Kỷ yếu Hội nghị Quốc gia lần thứ X về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR), Đà Nẵng, ngày 17-18/08/2017
DOI: 10.15625/vap.2017.000117

XÂY DỰNG VÀ THỬ NGHIỆM ĐÁNH GIÁ HIỆU NĂNG PHẦN MỀM DỰA TRÊN KỸ THUẬT BIẾN ĐỔI BIỂU ĐỒ TRÌNH TỰ THÀNH QUEUEING PETRI NETS

Vũ Văn Đốc, Nguyễn Trọng Bách, Huỳnh Quyết Thắng

Viện Công nghệ thông tin và Truyền thông, Trường Đại học Bách khoa Hà Nội

vvdoc@uneti.edu.vn, bachnt9x@gmail.com, thanghq@soict.hust.edu.vn

TÓM TẮT: *Đánh giá hiệu năng phần mềm là một trong những vấn đề được quan tâm. Một trong các hướng tiếp cận giải quyết vấn đề này là thông qua mô hình hóa và mô phỏng. Ngôn ngữ đặc tả Unified Modeling Language (UML) và Queueing Petri Net (QPN) có rất nhiều lợi thế trong việc đánh giá các thông số hiệu năng của phần mềm. Vì vậy, chuyển đổi từ UML sang mô hình QPN được xem là một trong những cách hiệu quả nhất để giải quyết các vấn đề đánh giá hiệu năng phần mềm. Tuy nhiên, sự chuyển đổi này cũng có rất nhiều khó khăn do sự khác biệt giữa ngữ nghĩa của hai mô hình. Trong bài báo này, chúng tôi trình bày một phương pháp sử dụng mô hình trung gian bao gồm các mô đun được trừu tượng từ các yếu tố trong UML và QPN theo chức năng và cấu trúc. Sự chuyển đổi bao gồm ba bước: mở rộng mô hình UML sang mô hình XMI, chuyển đổi chúng sang dạng XML của QPN, hiện tại mới thử nghiệm với biểu đồ trình tự. Các mô hình QPN được chuyển đổi có thể hoạt động chính xác và cung cấp cơ sở để đánh giá hiệu năng phần mềm dựa vào công cụ QPME (Queueing Petri net Modeling Environment).*

Từ khóa*: Đánh giá hiệu năng phần mềm, Queueing Petri Nets, Unified Modeling Language.*

I. GIỚI THIỆU

A. Hiệu năng phần mềm

Theo tiêu chuẩn ISO/IEC 25010, chất lượng phần mềm được định nghĩa là mức độ mà phần mềm đáp ứng các yêu cầu với đặc tính cụ thể. Một trong những đặc điểm chính ảnh hưởng đến chất lượng phần mềm đó là hiệu năng phần mềm, được chia thành ba đặc tính riêng: Hành vi thời gian, sử dụng tài nguyên và năng lực xử lý. Các định nghĩa của ba đặc tính này có thể đóng vai trò như một định nghĩa về hiệu năng phần mềm. Hiệu năng phần mềm bao gồm [1]: Hành vi thời gian: Bao gồm *thời gian đáp ứng (response time), thời gian xử lý (processing times) và Thông lượng (Throughput);* Sử dụng tài nguyên (*Resource utilization*): Lượng tài nguyên cần để đáp ứng yêu cầu của phần mềm; Công suất (*Capacity*): Là khả năng đáp ứng tối đa của phần mềm.

Sự phát triển của phần mềm quy mô lớn thường là theo vòng đời phát triển phần mềm [2], nhưng hiệu suất chỉ được đánh giá chính xác khi phần mềm hoàn thành. Điều đó có nghĩa là khi hiệu suất không đáp ứng được yêu cầu thì toàn bộ hệ thống bị thất bại. Do đó, việc xây dựng cơ chế đánh giá hiệu năng trong chu kỳ phát triển phần mềm là rất quan trọng để đảm bảo sự phát triển của các dự án phần mềm.

Trước đây, việc đánh giá hiệu năng phần mềm dựa trên ngôn ngữ mô hình hóa thống nhất UML là vấn đề rất khó khăn và phức tạp. Tuy nhiên, từ những năm 2002, nhiều nhà khoa học bắt đầu tiếp cận các hướng nghiên cứu đánh giá hiệu suất của phần mềm thông qua mạng hàng đợi, Petri net [3], mạng Generalized Stochastic Petri Nets [4] hay color Petri Net [2]. Đầu tiên có thể kể ra đó là nhóm ba tác giả Simona Bernardi, Susanna Donatelli, Jose' Merseguer [4] đưa ra phương pháp đánh giá hiệu năng dựa vào việc chuyển đổi tự động từ biểu đồ trình tự UML sang mạng Generalized Stochastic Petri Nets hay theo tác giả Tony Spiteri Staines [3] đưa ra phương pháp đánh giá hiệu năng phần mềm dựa trên chuyển đổi các biểu đồ trình tự UML thành Petri Nets. Mặc dù các hướng tiếp cận này là đáng tin cậy và chính xác nhưng sẽ rất khó khăn do sự bùng nổ của không gian trạng thái khi hệ thống đủ lớn. Theo hướng tiếp cận của hai tác giả ZHU Lian-Zhang, KONG Fan-Sheng [2] đó là mở rộng mô hình UML cho các mô hình hiệu suất và sau đó chuyển đổi sang mô hình CPN được xem là một trong những cách hiệu quả để giải quyết các vấn đề đánh giá hiệu năng phần mềm. Tuy nhiên, nhược điểm lớn nhất của hướng tiếp cận này đó là sự chuyển đổi cần rất nhiều sự can thiệp bằng tay vì sự khác biệt giữa ngữ nghĩa của hai mô hình. Vì thế trong bài báo này, chúng tôi đề xuất một phương pháp tiếp cận đánh giá hiệu năng phần mềm bằng phép biến đổi từ biểu đồ tuần tự UML sang mô hình trung gian XMI và sau đó chuyển sang mô hình QPN, giải quyết được vấn đề chuyển đổi thủ công. Mô hình QPN sau khi đã được chuyển đổi từ UML có thể được sử dụng trực tiếp để đánh giá hiệu năng nhờ vào công cụ QPME sẽ hạn chế được vấn đề bùng nổ không gian trạng thái của các phương pháp khác.

B. Ngôn ngữ mô hình UML và QPN

B.1. Ngôn ngữ mô hình hóa UML

UML là một ngôn ngữ mô hình gồm các ký hiệu đồ họa mà các phương pháp hướng đối tượng sử dụng để thiết kế các hệ thống thông tin một cách nhanh chóng [1]. UML sử dụng một hệ thống ký hiệu thống nhất biểu diễn các phần tử mô hình. Tập hợp các phần tử mô hình tạo thành các biểu đồ UML. Trong nội dung bài báo này, chúng tôi sẽ tập trung nghiên cứu về biểu đồ trình tự do biểu đồ này mô tả sự tương tác của các đối tượng theo trình tự về thời gian, có

sự liên kết chặt chẽ với biểu đồ lớp và mỗi biểu đồ trình tự mô tả một tình huống xử lý. Biểu đồ trình tự là một trong những biểu đồ tương tác UML phổ biến, được sử dụng để xác định và chỉ rõ vai trò của các đối tượng tham gia vào luồng sự kiện của use-case [5]. Đây là một loại biểu đồ mô tả mô hình tương tác giữa các đối tượng, trong đó nhấn mạnh vào trình tự thời gian của các thông điệp trao đổi giữa các đối tượng đó. Các đặc tả biểu đồ trình tự trong UML 2.5 đã được viết lại từ các phiên bản trước đó, làm nó dễ đọc hơn nhờ loại bỏ dư thừa [5]. Biểu đồ trình tự trong UML 2.5 chỉ ra: Các đối tượng tham gia vào tương tác, thời gian sống của các đối tượng, trình tự các thông điệp được trao đổi, mỗi biểu đồ trình tự là sự kết hợp của một tập các nút (node) và các cạnh/đường đi (edgs/path).

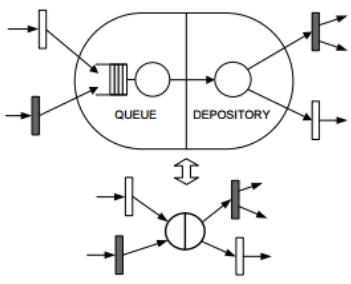
B.2. Queueing Petri nets (QPN)

Queueing Petri Nets (QPN) là sự kết hợp từ một số mở rộng khác nhau của Petri Nets thông thường (PN) gồm Petri Nets màu(CPN) và Generalized Stochastic Petri Nets (GSPN) [6], đồng thời bổ sung thêm một vài yếu tố khác. Mỗi PN là một đồ thị hai phía, có hướng, bao gồm một tập các *place* và một tập các *transition*. Mỗi cung (*arc*) của đồ thị chỉ kết nối một *place* và một *transition* [6]. Nếu một cung đi từ *place* p vào *transition* t thì *place* p được gọi là *place* vào của *transition* t và *transition* t được gọi là *transition* ra của *place* p. Nếu một cung đi từ *transition* t vào *place* p thì *place* p được gọi là *place* ra của *transition* t và *transition* t được gọi là *transition* vào của *place* p. Mỗi cung được gán một số nguyên dương gọi là trọng số (*weight*) của cung. Mỗi *place* chứa một lượng thẻ (*token)* nhất định. Khi mọi *place* vào của *transition* t chứa số thẻ không nhỏ hơn trọng số của cung nối nó với *transition* t thì *transition* này ở trạng thái sẵn sàng kích hoạt (*enabled*), và được gọi là *transition* sẵn sàng kích hoạt. Một *transition* sẵn sàng kích hoạt có thể kích hoạt (*fire*), khi đó nó hủy các thẻ có trong các *place* vào của nó và tạo ra các thẻ mới trong *place* ra của nó. Số lượng thẻ được hủy và tạo được xác định bởi trọng số của các cung kết nối.

Nếu PN chỉ có một loại thẻ duy nhất thì CPN đưa vào yếu tố màu (*color*) để phân loại thẻ [7]. CPN cho phép một hàm màu (*color function*) ánh xạ từ một tập các màu tới một *place*, xác định các loại thẻ có thể có trong *place* đó. Trong CPN, một *transition* sẵn sàng kích hoạt có thể có nhiều khả năng để kích hoạt, mỗi khả năng như thế được gọi là một *mode*. Các mode này có trọng số kích hoạt (*firing weight*) là tỷ lệ mà *mode* được chọn khi nhiều *mode* sẵn sàng kích hoạt.

Khác với PN, GSPN phân transition thành hai loại: *timed transtion* và *immediate transition*. Khi ở trạng thái sẵn sàng kích hoạt, một *immediate transition* kích hoạt trong thời gian 0 (*zero time*). Nếu có nhiều *immediate transition* ở trạng thái sẵn sàng kích hoạt tại cùng một thời điểm thì việc lựa chọn kích hoạt sẽ phụ thuộc vào trọng số kích hoạt (xác suất) gắn cho các *transition* đó. *Immediate transition* có được ưu tiên kích hoạt trước so với *timed transition*. Một *timed transition* kích hoạt sau một khoảng thời gian bằng phân phối mũ của độ trễ kích hoạt (*firing delay*) gắn với *transition* đó [6].

Kết hợp CPN và GSPN hình thành CGSPN. QPN thêm yếu tố hàng đợi vào các *place* trong CGSPN tạo ra *place* hàng đợi. Mỗi *place* hàng đợi bao gồm hai thành phần: thành phần hàng đợi cho phép các thẻ được đặt trong hàng đợi để chờ thực hiện dịch vụ và thành phần *depository* lưu giữ các thẻ đã hoàn thành dịch vụ của chúng ở hàng đợi [7, 8]. Hình 1 mô tả trực quan hình ảnh của *place* hàng đợi trong QPN.



Hình 1. Ký pháp và mô tả place hàng đợi trong QPN

Sau khi được kích hoạt bởi một *transition* vào của một *place* hàng đợi nào đó, các thẻ sẽ được thêm vào thành phần hàng đợi theo các chiến lược điều phối hàng đợi. Các thẻ trong hàng đợi không có sẵn cho các *transition* ra của *place* hàng đợi. Sau khi hoàn thành dịch vụ, một thẻ ngay lập tức được chuyển sang cho *depository*. Tại đây, các thẻ sẵn có cho các *transition* ra của *place* hàng đợi. *Place* hàng đợi loại này được gọi là *timed queueing place*. Ngoài ra, trong QPN còn có *immediate queueing place* cho phép mô tả việc điều phối thuần túy, các thẻ tại đây được xem như có thể phục vụ ngay lập tức. Điều phối cho những *place* này có độ ưu tiên cao hơn khi điều phối *timed queueing place* [6].

Để xây dựng mô hình QPN chúng tôi lựa chọn phần mềm QPME2.1 (32 bit) và chạy thử nghiệm mô hình QPN bằng SimQPN đã có trong phần mềm này.

C. Đánh giá hiệu năng hệ thống phần mềm

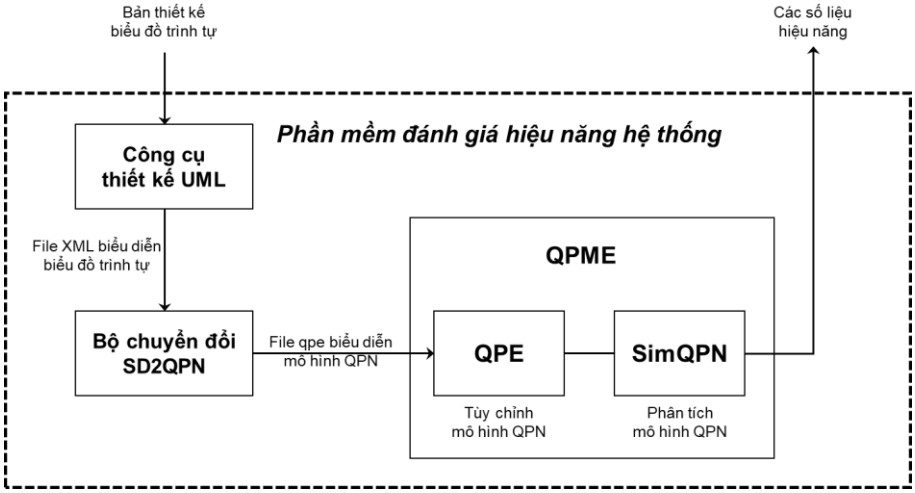
Trong phần này chúng tôi trình bày về yêu cầu đánh giá hiệu năng hệ thống phần mềm. Theo các đặc tính của vòng đời hiện tại trong kỹ thuật phần mềm, quá trình phát triển phần mềm nói chung tạo ra các tài liệu UML ở mọi giai đoạn. Các mô hình UML cùng với những điều kiện ràng buộc được tạo ra trong chu kỳ thiết kế phần mềm hệ thống.

Việc đánh giá hiệu năng chủ yếu là dựa trên các đặc tính của các mô hình UML, phân chia quá trình thực hiện thành các mô đun ... Phương pháp này có lợi điểm tính đơn giản và có kết quả duy nhất, nhưng việc thực hiện không phải lúc nào cũng dễ dàng diễn tả quá trình sử dụng chuỗi, các nhánh và vòng lặp hay các quá trình song song [4]. Để giải quyết vấn đề này, QPN được coi là phương pháp hiệu quả nhất để đánh giá hiệu năng phần mềm [6]. Tuy nhiên, do sự khác biệt về ngữ nghĩa giữa QPN và UML, rất khó để tạo ra sự tương ứng giữa chúng và luôn đòi hỏi nhiều sự can thiệp thủ công.

Như vậy cần thiết phải xây dựng mô hình trung gian để có thể giải quyết sự khác biệt về ngữ nghĩa giữa UML và QPN và đặt nền móng cho việc chuyển đổi tự động từ UML sang QPN. Điều này rất có ý nghĩa để đánh giá được hiệu năng của phần mềm chính xác và sẵn có trước thời gian trong chu trình phát triển phần mềm quy mô lớn.

II. XÂY DỰNG MÔ HÌNH

Sơ đồ khối hệ thống



Hình 2. Sơ đồ khối hệ thống

Hình 2 minh họa khối mô hình đánh giá hiệu năng phần mềm mà chúng tôi đề xuất xây dựng bao gồm các thành phần: Công cụ thiết kế UML, Bộ chuyển đổi SD2QPN, QPME

Khối *Phần mềm đánh giá hiệu năng hệ thống* là khối lớn nhất, nhận đầu vào là một bản thiết kế UML (ở đây chúng tôi quan tâm tới biểu đồ trình tự nên sẽ là bản thiết kế biểu đồ trình tự) và đầu ra là các số liệu hiệu năng qua quá trình mô phỏng và phân tích. Khối này bao gồm 3 khối con là khối *Công cụ thiết kế UML*, *Bộ chuyển đổi SD2QPN* và *QPME*.

Khối *Công cụ thiết kế UML* được dùng để tạo các bản vẽ biểu đồ trình tự và sinh ra file XML biểu diễn biểu đồ đó. File XML này qua *Bộ chuyển đổi SD2QPN* (*Sequence Diagram to Queueing Petri Nets)* cho ra file qpe – là file XML biểu diễn mô hình QPN.

Khối *QPME* thực chất là công cụ QPME [6] dùng để đánh giá hiệu năng các mô hình QPN. QPME bao gồm hai thành phần chính là QPE (QPN Editor) cung cấp giao diện đồ họa để thiết kế cũng như tùy chỉnh mô hình QPN và SimQPN (Simulator for QPNs) là bộ mô phỏng để phân tích mô hình QPN. Tương ứng với hai thành phần này là hai khối con *QPE* và *SimQPN* trong khối *QPME*.

Khối *QPE* sử dụng file qpe đã sinh ra được từ *Bộ chuyển đổi SD2QPN* để hiển thị mô hình QPN. Các nhà phát triển và chuyên gia có thể tiếp tục chỉnh sửa để hoàn thiện QPN.

Khối *SimQPN* sử dụng mô hình QPN cuối cùng và các cấu hình thử nghiệm do người dùng thiết lập để phân tích mô hình QPN. Kết quả sau khi quá trình phân tích này là các số liệu hiệu năng công cụ QPME đánh giá được và cũng là đầu ra của khối *Phần mềm đánh giá hiệu năng hệ thống* đã đề cập tới ở trên.

Trong phần tiếp theo, chúng tôi xin giới thiệu một số phép biến đổi cơ bản trong *Bộ chuyển đổi SD2QPN*.

Một số phép biến đổi cơ bản trong bộ chuyển đổi SD2QPN

Mỗi *frame* để vẽ các thành phần trong biểu đồ trình tự được ánh xạ thành một không gian để biểu diễn QPN.

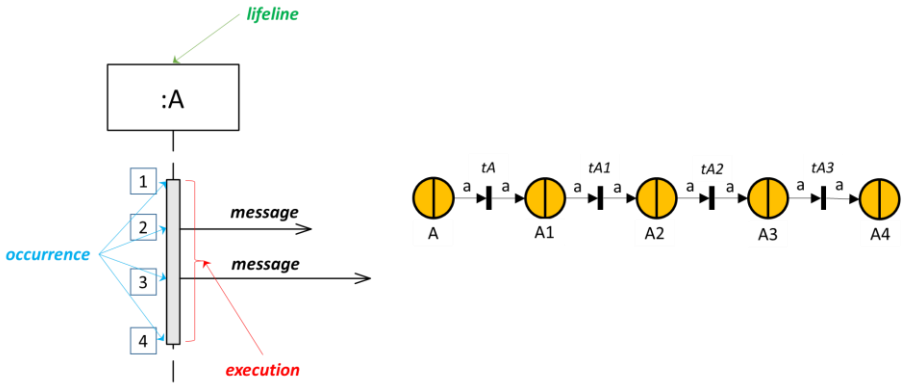
Mỗi *lifeline* (actor, object) trong *frame* được ánh xạ thành một queueing place (viết tắt q-place) trong không gian của QPN. Q-place này chứa *main-color* (color đặc trưng) cho *lifeline* tương ứng với nó.

Mỗi *occurrence* là một điểm trên *lifeline* biểu diễn sự kiện bắt đầu hoặc kết thúc một thông điệp hay bắt đầu hoặc kết thúc một *execution*. Ngữ nghĩa của biểu đồ trình tự được xác định thông qua dãy các *occurrence*. Mỗi

occurrence chỉ xuất hiện trên chính xác một *lifeline* và các *occurrence* của một *lifeline* được sắp xếp theo thứ tự thời gian từ trên xuống dưới. Khi ánh xạ sang QPN, mỗi *occurrence* trên một *lifeline* được chuyển thành một q-place dùng chung Queue với q-place ánh xạ từ *lifeline* đó và chứa main-color của *lifeline*. (Tên của *occurrence* tạm quy ước là tên của *lifeline* + số thứ tự của *occurrence* trên *lifeline* theo thứ tự từ trên xuống dưới). Hai *occurrence* trên hai *lifeline* khác nhau sẽ ánh xạ sang hai q-place không cùng main-color của *lifeline*. Q-place chứa main-color ứng với *lifeline* nào thì sẽ là ánh xạ từ một *occurrence* trên lifeline đó.

Một transition được sinh ra nối q-place ánh xạ từ *lifeline* sang q-place chuyển đổi từ *occurrence* đầu tiên trên *lifeline* đó. Một hoặc một số transition khác được sinh ra để chuyển từ q-place ứng với *occurrencei* tới q-place ứng với *occurrence*$^{i+1}$.Mục đích của những transition này là để lan truyền main-color ứng với *lifeline*. Ở thời điểm ban đầu, chỉ q-place ứng với *lifeline* được khởi tạo main-color, còn các q-place ánh xạ từ các *occurrence* trên *lifeline* không được khởi tạo main-color này mà chỉ có khả năng chứa nó. Qua các transition ở trên, main-color di chuyển qua từng q-place ứng với các *occurrence*. Như vậy, *occurrence*$^{i+1}$ chỉ đạt tới được khi đã có occurrence^{i}.

Execution biểu diễn một giai đoạn hoạt động trên *lifeline*, được xác định bởi hai *occurrence* đầu và cuối, được ánh xạ sang QPN giống như đã làm với *occurrence*.

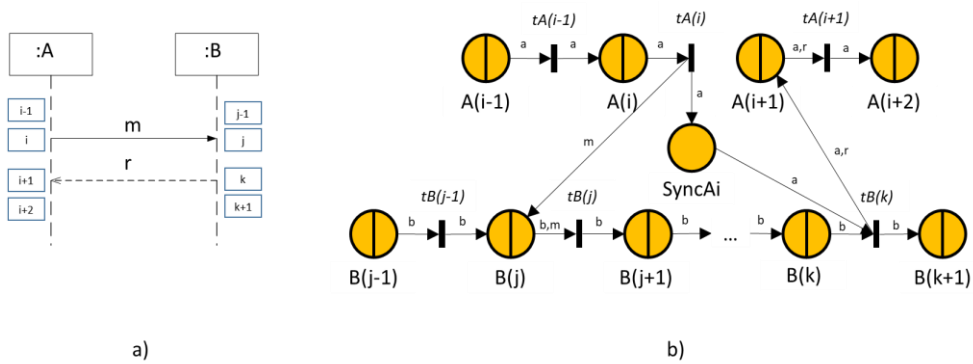


Hình 3. Các phần tử cơ bản trong biểu đồ trình tự và ánh xạ của chúng sang QPN

Hình 3 minh họa các phần tử *lifeline, execution, occurrence* trong biểu đồ trình tự. A được hiểu như một *lifeline* với 4 *occurrence* là A1, A2, A3, A4 trong đó A1, A4 là các *occurrence* đánh dấu hai điểm đầu và cuối của một *execution*, còn A2, A3 là các *occurrence* biểu diễn điểm bắt đầu thông điệp. *Lifeline* A và các *occurrence* A1, A2, A3, A4 được ánh xạ thành các q-place A, A1, A2, A3, A4 chứa main-color a đặc trưng cho *lifeline* A. Ở thời điểm ban đầu, q-place A được khởi tạo 1 color a, các q-place A1, A2, A3, A4 không được khởi tạo color a. Các transition tA, tA1, tA2, tA3 được sinh thêm làm nhiệm vụ lan truyền color a từ q-place A tới các q-place sau nó.

Tiếp theo, chúng tôi xin trình bày phép biến đổi biểu đồ trình tự có chứa lời gọi đồng bộ. Lời gọi đồng bộ từ một *lifeline* là một thông điệp mà khi gửi đi thì ngắt quá trình hoạt động của *lifeline* đó lại và phải chờ cho đến khi nhận được thông điệp phản hồi thì mới có thể quay lại hoạt động bình thường. Đối lập với lời gọi đồng bộ là lời gọi không đồng bộ, thông điệp được gửi đi không ngắt quá trình hoạt động của *lifeline* và *lifeline* có thể thực hiện các công việc khác.

Giả sử tại *occurrence* A(i), A gửi lời gọi đồng bộ m tới B với *occurrence* tương ứng trên B là B(j). Sau một khoảng thời gian khác 0, B từ *occurrence* B(j) tới *occurrence* B(k), bắt đầu gửi thông điệp phản hồi r cho A và đến A tại *occurrence* A(i+1). Hình 4a minh họa cho trường hợp này.



Hình 4. a) Biểu đồ trình tự có lời gọi đồng bộ. b) Biến đổi QPN tương ứng

Hình 4b biểu diễn một phần mô hình QPN liên quan đến lời gọi đồng bộ qua phép biến đổi SD2QPN.

Các *occurrence* trên *lifeline*A và B được chuyển thành các q-place chứa main-colora và b tương ứng.

Transition tA(i-1) hủy 1 color a từ q-place A(i-1) và tạo mới 1 color a cho q-place A(i). Transition tA(i) hủy 1 color a trong q-place A(i) và tạo mới 1 color m cho q-place B(j) thể hiện việc gửi thông điệp m từ A sang B, đồng thời sinh ra 1 color a cho ordinary place SyncAi được sinh thêm. Lúc này, A không thể gửi bất kỳ một thông điệp nào khác mà phải chờ đợi thông điệp phản hồi. (Nếu A gửi được một thông điệp khác sau A(i) thì rõ ràng tồn tại một q-place ứng với *occurrence* sau A(i) chứa color a, điều này không xảy ra vì color a lúc này nằm trong SyncAi).

Transition tB(j) chỉ kích hoạt được khi q-place B(j) có cả color m và color b. Q-place B(j) nhận được color b khi transition tB(j-1) kích hoạt. Việc transition tB(j-1) kích hoạt trước hay sau transition tA(i-1) không quan trọng vì nếu tB(j-1) kích hoạt trước thì đến tB(j) cũng phải dừng lại để đợi color m. Transition tB(j) hủy 1 color m và 1 color b đang có trong q-place B(j) rồi sinh ra 1 color b mới cho q-place B(j+1). Lúc này B có thể làm gì đó cho đến *occurrence* B(k).

Transition tB(k) hủy 1 color b trong q-place B(k) và 1 color a trong ordinary place SyncAi để sinh ra 1 color a và 1 color r mới cho q-place A(i+1) thể hiện sự đồng bộ và gửi thông điệp phản hồi tới A, đồng thời cũng sinh ra 1 color b mới gửi tới q-place B(k+1) để lan truyền.

Khi q-place A(i+1) có cả color r và color a, transition tA(i+1) có thể kích hoạt để hủy 2 color đó trong q-place A(i+1) và tạo mới color a cho q-place A(i+2).

Đối với lời gọi không đồng bộ, không cần thiết phải sinh thêm ordinary place SyncAi như ở trên. Khi đó, khi transition tA(i) kích hoạt, ngoài việc hủy 1 color a ở q-place A(i) và sinh mới 1 color m tới q-place B(j) thì còn tạo ra 1 color a khác gửi đến q-place A(i+1) để lan truyền ngay. Lúc này, A có thể gửi thông điệp khác mà không cần chờ phản hồi như trường hợp trước.

Ngoài các phép biến đổi kể trên, *Bộ chuyển đổi SD2QPN* có thể hỗ trợ nhiều phần tử khác trong biểu đồ trình tự như các loại thông điệp tạo, hủy, các phân đoạn tổ hợp (*combined fragment*) hay *interaction use*.

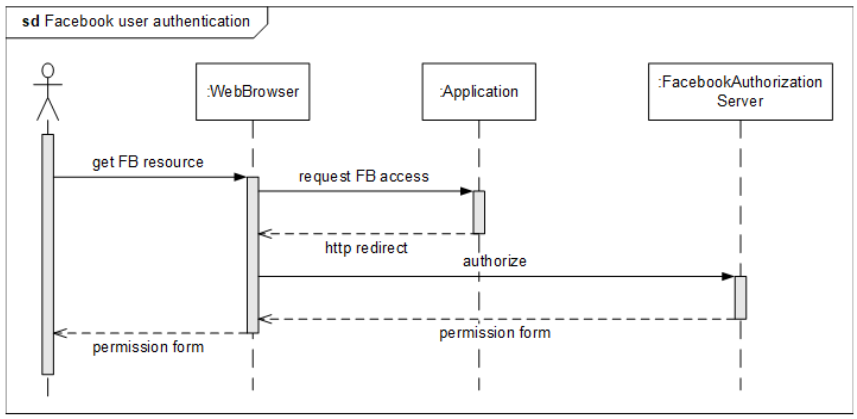
Theo cách chuyển đổi trên, biểu đồ trình tự UML được biểu diễn bằng 1 tập các occurrence, mỗi occurrence được chuyển đổi thành chỉ một q-place có một main-color, các q-place có cùng main-color sẽ cùng 1 lifeline, thứ tự các occurrence trên 1 lifeline dựa theo chiều của các transitions nối q-place tương ứng. Q-place nào được khởi tạo main-color thì là lifeline, ngược lại nếu q-place không được khởi tạo thì là occurrence trên lifeline. Các message thì được dựa vào transition và color ứng với message…. Chính vì vậy việc chuyển đổi các các thành phần từ biểu đồ trình tự UML sang QPN tương ứng sẽ không làm ảnh hưởng đến việc đánh giá các thông số hiệu năng phần mềm như *thời gian đáp ứng, thời gian xử lý, thông lượng, sử dụng tài nguyên* hay *công suất*.

III. THỬ NGHIỆM MÔ HÌNH

Trong phần này, chúng tôi áp dụng khối mô hình đã đề xuất trong phần II cho một phần hệ thống xác thực người dùng của Facebook. Phần III-A đưa ra biểu đồ trình tự cho hệ thống này. Chúng tôi chọn hệ thống phần mềm xác thực người dùng Facebook để thử nghiệm mô hình vì theo [13], biểu đồ trình tự của hệ thống được xác định và chỉ rõ vai trò của các đối tượng tham gia vào luồng sự kiện, mô tả mô hình tương tác giữa các đối tượng một cách rành mạch, trong đó nhấn mạnh vào trình tự thời gian của các thông điệp trao đổi giữa các đối tượng đó. Mặt khác hệ thống diễn tả quá trình sử dụng chuỗi, vòng lặp và các quá trình song song. Do vậy có thể áp dụng để chuyển đổi biểu đồ tuần tự UML sang mô hình trung gian XMI và sau đó chuyển sang mô hình QPN để đánh giá hiệu năng hệ thống.

Biểu đồ trình tự cho hệ thống xác thực người dùng của Facebook

Hệ thống xác thực người dùng của Facebook là một trong những ví dụ về biểu đồ trình tự được đưa ra trong [13] mô tả cách người dùng Facebook được xác thực trên ứng dụng Web để truy cập vào tài nguyên Facebook của mình. Hình 5 minh họa một phần biểu đồ trong hệ thống này.

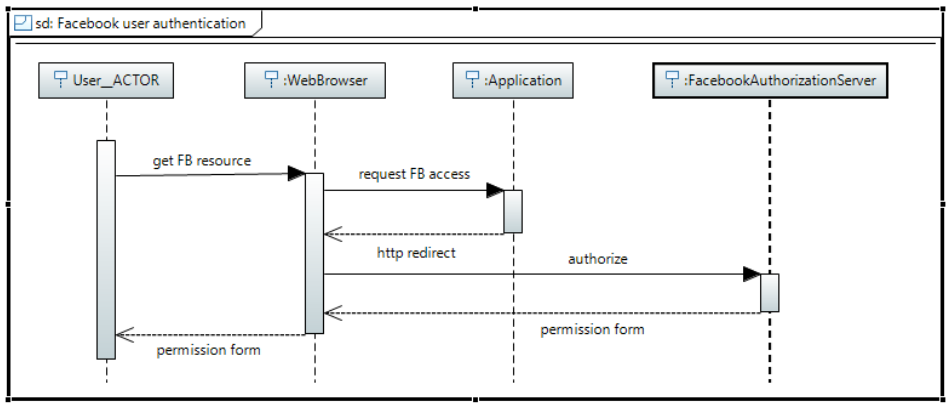


Hình 5. Biểu đồ trình tự cho hệ thống xác thực người dùng của Facebook

Trong Hình 5, Actor là người dùng, gửi thông điệp đồng bộ *get FB resource* tới trình duyệt web (*WebBrowser*) yêu cầu lấy tài nguyên Facebook. Trình duyệt Web sau khi nhận được thông điệp này, sẽ gửi thông điệp đồng bộ *request FB access* để yêu cầu truy cập tới ứng dụng của Facebook (*Application*). Ứng dụng này trả về cho trình duyệt web thông điệp phản hồi *http redirect* cho phép trình duyệt web giao tiếp với máy chủ FB. Sau đó, trình duyệt web gửi đi thông điệp *authorize* tới máy chủ xác thực Facebook (*FacebookAuthorizationServer*) cho biết cần xác thực người dùng. Máy chủ nhận được thông điệp này, tiến hành xử lý và trả về thông điệp *permission form* cho trình duyệt web để trình duyệt web trả lại cho người dùng. Lúc này, người dùng nhận được một biểu mẫu cấp phép. Các công việc tiếp theo trong hệ thống này có thể xem thêm trong [13]. Chúng tôi chỉ sử dụng đoạn biểu đồ này làm ví dụ minh họa việc áp dụng khối mô hình sẽ được trình bày ngay sau đây.

Áp dụng khối mô hình đề xuất

Chúng tôi sử dụng phần mềm Papyrus (phiên bản Neon – 4.6.0 [14]) làm khối *Công cụ thiết kế UML* để xây dựng lại biểu đồ trình tự trong Hình 5. Kết quả thu được ở bước này như Hình 6 Lifeline User_ACTOR được sử dụng để thay cho Actor trong Hình 6.



Hình 6. Biểu đồ trình tự cho hệ thống xác thực người dùng Facebook vẽ trên Papyrus

Từ biểu đồ trình tự thiết kế được trên Papyrus, một file XMI (Hình 7) mô tả biểu đồ này được trích xuất ra.

```
<uml:Model xmi:version="20131001" xmlns:xmi="http://www.omg.org/spec/XMI/20131001" xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML" xmi:id="_vcIWwFD2Eee6SP4k2omS5Q" name="RootElemer
    <packagedElement xmi:type="uml:Interaction" xmi:id="_vd5RUFD2Eee6SP4k2omS5Q" name="Facebook user authentication">
        <lifeline xmi:type="uml:Lifeline" xmi:id="_ITxlwFD3Eee6SP4k2omS5Q" name="User_ACTOR" coveredBy="_ln0HkFD3Eee6SP4k2omS5Q _ln0HklD3Eee6SP4k2omS5Q _ln0HkVD3Eee6SP4k2omS5Q _4ff_
        <lifeline xmi:type="uml:Lifeline" xmi:id="_NGu-cFD3Eee6SP4k2omS5Q" name=":WebBrowser" coveredBy="_s5N4oVD3Eee6SP4k2omS5Q _6erowlD3Eee6SP4k2omS5Q _CkTsEVD4Eee6SP4k2omS5Q _Mbw
        <lifeline xmi:type="uml:Lifeline" xmi:id="_QhZugFD3Eee6SP4k2omS5Q" name=":Application" coveredBy="_wFSMAVD3Eee6SP4k2omS5Q _CkTsElD4Eee6SP4k2omS5Q _Mbw5YVD4Eee6SP4k2omS5Q"/>
        <lifeline xmi:type="uml:Lifeline" xmi:id="_bVF6kFD3Eee6SP4k2omS5Q" name=":FacebookAuthorizationServer" coveredBy="_xzPioVD3Eee6SP4k2omS5Q _VE1JU1D4Eee6SP4k2omS5Q _m_nLgFD4Ee
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_4ff_QVD3Eee6SP4k2omS5Q" name="MessageSend" covered="_ITxlwFD3Eee6SP4k2omS5Q"/>
        <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_ln0HkFD3Eee6SP4k2omS5Q" name="BehaviorExecSpecStart" covered="_ITxlwFD3Eee6SP4k2omS5Q" execution="_ln0HkVD
        <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="_ln0HkVD3Eee6SP4k2omS5Q" name="BehaviorExecSpec" covered="_ITxlwFD3Eee6SP4k2omS5Q" finish="_ln0Hk1D3Eee6SP4k2
        <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="_s5N4oVD3Eee6SP4k2omS5Q" name="BehaviorExecSpec0" covered="_NGu-cFD3Eee6SP4k2omS5Q" finish="_7m2VQVD4Eee6SP4k
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_6erowlD3Eee6SP4k2omS5Q" name="MessageRecv" covered="_NGu-cFD3Eee6SP4k2omS5Q" message="_6erowFD3Eee6SP4k2omS5
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_6erowVD3Eee6SP4k2omS5Q" name="MessageSend0" covered="_ITxlwFD3Eee6SP4k2omS5Q" message="_6erowFD3Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_CkTsEVD4Eee6SP4k2omS5Q" name="MessageSend1" covered="_NGu-cFD3Eee6SP4k2omS5Q" message="_CkTsEFD4Eee6SP4k2omS
        <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="_wFSMAVD3Eee6SP4k2omS5Q" name="BehaviorExecSpec1" covered="_QhZugFD3Eee6SP4k2omS5Q" finish="_Mbw5YVD4Eee6SP4k
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_Mbw5YVD4Eee6SP4k2omS5Q" name="MessageSend2" covered="_QhZugFD3Eee6SP4k2omS5Q" message="_Mbw5YFD4Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_Mbw5YlD4Eee6SP4k2omS5Q" name="MessageRecv1" covered="_NGu-cFD3Eee6SP4k2omS5Q" message="_Mbw5YFD4Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_7m2VQlD4Eee6SP4k2omS5Q" name="MessageRecv5" covered="_ITxlwFD3Eee6SP4k2omS5Q" message="_7m2VQFD4Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_tKbH8VD4Eee6SP4k2omS5Q" name="MessageSend5" covered="_NGu-cFD3Eee6SP4k2omS5Q"/>
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_tKbH81D4Eee6SP4k2omS5Q" name="MessageRecv4" covered="_ITxlwFD3Eee6SP4k2omS5Q"/>
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_CkTsElD4Eee6SP4k2omS5Q" name="MessageRecv0" covered="_QhZugFD3Eee6SP4k2omS5Q" message="_CkTsEFD4Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_VE1JUVD4Eee6SP4k2omS5Q" name="MessageSend3" covered="_NGu-cFD3Eee6SP4k2omS5Q" message="_VE1JUFD4Eee6SP4k2omS
        <fragment xmi:type="uml:BehaviorExecutionSpecification" xmi:id="_xzPioVD3Eee6SP4k2omS5Q" name="BehaviorExecSpec2" covered="_bVF6kFD3Eee6SP4k2omS5Q" finish="_m_nLgFD4Eee6SP4k
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_VE1JUlD4Eee6SP4k2omS5Q" name="MessageRecv2" covered="_bVF6kFD3Eee6SP4k2omS5Q" message="_VE1JUFD4Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_m_nLgFD4Eee6SP4k2omS5Q" name="MessageSend4" covered="_bVF6kFD3Eee6SP4k2omS5Q" message="_m_eBkFD4Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_m_nLgVD4Eee6SP4k2omS5Q" name="MessageRecv3" covered="_NGu-cFD3Eee6SP4k2omS5Q" message="_m_eBkFD4Eee6SP4k2omS
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="_7m2VQVD4Eee6SP4k2omS5Q" name="MessageSend6" covered="_NGu-cFD3Eee6SP4k2omS5Q" message="_7m2VQFD4Eee6SP4k2omS
        <fragment xmi:type="uml:ExecutionOccurrenceSpecification" xmi:id="_ln0Hk1D3Eee6SP4k2omS5Q" name="BehaviorExecSpecFinish" covered="_ITxlwFD3Eee6SP4k2omS5Q" execution="_ln0HkV
        <message xmi:type="uml:Message" xmi:id="_6erowFD3Eee6SP4k2omS5Q" name="get FB resource" receiveEvent="_6erowlD3Eee6SP4k2omS5Q" sendEvent="_6erowVD3Eee6SP4k2omS5Q"/>
        <message xmi:type="uml:Message" xmi:id="_CkTsEFD4Eee6SP4k2omS5Q" name="request FB access" receiveEvent="_CkTsElD4Eee6SP4k2omS5Q" sendEvent="_CkTsEVD4Eee6SP4k2omS5Q"/>
        <message xmi:type="uml:Message" xmi:id="_Mbw5YFD4Eee6SP4k2omS5Q" name="http redirect" messageSort="reply" receiveEvent="_Mbw5YlD4Eee6SP4k2omS5Q" sendEvent="_Mbw5YVD4Eee6SP4k
        <message xmi:type="uml:Message" xmi:id="_VE1JUFD4Eee6SP4k2omS5Q" name="authorize" receiveEvent="_VE1JUlD4Eee6SP4k2omS5Q" sendEvent="_VE1JUVD4Eee6SP4k2omS5Q"/>
        <message xmi:type="uml:Message" xmi:id="_m_eBkFD4Eee6SP4k2omS5Q" name="permission form" messageSort="reply" receiveEvent="_m_nLgVD4Eee6SP4k2omS5Q" sendEvent="_m_nLgFD4Eee6SP
        <message xmi:type="uml:Message" xmi:id="_7m2VQFD4Eee6SP4k2omS5Q" name="permission form" messageSort="reply" receiveEvent="_7m2VQlD4Eee6SP4k2omS5Q" sendEvent="_7m2VQVD4Eee6SP
    </packagedElement>
</uml:Model>
```

Hình 7. File xmi thu biểu diễn biểu đồ trình tự trên Papyrus

Trong Hình 7, các thẻ lifeline tương ứng với các Lifeline, các thẻ message tương ứng với các lời gọi thông điệp, các thẻ fragment tương ứng với các occurrence và execution trong biểu đồ trình tự ở Hình 5.

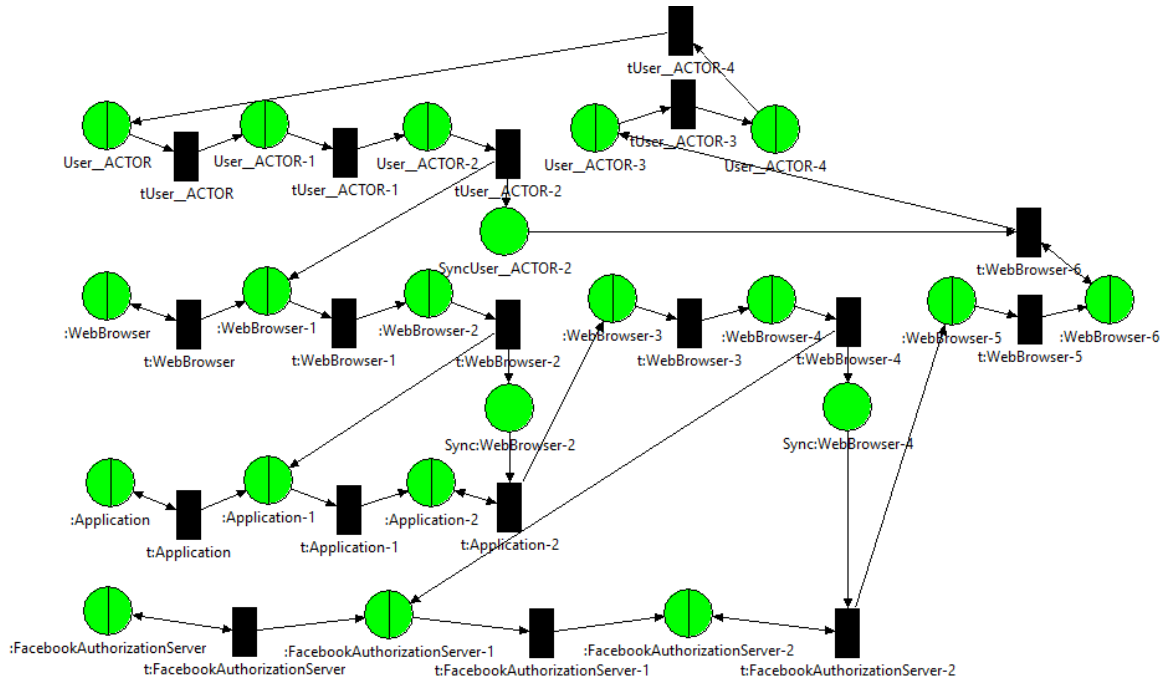
File XMI trên được đưa qua *Bộ chuyển đổi SD2QPN*, với các phép biến đổi đã đề cập tới trong phần II-B. Mô hình QPN mà khối *QPE* đọc được có dạng như Hình 7 với file qpe tương ứng như Hình 8.

Khi có được mô hình QPN này, có thể thiết lập khác cho các tham số mặc định hoặc có những tùy chỉnh riêng biệt hơn trước khi chạy thử nghiệm mô hình. Một phần kết quả sau khi phân tích mô hình QPN được thể hiện trong Hình 10.

Nhận xét kết quả:

Từ một phần kết quả nhận được ở hình 10 khi đánh giá QPN trên công cụ QPME ta thấy: mỗi hàng thể hiện các thông số hiệu năng của từng Queue ứng với từng đối tượng tham gia vào biểu đồ trình tự UML của Hệ thống xác thực

người dùng của Facebook. Các giá trị Mean Token Residence Time, Queue Utilization, Mean Total Token Population, Total Arrival Throughput, Total Departure Throughput trên các cột thể hiện các thông số hiệu năng tương ứng của các queue như: thời gian trung bình của một color trong queue (thời gian xử lý), mức độ sử dụng hàng đợi (sử dụng tài nguyên), số màu trong queue (công suất) hay tổng thông lượng đến queue và thông lượng đi ra khỏi queue. Như vậy ta có thể thấy thay vì đánh giá các thông số hiệu năng của hệ thống trên biểu đồ trình tự UML một cách khó khăn ta có thể đánh giá các thông số này trên QPN tương ứng bằng công cụ QPME một cách hiệu quả.



Hình 8. Mô hình QPN biến đổi được

```
<?xml version="1.0" encoding="UTF-8"?>

<net xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" qpme-version="2.1.0">
  <colors>
    <color real-color="#89c9b4" id="_1497442711254" name="User__ACTOR"/>
    <color real-color="#9ff789" id="_1497442711255" name=":WebBrowser"/>
    <color real-color="#41c96e" id="_1497442711256" name=":Application"/>
    <color real-color="#6720b8" id="_1497442711257" name=":FacebookAuthorizationServer"/>
    <color real-color="#ea9922" id="_1497442711258" name="get FB resource"/>
    <color real-color="#b1b060" id="_1497442711259" name="request FB access"/>
    <color real-color="#814eb0" id="_1497442711260" name="http redirect"/>
    <color real-color="#050481" id="_1497442711261" name="authorize"/>
    <color real-color="#94e5c8" id="_1497442711262" name="permission form"/>
    <color real-color="#d84ea1" id="_1497442711263" name="permission form 2"/>
  </colors>
  <queues>
    <queue strategy="FCFS" number-of-servers="1" id="_1497442711248" name="QUser__ACTOR"/>
    <queue strategy="FCFS" number-of-servers="1" id="_1497442711249" name="Q:WebBrowser"/>
    <queue strategy="FCFS" number-of-servers="1" id="_1497442711250" name="Q:Application"/>
    <queue strategy="FCFS" number-of-servers="1" id="_1497442711251" name="Q:FacebookAuthorizationServer"/>
  </queues>
  <places>
    <place id="_1497442711264" departure-discipline="NORMAL" xsi:type="queueing-place" name="User__ACTOR" queue-ref="_1497442711248">
      <meta-attributes>
        <meta-attribute xsi:type="location-attribute" location-x="55" location-y="82"/>
        <meta-attribute xsi:type="simqpn-place-configuration" id="_1497442711453" configuration-name="fas-test" statsLevel="3"/>
      </meta-attributes>
      <color-refs>
        <color-ref color-id="_1497442711254" maximum-capacity="0" id="_1497442711282" xsi:type="queueing-color-reference" ranking="0" priority="0" distribution-function="Exponen
          <meta-attributes>
            <meta-attribute xsi:type="simqpn-batch-means-queueing-color-configuration" configuration-name="fas-test" id="_1497442711474" signLev="0.05" reqAbsPrc="50" reqRelPrc=
          </meta-attributes>
        </color-ref>
      </color-refs>
    </place>
```

Hình 9. Một phần file QPE tương ứng với mô hình QPN biến đổi được

Queue	Mean Token Residence Time	Queue Utilization	Mean Total Token Population	Total Arrival Throughput	Total Departure Throughput
Q:Application (queue)	2,641	1	2,639	0,999	0,999
Q:FacebookAuthorizationServer (queue)	2,643	1	2,638	0,998	0,998
Q:WebBrowser (queue)	3,121	1	3,119	0,999	0,999
QUser__ACTOR (queue)	1,016	0,129	0,154	0,151	0,151

Hình 10. Một phần kết quả thu được khi mô phỏng QPN trên QPME

IV. KẾT LUẬN

Bài báo này đề xuất mô hình đánh giá hiệu năng phần mềm bằng phép biến đổi từ biểu đồ tuần tự sang mô hình trung gian XMI và sau đó chuyển sang mô hình QPN, giải quyết được vấn đề chuyển đổi thủ công. Mô hình QPN sau khi đã được chuyển đổi từ UML có thể được sử dụng trực tiếp để đánh giá hiệu năng nhờ vào công cụ QPME, cung cấp cơ sở để tự động hóa việc đánh giá hiệu năng phần mềm quy mô lớn. Do đó, trong vòng đời phát triển phần mềm, đánh giá hiệu năng có thể được thực hiện nhanh chóng bất kỳ lúc nào để hạn chế rủi ro của phần mềm do các vấn đề về hiệu năng.

Trong tương lai, chúng tôi hướng đến việc nghiên cứu phương pháp tự động chuyển đổi từ mô hình UML (có thể thêm một số ràng buộc nhất định) sang mô hình QPN để đánh giá hiệu năng phần mềm. Đồng thời, chúng tôi cũng tập trung phát triển công cụ QPME hỗ trợ thêm nhiều chức năng khác, có thể tích hợp với công cụ UML mã nguồn mở nào đó thành một phần mềm duy nhất, vừa để xây dựng mô hình UML, vừa cho phép đánh giá hiệu năng phần mềm mới tạo dựng được.

TÀI LIỆU THAM KHẢO

[1] JanWaller, "Performance Benchmarking of Application Monitoring Frameworks", Department of Computer Science, Kiel University, 2014.

[2] ZHU Lian-Zhang a*, KONG Fan-Sheng, "Automatic Conversion from UML to CPN for Software Performance Evaluation", 2012.

[3] Tony Spiteri Staines, "Transforming UML Sequence Diagrams into Petri Nets", Department of Computer Information Systems, Faculty of ICT, University of Malta Msida MSD 208, Malta, 2013.

[4] Simona Bernardi, Susanna Donatelli, Jose' Merseguer, "From UML Sequence Diagrams and Statecharts to analysable Petri Net models", In 3rd International Workshop on Software and Performance, Rome, July 2002.

[5] OMG Unified Modeling Language ™ (OMG UML), 2015.

[6] Samuel Kounev, Simon Spinner, "Queueing Petri net Modeling Environment User's Guide - A software tool for performance modeling and analysis using Queueing Petri Nets", 5/2011.

[7] Paola Bracchi, 2006, A Methodology for Software Performance Modeling and its Application to a Border Inspection System, College of Engineering and Mineral Resources at West Virginia University, Master of Science in Computer Science.

[8] F. Bause, "Queueing Petri Nets – a formalism for the combined qualitative and quantitative analysis of systems" in Proc. on Petri Nets and Performance Models, 1993, pp.14–23.

[9] Samuel Kounev, Simon Spinner, Philipp Meier, "Introduction to Queueing Petri Nets: Modeling Formalism, Tool Support and Case Studies", 2012.

[10] C. U. Smith, "Performance Engineering of Software Systems" Addison-Wesley, 1990.

[11] Thijmen de Gooijer, "Performance Modeling of ASP.Net Web Service, Applications: an industrial case study", 2011.

[12] F. Bause and P.S.Kritzinger, Stochastic Petri Nets–An Introduction to the Theory, Vieweg Verlag, 2013.

[13] http://www.uml-diagrams.org/facebook-authentication-uml-sequence-diagram-example.html.

[14] https://projects.eclipse.org/projects/rt.equinox/releases/4.6.0-neon.

DEVELOPMENT AND EXPERIEMENT OF A METHOD FOR SOFTWARE PERFORMANCE EVALUATION BASED ON TRANSFORMATION FROM SEQUENCE DIAGRAMS INTO QUEUEING PETRI NETS

Vu Van Doc, Nguyen Trong Bach, Huynh Quyet Thang

vvdoc@uneti.edu.vn, bachnt9x@gmail.com, thanghq@soict.hust.edu.vn

ABSTRACT: *At present, performance holds a key position in the success of software systems. However, quite a number of software products do not meet the performance requirements from the beginning. One of the best way to solve this problem is to introduce a software performance evaluation mechanism into the software development life cycle through modeling and simulation. The Unified Modeling Language (UML) and PetriNet Queueing (QPN) specification languages have a number of advantages in evaluating software performance parameters. Therefore, transform from UML to QPN model is considered as one of the most effective ways to solve software performance evaluation problems. However, this transformation is very difficult because the difference between the semantic of models. In this paper, we propose an approach based on modular transformation, provides an intermediate model consisting of abstract modules from elements in UML and QPN according to function and structure. The transformation consists of three steps: Extending the UML model to the XMI model, converting this model to the XML format of QPN. Transformed QPN models can work correctly and provide a framework for evaluating software performance against the Queueing Petri net Modeling Environment (QPME) tool.*

Keywords: *Software Performance Evaluation, Queueing Petri Nets, Unified Modeling Language.*