tạo tự động để cho phép quá trình phát triển theo hướng mô hình lặp đi lặp lại.



Hình 2. 3. Đoạn trích mô hình sMVCML và mã nguồn tương ứng.

Đối với giai đoạn đầu, cụ thể là tải/nạp các mô hình, API EMF, cung cấp các lớp để tải tài nguyên (trong trường hợp của này) các mô hình sMVCML vào bộ nhớ được sử dụng. Trong giai đoạn hai, tất cả các phần tử của mô hình được truy vấn từ mô hình đầu vào, và sau đó, được lặp lại. Nếu phần tử mô hình là một lớp (kiểm tra kiểu instanceof), thì một biến String được đạt tên sẽ được khởi tạo và tiếp tục được hoàn thiện bằng các câu lệnh Java dưới dạng giá trị String. Trong giai đoạn ba, một luồng được xác định cho một tệp Java với tên của lớp được xử lý và giá trị của biến mã được duy trì trong tệp này. Tất nhiên, các bộ tạo mã dựa trên GPL phức tạp hơn có thể được phát triển bằng cách sử dụng các mẫu thiết kế (pattern) như Visitor pattern, nhưng những hạn chế nêu trong phần sau cũng áp dụng cho các giải pháp đó.

```
// PHASE 1: load the sMVCML model using the EMF API
ResourceSet resourceSet = new ResourceSetImpl();
Resource resource =
resourceSet.getResource(URI.create("model.smvcml"));

// PHASE 2: collect the code statements in variable
// traverse the complete model using the EMF API
TreeIterator allElementsIter = resource.getAllContents();
while (allElementsIter.hasNext()) {
    Object object = allElementsIter.next();
    if (!object instanceof Class) continue;
    Class cl = (Class) object;

    // String variable for collecting code statements
    String code = "package entities;\n\n",
    code += "import java.io.Serializable;\n\n";
    code += "public class " + cl.getName() + "implements Serializable{\n";

    // generate Attributes:
    Iterator<Attribute> attIter = cl.getAtts(); ... code += ...

    // generate Methods:
    Iterator<Operation> opIter = cl.getOps(); ... code += ...

    code += "}";

    // PHASE 3: print code to file
    try {
        FileOutputStream fos = new FileOutputStream(cl.getName() +".java");
        fos.write(code.getBytes());
        fos.close();
    } catch (Exception e) {.}
}
```

Hình 2. 4. Sinh mã nguồn dựa trên ngôn ngữ lập trình (Java).

Phương pháp sinh mã nguồn bằng ngôn ngữ lập trình có những ưu điểm sau. Thứ nhất: không cần thêm kỹ năng lập trình, hỉ cần biết ngôn ngữ lập trình được chọn để phát triển trình tạo và quen thuộc với API model là đủ. Thứ hai: không cần công cụ bổ sung nào, cho cả thời gian thiết kế và thời gian chạy.

Tuy nhiên, theo cách tiếp cận như vậy cũng có một số hạn chế:

- **Mã tĩnh / động xen kẽ (Intermingled static/dynamic code)**: Không có sự tách biệt của mã tĩnh, tức là mã được tạo theo cùng một cách cho mọi phần tử mô hình, ví dụ: định nghĩa package v.v. và mã động được bắt nguồn từ thông tin mô hình, ví dụ, tên lớp, tên biến…
- **Cấu trúc đầu ra không thể nắm bắt được (Non-graspable output structure)**: Cấu trúc của đầu ra không dễ dàng nắm bắt được trong đặc tả kỹ thuật của bộ tạo mã. Vấn đề là mã được tạo ra được nhúng vào mã tạo. Do đó, cấu trúc điều khiển của trình tạo mã là rõ ràng, nhưng không phải là định dạng đầu ra. Sự cố này cũng được biểu hiện trong các phương pháp tiếp cận bộ tạo dựa trên GPL khác, chẳng hạn như trong Java Servlets để tạo mã HTML bằng các câu lệnh được nhúng trong mã sản xuất.

- **Thiếu ngôn ngữ truy vấn khai báo (Missing declarative query language)**: Không có sẵn ngôn ngữ truy vấn khai báo để truy cập thông tin mô hình. Do đó, nhiều vòng lặp và điều kiện dẫn đến một lượng lớn mã. Ngoài ra, hãy lưu ý rằng kiến thức về API model đã tạo là bắt buộc. Ví dụ: để truy cập các tính năng của các phần tử mô hình, các phương thức getter phải được sử dụng thay vì truy vấn các giá trị của đối tượng chỉ bằng cách sử dụng tên đối tượng được xác định trong meta-model.
- **Thiếu chức năng cơ sở có thể tái sử dụng (Missing reusable base functionality**: Mã phải được phát triển để đọc các mô hình đầu vào và duy trì mã đầu ra lặp đi lặp lại cho mỗi bộ tạo mã.

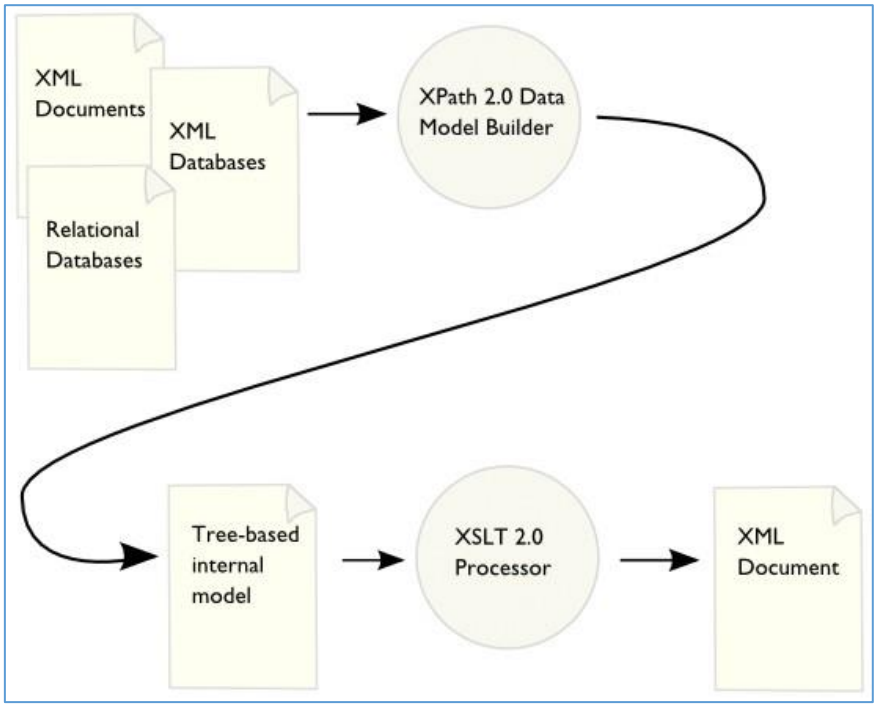
Để loại bỏ những nhược điểm đã đề cập bên trên, DSL đã được phát triển để tạo văn bản từ các mô hình. Điều này cũng dẫn đến một tiêu chuẩn OMG được gọi là Mô hình Ngôn ngữ chuyển đổi mô hình sang văn bản MOF (MOFM2T – Model to Text Transformation Language). Trong phần sau, luận văn sẽ chỉ ra cách bộ tạo mã dựa trên Java có thể được triển khai lại bằng ngôn ngữ chuyển đổi M2T chuyên dụng và thảo luận về lợi ích của cách tiếp cận như vậy.

2.3. Sinh mã nguồn bằng ngôn ngữ chuyển đổi mô hình

Trong phần này, luận văn sẽ trình bày cách các ngôn ngữ chuyển đổi mô hình sang văn bản, cụ thể là các ngôn ngữ chuyển đổi dựa trên mẫu, nhằm giúp dễ dàng phát triển bộ tạo mã, cung cấp tổng quan về các phần chính hiện có và chỉ ra cách sử dụng một trong số chúng để triển khai trình tạo mã cho ví dụ đang được thực thi trong phần luận văn này.

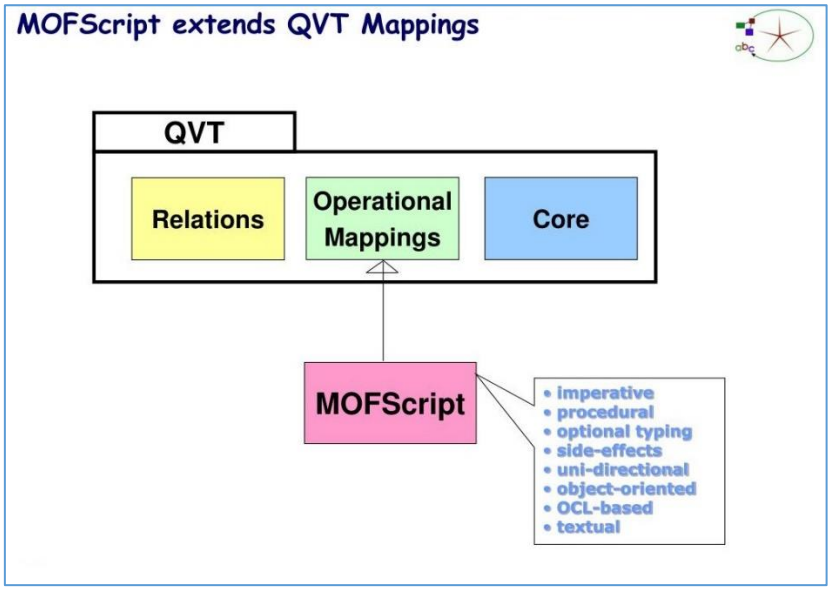
Phần dưới đây sẽ mô tả các ngôn ngữ chuyển đổi mô hình dựa trên mẫu phổ biến có thể được sử dụng để tạo văn bản từ các mô hình [8]:

- **XSLT**: chuyển đổi XSL (XSLT 2.0) là một ngôn ngữ để chuyển đổi các tài liệu XML thành các tài liệu XML, tài liệu văn bản hoặc tài liệu HTML khác. Với XSLT 2.0, việc có thể hoạt động không chỉ trên XML mà còn trên bất kỳ thứ gì có thể được tạo ra để trông giống như XML: bảng cơ sở dữ liệu quan hệ, hệ thống thông tin địa lý, hệ thống tệp… XSLT có khả năng hoạt động trên nhiều tệp đầu vào ở nhiều định dạng và coi tất cả chúng như thể chúng là tệp XML.



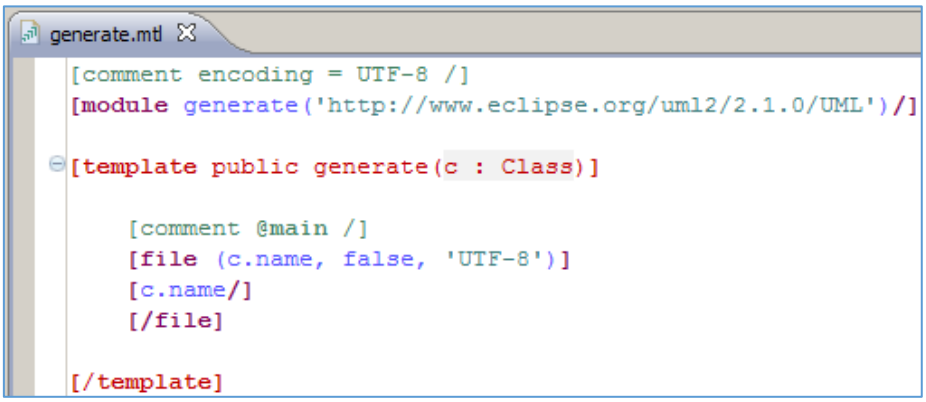
Hình 2. 5. Một ứng dụng điển hình có thể chuyển.

- **JET**: Dự án Java Emitter Template (JET) là một trong những cách tiếp cận đầu tiên để phát triển việc tạo mã cho các mô hình dựa trên EMF. Nhưng JET không giới hạn ở các mô hình dựa trên EMF. Nói chung, với JET, mọi đối tượng dựa trên Java đều có thể chuyển đổi thành văn bản. JET cung cấp một cú pháp giống như JSP được điều chỉnh để viết các mẫu cho các phép biến đổi M2T. Đối với JSP, các biểu thức Java tùy ý có thể được nhúng trong các mẫu JET. Hơn nữa, các mẫu JET được chuyển đổi sang mã Java thuần túy cho các mục đích thực thi. Tuy nhiên, không có ngôn ngữ truy vấn dành riêng cho các mô hình có sẵn trong JET.
- **Xtend**: Xtend là một ngôn ngữ lập trình hiện đại chủ yếu dựa trên Java nhưng cung cấp một số tính năng ngôn ngữ bổ sung. Ví dụ, nó cung cấp hỗ trợ chuyên dụng cho việc tạo mã dưới dạng các biểu thức mẫu. Hơn nữa, nó hỗ trợ lập trình chức năng, có lợi cho các mô hình truy vấn (đặc biệt là nhiều hoạt động dựa trên trình lặp từ OCL có sẵn ngay lập tức).
- **MOFScript**: dự án này cung cấp một ngôn ngữ chuyển đổi M2T khác cung cấp các tính năng tương tự như Xtend. MOFScript đã được phát triển như một đề xuất ứng viên trong nỗ lực tiêu chuẩn hóa OMG cung cấp một ngôn ngữ chuẩn hóa cho các phép biến đổi M2T. MOFScript có sẵn như một trình cắm thêm Eclipse và hỗ trợ các mô hình dựa trên EMF.



Hình 2. 6. Ngôn ngữ chuyển đổi MOFScript.

- **Acceleo**: Mục đích của dự án này là cung cấp một phiên bản thực dụng của tiêu chuẩn chuyển đổi M2T của OMG cho các mô hình dựa trên EMF. Ngôn ngữ này cung cấp hỗ trợ OCL đầy đủ cho các mô hình truy vấn và hỗ trợ công cụ hoàn thiện, đã được chứng minh là hữu ích trong ngành.



Hình 2. 7. Khung triển khai ngôn ngữ Acceleo.

2.4. Kỹ thuật sinh mã nguồn sử dụng ngôn ngữ chuyển đổi Acceleo

2.4.1. Tổng quan

Acceleo được chọn làm sự biểu diễn chính để chứng minh các ngôn ngữ chuyển đổi M2T, vì tính phù hợp với thực tế và đầy đủ sự hỗ trợ từ các công cụ. Lưu ý rằng các tính năng ngôn ngữ của Acceleo hầu hết được hỗ trợ bởi các ngôn ngữ chuyển đổi M2T khác như Xtend hoặc MOFScript.

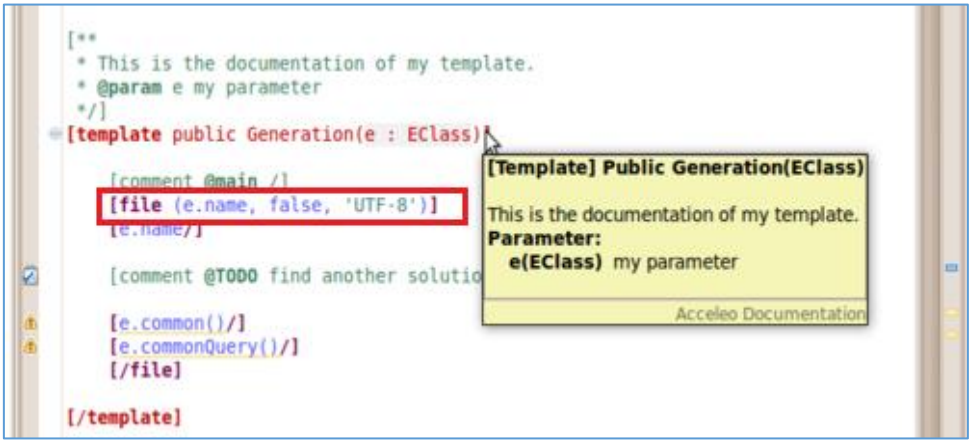
Acceleo cung cấp một ngôn ngữ dựa trên mẫu để xác định các mẫu tạo mã. Ngôn ngữ này đi kèm với một API mạnh mẽ hỗ trợ OCL cũng như các hoạt động bổ sung hữu ích để

làm việc với các tài liệu dựa trên văn bản nói chung, ví dụ: các hàm nâng cao để thao tác chuỗi. Acceleo được cung cấp với công cụ mạnh mẽ như trình chỉnh sửa với đánh dấu cú pháp, phát hiện lỗi, hoàn thành mã, tái cấu trúc, trình gỡ lỗi, trình biên dịch và API truy xuất nguồn gốc cho phép theo dõi các phần tử mô hình với mã được tạo và ngược lại.

Trước khi có thể xác định mẫu trong Acceleo, một mô-đun phải được tạo hoạt động như một vùng chứa cho các mẫu. Mô-đun cũng nhập định nghĩa meta-model mà các mẫu được xác định. Điều này làm cho mẫu nhận biết các lớp meta-model hiện có thể được sử dụng như các loại trong mẫu. Mẫu trong Acceleo luôn được xác định cho một lớp meta-model cụ thể. Ngoài loại phần tử mô hình, điều kiện trước có thể được xác định có thể so sánh với điều kiện lọc trong ATL, ví dụ: để áp dụng một mẫu riêng cho các phần tử mô hình được yêu cầu có một loại cụ thể cũng như các giá trị cụ thể.

Acceleo cung cấp một số thẻ đánh dấu phổ biến trong các ngôn ngữ chuyển đổi M2T có sẵn khác [8]. Một số khái niệm chính được mô tả trong mẫu Acceleo được dùng để chuyển đổi mô hình:

- File: Để tạo mã, tệp phải được mở, điền và đóng như chúng ta đã thấy trước đây đối với trình tạo mã dựa trên Java. Trong Acceleo, có một thẻ tệp đặc biệt được sử dụng để in nội dung được tạo giữa phần đầu và phần cuối của thẻ tệp cho một tệp nhất định. Đường dẫn và tên tệp đều được xác định bởi một thuộc tính của thẻ.



Hình 2. 8. Mô-đun trình chỉnh sửa Acceleo với cú pháp.

- Cấu trúc điều khiển: Có các thẻ để xác định cấu trúc điều khiển như vòng lặp để lặp qua các tập hợp các phần tử, ví dụ: đặc biệt hữu ích để làm việc với các tham chiếu đa giá trị thu được khi điều hướng dẫn đến một tập hợp các phần tử và các nhánh có điều kiện (nếu được gắn).

- Truy vấn: Các truy vấn OCL có thể được xác định (thẻ truy vấn), tương tự như các trình trợ giúp trong ATL. Các truy vấn này có thể được gọi trong toàn bộ mẫu và được sử dụng để tính toán mã định kỳ.
- Biểu thức: Có các biểu thức chung cho các giá trị tính toán để tạo ra các phần động của văn bản đầu ra. Biểu thức cũng được sử dụng để gọi các mẫu khác để bao gồm mã được tạo bởi các mẫu được gọi trong mã được tạo bởi mẫu người gọi. Việc gọi các mẫu khác có thể được so sánh với các cuộc gọi phương thức trong Java.
- Phạm vi truy cập: Một tính năng quan trọng của ngôn ngữ M2T là hỗ trợ các dự án mà chỉ có thể tạo một phần mã. Đặc biệt, cần hỗ trợ đặc biệt để bảo vệ mã được thêm thủ công khỏi các sửa đổi tệp trong các lần chạy trình tạo mã tiếp theo. Đối với nhiệm vụ này, một khái niệm đặc biệt có tên là các phạm vi truy cập đã được chứng minh là hữu ích, được Acceleo hỗ trợ thông qua thẻ được bảo vệ. Các khu vực được bảo vệ được sử dụng để đánh dấu các phần trong mã đã tạo mà sẽ không bị ghi đè lại bởi các lần chạy máy phát tiếp theo. Các phần này thường chứa mã được viết thủ công.

2.4.2. Ví dụ

Danh sách sau đây hiển thị một mẫu Acceleo tương đương với bộ tạo mã dựa trên Java đã trình bày trước đó. Trong dòng đầu tiên, mô-đun được gọi là ***createJavaClass*** được định nghĩa bao gồm việc nhập meta-model sMVCML. Bản thân mô-đun được cấu trúc thành một mẫu chính (xem mẫu javaClass) nhắm mục tiêu tạo mã cho các lớp các lớp sMVCML [8]. Mẫu này ủy quyền việc tạo mã cho các thuộc tính và hoạt động cho các mẫu cụ thể bổ sung.

```
[module generateJavaClass('http://smvcml/1.0')]

[query public getter(att : Attribute) : String = 'get'+att.name.toUpperFirst() /]

[query public returnStatement(type: String) : String = if type = 'Boolean'
    then 'return true;' else '...' endif  /]

[template public javaClass(aClass : Class)]

[file (aClass.name.toUpperFirst()+'.java', false, 'UTF-8')]
package entities;

import java.io.Serializable;

public class [aClass.name/] implements Serializable {

[for (att : Attribute | aClass.atts) separator ('\n')]
[javaAttribute(att)/]
[/for]

[for (op : Operation | aClass.ops) separator ('\n')]
[javaMethod(op)/]
[/for]

}

[/file]
[/template]

[template public javaAttribute(att : Attribute)]
  private [att.type/] [att.name/];

  public [att.type/] [att.getter()/]() {
    return [att.name/];
  }
    ...
[/template]

[template public javaMethod(op : Operation)]
  public [op.type/] [op.name/]() {
    // [protected (op.name)]
    // Fill in the operation implementation here!
    [returnStatement(op.type)/]
    // [/protected]
  }
[/template]
```

Hình 2. 9. Sinh mã nguồn dựa trên Acceleo.

Trong danh sách, một số thẻ khác nhau được sử dụng. Thẻ Query được sử dụng để tạo tên chữ ký cho các phương thức getter và setter cần thiết để truy cập / sửa đổi các thuộc tính cũng như để tính toán câu lệnh trả về mặc định dựa trên kiểu trả về để đảm bảo tạo ra mã có thể biên dịch được. Thẻ tệp được sử dụng để mở và đóng tệp trong đó mã cho lớp Java được in. Thẻ For được sử dụng để lặp qua các thuộc tính và hoạt động của một lớp đã xử lý để gọi các mẫu cụ thể.

Biểu thức được sử dụng nhiều lần. Ví dụ, [cl.name/] in tên của lớp vào luồng văn bản. Các biểu thức khác đang gọi các mẫu, ví dụ: [javaAttribute (att) /] được sử dụng để gọi mẫu để tạo mã cho các thuộc tính và [att.getter () /] được sử dụng để gọi truy vấn để tạo ra tên của các phương thức getter.

Trong khuôn mẫu javaMethod, một vùng được bảo vệ được sử dụng để xác định không gian bao gồm việc thực hiện các hoạt động. Đầu ra được tạo cho mẫu này được hiển thị trong hình dưới đây. Xin lưu ý rằng trong lần chạy trình tạo đầu tiên, vùng được bảo vệ được tạo chỉ bao gồm nhận xét chuẩn và giá trị trả về mặc định như được cung cấp trong mẫu dưới dạng trình giữ chỗ cho việc triển khai phương pháp thực tế. Trong tất cả các lần chạy trình tạo tiếp theo, khoảng trống này không bị thay đổi lại, có nghĩa là người dùng có thể triển khai phương pháp và mã được thêm theo cách thủ công sẽ không bị mất trong các lần chạy trình tạo sau đó.

```
public boolean checkAvailability(){
    // Start of user code checkAvailability
    // Fill in the operation implementation here!
    return true;
    // End of user code
}
```

Hình 2. 10. Vùng bảo vệ của phương thức Java.

2.5. Tổng kết chương

Chuyển đổi mô hình sang văn bản, cụ thể là sinh mã nguồn tự động, có thể được triển khai bằng cách sử dụng ngôn ngữ lập trình hoặc ngôn ngữ chuyển đổi mô hình. Mặc dù dễ nắm bắt, dễ lập trình tuy nhiên việc sử dụng ngôn ngữ lập trình vẫn còn nhiều hạn chế. Từ những hạn chế đó, các ngôn ngữ chuyển đổi mô hình dựa trên mẫu được tạo ra nhằm dễ dàng đọc các mô hình đầu vào và khai báo mã nguồn đích ứng với ngôn ngữ cụ thể. Có khá nhiều các ngôn ngữ chuyển đổi, trong số đó có Acceleo là một trong số những ngôn ngữ phổ biến và dễ triển khai.

CHƯƠNG 3. SINH TỰ ĐỘNG MÃ NGUỒN JAVA TỪ BIỂU ĐỒ LỚP BẰNG ACCELEO

Nội dung chương này sẽ hướng tới khía cạnh sinh tự động mã nguồn từ mô hình, cụ thể hơn là biểu đồ lớp, một khía cạnh chính của kỹ thuật chuyển đổi mô hình sang văn bản. Chương sẽ giới thiệu về tình huống nghiên cứu, khó khăn và thách thức gặp phải, tiếp đến sẽ đi sâu vào các quy tắc chuyển đổi được áp dụng bằng việc sử dụng ngôn ngữ chuyển đổi Acceleo. Tiếp đến sẽ đưa ra ví dụ minh họa về một hệ thống cụ thể, các dữ liệu mẫu cần thiết, và cuối cùng là tổng kết các vấn đề đã nêu trong chương.

3.1. Giới thiệu

Sinh mã nguồn là một khía cạnh không phải mới, tuy nhiên áp dụng một cách hiệu quả là một điều không phải đơn giản. Luận văn không hướng đến toàn bộ các nghiệp vụ của các hệ thống để từ đó có thể sinh mã nguồn, mà sẽ tập trung hướng tới nghiệp vụ của một hệ thống cụ thể, xoay quanh những chức năng cụ thể nhằm đạt được hiểu quả tốt nhất. Bên cạnh việc xác định phạm vi hệ thống cần nghiên cứu, luận văn cũng xác định phạm vi mô hình làm đầu vào cho việc sinh mã tự động. Để mô hình hóa một hệ thống thì có rất nhiều dạng biểu đồ có thể làm được, có thể kể đến: biểu đồ ca sử dụng (use case diagram), biểu đồ trình tự (sequence diagram), biểu đồ hoạt động (activity diagram), biểu đồ lớp (class diagram)… Ý tưởng luận văn hướng tới đó là dựa vào đặc tả về thuộc tính và các chức năng của hệ thống để chuyển đổi sang văn bản, cụ thể ở đây là sinh mã nguồn. Các biểu đồ ca sử dụng, biểu đồ trình tự hay biểu đồ hoạt động chú trọng tới diễn trình tự hoạt động của hệ thống mà không mô tả được đặc điểm của hệ thống là gì. Chính vì vậy luận văn áp dụng mô hình hóa hệ thống qua biểu đồ lớp và mã nguồn sinh ra là Java. Như vậy với đầu vào là biểu đồ lớp mô hình hóa một hệ thống cụ thể sẽ được chuyển đổi, hay sinh tự động thành mã nguồn Java.

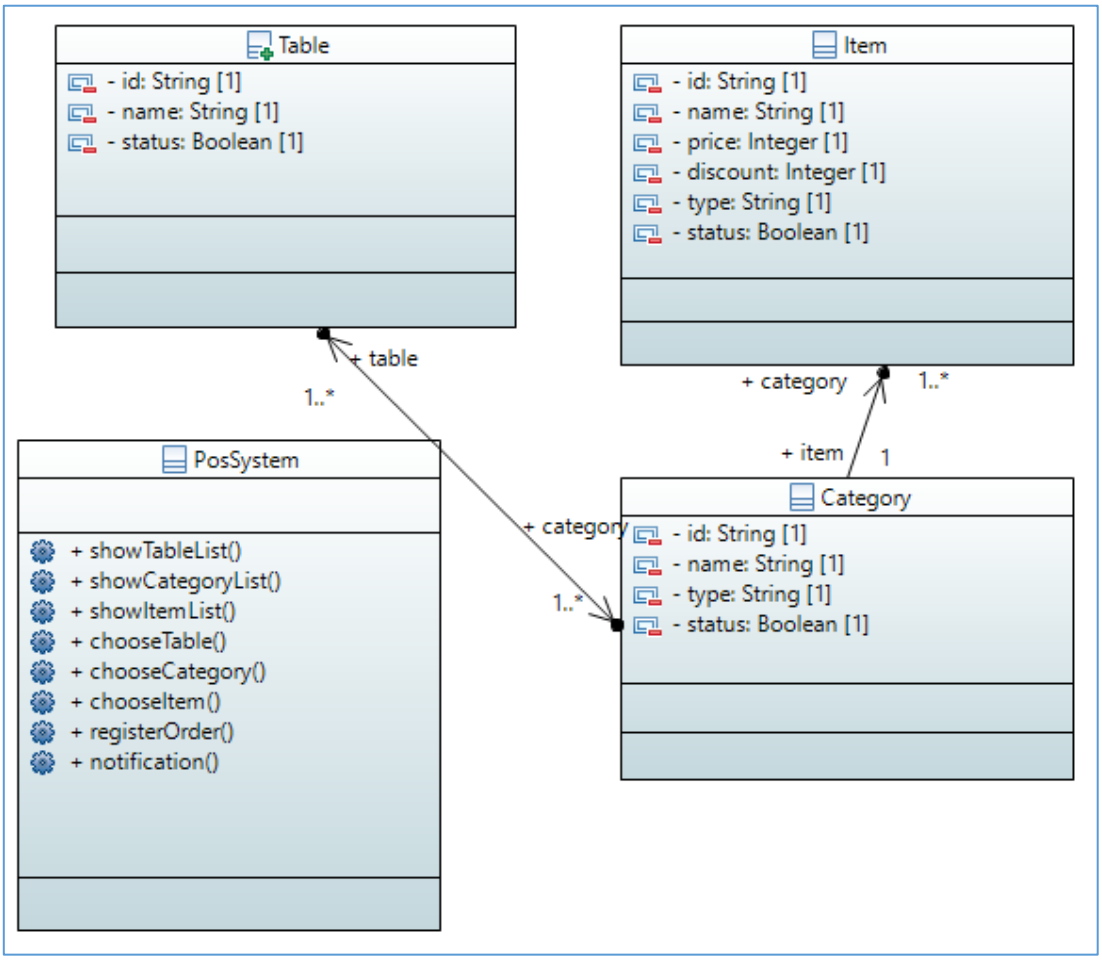
3.2. Nghiên cứu tình huống

Để sinh tự động mã nguồn thì hai điều quan trọng nhất đó là mô hình đầu vào và phương pháp. Thứ nhất, mô hình đầu vào là kết quả của việc mô hình hóa một hệ thống hoặc một chức năng cụ thể. Mô hình đầu vào luận văn áp dụng là biểu đồ lớp dưới dạng file UML. Và thứ hai đó chính là phương pháp, phương pháp chính là cách thức thực hiện hay cụ thể hơn là các quy tắc. Câu hỏi đặt ra là các quy tắc bám sát theo đặc điểm của hệ thống như thế nào? Có rất nhiều cách trích xuất thông tin từ mô hình. Nhìn chung sẽ phải dựa vào chuỗi ký tự mô tả phương thức trong từng lớp của biểu đồ lớp, tuy vậy sẽ thật khó để trích xuất chính xác ví điều này cần tới xử lý ngôn ngữ tự nhiên và không phải hệ thống nào cũng có thể xử lý được, do đặc thù về nghiệp vụ phức tạp… Chính vì lý do này, luận văn hướng tới khai thác những đặc điểm, tính chất của hệ thống, được mô tả qua thuộc tính và phương thức của lớp trong biểu đồ lớp.

Như đã trình bày trong mục trước, luận văn sẽ xoay quanh một phạm vi cụ thể nhằm mục đích mã nguồn sau khi sinh tự động có thể được thực thi hiệu quả nhất. Luận văn sẽ hướng tới bài toán ***Điểm bán hàng tự động (POS – Point of Sale)*** và sẽ tập trung khai thác tính năng chọn món (***Ordering***) của hệ thống này. Các đặc điểm chính của tính năng chọn món có thể kể đế như chọn bàn còn trống, chọn danh mục các món và chọn một món bất kỳ…

3.2.1. Biểu đồ lớp

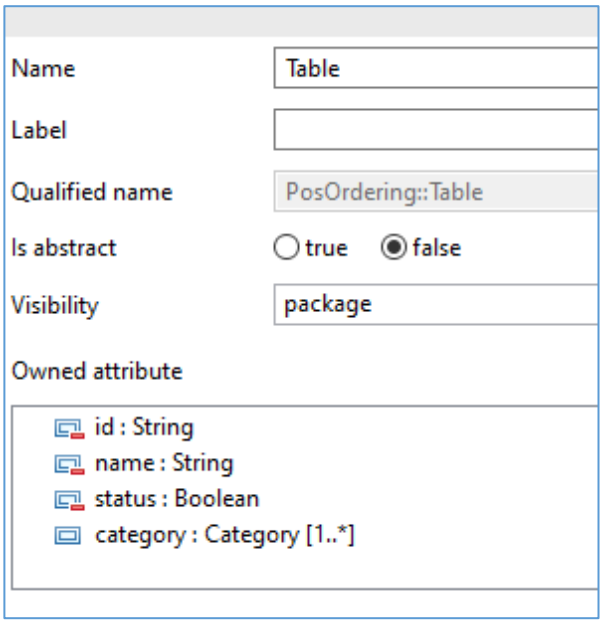
Phần này luận văn sẽ ví dụ minh họa với các lớp trong tính năng ***Ordering*** của hệ thống ***POS***. Hình dưới đây mô tả biểu đồ lớp:



Hình 3. 1. Biểu đồ lớp cho chức năng đặt hàng (ordering).

Biểu đồ lớp hình trên mô tả các lớp ứng với một hệ thống quản lý bán hàng. Các lớp được cài đặt bao gồm: ***PosSystem***, ***Table***, ***Category*** và ***Item***. Đây là những đối tượng cơ bản và chủ yếu bao quát bài toán ***Ordering***, mọi yêu cầu nâng cấp, sửa đổi bổ sung sẽ được thêm xoay quanh các đối tượng này. Phần tiếp theo sẽ mô tả cụ thể các lớp và chi tiết ý nghĩa các phương thức, thuộc tính của từng lớp.

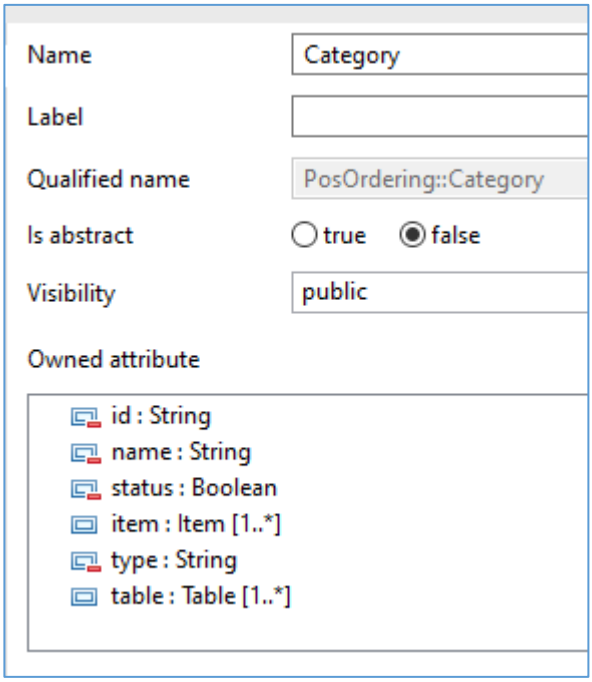
- ***Table***: đại diện cho các bàn trong một nhà hàng, cửa tiệm.
  - ***id***: định danh bàn, mỗi bàn sẽ có một id định danh duy nhất.
  - ***name***: tên bàn.
  - ***status***: trạng thái hiện tại của bàn (true: bàn đang trống, false: bàn đã có khách ngồi).
  - ***category***: ngoài ra còn có thuộc tính category sinh ra bởi mối liên hệ giữa table và category là liên kết n – n. Điều này có nghĩa một bàn có thể được order nhiều danh mục và một danh mục có thể được order trên nhiều bàn miễn là trạng thái danh mục đang là true.



Hình 3. 2. Thuộc tính và phương thức lớp Table.

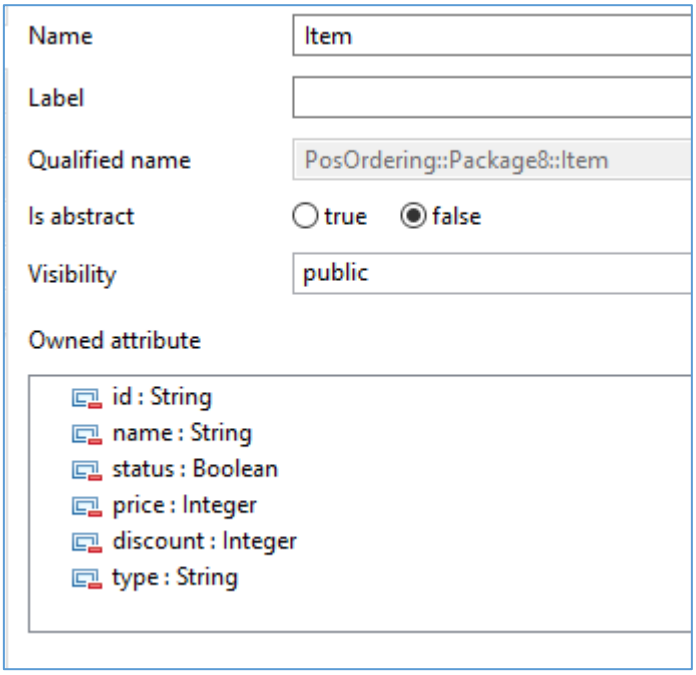
- ***Category***: là danh mục các hàng hóa đang lưu trữ tại cửa hàng, bao gồm các thuộc tính:
  - ***id***: định danh danh mục. Định danh này cũng phải là duy nhất cho từng loại danh mục.
  - ***name***: tên danh mục.
  - ***type***: tên loại danh mục (đồ ăn, đồ uống, topping…)
  - ***status***: trạng thái hiện tại của danh mục (true: danh mục hiện có tại cửa hàng, false: danh mục hiện chưa/không có tại cửa hàng…).
  - ***item***: ngoài ra còn có thuộc tính item sinh ra bởi mối liên hệ giữa category và item là liên kết 1 – n.

- ***table***: thuộc tính này giống với lớp Table phía trên, mô tả mối liên hệ giữa Table và Category là n – n.



Hình 3. 3. Thuộc tính và phương thức lớp Category.

- ***Item***: là các món hàng hóa có tại nhà hàng, các thuộc tính cụ thể gồm:
  - ***id***: định danh hàng hóa. Định danh này cũng phải là duy nhất cho từng loại hàng hóa.
  - ***name***: tên hàng hóa.
  - ***price***: giá một đơn vị hàng hóa (chiếc, cái, cốc…).
  - ***discount***: giá được giảm bao nhiêu tiền (nếu có > 0).
  - ***type***: tên danh mục hàng hóa.
  - ***status***: trạng thái hiện tại của hàng hóa (true: hàng hóa hiện có tại cửa hàng, false: hàng hóa hiện chưa/không có tại cửa hàng…).



Hình 3. 4. Thuộc tính và phương thức lớp Item.

- ***PosSystem***: đây là lớp chính nhằm cung cấp các phương thức hiển thị thông tin ngoài màn hình, chọn các table, category hoặc item, update và thông báo kết quả, đây là lớp bao quát, điều khiển hoạt động của toàn bộ chức năng có trong hệ thống, bao gồm các phương thức sau:
  - ***notification()***: thông báo cho người dùng các thông tin liên quan đến order hàng hóa tại cửa hàng.
  - ***showTableList()***: hiển thị danh sách tất cả các bàn đang thuộc cửa hàng.
  - ***showCategoryList()***: hiển thị danh sách tất cả các danh mục đang thuộc cửa hàng.
  - ***showItemList()***: hiển thị danh sách tất cả các món hàng hóa đang thuộc danh mục lựa chọn.
  - ***chooseTable()***: chọn bàn theo yêu cầu của khách hàng.
  - ***chooseCategory()***: chọn danh mục theo yêu cầu của khách hàng.
  - ***chooseItem()***: chọn hàng hóa theo yêu cầu của khách hàng.
  - ***registerOrder()***: order danh sách sau khi chốt.

3.2.2. Cách thức thực hiện

Sau khi có được biểu đồ lớp mô hình hóa tính năng ***Ordering***, phần này sẽ trình bày các quy tắc nhằm ánh xạ biểu đồ lớp sang mã nguồn Java.

Quy tắc số 1: Ánh xạ các lớp và thuộc tính lớp của biểu đồ sang Class và Attribute tương ứng trong mã nguồn Java.

Quy tắc đầu tiên sẽ ánh xạ từ các lớp và tên thuộc tính, kiểu thuộc tính sang Class và Attribute tương ứng trong mã nguồn Java được mô tả trong bảng dưới đây:

Bảng 3. 1. Quy tắc ánh xạ từ biểu đồ lớp sang mã nguồn Java

Biểu đồ lớp	Class (mã nguồn Java)
Tên lớp	→ File mã nguồn Java (tên trùng với tên lớp) và tên Class. → Tên hàm khởi tạo Constructor. → Tên hàm khởi tạo với tham số "name" nếu có.
Thuộc tính	→ Thuộc tính trong Java Class gồm: tên thuộc tính, tên kiểu trả về (String, Boolean hoặc Integer…), phạm vi truy cập thuộc tính (public, protected hoặc private). → Phương thức get() → Phương thức set()

Bảng 3.1 là mô tả Quy tắc 1 cho việc ánh xạ từ lớp trong biểu đồ lớp sang Class tương ứng mã nguồn Java. Cụ thể, với tên lớp trong biểu đồ lớp sẽ được ánh xạ sang các thành phần như: file mã nguồn, tên Class, hàm Constructor… Ứng với từng thuộc tính sẽ được ánh xạ sang các thành phần như thuộc tính (gồm tên, kiểu trả về, phạm vi truy cập…)

Quy tắc số 2: Ánh xạ phương thức hiển thị danh sách chuyển sang thân hàm trong Java với vòng lặp forEach().

Từ phương thức ***showList*** trong biểu đồ lớp sẽ chuyển sang thân phương thức mã nguồn Java:

Bảng 3. 2. Quy tắc ánh xạ từ phương thức showItemList() sang mã nguồn Java

Biểu đồ lớp	Class (mã nguồn Java)
Phương thức showList()	→ Vòng lặp forEach duyệt các phần tử mảng. Nếu phần tử mảng có các thuộc tính trùng với tên danh mục thì sẽ hiển thị phần tử khớp, nếu không sẽ hiển thị toàn bộ item

Như mô tả trong 3.6 phương thức hiển thị danh sách sẽ được ánh xạ sang một vòng lặp forEach duyệt các phần tử của mảng và đưa vào một khung giao diện được dựng sẵn.

Quy tắc số 3: Từ phương thức chọn phần tử chuyển sang thân hàm trong Java với câu lệnh điều khiển if/else kiểm tra trạng thái item.

Bảng 3. 3. Quy tắc ánh xạ phương thức chooseItem() sang mã nguồn Java

Biểu đồ lớp	Class (mã nguồn Java)
Phương thức choose()	→ Câu lệnh điều khiển if/else kiểm tra tên phần tử xem có khớp với tên phần tử được chọn hay không, sau đó kiểm tra trạng thái phần tử.

Từ phương thức ***choose*** trong biểu đồ lớp sẽ ánh xạ sang thân phương thức mã nguồn Java, với câu lệnh điều khiển if/else nhằm chọn phần tử có tên ứng với tên được chọn trên giao diện khung mẫu, sau đó kiểm tra trạng thái phần tử, nếu đúng (true/active) thì lưu lại phần tử, nếu trạng thái là sai (false/inactive) thì đưa ra thông báo.

Quy tắc số 4: Đăng ký và lưu, hiển thị kết quả

Bảng 3. 4. . Quy tắc ánh xạ phương thức register()
và notification() sang mã nguồn Java

Biểu đồ lớp	Class (mã nguồn Java)
Phương thức register()	→ Câu lệnh điều khiển if/else kiểm tra giá trị biến toàn cục xem có null hay không. Nếu có giá trị thì gán vào một biến toàn cục, sau đó ghi kết quả ra file.
Phương thức notification()	→ Cũng gán vào một biến toàn cục khác nhưng dùng cho kết quả tạm, khi người dùng vừa thao tác với các phần tử trên màn hình

Phương thức lưu và hiển thị kết quả được chuyển đổi thông qua việc gán vào các biến toàn cục và hiển thị dưới dạng text. Việc đăng ký sẽ ghi các biến cục bộ đã được lưu giá trị dưới dạng biến String, đồng thời lưu ra file kết quả tương ứng.

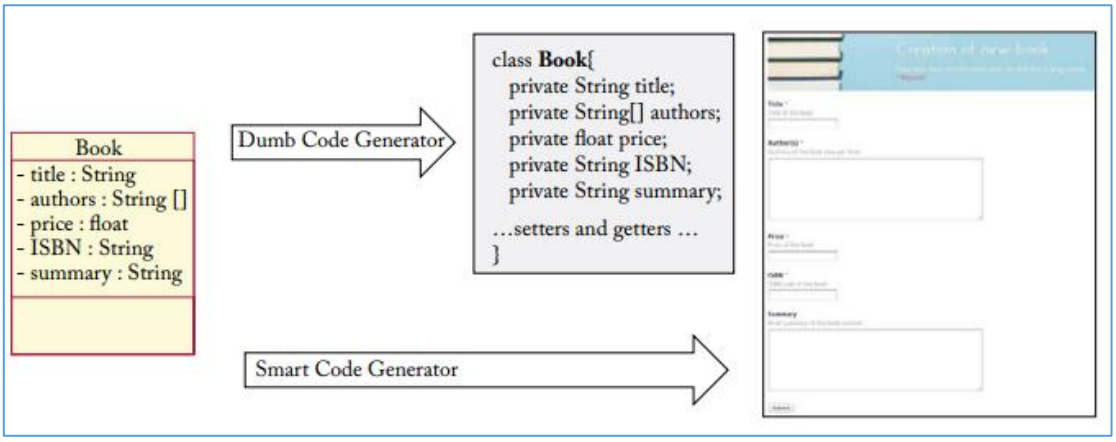
3.3. Đặc tả chuyển Acceleo

Trong phần tiếp theo, luận văn sẽ trình bày các quy tắc chuyển đổi nhằm cụ thể hóa việc ánh xạ từ biểu đồ lớp sang mã nguồn Java, sử dụng ngôn ngữ chuyển đổi mô hình là Acceleo.

3.3.1. Quy tắc chuyển đổi

3.3.1.1. Quy tắc chuyển đổi tĩnh

Quy tắc chuyển đổi tĩnh (***dumb code generation***), có thể coi là việc chuyển đổi câm, nghĩa là toàn bộ mã nguồn sau khi được chuyển đổi từ model hoàn toàn không thực thi được, mà chỉ được coi là mã nguồn thụ động. Tuy vậy đây là phần xương sống, là phần khung khởi tạo ban đầu cho mọi mã nguồn, đặc biệt là mã nguồn hướng đối tượng, cụ thể ở đây là Java. Luận văn áp dụng quy tắc chuyển đổi tĩnh nhằm sinh mã nguồn dưới dạng các thuộc tính, phương thức cơ bản như set(), get() hay một vài phương thức cơ bản khác… Trong phạm vi nghiên cứu và áp dụng, luận văn sẽ sinh mã nguồn tĩnh trước tiên, từ biểu đồ lớp class diagram, trong đó chuyển đổi các lớp được mô tả trong biểu đồ thành các lớp tương ứng trong lập trình Java. Hình 3.1 mô tả việc chuyển đổi tĩnh.



Hình 3. 5. Ánh xạ từ lớp trong biểu đồ sang lớp trong mã nguồn Java [8].

Như vậy Quy tắc số 1 chính là việc chuyển đổi tĩnh và sẽ được thể hiện bằng cách sử dụng ngôn ngữ Acceleo, chi tiết trong bảng dưới đây:

Quy tắc số 1: Ánh xạ các lớp và thuộc tính lớp của biểu đồ sang Class và Attribute tương ứng trong mã nguồn Java.

Bảng 3. 5. Sử dụng Acceleo ánh xạ từ biểu đồ lớp sang mã nguồn Java

Biểu đồ lớp	Class (mã nguồn Java)

Tên lớp	→ File (tên trùng với tên lớp) và tên lớp:
	`[file (aClass.name.concat('.java'), false, 'UTF-8')]`
	`public class [aClass.name.toUpperFirst()/]`
	→ Tên hàm khởi tạo:
	`public [aClass.name.toUpperFirst()/]() { }`
	→ Tên hàm khởi tạo với tham số "name" nếu có:
	`[if (p.name = 'name')]`
	`public [aClass.name.toUpperFirst()/]([p.type.name/] [p.name/]) {`
	`this.set[p.name.toUpperFirst()/]([p.name/]);`
	`}`
	`[/if]`

Thuộc tính	Thuộc tính lớp gồm: tên thuộc tính, tên kiểu return, mức độ truy cập thuộc tính
	`[for (p: Property | aClass.attribute) separator('\n')]`
	`[p.visibility/] [p.type.name/] [p.name/];`
	`[/for]`
	Phương thức get()
	`public [p.type.name/] get[p.name.toUpperFirst()/]()`
	`{`
	`return this.[p.name/];`
	`}`
	Phương thức set()
	`public void set[p.name.toUpperFirst()/]([p.type.name/] [p.name.toUpperFirst()/]) {`
	`this.[p.name/] = [p.name.toUpperFirst()/];`
	`}`

Như vậy, mã nguồn Acceleo cho quy tắc số 1 như sau:

generate.mtl

```
502 [comment] Chuyen doi tinh [/comment]
503 [file (aClass.name.concat('.java'), false, 'UTF-8')]
504 package Target;
505
506 public class [aClass.name.toUpperFirst()/] {
507
508     public [aClass.name.toUpperFirst()/]() { }
509
510     [for (p: Property | aClass.attribute) separator('\n')]
511     [p.visibility/] [p.type.name/] [p.name/];
512
513     [if (p.name = 'name')]
514     public [aClass.name.toUpperFirst()/]([p.type.name/] [p.name/]) {
515         this.set[p.name.toUpperFirst()/]([p.name/]);
516     }
517     [/if]
518
519     public [p.type.name/] get[p.name.toUpperFirst()/]() {
520         return this.[p.name/];
521     }
522
523     public void set[p.name.toUpperFirst()/]([p.type.name/] [p.name.toUpperFirst()/]) {
524         this.[p.name/] = [p.name.toUpperFirst()/];
525     }
526     [/for]
527 }
528 [/file]
```

Hình 3. 6. Mã nguồn Acceleo chuyển đổi tĩnh.

3.3.1.2. Quy tắc chuyển đổi mở rộng

Ngoài các quy tắc chuyển đổi tĩnh được mô tả ở phần trước, phần này sẽ mô tả các quy tắc chuyển đổi mở rộng từ biểu đồ lớp sang mã nguồn Java. Bản chất cơ chế của các quy tắc chuyển đổi mở rộng dựa trên hai yếu tố chính. Thứ nhất, cơ chế của quy tắc chuyển đổi mở rộng phải dựa trên phạm vi miền ứng dụng cụ thể. Ứng với từng ứng dụng sẽ có quy tắc riêng, khó có thể áp dụng quy tắc chung cho toàn bộ mọi vấn đề, mọi bài toán đặt ra, và như đã trình bày bài toán nghiên cứu, phạm vi luận văn sẽ hướng tới tính năng ***Ordering***. Yếu tố thứ hai đó là tên phương thức. Bên cạnh việc áp dụng vào một bài toán cụ thể, luận văn cũng dựa khá nhiều trên tên đặt cho các phương thức, với các tiết đầu ngữ ví dụ như: show, choose, register hay notification…

Bên cạnh khung mã nguồn được sinh từ quy tắc chuyển đổi tĩnh, quy tắc chuyển đổi mở rộng nhằm chuyển đổi tên các phương thức được định nghĩa thành thân các hàm tương ứng mã nguồn Java một cách đầy đủ và có thể thực thi được. Phần này sẽ cụ thể hóa những quy tắc tiếp theo trong miền ứng dụng cho tính năng ***Ordering*** bằng cách sử dụng Acceleo.

Quy tắc số 2: Ánh xạ phương thức hiển thị danh sách chuyển sang thân hàm trong Java với vòng lặp forEach().

Trước tiên sẽ đọc vào tên các lớp và ghi ra tên file Java tương ứng “aClass.name.concat(‘.java’)” và tên class “[aClass.name.toUpperFirst()/]”. Tiếp theo sẽ khởi tạo thuộc tính lớp, phương thức set() và get() theo từng thuộc tính, cùng với đó là hàm khởi tạo (constructor). Sau khi khởi tạo các lớp với bộ khung là các thuộc tính, phần tiếp theo sẽ sinh mã nguồn áp dụng quy tắc chuyển đổi mở rộng. Trước hết, ứng với phương

thức showItemList sẽ duyệt mảng các item, nếu tên danh mục được chọn khớp với tên danh mục của item thì sẽ tạo một RadioButton và bổ sung các mã nguồn liên quan đến giao diện, cụ thể như hình dưới đây:

generate.mtl

```
[elseif (o.name.contains('showItemList'))]
[o.visibility/] void [o.name/]() {
    itemButton.addActionListener(new ActionListener() {
        int yCoordinates = 30;

        public void actionPerformed(ActionEvent e) {
            JFrame chosenFrame = new JFrame("Item");
            ButtonGroup bg = new ButtonGroup();
            JButton jButton = new JButton("OK");
            if (finalCategory != null) {
                itemList.forEach(item -> {
                    if (item.getType().equals(finalCategory.getName())) {
                        JRadioButton radioButton = new JRadioButton(item.getName());
                        radioButton.setBounds(30, yCoordinates, 100, 30);
                        bg.add(radioButton);
                        chosenFrame.add(radioButton);
                        yCoordinates = yCoordinates + 30;
                        chooseItem(radioButton);
                    }
                });
            } else {
                itemList.forEach(item -> {
                    JRadioButton radioButton = new JRadioButton(item.getName());
                    radioButton.setBounds(30, yCoordinates, 100, 30);
                    bg.add(radioButton);
                    chosenFrame.add(radioButton);
                    yCoordinates = yCoordinates + 30;
                    chooseItem(radioButton);
                });
            }

            chosenFrame.setSize(500, 500);
            chosenFrame.setLayout(null);
```

Hình 3. 7. Mã nguồn Acceleo chuyển đổi phương thức showItem.

Quy tắc số 3: Từ phương thức chọn phần tử chuyển sang thân hàm trong Java với câu lệnh điều khiển if/else kiểm tra trạng thái item.

Tiếp đến là quy tắc chuyển đổi phương thức “chooseItem”.

*generate.mtl

```
[elseif (o.name.contains('chooseItem'))]
private class ItemAgreeActionListener implements ItemListener {
    @Override
    public void itemStateChanged(ItemEvent e) {
        JRadioButton radio = (JRadioButton) e.getSource();
        if (radio.isSelected()) {
            instance.chosenItemName = radio.getText().trim();
            for (Item item : PosSystem.itemList) {
                if (item.getName().equals(instance.chosenItemName))
                    instance.finalItem = item;
            }
            if (instance.finalItem.getStatus().booleanValue() == true)
                refreshResult(instance.chosenTableName,
                    instance.chosenCategoryName, instance.chosenItemName);
            else
                JOptionPane.showMessageDialog(new JFrame(), "ITEM STATUS NOT ACTIVE!");
        }
    }
}
```

Hình 3. 8. Mã nguồn Acceleo chuyển đổi phương thức chooseItem.

Nếu tên phương thức được duyệt chứa "chooseItem" thì sẽ sinh ra mã nguồn tương ứng, mã nguồn này sẽ cài đặt sẵn các RadioButton trong trường hợp được chọn (click), sau đó sẽ kiểm tra trạng thái của item, nếu hợp lệ sẽ gán vào biến kết quả.

Quy tắc số 4: Đăng ký và lưu, hiển thị kết quả

Cuối cùng là quy tắc chuyển đổi phương thức thông báo và đăng ký món (notification, register). Phương thức đăng ký sẽ tính giá item và gán kết quả cuối cùng vào biến toàn cục và ghi kết quả ra file (có thể dùng cho in hóa đơn sau này nếu cần). Phương thức thông báo chủ yếu gán kết quả cuối cùng vào biến toàn cục nhằm hiển thị thông tin tại thời điểm thao tác (VD: chọn bàn, chọn danh mục hay chọn món…).

generate.mtl

```
476
477     [elseif (o.name.contains('notification'))]
478     [o.visibility/] void [o.name/](String tableName, String cateName, String itemName) {
479         String resultText = "Table: " + tableName + "\nCategory: "
480             + cateName + "\nItem: " + itemName;
481         PosSystem.result.setText(resultText);
482     }
483     [/if]
484     [/for]
485
486     [elseif (o.name.contains('registerOrder'))]
487     [o.visibility/] void [o.name/]() {
488         registerButton.addActionListener(new ActionListener() {
489             public void actionPerformed(ActionEvent e) {
490                 if (finalItem != null) {
491                     orderDetail = "YOU HAVE JUST ORDER ITEM WITH THE DETAIL" + "\nItem name: "
492                             + finalItem.getName()
493                             + "\nItem price: " + finalItem.getPrice() + "\nItem discount price: "
494                             + finalItem.getDiscount() + "\nTotal: "
495                             + (finalItem.getPrice() - finalItem.getDiscount());
496                     JOptionPane.showMessageDialog(new JFrame(), orderDetail);
497                     PosData.writeResultTofile(orderDetail);
498                 } else {
499                     JOptionPane.showMessageDialog(new JFrame(), "NO ORDERED RESULT!");
500                 }
501             }
502         });
503     }
504     [/if]
505     [/for]
```

Hình 3. 9. Mã nguồn Acceleo chuyển đổi phương thức register và notification.

3.4. Template và dữ liệu mẫu

Luận văn áp dụng các quy tắc chuyển đổi nhằm sinh mã nguồn từ class diagram, tuy nhiên sẽ không thể thực thi mã nguồn ngay lập tức, mà sẽ phải dựa trên một template cụ thể, xây dựng sẵn áp dụng cho bài toán cụ thể. Template mẫu ở đây chính là giao diện người dùng, áp dụng mã nguồn Java swing với các component sau: JFrame, ButtonGroup, ImageIcon, JButton, JLabel, JMenu, JRadioButton... Trong đó JFrame nhằm khởi tạo khung template mẫu, các ButtonGroup, JButton, JRadioButton để hỗ trợ việc hiển thị danh sách các phần tử khi áp dụng Quy tắc số 2, Quy tắc số 3, bên cạnh đó các component JOptionPanel, JFrame hỗ trợ việc thông báo, đăng ký trong Quy tắc số 4.

Bên cạnh đó, để thực thi mã nguồn, ngoài template và mã nguồn chuyển đổi, cũng cần bộ dữ liệu mẫu để kiểm tra tính đúng của các quy tắc. Tính đúng đắn được kiểm tra dựa

theo dữ liệu mẫu, xem output hiển thị có đúng với dữ liệu mẫu hay không. Bộ dữ liệu mẫu sẽ được đưa vào dưới dạng JSON (JavaScript Object Notation) và sẽ được nạp vào các biến trong mã nguồn.

3.5. Tổng kết chương

Từ khía cạnh chuyển đổi mô hình sang văn bản, sinh mã nguồn Java từ biểu đổ lớp là một bài toán cụ thể. Biểu đồ lớp là một biểu đồ UML được thiết kế gồm nhiều thành phần, chủ yếu xoay quanh thuộc tính và phương thức của lớp. Sau khi đã có mô hình đầu vào, để chuyển đổi mô hình tự động thì yếu tố quan trọng nhất là phép chuyển đổi, cụ thể hơn là quy tắc chuyển đổi. Để áp dụng sinh mã nguồn Java từ biểu đồ lớp thì cần quy tắc chuyển đổi tĩnh và quy tắc chuyển đổ mở rộng. Ngoài việc sinh mã nguồn tĩnh, chưa thể thực thi như quy tắc tĩnh, quy tắc mở rộng sẽ linh động hơn, có nhiều mã nguồn logic được sinh tự động hơn nhằm mô tả chính xác các thành phần của hệ thống đã được mô hình hóa. Tuy nhiên quy tắc mở rộng cần áp dụng dựa trên mô hình cho một miền ứng dụng cụ thể. Luận văn sẽ nghiên cứu tình huống với tính năng ***Ordering*** trong miền ứng dụng ***POS*** và cụ thể hóa các quy tắc chuyển đổi bằng ngôn ngữ Acceleo. Bên cạnh đó cũng cần bổ sung template mẫu và dữ liệu mẫu nhằm chuẩn bị dữ liệu cho việc thực thi mã nguồn sau khi được sinh tự động.

CHƯƠNG 4. CÀI ĐẶT VÀ THỰC NGHIỆM

4.1. Môi trường cài đặt

Phần này luận văn mô tả môi trường cho việc cài đặt các cấu phần cần thiết cho việc thực nghiệm ví dụ. Môi trường cài đặt gồm các yêu cầu phần cứng, phần mềm, dữ liệu đầu vào và cuối cùng sẽ trình bày các bước thức hiện.

4.1.1. Cấu hình phần cứng, phần mềm

Chương này sẽ đi vào cài đặt chương trình và thực nghiệm mã nguồn sau khi sinh tự động. Về mặt cấu hình phần cứng, bảng dưới sẽ mô tả cấu hình tối thiểu phần cứng đáp ứng việc thực nghiệm.

Bảng 4. 1. Cấu hình phần cứng

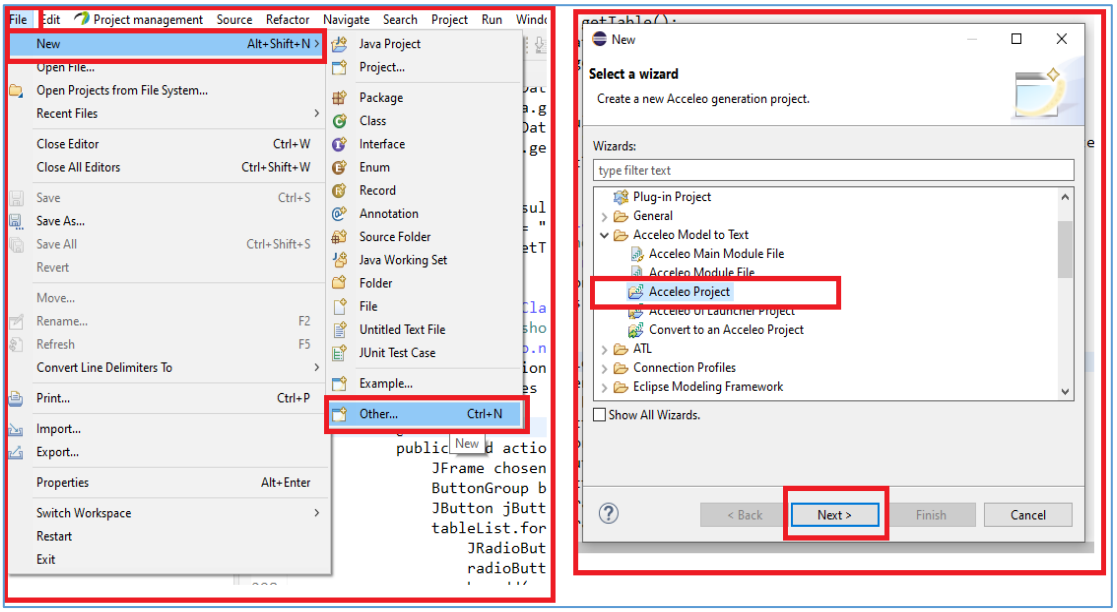
Thành phần	Cấu hình tối thiểu / phiên bản
Bộ xử lý (CPU)	1GHZ
Bộ nhớ (RAM)	1GB
Bộ nhớ trống	16GB

Về mặt cấu hình phần mềm cần cài đặt các ứng dụng và phần mềm bổ trợ sau, trong đó Acceleo được thêm vào như một trình cài thêm (plugin) trong Eclipse:

Bảng 4. 2. Cấu hình phần mềm

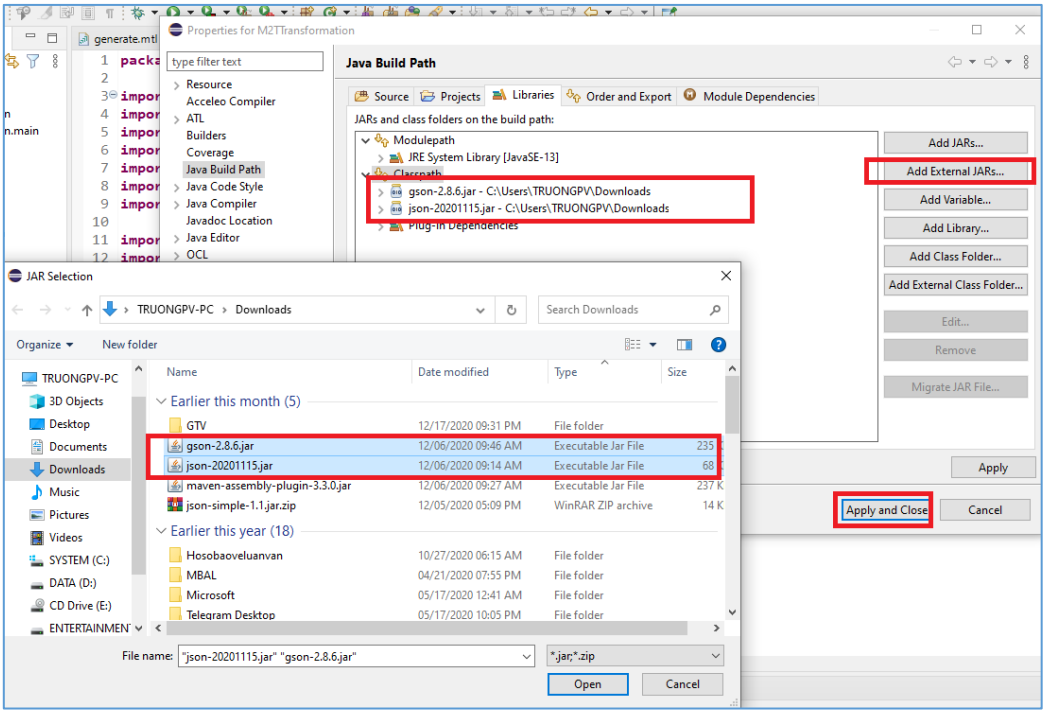
Thành phần	Cấu hình tối thiểu / phiên bản
Hệ điều hành	Windows / macOS / Linux
Java JDK	7u80
Eclipse	Eclipse 2020-06
Acceleo	3.1

Sau khi cài đặt các ứng dụng cần thiết, phần tiếp theo sẽ tạo mới dự án Acceleo trong trình Eclipse theo các bước như trong hình dưới đây:



Hình 4. 1. Tạo mới dự án Acceleo trong Eclipse.

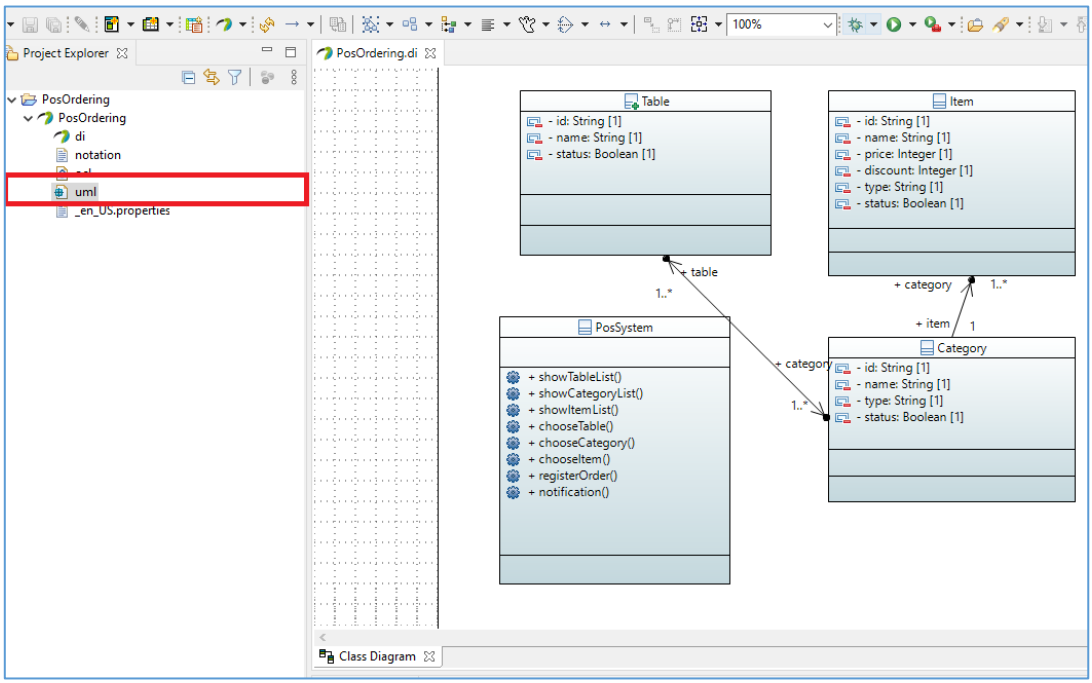
Sau khi tạo dự án Acceleo, điều tiếp theo cần phải làm là bổ sung thư viện hỗ trợ, chính là các gói .jar nhằm hỗ trợ việc nhập xuất dữ liệu mẫu. Cụ thể các gói cần thiết là: ***gson-2.8.6.jar*** và ***json-20201115.jar***, được nạp vào phần thư viện chung của dự án như được mô tả trong hình sau:



Hình 4. 2. Import các gói thư viện vào dự án Acceleo

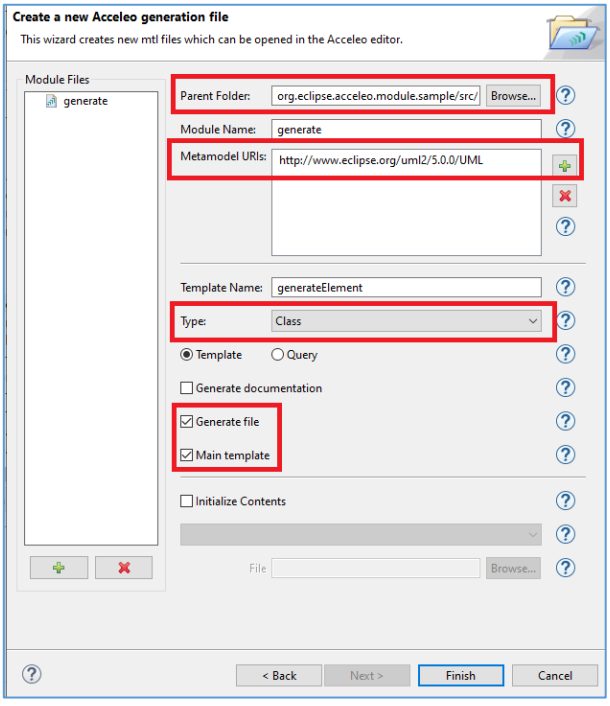
4.1.2. Dữ liệu đầu vào

Với việc chuyển đổi mô hình sang văn bản thì dữ liệu đầu vào quan trọng nhất chính là mô hình, cụ thể hơn là biểu đồ lớp mô hình hóa tính năng ***Ordering***. Sau khi vẽ biểu đồ lớp với các công cụ hỗ trợ, ta trích xuất được file UML như hình sau đây:



Hình 4. 3. Trích xuất file UML mô hình hóa hệ thống.

Sau khi tạo mới dự án Acceleo, cần đưa file UML như một mô hình đầu vào như sau:



Hình 4. 4. Import và cấu hình model đầu vào cho dự án Acceleo.

Chú ý sau khi import file UML, cần chọn đúng metamodel là UML và với Type là Class như trong hình.

4.1.3. Cách thức thực hiện

4.1.3.1. Cài đặt dữ liệu mẫu

Như đã trình bày trong chương trước, dữ liệu mẫu giúp cho mã nguồn sau khi chuyển đổi có thể thực thi với input là các dữ liệu cấu hình sẵn. Với bài toán áp dụng chức năng ***Ordering*** của hệ thống ***POS***, dữ liệu mẫu được giả lập dưới dạn JSON như sau:

Bảng 4. 3. Bảng dữ liệu mẫu dưới dạng JSON

Dữ liệu Table	{
	  "tableInformation": [
	    {
	      "id": "1",
	      "name": "Table number 1",
	      "status": true
	    },
	    {
	      "id": "2",
	      "name": "Table number 2",
	      "status": true
	    },
	    {
	      "id": "3",
	      "name": "Table number 3",
	      "status": true
	    },
	    {
	      "id": "4",
	      "name": "Table number 4",
	      "status": false
	    },
	    {
	      "id": "5",
	      "name": "Table number 5",
	      "status": true
	    }
	  ]
	}

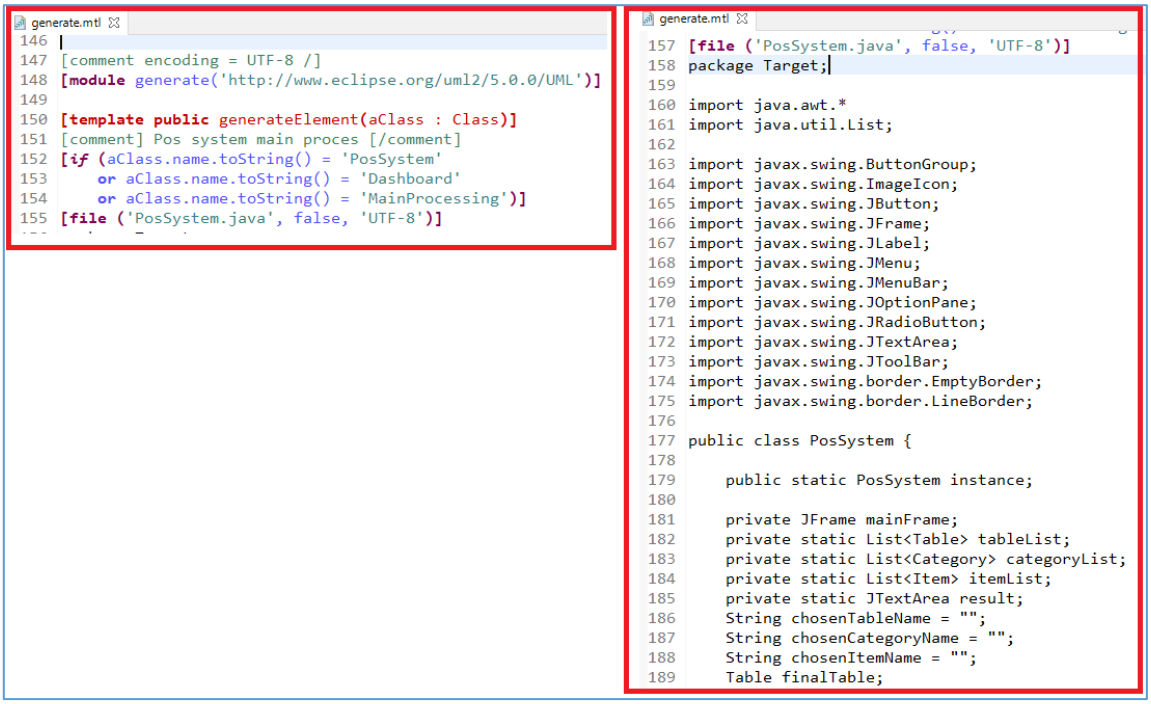
Dữ liệu Category	{
	"categoryInformation": [
	{
	"id": "1",
	"name": "Fish",
	"type": "Food",
	"status": true
	},
	{
	"id": "2",
	"name": "Chicken",
	"type": "Food",
	"status": true
	},
	{
	"id": "3",
	"name": "Pizza",
	"type": "Food",
	"status": true
	},
	{
	"id": "4",
	"name": "Beer",
	"type": "Drink",
	"status": true
	},
	{
	"id": "5",
	"name": "Water",
	"type": "Drink",
	"status": false
	}
	]

	}
	{
	"itemInformation": [
	{
	"id": "1",
	"name": "Shark",
	"price": 75000,
	"discount": 0,
	"type": "Fish",
	"status": true
	},
	{
	"id": "2",
	"name": "Ray",
	"price": 60000,
	"discount": 0,
	"type": "Fish",
	"status": true
	},
	{
	"id": "3",
	"name": "Turkey",
	"price": 50000,
	"discount": 10000,
	"type": "Chicken",
	"status": true
	},
	{
	"id": "4",
	"name": "Cock",
	"price": 25000,
	"discount": 5000,
	"type": "Chicken",
	"status": false
	},
	{
	"id": "5",
	"name": "Seafood pizza",
	"price": 30000,
	"discount": 0,
	"type": "Pizza",
	"status": true
	}
	]

Dữ liệu Item	}

4.1.3.2. Cài đặt mã nguồn Acceleo

Sau khi đã nạp dữ liệu mẫu, mô hình mẫu, luận văn sẽ tiến hành cài đặt mã nguồn Acceleo. Mã nguồn Acceleo gồm các phần cơ bản: module và template, chuyển đổi tĩnh và chuyển đổi mở rộng. Thứ nhất là module và template, phần này sẽ định nghĩa mô hình đầu vào là UML, mã nguồn sinh dưới dạng các Java class.



Hình 4. 5. Cài đặt module và template mã nguồn Acceleo.

Hình trên là module và template thực nghiệm, trong đó mã nguồn bên trái là điều kiện đọc vào từ mô hình, nếu tên lớp ứng với ***PosSystem***, ***Dashboard*** hoặc ***MainProcessing*** thì tương ứng với lớp mã nguồn được sinh ra là ***PosSystem***. Trong khi đó mã nguồn bên phải là mã nguồn Java được cài đặt sẵn của lớp PosSystem, gồm import các gói thư viện, khởi tạo các biến toàn cục… Tiếp đến sẽ cài đặt mã nguồn Acceleo cho các quy tắc chuyển đổi như đã trình bày trong Chương 3.

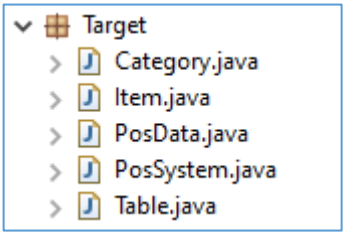
Lưu ý: do đặc thù ngôn ngữ nên khi chuyển đổi từ mô hình sang mã nguồn cần thay thế ký tự đặc biệt bằng một ký hiệu, chuỗi ký tự bất kỳ, và sau khi chuyển đổi sẽ thay thế ngược trở lại những ký tự đặc biệt đó. Ví dụ như ký tự "[]", trong Acceleo thì đây là môn câu lệnh mang tính cú pháp, sẽ không được sinh khi chuyển đổi. Luận văn sẽ thay thế "[]" thành chuỗi ký tự "`_SQUARE_BRACKET_`". Để thực thi mã nguồn sau khi chạy cũng cần import thư viện cần thiết: ***json-simple-1.1.jar***. Ngoài ra khi nạp mô hình (biểu đồ lớp) không thể nạp kiểu dữ liệu nguyên thủy, mà cần cấu hình thêm thư viện:

```
// BEGIN - Add UML2 resources (for Primitive type)
resourceSet
    .getPackageRegistry()
    .put(UMLPackage.eNS_URI, UMLPackage.eINSTANCE);
resourceSet
    .getResourceFactoryRegistry()
    .getExtensionToFactoryMap()
    .put(UMLResource.FILE_EXTENSION, UMLResource.Factory.INSTANCE);
Map uriMap = resourceSet
    .getURIConverter().getURIMap();
URI uri = URI
        .createURI("jar:file:/C:/eclipse/plugins"
                + "/org.eclipse.uml2.uml.resources_5.5.0.v20200302-1312.jar!/");
uriMap.put(URI.createURI(UMLResource.LIBRARIES_PATHMAP),
        uri.appendSegment("libraries").appendSegment(""));
uriMap.put(URI.createURI(UMLResource.METAMODELS_PATHMAP),
        uri.appendSegment("metamodels").appendSegment(""));
uriMap.put(URI.createURI(UMLResource.PROFILES_PATHMAP),
        uri.appendSegment("profiles").appendSegment(""));
// END - Add UML2 resources
```

Hình 4. 6. Cài đặt thư viện hỗ trợ biến kiểu nguyên thủy Java trong Acceleo.

4.2. Kết quả thực nghiệm

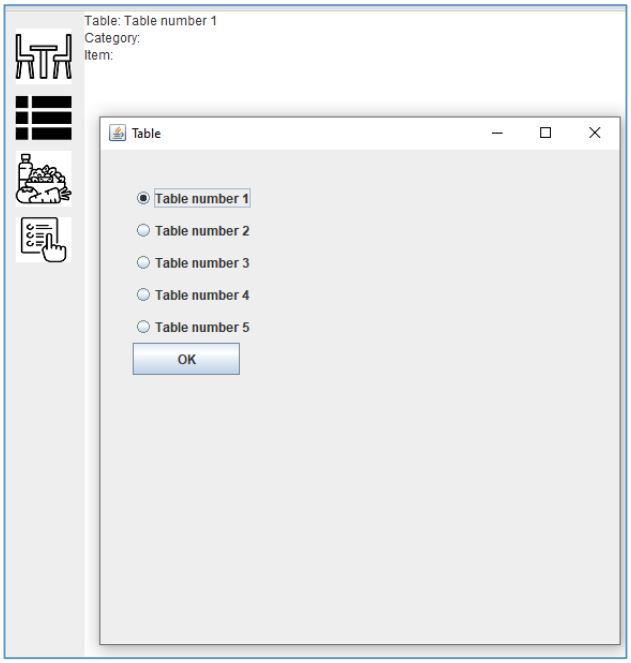
Sau khi chuẩn bị dữ liệu đầu vào gồm biểu đồ lớp, dữ liệu mẫu và cài đặt mã nguồn Acceleo, chúng ta tiến hành chạy mã nguồn với trình IDE. Kết quả cuối cùng sẽ bao gồm các Java class. Cụ thể sẽ có một file sinh dữ liệu mẫu PosData.java, ba file java tương ứng ba lớp Table.java, Category.java, Item.java, và một file mã nguồn chứa thân hàm logic được sinh ra PosSystem.java.



Hình 4. 7. Lớp mã nguồn Java được tạo ra.

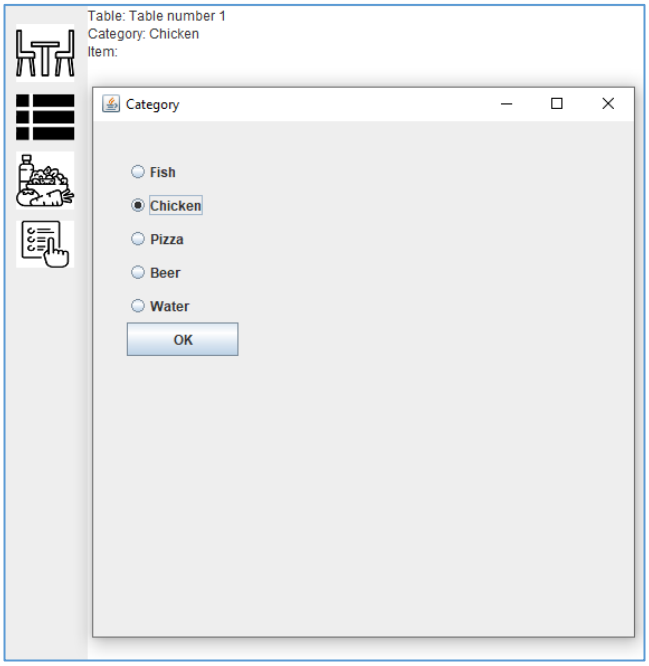
Kết quả template mẫu kết hợp mã nguồn sinh ra sẽ cho giao diện như sau:

- Hiển thị và chọn bàn:



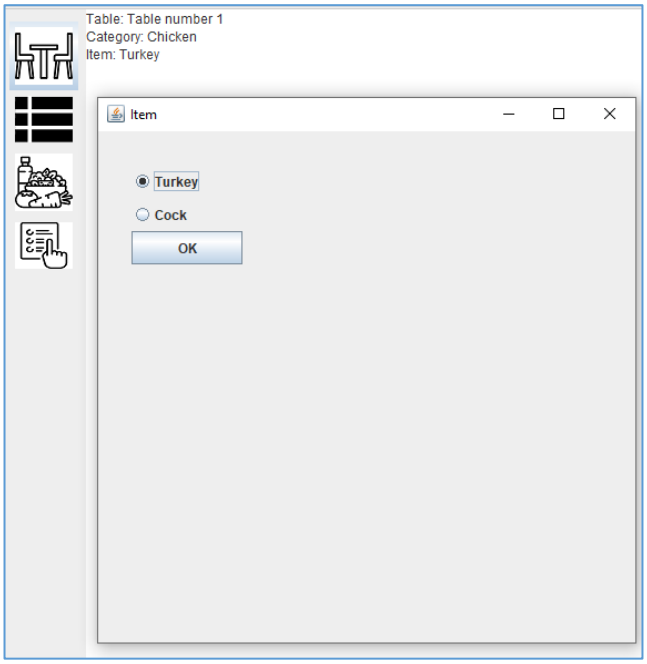
Hình 4. 8. Template showTable và chooseTable.

- Hiển thị và chọn danh mục



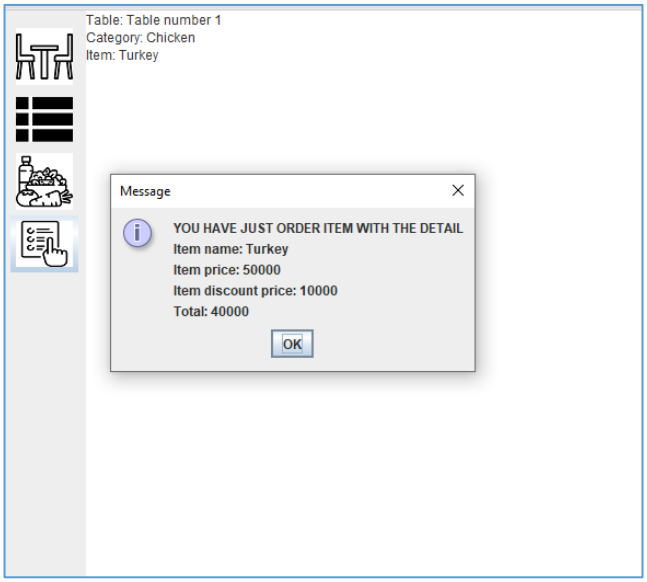
Hình 4. 9. Template showCategory và chooseCategory.

- Hiển thị và chọn item



Hình 4. 10. Template showItem và chooseItem.

- Hiển thị kết quả order:



Hình 4. 11. Template register và notification.

Sau khi chạy thực nghiệm, kết quả chuyển từ mô hình sang mã nguồn Java đúng với mong đợi và những gì đã cài đặt. Dưới đây luận văn sẽ chỉ ra các thông số mô tả phép đo khối lượng đầu vào va kết quả sau khi sinh mã nguồn. Thứ nhất, quá trình cài đặt và chạy thực nghiệm kéo dài khoảng 10 – 15 phút. Thứ hai, mã nguồn sau khi sinh có tổng cộng 549 dòng lệnh, tuy nhiên cũng cần phải bổ sung 135 dòng lệnh cho việc lấy dữ liệu mẫu, 34 dòng lệnh nhằm xử lý mã nguồn sau khi sinh đảm bảo do cú pháp của ngôn ngữ chuyển

đổi. Như vậy cần đến 169 dòng lệnh hỗ trợ chiếm 23.5% khối lượng công việ để đi đến kết quả cuối cùng.

Như vậy, sau khi nghiên cứu cơ sở lý thuyết và thực hiện luận văn này tác giả đã có cái nhìn bao quát về một lĩnh vực nghiên cứu mới liên quan đến mô hình nói chung và các phương pháp phát triển phần mềm định hướng mô hình nói riêng. Quá trình trong và sau khi thực nghiệm, có một vài phát sinh và lưu ý. Điều đầu tiên phát sinh khi thực nghiệm đó là cần bổ sung thêm thư viện các gói jar nhằm mục đích đọc các dữ liệu mẫu và xử lý chúng. Thứ hai cần phải lưu ý là việc thay đổi mô hình, thay đổi các phương thức và tên thuộc tính. Việc thay đổi này không chỉ là chỉnh sửa mã nguồn chuyển đổi phải thay đổi cấu hình, vì khi lưu lại mã nguồn Acceleo thì cấu hình về biến kiểu nguyên thủy (đã được cấu hình trước đó) sẽ bị đè bởi cấu hình mặc định, và cần phải sao chép lại, tốn khá nhiều thời gian. Cuối cùng, các file mã nguồn sau khi sinh tự động đặt chung hết trong một package, điều này mặc dù không ảnh hưởng tới việc thực thi nhưng cần cấu trúc lại sang các package hợp lý hơn, theo một số mô hình hiện có như MVC (Model – View – Controller)…

4.3. Tổng kết chương

Với kết quả thu được qua thực nghiệm, có thể thấy phần việc quan trọng và mất nhiều công sức nhất nằm ở quy tắc chuyển đổi mở rộng hơn là việc mô hình hóa hệ thống dưới dạng biểu đồ lớp. Tuy nhiên việc mô hình hóa phải đúng và đầy đủ sẽ cho kết quả tốt hơn. Nhìn chung việc sinh mã nguồn tự động đạt tỉ lệ tỉ tự động khá cao, tuy vậy vẫn cần lưu ý một vài điểm phát sinh bên cạnh việc triển khai các đoạn mã một cách thủ công để chương trình sau khi sinh mã nguồn có thể thực thi được.

KẾT LUẬN

Qua trình nghiên cứu, tìm hiểu và thực hiện đề tài "***Nghiên cứu kỹ thuật chuyển đổi mô hình sang văn bản và ứng dụng vào sinh mã nguồn Java***", luận văn có một số đóng góp nhất định và mở ra hướng phát triển trong tương lai. Về mặt kiến thức, luận văn đã trình bày được kiến thức tổng quát về phát triển hướng mô hình, trình bày các thuật ngữ trong phạm vi nghiên cứu về phát triển hướng mô hình. Trong đó nổi bật là kiến thức về mô hình và chuyển đổi mô hình, và từ đó áp dụng chuyển đổi mô hình vào sinh mã nguồn Java từ biểu đồ lớp class diagram qua thực tiễn sử dụng ngôn ngữ dựa trên mẫu Acceleo cùng bộ cài đặt dữ liệu mẫu. Về mặt thực nghiệm, luận văn đã định nghĩa các quy tắc chuyển đổi mô hình, xây dựng các luật chuyển áp dụng cho ví dụ cụ thể và sử dụng ngôn ngữ chuyển đổi mô hình để hiện thực hóa các quy tắc chuyển đổi đã nêu.

Qua quá trình nghiên cứu tìm hiểu và thực nghiệm, luận văn đã cho thấy hướng phát triển sinh mã nguồn từ mô hình biểu đồ lớp. Tuy không còn mới nhưng việc sinh mã nguồn Java đáp ứng chức năng cơ bản cho các phương thức của một mô hình biểu đồ lớp thì chưa có nghiên cứu cụ thể nào đưa ra và áp dụng. Bên cạnh đó, việc sinh mã nguồn mang một ý nghĩa giúp mở ra hướng nghiên cứu tiếp theo xoay quanh các quy tắc chuyển đổi, làm sao để cải thiện và nâng cao hơn nữa tính tự động trong các quy tắc chuyển đổi.

Tuy nhiên vẫn còn một số thiếu xót cũng như những hạn chế mà trong thời gian ngắn luận văn chưa thể thực hiện. Điều đầu tiên đó chính là tính đơn giản trong bài toán áp dụng, biểu đồ lớp mô hình hóa ví dụ còn chưa có nhiều phương thức khó, giống nhất với thực tế. Tiếp đến, việc sinh mã nguồn còn cần xây dựng mẫu và thiết kế, lập trình thêm về mặt giao diện, mặc dù giao diện còn cơ bản và chưa phù hợp về mặt trải nghiệm người dùng. Trong lương lai nếu có cơ hội, tác giả sẽ tìm hiểu sâu hơn về hướng mô hình và việc sinh mã nguồn Java và cải tiến chất lượng đầu ra nhằm tăng tỉ lệ chạy tự động mà ít phải cài đặt hay cấu hình thủ công. Nếu có thể sẽ cải tiến quy tắc chuyển đổi nhằm tăng khả năng tự động hơn nữa và áp dụng vào các bài toán khó hơn nữa nhằm khẳng định tính hiệu quả của phương pháp.

TÀI LIỆU THAM KHẢO

Tiếng Anh:

1. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, Addison-Wesley, 2007. *Compilers: Principles, Techniques, and Tools, 2nd ed*.
2. Cuadrado, J.S., Molina, J.G., Tortosa, M.M. In Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 158–172. Springer, Heidelberg (2006): *RubyTL: A Practical, Extensible Transformation Language*.
3. Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. 72(1-2):31–39, 2008. *ATL: A model transformation tool. Science of Computer Programming.*
4. Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Springer, 2006. *Fundamentals of Algebraic Graph Transformation.*
5. Jean Bézivin, 2005. *On the unification power of models: Software and System Modelling*.
6. Kolovos, D.S., Paige, R.F., Polack, F.A.C. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 46–60. Springer, Heidelberg (2008). *The Epsilon Transformation Language.*
7. Krzysztof Czarnecki and Simon Helsen. IBM Systems Journal, 45(3):621–645, 2006. *Feature-based survey of model transformation approaches.*
8. Marco Brambilla, Jordi Cabot, Manuel Wimmer: *Model-Driven Software Engineering in Practice, Second Edittion*.
9. Markus Völter. In Kevlin Henney and Dietmar Schütz, editors, Proc. of the 8th European Conference on Pattern Languages of Programs (EuroPLoP'03), pages 285–320, 2003. *A catalog of patterns for program generation.*
10. Nickel, U., Niere, J., Z¨undorf, (ICSE 2000), pp. 742–745 (2000). *The FUJABA environment. In: Int. Conf. on Software Engineering.*
11. "*OMG Unified Modeling Language*", [Online]. Available: http://www.uml.org [Accessed: Sept. 24, 2010].
12. Sébastien Gérard, Patrick Tessier, Bran Selic, Cedric Dumoulin. *Papyrus: A UML2 Tool for Domain-Specific Language Modeling Model-Based Engineering of Embedded Real-Time Systems.*
13. Shane Sendall and Wojtek Kozaczynski. IEEE Software, 20(5):42–45, 2003. *Model transformation: The heart and soul of model-driven software development*.

14. Taentzer, G. In: Pfaltz, J.L., Nagl, M., B¨ohlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 446–453. Springer, Heidelberg (2004). *AGG: A Graph Transformation Environment for Modeling and Validation of Software.*

15. Tom Mens and Pieter Van Gorp. Electronic Notes in Theoretical Computer Science, 152:125–142, 2006. *A taxonomy of model transformation.*

Tiếng Việt:

16. Nguyễn Huy Hoàng, (2014). *Thao tác mô hình trong phát triển hướng mô hình*.