XÂY DỰNG CHƯƠNG TRÌNH CHUYỂN ĐỔI MỘT SỐ ĐỐI TƯỢNG CỦA BIỂU ĐỒ TRÌNH TỰ SANG MẠNG PETRI HÀNG ĐỢI

BUILDING A PROGRAM TO CONVERT SOME OBJECTS OF SEQUENCE DIAGRAM INTO QUEUEING PETRI NETS

Vũ Văn Đốc

Khoa Công nghệ thông tin, Trường Đại học Kinh tế - Kỹ thuật Công nghiệp

Đến Tòa soạn ngày 02/09/2021, chấp nhận đăng ngày 30/09/2021

Tóm tắt: Biểu đồ trình tự là một sự trừu tượng hóa của mô hình giao tiếp giữa các thực thể, đối tượng hoặc lớp khác nhau. Nó được sử dụng để mô tả một dấu vết thực thi của một hệ thống cụ thể, tại một thời điểm cụ thể. Mạng Petri hàng đợi (Queueing Petri Nets - QPNs) là các hình thức đồ họa, ở mức độ trừu tượng thấp hơn, có sẵn các kỹ thuật giải pháp dựa trên mô phỏng hiệu quả và chuyên nghiệp. Bài báo trình bày cách xây dựng chương trình chuyển đổi tự động một số đối tượng trong biểu đồ trình tự sang mạng Petri hàng đợi. Cách tiếp cận được trình bày trong bài báo có thể được sử dụng để chuyển đổi tự động một biểu đồ trình tự phức hợp thành mạng Petri hàng đợi.

Từ khóa: biểu đồ trình tự, mạng hàng đợi Petri, mô hình chuyển đổi

Abstract: A sequence diagram is an abstraction of the communication model between different entities, objects, or classes. It is used to describe an execution trace of a particular system at a particular time. Queueing Petri Nets (QPNs) are graphical forms, at a lower level of abstraction, with efficient and professional simulation-based solution techniques available. This paper presents how to build a program to automatically convert some objects in the sequence diagram to QPNs. Our approach can be used to automatically convert a complex sequence diagram into a QPNs.

Keywords: sequence diagrams, queueing Petri Nets, model Transformation

1. GIỚI THIỆU

Một mạng hàng đợi Petri Net thông thường là một ngôn ngữ mô hình toán học cho mô tả hệ thống. Mỗi Queueing Petri Nets (QPNs) bao gồm một tập hợp các *hàng đợi* (*queueing*), *vị trí* (place), *chuyển tiếp* (*transition*) và một tập hợp các cung kết nối (*connection*). Mỗi *hàng đợi* có một khả năng nhất định và có chiến lược để điều phối các yêu cầu truy cập trong mạng. Một *place* có thể chứa nhiều *Color* (*màu*) - là khái niệm mở rộng từ thẻ (token) *(token trong Petri Nets*) để phân biệt các token với nhau khi có nhiều loại. Có ba loại *place* là OrdinaryPlace (o-*place*) là dạng place bình thường, QueueingPlace (*q-place*) là dạng place được tích hợp thêm thành phần *Queue*, khi đó các color trong place này sẽ được điều phối truy cập theo chiến lược của Queue; SubnetPlace (s-place) là dạng place thay thế cho cả một Queueing Petri Nets, là phần tử cho phép QPN có khả năng phân cấp. Điều này cho phép dễ dàng trình bày các chiến lược lập lịch và mang lại lợi ích của Mạng xếp hàng vào thế giới Petri Nets. Một Chuyển tiếp

có thể có nhiều Mode (*chế độ*) - là khái niệm tương tự color dùng để phân biệt các cách mà một transition có thể thực hiện. Có hai loại chuyển tiếp đó là Timed Transition (*t-transition*) là dạng chuyển tiếp có liên quan đến yếu tố thời gian và Immediate Transition (*i-transition*) là dạng chuyển tiếp mà thời gian không phải nhân tố ảnh hưởng đến nó. Có hai loại kết nối đó là, Place Transition Connnection (*pt-connection*) dùng để kết nối một *place* với một *transition* và ngược lại; Incidence Function Connection (*if-connection*) dùng để kết nối có trọng số một color với một mode và ngược lại. Như đã nói ở trên, so với các loại mạng khác, *QPNs* giới thiệu một loại vị trí mới: *q-place*. Mỗi *q-place* 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. Máy chủ xử lý các mã thông báo trong hàng đợi theo một chiến lược lập lịch nhất định. Thời gian thẻ token chiếm máy chủ được xác định thông qua chiến lược phân phối. Khi thời gian của token kết thúc, token được đưa vào kho lưu trữ, sau đó hoạt động giống như một Immediate place cho các chuyển đổi được kết nối. Chỉ các thẻ *token* trong kho lưu trữ được coi là có sẵn cho hàm tỷ lệ [7].

Biểu đồ trình tự (*SDs*) là một loại biểu đồ tương tác *UML* [1]. *SDs* là biểu đồ hai chiều trong đó trục tung thể hiện thời gian và trục hoành thể hiện sự tương tác. Biểu đồ trình tự thường được sử dụng để mô tả luồng thông tin trong một hệ thống thông qua giao tiếp giữa các đối tượng.

Việc chuyển đổi mô hình từ *SDs* thành Queueing Petri Nets giúp cho hai mô hình này nhất quán với nhau, từ đó có thể áp dụng mô phỏng vào mô hình *Hàng đợi Petri Nets* để đánh giá hiệu năng phần mềm một cách đơn giản hơn.

Bài báo này đưa ra những đóng góp sau: (i) Xác định mô hình chuyển đổi từ biểu đồ trình tự sang mạng Petri hàng đợi; (ii) Xây dựng chương trình chuyển đổi từ một số đối tượng *SDs* sang mô hình *QPNs*.

2. NHỮNG NGHIÊN CỨU LIÊN QUAN

Một số nghiên cứu gần đây thảo luận về việc chuyển đổi biểu đồ trình tự SDs (mô hình dựa trên kịch bản) sang mô hình mạng Petri Nets (mô hình dựa trên trạng thái). Trong [2] các tác giả phân tích ưu điểm và nhược điểm của UML và Petri net trong việc mô hình hóa các phần mềm phức tạp và đề xuất các quy tắc chuyển đổi từ UML sang mạng Petri Net. Trong [6] tác giả đề xuất chuyển đổi mô hình từ Biểu đồ trình tự sang Petri Nets và một cách tiếp cận sử dụng Cấu trúc nhãn sự kiện (LES), cũng như các phương pháp để dịch cả Biểu đồ trình tự và Petri Nets sang LES. Tác giả cũng đưa ra một ứng dụng chuyển đổi mô hình từ UML sang Petri Nets, được đặt tên là SD2PN. Trong [1], tác giả cũng đã đưa ra một phương pháp chuyển SDs thành Petri Nets một cách hiệu quả. Mặt khác, trong [3,4] các tác giả giới thiệu một giải pháp để chuyển đổi các tính năng có liên quan nhất của Biểu đồ trình tự UML để mô hình hóa các hệ thống phân tán thành các mạng Petri Nets màu tương đương và chấp nhận các dấu vết thực thi giống nhau (chuỗi sự kiện) như mô hình ban đầu. Biểu đồ trình tự UML được xây dựng bằng công cụ Papyrus được chuyển đổi thành Petri Nets màu có thể thực thi được với Công cụ CPN. Các tác giả đã định nghĩa và mô tả sự chuyển đổi từ biểu đồ trình tự UML2 sang Petri Nets màu và chứng minh tính đúng đắn về cú pháp và ngữ nghĩa của sự chuyển đổi [4]. Họ cũng trình bày công cụ chuyển đổi mô hình dựa trên kịch bản với khả năng phân tích SD2CPN. Cuối cùng [7] đưa ra một ánh

xạ từ mô hình PCM sang QPN, được thực hiện bằng cách chuyển đổi tự động từ mô hình PCM sang Queueing Petri Nets. Theo cách tiếp cận để chuyển đổi mô hình từ SDs thành Queueing Petri Nets tôi sẽ xây dựng chương trình chuyển đổi từng SD sang QPN. Thông tin chi tiết được trình bày trong Phần 4. Dựa trên [5], tôi phát triển các chuyển đổi cho các đối tượng Message như: Create Message, Delete Message, Lost Message, Found Message, Synchronous Call và Asynchronous Call.

3. BIỂU ĐỒ TRÌNH TỰ VÀ MẠNG PETRI HÀNG ĐỢI

3.1. Biểu đồ trình tự

Biểu đồ trình tự (Sequence Diagram - SD) là một trong những biểu đồ tương tác UML phổ biến, tập trung vào dãy thông điệp trao đổi giữa một số bên tham gia. [6]. Biểu đồ trình tự dùng để xác định các trình tự diễn ra sự kiện của một nhóm đối tượng nào đó. Nó miêu tả chi tiết các thông điệp được gửi và nhận giữa các đối tượng đồng thời cũng chú trọng đến trình tự về mặt thời gian gửi và nhận các thông điệp đó. Biểu đồ trình tự được thể hiện theo hai trục:

- Trục dọc từ trên xuống chỉ thời gian xảy ra các sự kiện, hay sự truyền thông điệp, được biểu diễn bằng các đường thẳng đứng đứt nét bắt đầu từ đỉnh đến đáy của biểu đồ.
- Trục ngang từ trái qua phải là dãy các đối tượng tham gia vào việc trao đổi các thông điệp với nhau theo chiều ngang, có thể có cả các tác nhân.

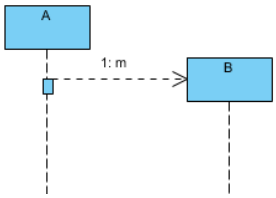
Biểu đồ trình tự gồm một số đối tượng chính đó là: Interaction (Frame), Lifeline, Message, Interaction Fragment… [6].

Các đối tượng thông báo (messages) trong biểu đồ trình tự:

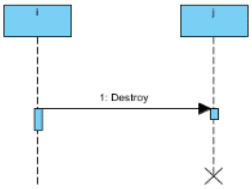
Message là một thông báo xác định một loại giao tiếp cụ thể giữa các lifeline trong biểu đồ trình tự. Điểm bắt đầu hoặc kết thúc của một message được gọi là message end.

Tùy thuộc vào các loại hành động sẽ sinh ra các loại tin nhắn khác nhau như: lời gọi đồng bộ (Synchronous Call), lời gọi không đồng bộ (Asynchronous Call), thông báo khởi tạo (Create Message), thông báo hủy (Destroy Message), Found Message và Lost message.

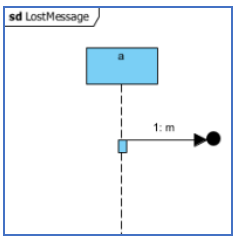
Create Message là một thông báo gửi tới một lifeline và tạo ra lifeline đó:



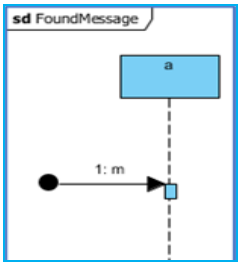
Delete Message là một message gửi đi để kết thúc một lifeline khác:



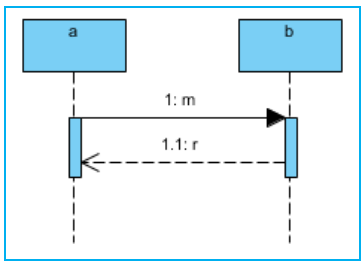
Lost Message là một message mà biết sự kiện gửi nhưng không biết sự kiện nhận.



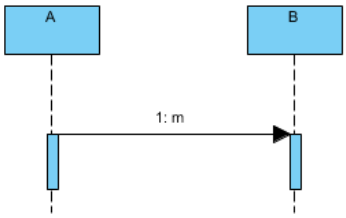
Found message là một thông báo mà biết sự kiện nhận nhng không biết sự kiện gửi.



Lời gọi đồng bộ (Synchronous Call) là một thông báo mà lifeline gửi đợi lifeline nhận trả thông báo về rồi mới thực hiện thông báo tiếp theo [6].

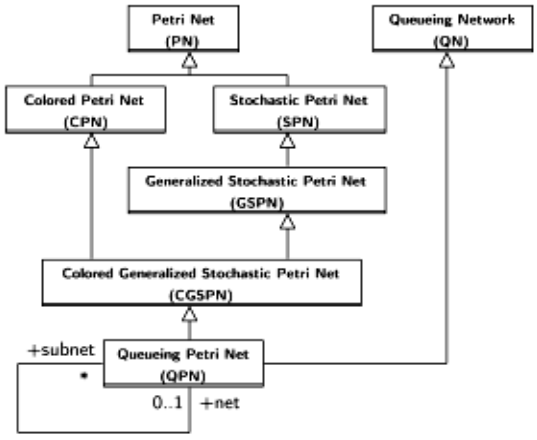


Lời gọi không đồng bộ (Asynchronous Call) là một thông báo mà lifeline gửi có thể thực hiện thông báo tiếp theo mà không cần đợi lifeline nhận trả thông báo về.



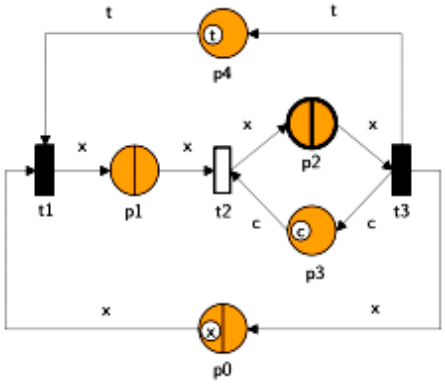
3.2. Mạng Petri hàng đợi

Mạng Petri hàng đợi là một mô hình hiệu năng với cơ sở toán học mạnh mẽ, được hình thành từ một số mở rộng của mạng Petri (Petri Net) như CPN, SPN, GSPN, CGSPN và được bổ sung thêm các tính chất về hàng đợi của mạng hàng đợi (Queueing Network). Một phương trình đơn giản để thể hiện điều này là: QPN = PN + QN và được thể hiện ở hình 1.



Hình 1. Queueing Petri Nets = PN + QN

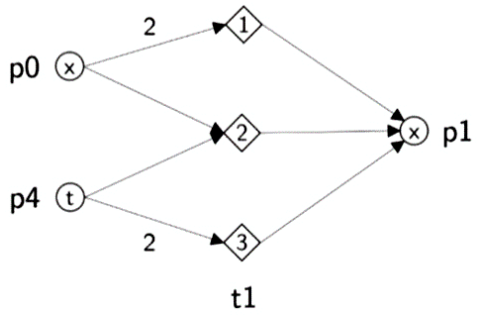
Để hiểu rõ hơn về Queueing Petri Nets ta tìm hiểu một *ví dụ về QPN với năm place (p0, p1, p2, p3, p4), ba transition (t1, t2, t3) và mười pt-connection trong đó: (p3, p4) là các o-place, (p0, p1) là các q-place, p2 là một s-place, (t1, t3) là các i-transition, t2 là một t-transition.*



Hình 2. Một ví dụ về QPN

t1 là input-transition của place p1, p1 là input-place của transition t2, t2 là output-transition của place p3, p3 là output-place của transition t3. Transition t1 có hai input-place (p0, p4) và một output-place (p1). Transition t3 có một input-place (p2) và ba output- place (p0, p3, p4). Mỗi place trong hình vẽ này đều có một input-transition và một output-transition. Ngoài những ký pháp thông thường, trong hình 2 có xuất hiện các ký pháp mở rộng. Các color x, c, t lần lượt được đặt nằm bên trong các place p0, p3, p4, thể hiện rằng: ở thời điểm khởi tạo mạng (initial marking - hình trạng khởi tạo), những place này sẽ được khởi tạo color tương ứng. Các place không có ký pháp color đặt bên trong sẽ không được khởi tạo color tại thời điểm ban đầu (trong ví dụ này là p1, p2). Những connection chứa trọng số là những if-connection: kết nối color với mode và ngược lại như đã nói từ trước. Một hàm tỷ lệ được liên kết với mỗi transition, xác định cách thức transition hoạt động. Giả sử rằng transition t1 có ba mode là mode 1,

mode 2 và mode 3. Hình 3 minh hoạ một ví dụ về hàm tỷ lệ liên kết với transition t1:



Hình 3. Một ví dụ về hàm tỷ lệ của transition

Các color xuất hiện trong hình vẽ là các color có thể được chứa trong input-place và output-place của transition t1, đó là color x trong place p0 là input-place của transition t1, color t trong place p4 là input-place của transition t1 và color x trong place p1 là output-place của transition t1. Các if-connection nối các color và mode với nhau chứa trọng số mặc định là 1 nếu không biểu diễn trọng số khác. Trọng số trên if-connection nối color p0.x với mode 1 và trên if-connection nối color p4.t với mode 3 là 2, trọng số trên các if-connection còn lại bằng 1.

Một transition ở trạng thái enabled (sẵn sàng) nếu nó có một mode mà trọng số trên mọi if-connection nối tới mode đó từ một color trong một input- place của transition nhỏ hơn hoặc bằng số lượng color tương ứng mà input-place này đang chứa [3]. Giả sử place p0 đang chứa hai color x, place p4 đang chứa một color t. Khi đó, transition ở trạng thái enabled, do mode 1 và mode 2 đều thoả mãn điều kiện trên. Mode 3 không thoả mãn vì place p4 chỉ chứa một color t trong khi if- connection nối p4.t tới mode 3 lại có trọng số bằng 2 > 1.

Một transition ở trạng thái enabled được gọi là e-transition. Một e-transition có thể đốt cháy (fire / kích hoạt) một mode thoả mãn ràng buộc để transition đó ở trạng thái enabled. Khi đốt cháy một mode m, transition này hủy số lượng color trong mỗi input-place liên quan đến mode m đúng bằng trọng số trên if-connection nối color trong place với mode m, đồng thời tạo mới một lượng các color cho mỗi output-place liên quan đến mode m đúng bằng trọng số của if-connection nối mode m với color trong place [3]. Nếu transition t1 đốt cháy mode 2 thì một color x trong place p0 bị huỷ, một color t trong place p4 bị hủy, đồng thời một color x khác được thêm vào place p1. Trong trường hợp transition t1 đốt cháy mode 1, hai color x trong place p0 bị huỷ, không có color t nào trong place p4 bị hủy nhưng vẫn có một color x khác được thêm vào place p1.

Tại một thời điểm, một e-transition không được đốt cháy nhiều hơn hai mode và không có hai e-transition nào cùng đốt cháy mode của nó. Thứ tự đốt cháy một transition sẽ phụ thuộc vào loại transition. Các i-transition được ưu tiên đốt cháy trước so với t-transition.

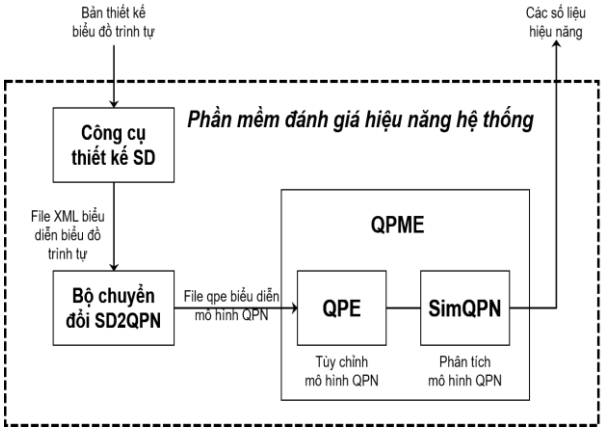
Trên hình số 2, p0 là một q-place, bản chất là một phần tử được hình thành bằng cách đặt queue (trong QN) vào o-place (trong PN). Mỗi q-place gồm hai thành phần: queue và depository. Thành phần queue cho phép điều phối các color chờ thực hiện dịch vụ. Thành phần depository lưu giữ các color đã hoàn thành dịch vụ trong thành phần queue. Đồng thời trên hình số 2 cũng có phần tử p2 là một S-place có khả năng chứa một QPN con có các o-place chuyên dụng là input, output và actual population. Trong QPN con này, có thể có các s-place khác, quá trình phân cấp không bị hạn chế.

4. CHUYỂN ĐỔI SD2QPN

4.1. Tổng quan

Để xây dựng bộ chuyển đổi SD2QPN, thiết kế

các đối tượng của vbieeur đồ trình tự trên công cụ Visual Paradigm 15.0, sau đó sử dụng ngôn ngữ lập trình Java để chuyển đổi file XML trên Visual paradigm 15.0 thành file qpe chạy trên công cụ mô phỏng QPME.



Hình 4. Mô hình chuyển đổi từ biểu đồ trình tự sang mạng petri hàng đợi

4.2. Chuyển đổi các đối tượng

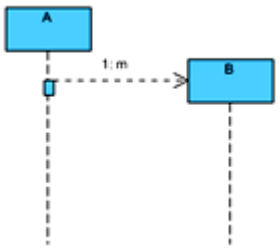
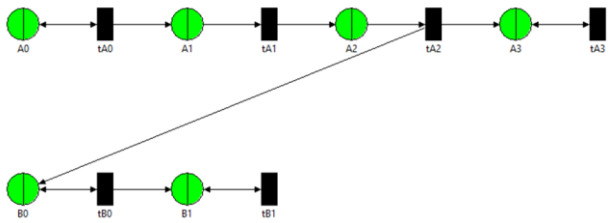
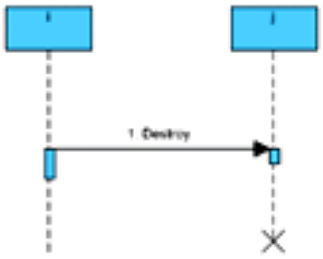
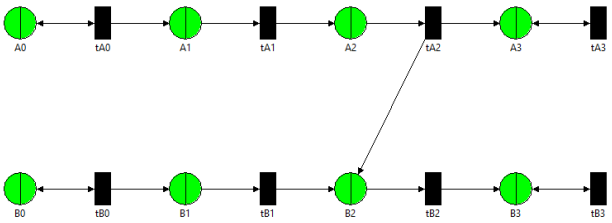
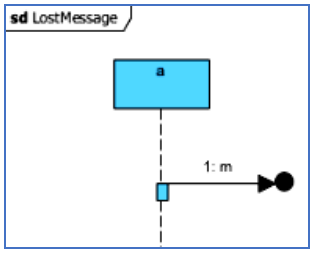
a. Cấu trúc chương trình chuyển đổi

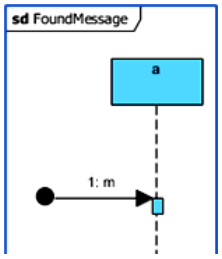
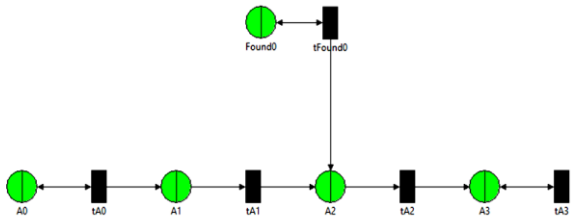
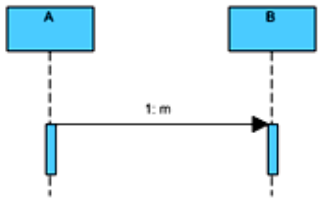
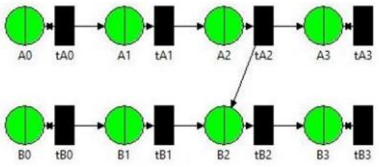
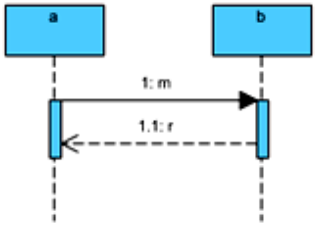
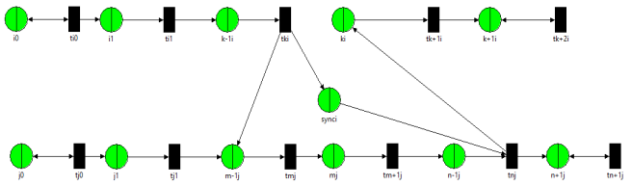
Trong chương trình chuyển đổi, xây dựng 2 gói (package):

- ***models*** chứa 2 gói ***SD*** (bao gồm các lớp biểu diễn các đối tượng trong biểu đồ trình tự) và gói ***qpn*** (bao gồm các lớp biểu diễn các đối tượng trong QPN)
- ***Ultilities*** bao gồm các lớp đọc các file đầu vào, chuyển đổi chúng sang Queueing Petri Nets
- Lớp Project.java để khởi tạo chương trình.

b. Kết quả chuyển đổi

Sau khi áp dụng chương trình vào việc chuyển đổi một số đối tượng trong biểu đồ trình tự tôi thu được một số kết quả như sau:

Tên đối tượng	Biểu đồ trình tự	Kết quả chuyển đổi dạng QPN
Create Message		
Destroy Message		
Lost Message		

Tên đối tượng	Biểu đồ trình tự	Kết quả chuyển đổi dạng QPN
Found message		
Asynchronous Call		
Synchronous Call		

Hình 5. Mô hình chuyển đổi từ biểu đồ trình tự sang mạng petri hàng đợi

4. KẾT LUẬN

Trong bài báo này trình bày về các đặc tính cơ bản của biểu đồ trình tự cũng như mô hình mạng Petri hàng đợi. Đồng thời cũng đã trình bày cách triển khai phần mềm chuyển đổi SD2QPN cho phép chuyển đổi từ tệp XML biểu thị một số đối tượng của biểu đồ trình tự sang mô hình Queueing Petri Nets. Kết quả chuyển đổi được trình bày trong hình 5. Các QPNs thu được từ việc sử dụng chương trình SD2QPN chuyển đổi một số đối tượng trong SDs như Create Message, Destroy Message, Lost Message, Found message, Asynchronous Call và Synchronous Call hoàn toàn phù hợp với lý thuyết đã được nêu ra ở [5]. Bằng cách sử dụng chương trình chuyển đổi này, QPNs thu được có thể được sử dụng để dự đoán hiệu suất phần mềm. Trong tương lai, tác giả sẽ nghiên cứu thêm về các biểu đồ trình tự khác, tìm ra các cách tiếp cận mới để có thể chuyển đổi một biểu đồ trình tự phức tạp thành Queueing Petri Nets nhằm hỗ trợ đánh giá hiệu suất phần mềm được tốt hơn.

TÀI LIỆU THAM KHẢO

[1] Tony Spiteri Staines: *Transforming UML Sequence Diagrams into Petri Nets*. Journal of Communication and Computer 10, pp. 72-81, 2013.

[2] Chun Jian Wang, Hong Jun Fan and Shuang Pan: *Research on Mapping UML to Petri-Net in System Modeling*. In MATEC Web of Conferences 44, 02038, 2016.

[3] Joao António Custódio Soares, Bruno Lima and Joao Pascoal Faria: *Automatic Model Transformation from UML Sequence Diagrams to Coloured Petri Nets*. In: 6th International Conference on Model-Driven Engineering and Software Development MODELSWARD, 2018.

[4] Dulani A. Meedeniya: *Correct model-to-model transformation for formal verification*. A Thesis Submitted for the Degree of PhD at the University of St Andrews, 2013.

https://research-repository.st-andrews.ac.uk/handle/10023/3691

[5] Vu Van Doc, Huynh Quyet Thang, Nguyen Trong Bach: *Formal Transformation from UML Sequence Diagrams to Queueing Petri Nets*. Series: Frontiers in Artificial Intelligence and Applications Volume 318: Advancing Technology Industrialization Through Intelligent Software Methodologies, Tools and Techniques Pages: 588 - 601, 2019.

[6] Mohamed Ariff Ameedeen*: A model driven approach to analysis and synthesis of sequence diagrams.* A thesis submitted to The University of Birmingham for the degree of Doctor of Philosophy, 2011. http://etheses.bham.ac.uk/3282/.

[7] Philipp Meier, Samuel Kounev, Heiko Koziolek: *Automated Transformation of Component-based Software Architecture Models to Queueing Petri Nets*. In: IEEE 19th Annual Interna-tional Symposium, pp. 339-348, 2011.

Thông tin liên hệ: **Vũ Văn Đốc**

Điện thoại : 0912648561 - Email: vvdoc@uneti.edu.vn

Khoa Công nghệ thông tin, Trường đại học Kinh tế - Kỹ thuật Công nghiệp.