ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

PHẠM THỊ TỐ NGA

ĐẶC TẢ VÀ KIỂM CHỨNG CÁC HỆ THỐNG THỜI GIAN THỰC SỬ DỤNG UPPAAL

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2017

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

PHẠM THỊ TỐ NGA

ĐẶC TẢ VÀ KIỂM CHỨNG CÁC HỆ THỐNG THỜI GIAN THỰC SỬ DỤNG UPPAAL

Ngành: Công nghệ thông tin
Chuyên ngành:Kỹ Thuật Phần Mềm
Mã số: 60480103

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS.TS PHẠM NGỌC HÙNG

Hà Nội – 2017

MỤC LỤC

LỜI CAM ĐOAN

Tôi xin cam đoan luận văn tốt nghiệp với đề tài **"Đặc tả và kiểm chứng các hệ thống thời gian thực sử dụng Uppaal"** này là công trình nghiên cứu của riêng tôi dưới sự hướng dẫn của PGS.TS Phạm Ngọc Hùng. Các kết quả tôi trình bày trong luận văn là hoàn toàn trung thực và chưa từng được công bố trong bất cứ công trình nào khác.

Tôi đã trích dẫn đầy đủ các tài liệu tham khảo, các công trình nghiên cứu liên quan ở trong nước và quốc tế trong phần tài liệu tham khảo. Ngoại trừ các tài liệu tham khảo này, luận văn này hoàn toàn là công việc của riêng tôi.

Nếu có bất cứ phát hiện nào về sự gian lận sao chép tài liệu, công trình nghiên cứu của tác giả khác mà không ghi rõ trong phần tài liệu tham khảo, tôi xin chịu hoàn toàn trách nhiệm về kết quả luận văn của mình.

Hà nội, tháng 10 năm 2017

Học viên

Phạm Thị Tố Nga

LỜI CẢM ƠN

Tôi xin bày tỏ lòng cảm ơn chân thành và sâu sắc nhất đến PGS.TS Phạm Ngọc Hùng vì sự hướng dẫn và chỉ bảo tận tình cùng với những định hướng, những lời khuyên, những kiến thức vô cùng quý giá của Thầy trong quá trình tôi theo học cũng như làm luận văn.

Tôi xin được gửi lời cảm ơn tới các Thầy Cô trong khoa Công nghệ thông tin - trường Đại học Công Nghệ- Đại học Quốc gia Hà Nội đã trang bị cho tôi những kiến thức quý báu trong quá trình tôi theo học tại khoa. Đây cũng chính là tiền đề để m có được những kiến thức cần thiết để hoàn thiện luận văn này.

Tôi xin được gửi lời cảm ơn tới các Thầy Cô giáo cùng các anh chị em bạn bè đang theo học tại bộ môn Công nghệ Phần mềm đã rất tận tình chỉ bảo và tạo điều kiện tốt nhất để tôi được làm việc trên bộ môn với đầy đủ trang thiết bị cần thiết để tôi có thể hoàn thiện tốt nhất luận văn này.

Tôi cũng xin được gửi lời cảm ơn chân thành đến lãnh đạo và các anh chị em đồng nghiệp tại trường Đại học Đại Nam nơi tôi đang công tác cũng như gia đình, bạn bè, người thân đã giúp đỡ tôi cả về vật chất lẫn tinh thần để tôi hoàn thành được luận văn này.

Mặc dù đã rất cố gắng nhưng luận văn chắc chắn không tránh khỏi những thiếu sót, tôi rất mong nhận được những ý kiến đánh giá và phê bình từ phía các Thầy Cô để luận văn được hoàn thiện hơn.

Tôi xin chân thành cảm ơn!

Hà nội, tháng 11 năm 2017

Học viên

Phạm Thị Tố Nga

DANH MỤC HÌNH VẼ

CHƯƠNG 1. GIỚI THIỆU

1.1 Đặt vấn đề

Trong thời đại ngày nay, các hệ thống có yếu tố thời gian và đặc biệt các hệ thống thời gian thực là một trong những lĩnh vực nhận được rất nhiều sự quan tâm của giới khoa học nói chung và giới khoa học nghiên cứu về công nghệ nói riêng. Thật vây, hệ thống thời gian thực được ứng dụng rất nhiều trong đời sống xã hội, trong sản xuất, trong y tế, trong hàng không vũ trụ và trong quân sự, gần như trong mọi lĩnh vực ta đều thấy có sự góp mặt của những ứng dụng trong hệ thống thời gian thực. Không chỉ góp mặt trong nhiều lĩnh vực mà sự góp mặt của nó còn có tầm quan trọng rất lớn đối với hệ thống. Trong hệ thống thời gian thực, các công vệc và các tác vụ cần phải hoàn thành trong một khoảng thời gian cho phép (deadline), nếu không đáp ứng được yêu cầu thời gian thì hệ thống sẽ sụp đổ hoặc sẽ gây ra hậu quả nghiêm trọng (hệ thời gian thực cứng: Hard Real-Time) hoặc sẽ bị suy giảm về chất lượng dịch vụ (hệ thời gian thực mềm: Soft Real-Time) [6, 9].

Chính vì tầm quan trọng của yếu tố thời gian trong hệ thống thời gian thực như vậy nên việc kiểm tra tính đúng đắn đối với hệ thống này là rất cần thiết. Việc kiểm tra tính đúng đắn của một hệ thống có thể được thực hiện ở khâu kiểm thử và kiểm chứng. Tuy nhiên, với những hệ thống có ràng buộc về thời gian cũng như tầm quan trọng của hệ thống mà việc kiểm thử không kiểm tra được hết mà chủ yếu tập chung ở khâu kiểm chứng [10]. Việc kiểm chứng tính đúng đắn của hệ thống nhằm kiểm tra xem hệ thống có vận hành đúng như yêu cầu không, muốn vậy cần phải có mô phỏng sự vận hành của hệ thống, cần có bước kiểm tra quy trình vận hành đó có đảm bảo các tính chất cơ bản của một hệ thống như: tính đến được của một trạng thái, tính an toàn, tính liên tục theo thời gian (không bị dừng – not deadlock) [10, 11]. Muốn làm được điều đó cần phải có một bộ công cụ có thể mô tả được sự vận hành của hệ thống, qua đó mô phỏng được sự vận hành đó và từ đó kiểm tra được sự vận hành đó có thỏa mãn các yêu cầu của hệ thống hay không. Đây thực sự là mối quan tâm lớn đối với vấn đề kiểm chứng nói riêng và công nghệ phần mềm nói chung.

Hiện nay có rất nhiều bộ công cụ cho phép kiểm chứng tự động các hệ thống phần mềm có yếu tố thời gian như: SPIN, LTSmin, mCRL2, MRMC, PAT, TAPAL, DREAM, ROMEO, UPPAAL … Trong đó bộ công cụ Uppaal thể hiện

những tính năng mạnh mà những bộ công cụ khác không có như việc mô phỏng được sự vận hành của hệ thống thời gian thực và kiểm chứng sự vận hành bởi các hệ thống câu lệnh rất đơn giản. Công cụ này giúp ta có thể kiểm chứng những hệ thống được mô hình hóa thành những hệ thống ô-tô-mát thời gian với nhứng biến số nguyên, cấu trúc dữ liệu, hàm người dùng và đồng bộ các kênh. Với việc đặc tả và kiểm chứng một hệ thống thời gian thực thì bộ công cụ Uppaal được đánh giá là tốt nhất hiện nay và được sử dụng rộng rãi trong công nghiệp. Nhưng bên cạnh đó việc sử dụng bộ công cụ Uppaal trong kiểm chứng hệ thống có yếu tố thời gian cũng đòi hỏi người sử dụng có trình độ nhất định trong việc đặc tả hệ thống dưới dạng các ô-tô-mát thời gian cũng như điều khiển sự vận hành và tương tác giữa các ô-tômát đó thông qua một ngôn ngữ lập trình [7, 8].

1.2 Mục tiêu và phạm vi của đề tài

Trong luận văn này tác giả đã tập trung tìm hiểu về bộ công cụ kiểm chứng Uppaal, đi sâu vào tìm hiểu ngôn ngữ đặc tả của Uppaal, tìm hiểu cách đặc tả một hệ thống phần mềm dưới dạng các ô-tô-mát thời gian và điều khiển sự vận hành của hệ thống thông qua ngôn ngữ lập trình C++, cũng như tìm hiểu cơ chế kiểm chứng của bộ công cụ này cho các hệ thống thời gian thực. Từ đó tác giả xây dựng một số ví dụ (cụ thể tác giả đã xây dựng được 4 ví dụ) về một số hệ thống thời gian thực áp dụng vào đặc tả và kiểm chứng hệ thống đó bởi bộ công cụ Uppaal.

Đối với mỗi ví dụ tác giả giả định là một hệ thống thời gian thực, tiến hành đặc tả, mô hình hóa dưới hệ ô-tô-mát thời gian trên trình soạn thảo của Uppaal sau đó chạy mô phỏng và kiểm chứng sự hoạt động của hệ thống đó.

1.3 Cấu trúc của luận văn

Phần còn lại của luận văn được trình bày thành ba chương: Ở chương 2, tác giả trình bày những cơ sở lí thuyết cần thiết cho việc nghiên cứu đề tài. Sang chương 3, tác giả nêu những hiểu biết của tác giả về bộ công cụ Uppaal cũng như cách đặc đả một hệ thống phần mềm dưới dạng các ô-tô-mát thời gian và phương pháp mô phỏng và kiểm chứng trong Uppaal. Chương 4, tác giả trình bày một số ví dụ áp dụng mà tác giả đã xây dựng được sau khi tìm hiểu về bộ công cụ Uppaal. Chương này tác giả trình bày bốn ví dụ mà tác giả đã xây dựng được về bốn hệ thống thời gian và tiến hành đặc tả và kiểm chứng các hệ thống đó qua công cụ Uppaal. Cuối cùng là phần kết luận, ở đây tác giả tóm tắt những công việc mà tác

giả đã làm được trong luận văn cũng như đưa ra những hạn chế và hướng mở rộng tiếp theo của đề tài.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Đặc tả hệ thống

Dựa trên những yêu cầu đối với việc sử dụng hệ thống để đưa ra các đặc tả cho hệ thống. Muốn đặc tả tốt hệ thống thì người đặc tả cần hiểu rõ nguồn gốc, các dạng thông tin cần cung cấp cho hệ thống hoạt động, cần nắm được hệ thống sẽ phải giải quyết những vấn đề gì, những kết quả cần phải có khi vận hành hệ thống là gì. Xác định được mối quan hệ giữa cái vào và cái ra cho quá trình hoạt động của hệ thống. Các đặc tả chi tiết hệ thống nhằm phục vụ cho việc xây dựng và trắc nghiệm về hệ thống để kiểm tra xem những nhiệm vụ đã đặt ra cho hệ thống có hoàn tất được hay không [5].

Việc sử dụng ngôn ngữ tự nhiên để đặc tả hệ thống dẫn đến rất nhiều bất tiện trong đó nhữn bất tiện tiêu biểu có thể kể đến như: Việc nhầm lẫn do cách hiểu các khái niệm khác nhau giữa hai bên; Đặc tả yêu cầu ngôn ngữ tự nhiên quá mềm dẻo do đó một vấn đề có thể được mô tả bằng quá nhiều cách khác nhau; Các yêu cầu không được phân hoạch tốt, khó tìm các mối quan hệ Do vậy người ta thường dùng các thay thế khác để đặc tả các yêu cầu như: Đặc tả bằng ngôn ngữ tự nhiên có cấu trúc; đặc tả bằng ngôn ngữ mô tả thiết kế, giống ngôn ngữ lập trình nhưng có mức trừu tượng cao hơn; Đặc tả bằng ngôn ngữ đặc tả yêu cầu; Đặc tả bằng ghi chép graphic; Đặc tả toán học,..

Đặc tả hệ thống có thể chia thành hai loại: đặc tả phi hình thức (ngôn ngữ tự nhiên) và đặc tả hình thức (dựa trên kiến trúc toán học).

Đặc tả phi hình thức: Đặc tả phi hình thức là đặc tả sử dụng ngôn ngữ tự nhiên. Như đã trình bày ở trên, đặc tả theo hình thức này có thể không được chặt chẽ nhưng nhiều người có thể dễ dàng tiếp cận và có thể dùng để trao đổi với nhau để làm chính xác hóa các điểm chưa rõ, chưa thống nhất giữa các bên phát triển hệ thống.

Đặc tả hình thức: Đặc tả hình thức là đặc tả mà ở đó các từ ngữ, cú pháp, ngữ nghĩa được định nghĩa hình thức dựa vào toán học. Đặc tả hình thức có thể coi là một phần của hoạt động đặc tả phần mềm. Với hình thức đặc tả này các đặc tả yêu cầu được phân tích một cách chi tiết, các mô tả trừu tượng của các chức năng chương trình có thể được tạo ra để làm rõ yêu cầu.

Đối với nhữn hệ thống tương đối phức tạp, có hai hướng tiếp cận đặc tả hình thức để phát triển các hệ thống đó là: Hướng tiếp cận đại số, hệ thống được mô tả dưới dạng các toán tử và các quan hệ và hướng tiếp cận mô hình, mô hình hệ thống được cấu trúc sử dụng các thực thể toán học như là các tập hợp và các thứ tự.

Việc sử dụng đặc tả hình thức để đặc tả một hệ thống mang lại nhiều thuận lợi. Thứ nhất đặc tả hình thức cho phép chúng ta thấy và hiểu được bản chất bên trong của các yêu cầu, đây chính là cách tốt nhất để làm giảm các lỗi, các thiếu sót có thể xảy ra ngay từ bước đầu và giúp cho công việc thiết kế được thuận lợi. Thứ hai do sử dụng suy luận toán học cho việc đặc tả nên có thể dựa vào các công cụ toán học khi phân tích và điều này làm tăng thêm tính chắc chắn và tính đầy đủ của hệ thống và cuối cùng việc đặc tả hình thức một hệ thống đem lại một cách thức cho việc kiểm tra hệ thống đó một cách thuận tiện.

Việc đặc tả một hệ thống cần đảm bảo những nguyên tắc: Phân tách chức năng với cài đặt; đặc tả hệ thống hướng tiến trình; đặc tả hệ thống phải là một mô hình nhận thức; đảm bảo đặc tả mô tả được sự vận hành của hệ thống; khi đặc tả phải tính toán đến khả năng không đầy đủ do môi trường phức tạp và cuối cùng việc đặc tả phải cục bộ và lỏng lẻo để thuận tiện cho việc thay đổi phát sinh.

2.2 Kiểm chứng hệ thống phần mềm

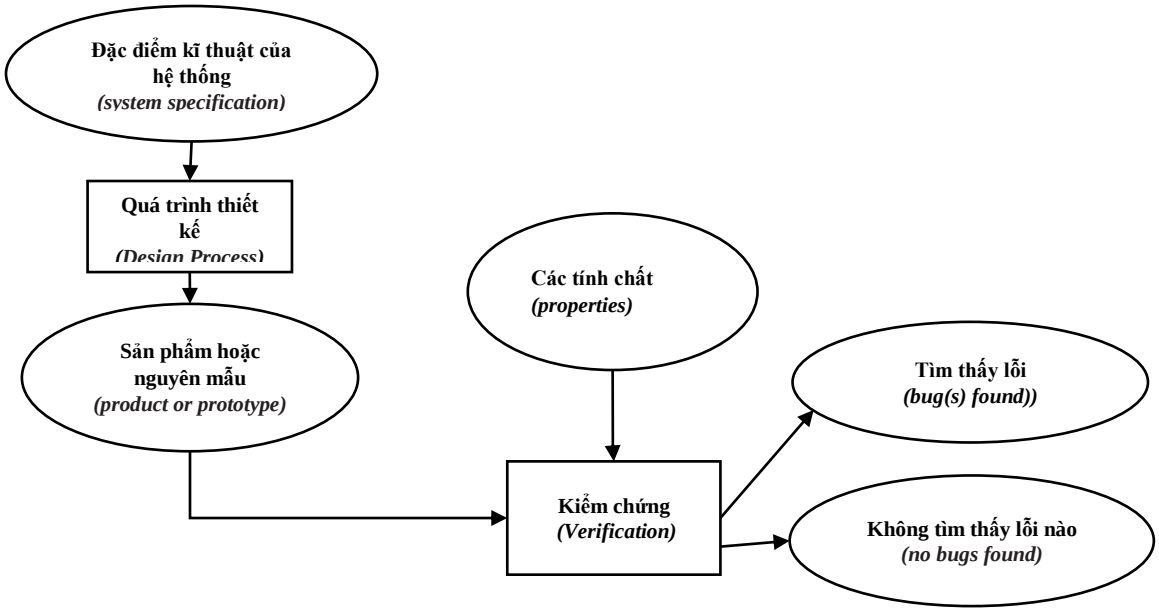
Kiểm chứng hệ thống phần mềm thực hiện việc xác minh một thiết kế hay một sản phẩm phần mềm thỏa mãn những những thuộc tính được nêu ra trong đặc tả của hệ thống [5]. Đặc tả hệ thống trở thành cơ sở của mọi hoạt động kiểm chứng. Một thiếu sót sẽ được phát hiện khi hệ thống không thỏa mãn một trong các thuộc tính đã được đặc tả. Hệ thống được xác minh là đúng khi nó thỏa mãn tất cả các thuộc tính mà nó đã được đặc tả. Quy trình kiểm chứng hệ thống được thể hiện thông qua sơ đồ (xem hình 2.1)

***Phương pháp hình thức*:** Việc áp dụng các kỹ thuật toán học vào đặc tả và phân tích hệ thống từ đó xây dựng mô hình hệ thống giúp ta xây dựng được những phương pháp kiểm chứng hệ thống dựa trên mô hình.

Model checking: Phương pháp kiểm tra chứng dựa trên mô hình, kèm theo đó là các phần mềm tự động hóa các bước kiểm chứng khác nhau. Cơ sở của phương pháp kiểm tra mô hình là các hành vi của hệ thống được mô tả một cách rõ ràng không bị nhập nhằng, đây cũng chính là điểm mấu chốt giúp phát hiện những điểm

mâu thuẫn, chưa hoàn thiện trong đặc tả phi hình thức của hệ thống. Việc sử dụng phương pháp kiểm tra mô hình đặc biệt quan trọng đối với các hệ thống

cần có tính toàn vẹn cao, góp phần đảm bảo rằng quá trình phát triển hệ thống sẽ

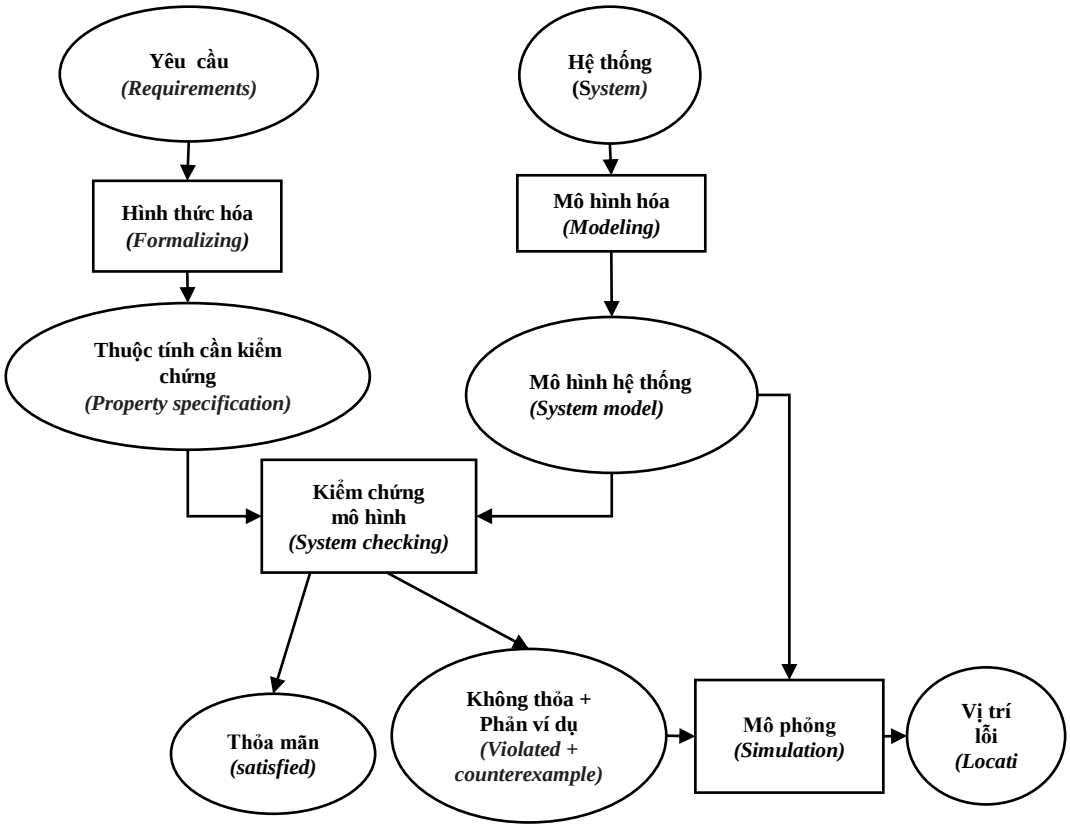


Hình 2.1 Sơ đồ việc kiểm chứng hệ thống [5]

không có lỗi. Phương pháp này đặc biệt hiệu quả tại giai đoạn đầu của quá trình phát

triển đó là ở các mức yêu cầu và đặc tả hệ thống, nhưng bên cạnh đó nó cũng được sử dụng cho một quá trình phát triển hoàn chỉnh của một hệ thống.

Hoạt động của phương pháp kiểm tra mô hình được thể hiện trong sơ đồ hoạt động của phương pháp kiểm tra mô hình (xem hình 2.2). Theo đó để kiểm chứng hệ thống ta xuất phát từ yêu cầu hệ thống ta đặc tả hình thức hệ thống, biểu diễn các thuộc tính cần kiểm tra cảu hệ thống. Cùng với đó ta mô hình hóa hệ thống, khi đó bộ công cụ kiểm chứng sẽ sinh ra tất cả các trạng thái của hệ thống và kiểm tra hệ thống có thỏa mãn các thuộc tính cần kiểm tra hay không, nếu không công cụ sẽ chỉ ra vị trí dẫn đến lỗi đó.



Hình 2.2 Sơ đồ hoạt động của phương pháp kiểm tra mô hình [5]

2.3 Ô-tô-mát thời gian

Một ô-tô-mát thời gian (Time automata-TA) là một tập gồm sáu thành phần (L, l_0, C, A, E, I) [1, 2, 5, 12], trong đó:

- L: tập các trạng thái
- l_0: trạng thái ban đầu
- C: tập các đồng hồ
- A: tập các hành động
- $E \in L \times A \times B(C) \times 2^C \times L$: tập các cạnh giữa các trạng thái
- I: $L \to B(C)$: chỉ định bất biến cho các trạng thái.

- Về mặt ngữ nghĩa:

Một ô-tô-mát thời gian được hiểu như một hệ dịch chuyển được gán nhãn $< S, s_0, \to >$, trong đó:

- $S \subseteq L \times R$ là tập các trạng thái.
- $s_0=(l_0,u_0)$ là trạng thái ban đầu.
- $\rightarrow S \times \{\mathrm{IR}_{\geq 0} \wedge A\} \times S$ là sự truyền trạng thái có quan hệ như sau:
- (l,u+d) nếu $0 \leq d' \leq d = u + d' \in I(l)$.
- (l',u') nếu có $e = (l, a, g, r, l') \in E$ sao cho $u \in g, u' = [r \rightarrow 0]u$ và $u' \in I(l)$ mà $d \in R_{\geq 0}, u + d$ ánh xạ mỗi đồng hồ x trong C đến giá trị u(x)+d, và $[r \rightarrow 0]u$ biểu thị giá trị đồng hồ, mỗi giá trị trên đồng hồ là r=0 và u nếu u thuộc C\r.

- Một mạng ô-tô-mát thời gian bao gồm nhiều các ô-tô-mát thành phần được biểu diễn như sau [1, 2, 5, 12, 14]:

Cho $A_i = \left(L_i,\ l_i^o,\ C,\ A,\ E_i,\ I_i\right)$. trong đó $1 \leq i \leq n$, một vectơ trạng thái $\bar{l} = (l_1, l_2, ..., l_n)$, $\bar{l}[l_i / l'_i]$ kí hiệu của véc tơ trạng thái này là trạng thái l'_i thay thế cho trạng thái l_i.

Ngữ nghĩa được định nghĩa như một hệ chuyển dịch $< S, s_0, \rightarrow >$ trong đó $S = (l_1, l_2, ..., l_n) \times R^C$ là tập các trạng thái, $s_0=(l_o,u_0)$ là trạng thái ban đầu, và $\rightarrow S \times S$ có quan hệ truyền như sau:

- $\left(\bar{l}, u\right) \rightarrow \left(\bar{l}, u + d\right)$ với $0 \leq d' \leq d, u + d' \in I\left(\bar{l}\right)$
- $\left(\bar{l}, u\right) \rightarrow \left(\bar{l}[l'_i / l_i], u'\right)$ với $l_i \xrightarrow{g,r} l'_i \quad s,t,u \in g, u' = [r \rightarrow 0]u$ và $u' \in I(\bar{l})$
- $\left(\bar{l}, u\right) \rightarrow \left(\bar{l}[l'_j / l_j, l'_i / l_i], u'\right)$ với $l_i \xrightarrow{g,r} l'_i$ và $l_j \xrightarrow{g_j, r_j} l'_j$ $s,t,u \in g_i; g_j, u' = \left[r_i \cup r_j \rightarrow 0\right]u$ và $u' \in I(\bar{l})$

CHƯƠNG 3. ĐẶC TẢ VÀ KIỂM CHỨNG TRONG UPPAAL

3.1 Bộ công cụ Uppaal

3.1.1 Giới thiệu về bộ công cụ Uppaal

Uppaal là một phần mềm để kiểm chứng những hệ thống thời gian thực, được phát triển bởi Đại học Uppsala và Đại học Aalborg [4]. Phần mềm được ứng dụng thành công trong những nghiên cứu ở nhiều lĩnh vực như bộ điều khiển, giao thức truyền thông hay ứng dụng multimedia – những lĩnh vực mà yếu tố thời gian là rất quan trọng. Công cụ này dùng để kiểm định những hệ thống được mô hình hóa thành hệ thống những automat thời gian với những biến số nguyên, cấu trúc dữ liệu, hàm người dùng, và đồng bộ các kênh [7, 8].

Phiên bản đầu tiên của Uppaal ra đời vào 1995. Kể từ đó phần mềm đã phát triển không ngừng để theo kịp với những tiến bộ về cấu trúc dữ liệu, giảm bậc hệ thống, bản phân phối của Uppaal, hỗ trợ tính năng tối thiểu hóa chi phí, hỗ trợ UML. Bản phát hành chính thức hiện nay là 4.0.12. Nó hỗ trợ Java và có phần kiểm chứng được viết bằng ngôn ngữ **C++**. Cho đến nay, bộ công cụ Uppaal được xem là bộ công cụ kiểm chứng tự động tốt nhất hiện nay, được sử dụng rộng rãi trong công nghiệp [13].

Cài đặt Uppaal khá đơn giản, ta chỉ cần tải zip-file Uppaal phát hành chính thức hiện nay (bản 4.0.12) về máy (lưu ý là máy đã cài đặt Java và C++). Để cài đặt, giải nén zip-file, khi đó sẽ tạo thư mục uppaal-4.0.12 có chứa ít nhất các tập tin uppaal, uppaal.jar, vàthư mục lib, bin-Linux, Win32-bin, lib, và demo. Các bin-thư mục tất cả cần có các verifyta hai tập tin (exe). Và máy chủ (exe). cộng với một số tập bổ sung, tùy thuộc vào nền tảng. Các thư mục demo-nên chứa một số demo file với xml và hậu tố.. q.

Lưu ý rằng UPPAAL sẽ không chạy mà không có Java 2 cài đặt trên máy chủ hệ thống. Java 2 cho SunOS, Windows95/98/Me/NT/2000/XP, và Linux có thể được download từ http://java.sun.com. Các phiên bản hiện tại của UPPAAL hiện không có phiên bản hỗ trợ, nó chạy trên môi trường Java (JRE), cần phải sử dụng phiên bản gần đây nhất sẵn sàng cho nền tảng đang sử dụng. Để chạy trên các hệ thống Windows 95/98/ME/NT/2000/XP mở tập tin uppaal.jar đã tải về [7, 8, 16].

3.1.2 Tổng quan về bộ công cụ Uppaal

UPPAAL sử dụng kiến trúc máy trạm-máy chủ mà trong đó phân chia chương trình ra gồm hai phần: giao diện đồ họa và hệ thống kiểm tra mô hình [1]. Giao

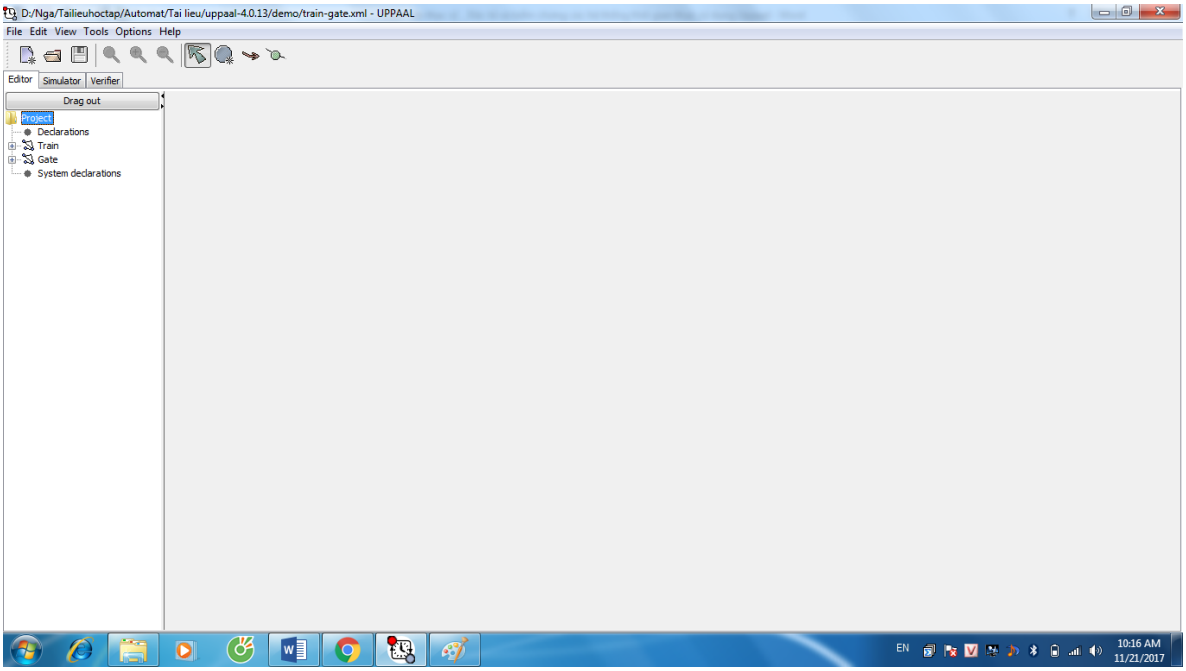
diện đồ họa, hay client, viết bằng Java, và server được đóng gói tùy vào hệ điều hành (Linux, Windows, Solaris). Do có kiến trúc như vậy nên hai phần của chương trình sẽ kết nối với nhau qua giao thức TCP/IP [7].

3.1.2.1 Java Client

Ý tưởng của chương trình là mô hình hóa hệ thống ô-tô-mát thời gian sử dụng công cụ đồ họa, mô phỏng và kiểm chứng sự hoạt động của nó, và cuối cùng là kiểm tra xem nó có thỏa mãn những tính chất cho trước hay không. Giao diện đồ họa - GUI (graphical user interface) của Java client thực hiện ý tưởng này bằng cách chia thành 3 phần: Khung soạn thảo (Editor), Mô phỏng (Simulator) và Kiểm chứng (Verifier) được xếp vào các thanh công cụ (Tab). Cụ thể các phần có các chức năng như sau [7, 16]:

- **Khung soạn thảo**

Khung soạn thảo của hệ thống được định nghĩa là một tập hợp các ô-tô-mát thời gian được tiến hành song song gọi là quá trình. Quá trình được tạo ra từ một khuôn mẫu (template) định trước.



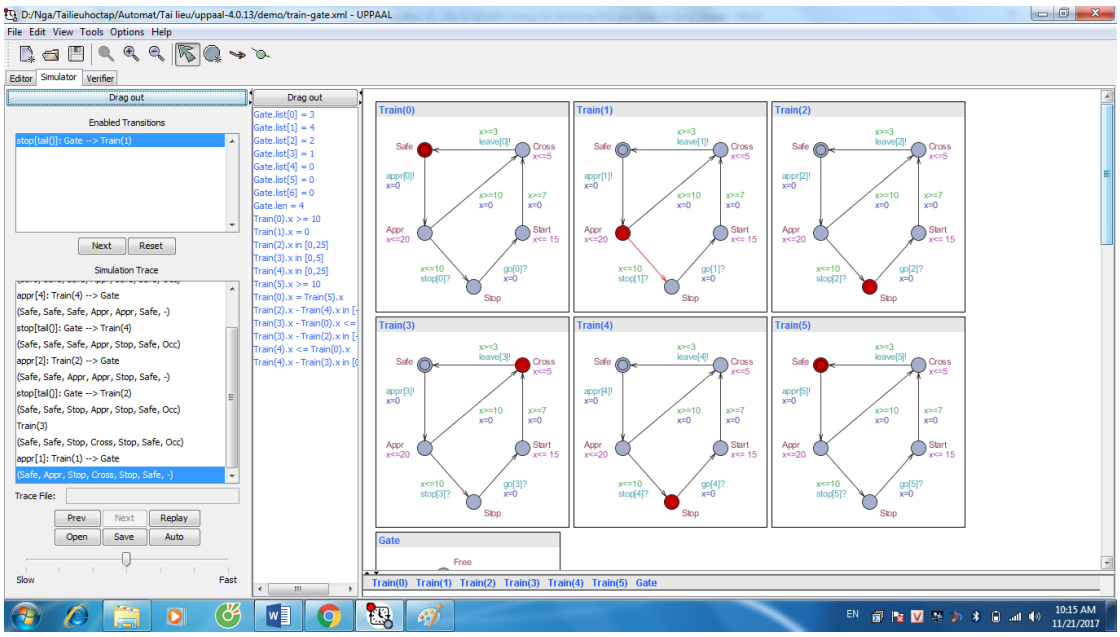
Hình 3.1. Màn hình khung soạn thảo của Uppaal.

Khung soạn thảo được chia làm hai phần: phần cửa sổ dạng cây để chọn các template, khai báo khác nhau và phần còn lại là bản vẽ để vẽ các ô-tô-mát thời gian ứng với từng khuôn mẫu. Thực hiện soạn thảo một hệ thống về mặt bản chất

là ta đang viết chương trình (code) cho hệ thống đó bằng công cụ Uppaal. Việc code này được thực hiện trong hai phần [7]. (xem hình 3.1)

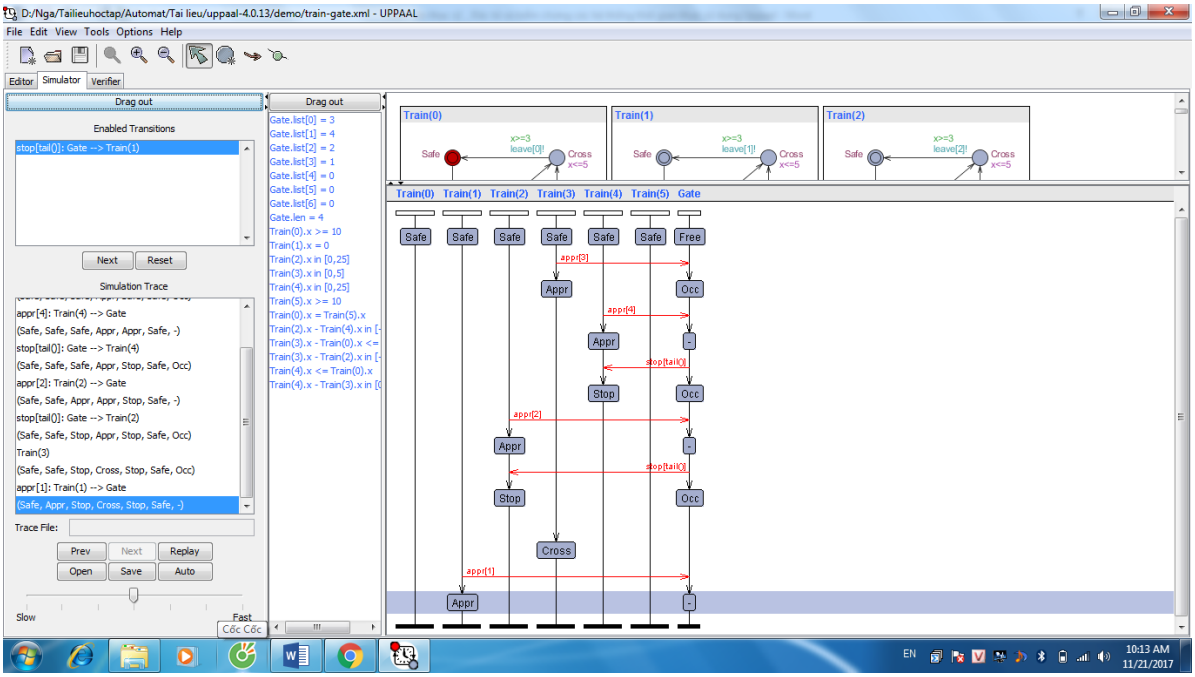
Với sơ đồ cây bên trái màn hình cho phép ta lựa chọn biểu diễn nhiều phần thông tin của hệ thống, trong đó tính năng Global Declaration cho phép ta khai báo biến toàn cục, đồng hồ, kênh đồng bộ và hằng, tính năng Template declaration để có khai báo biến, kênh và hằng cục bộ. Tính năng Process Assignment từ Template ta tạo ra các quá trình, khi gán quá trình các Template sẽ được gán với các biến kèm theo. Tính năng System Definition cho phép khai báo danh sách những quá trình trong hệ thống [7]. Với phần đồ họa bên phải cho phép ta vẽ các ô-tô-mát thời gian cho từng quá trình, tại đây ta biểu diễn các trạng thái của các quá trình và các bước chuyển trạng thái. Sự vận hành của các ô-tô-mát này còn phụ thuộc vào những khai báo mà ta đã khai báo trong Template declaration ở bên trái [7][8].

- **Mô phỏng (Simulator):** Chức năng mô phỏng cho phép mô phỏng việc thực thi hệ thống một cách ngẫu nhiên, đồng thời thông qua mô phỏng này, bước đầu ta có thể kiểm tra được hệ thống có vận hành được không, có xung đột gì không.



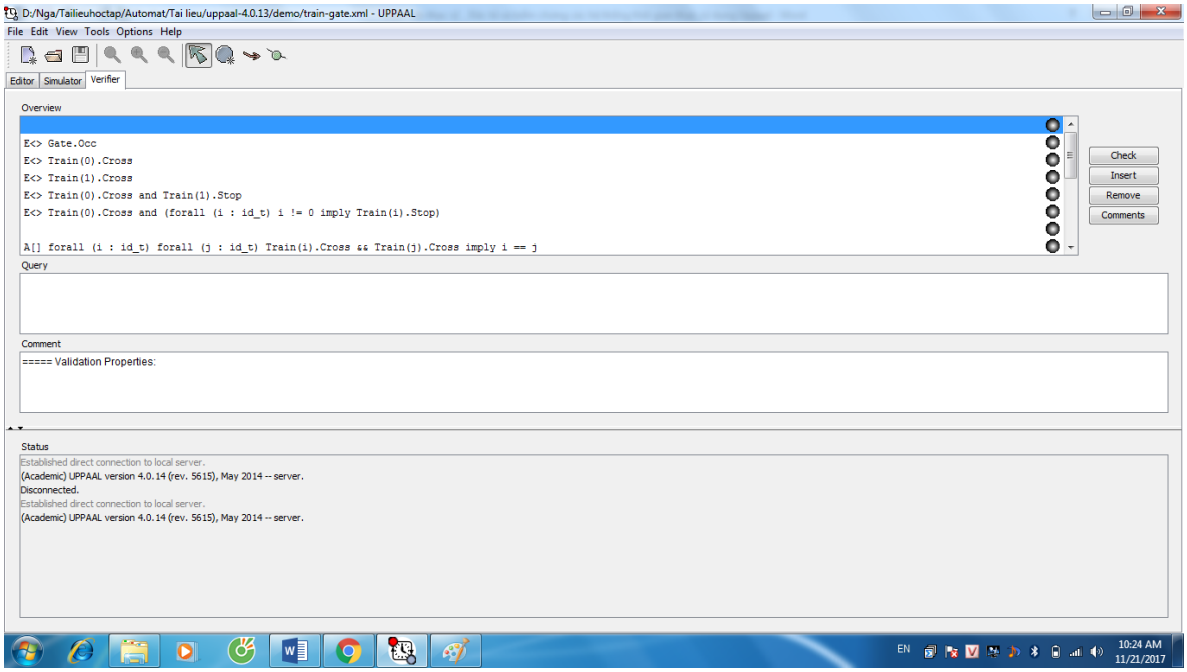
Hình 3.2. Màn hình khung mô phỏng các bước chuyển trạng thái của các quá trình của hệ thống Train-Gate trong Uppaal.

Trong tính năng mô phỏng này công cụ cho phép ta kiểm tra các bước chuyển trạng thái của các quá trình cũng như bước chuyển của hệ thống theo thời gian [7, 8] (xem hình 3.2; 3.3).



Hình 3.3. Màn hình khung mô phỏng các bước chuyển của hệ thống theo thời gian Train-Gate trong Uppaal.

- **Kiểm chứng (Verifier):**



Hình 3.4. Màn hình khung kiểm chứng của hệ thống Train-Gate

Chức năng kiểm hứng cho phép ta kiểm chứng tính thực hiện được của hệ thống, tính đến được của các trạng thái trong mô hình, tính đúng đắn của hệ thống, kiểm chứng hệ thống có bị deadlock không bằng những câu lệnh cụ thể theo ngôn ngữ của Uppaal, những câu lệnh này được viết theo cú pháp của TCTL. Màn hình của phần kiểm chứng được chia thành ba vùng: Overview cho phép hiển thị các câu lệnh đã gõ khi kiểm chứng; Query cho phép gõ câu lệnh; Comment cho phép ghi chú thích cho câu lệnh kiểm chứng và Status hiển thị kết quả của kiểm chứng [7, 8, 16]. (xem hình 3.4)

Để hỗ trợ cho các thao tác trên, Java client thiết kế giao diện khá thân thiện giống như màn hình soạn thảo của Microsoft Word khiến cho việc viết chương trình trong Uppaal trở nên dễ dàng hơn.

Trên cùng của màn hình là thanh thực đơn (Menu) bao gồm các thành phần: File; Edit; View; Tools; Option; Help. Trong mỗi thành phần này cho phép thực hiện các thao tác cụ thể sau:

- Menu File được biểu diễn ở tận cùng bên trái của thanh menu, chủ yếu được sử dụng để mở và lưu lại (một phần của) mô tả hệ thống hay chi tiết kỹ thuật yêu cầu được tạo ra trong UPPAAL. Các mục có sẵn là:

- New System (hệ thống mới): tạo một hệ thống mới
- Open System (mở hệ thống): tải một hệ thống đã có từ tập tin, các yêu cầu đặc điểm kỹ thuật tương ứng (nghĩa là cùng một tên tập tin, nhưng với các hậu tố .q) được nạp vào xác minh, nếu nó tồn tại.
- Save System (Lưu hệ thống): hệ thống lưu trong trình soạn thảo cho tập tin.
- Save System As (lưu hệ thống vào địa chỉ cụ thể): lưu hệ thống trong trình soạn thảo một tập tin chỉ định.
- Import Template: Một cửa sổ hộp thoại sẽ được hiển thị cho phép một tập hợp các sẵn các mẫu để được nhập vào.
- Export Template: Hiện tại mẫu ở định dạng tập tin Encapsulated Postscript.
- New Queries: đặc tả yêu cầu biên tập với một tập tin trống.
- Open Queries (mở câu hỏi): Tải một bộ hiện có của các chi tiết kỹ thuật yêu cầu từ tập tin.
- Save Queries(lưu câu hỏi): Để lưu thông số kỹ thuật yêu cầu trong biên tập các tập tin.

- Save Queries As(lưu truy vấn theo): Chi tiết kỹ thuật yêu cầu trong trình soạn thảo để một tên file được chỉ định.
- Exit (thoát):

• Menu Edit. Các thành phần menu edit cung cấp một tập hợp các lệnh được hỗ trợ trong trình soạn thảo hệ thống. Mục này bao gồm các chức năng là: Hoàn tác (Undo), làm lại (Redo), Cắt (Cut), Sao chép (Copy), Dán (Paste), Xóa (Delete), Chèn khuôn mẫu (Insert Template), Xóa khuôn mẫu (Remove Template).

• Menu View. Các thành phần Menu View được sử dụng để sửa đổi sự xuất hiện của hệ thống hiện đang được hiển thị trong hệ thống biên tập. Mục này gồm các chức năng cụ thể sau:

- Zoom: hiển thị một menu phụ với zoom giá trị cố định, zoom để phù hợp với phóng to trong, trên, hoặc để kích thước bình thường. Một sự thay đổi trong giá trị zoom ảnh hưởng đến các mẫu trình biên tập hoặc các quá trình trong giả lập (nếu một trong những công cụ được kích hoạt).

- Labels:hiển thị một menu phụ mà từ đó ta có thể chọn những loại nhãn sẽ được hiển thị trong vùng vẽ. Ngay cả khi ẩn, tất cả các nhãn có thể được nhìn thấy trong *tooltip* của địa điểm và góc cạnh.

- Show Grid: lưới vẽ được hiển thị khi mặt hàng này được kiểm tra.

- Snap to Grid: làm cho các đối tượng bản vẽ mới (như địa điểm, nhãn) sắp xếp cho lưới điện snap. Kích thước của lưới snap có liên quan đến kích thước của lưới vẽ.
Ở đây ta chú ý rằng các tùy chọn Grid Snap để có thể sử dụng ngay cả khi lưới bản vẽ không được hiển thị.

- Reload Simulator: Tác phẩm của hệ thống hiện đang được nạp vào trình soạn thảo, để giả lập và xác minh các.

- Processes: hiển thị một cửa sổ hộp thoại cho ẩn và hiện các quy trình trong bảng điều khiển quá trình của mô phỏng này.

- Variables: hiển thị một cửa sổ hộp thoại cho ẩn và hiển thị các biến trong bảng các biến của mô phỏng này.

- Full DBM: hiển thị tất cả các hạn chế trên đồng hồ trong bảng các biến của mô phỏng này. Nếu không được chọn một số lượng tối thiểu các khó khăn sẽ được hiển thị.

• Menu Tool (công cụ): Các thành phần Menu Tool chứa một bộ công cụ hữu ích trong hệ thống soạn thảo, thành phần này gồm các chức năng cụ thể sau:

- Check syntax (Kiểm tra cú pháp): kiểm tra nếu tìm thấy bất kỳ lỗi nào đều được cảnh báo và liệt kê ở phần dưới của khu vực vẽ của hệ thống biên tập.

- Convert syntax (Chuyển đổi cú pháp): hỗ trợ trong một hệ thống phù hợp với các cú pháp được sử dụng trong UPPAAL 3.4 đến cú pháp hiện hành.

- Align to Grid: làm cho tất cả các đối tượng hiện có của mẫu hiện tại hiển thị trên lưới.

• Menu option: Các thành phần Menu Option gồm các menu tùy chọn các thiết lập để kiểm soát kiểm tra mô hình, bao gồm:

- Search Order
- State Space Reduction
- State Space Representation
- Diagnostic Trace
- Extrapolation
- Hash table size
- Reuse

• Menu help: Danh sách trợ giúp có hai mục, trợ giúp mở ra một cửa sổ riêng biệt hiển thị các trang trợ giúp (Help…F1), và mở ra một cửa sổ hiển thị số phiên bản và thông tin về bản quyền của UPPAAL (About Uppaal)

Trong màn hình giao diện của Uppaal cũng có thiết kế các thanh công cụ (Tool Bar), Thanh công cụ của Uppaal được thiết kế nằm ngay dưới thanh thực đơn (Menu file). Thanh công cụ được chia thành ba nhóm trong đó hai nhóm tận cùng bên trái cung cấp truy cập nhanh vào một số các trình đơn được sử dụng các mục thường xuyên nhất, nhóm bên phải chứa các công cụ chỉnh sửa. Nhóm đầu tiên chứa các nút sau đây: *New*, *Open Project*, và *Save*. Những tính năng này được mô tả cụ thể trong menu File. Nhóm thứ hai chứa các nút sau đây: *Phóng to để Fit*, *Zoom In*, và *Zoom Out*. Tính năng này được mô tả trong phần menu View . Nhóm thứ ba bao gồm các công cụ được sử dụng trong trình soạn thảo để chọn và di chuyển các yếu tố của một *automaton*, và để thêm địa điểm, các cạnh và nhãn. Tính năng này được mô tả kĩ hơn trong phần thực hiện thao tác vẽ ô-tô-mát thời gian [6, 7, 15].

3.1.2.2 Stand-alone Verifier

Khi thực hiện những bài toán lớn, khó có thể làm hết được trong GUI. Khi đó ta có thể dùng kiểm chứng riêng bằng dòng lệnh bằng chương trình verifier. Nó cho phép ta thực hiện chức năng tương tự như GUI bằng các dòng lệnh. Cách này thích hợp cho những máy có cấu hình yếu và phải chia sẻ bộ nhớ cho những ứng dụng khác [7, 8, 13].

3.2 Mạng Ô-tô-mát thời gian trong Uppaal

3.2.1 Ô-tô-mát thời gian trong Uppaal

Ô-tô-mát thời gian thực là một máy hữu hạn trạng thái với một tập các đồng hồ. Mỗi đồng hồ là một hàm số ánh xạ vào một tập số thực không âm, nó ghi lại thời gian trôi qua giữa các sự kiện. Các đồng hồ được đồng bộ hóa về mặt thời gian. Trong UPPAAL, một hệ thống được mô hình là mạng của những automat thời gian sắp xếp song song. Mô hình được mở rộng với các biến rời rạc bị chặn ở trạng thái. Những biến này được sử dụng trong ngôn ngữ lập trình: có thể đọc, ghi và thực hiện các phép tính số học. Một trạng thái của hệ thống định nghĩa bởi vị trí của các automat, giá trị đồng hồ và các biến rời rạc. Mỗi automat có thể thay đổi (không nên hiểu là sự chuyển tiếp) độc lập hay đồng bộ với một automat khác, dẫn đến một trạng thái khác [5, 7, 8, 13].

Ô-tô-mát thời gian trong Uppaal được mở rộng với các tính năng cụ thể sau:

- Templates Automat có thể được khai báo với tập những tham số nhận mọi kiểu giá trị (int, char..). Những tham số này thay thế cho đối số trong khai báo quá trình.
- Constants khai báo là const name value. Hằng không thể thay đổi và phải có giá trị là số nguyên.
- Bounded Integer Variables biến nguyên bị chặn, được khai báo là int[min, max] name, min và max là cận trên và dưới. Guard, invariant và assignment có thể diễn tả bởi các biến này. Nếu vi phạm các cận thì sẽ dẫn tới trạng thái không tồn tại. Nếu không có cận thì mặc định là từ -2768 đến 32768.
- Binary synchronisation kênh khai báo là chan c. Một cạnh dán nhãn c! đồng bộ với nhãn khác là c?. Cặp đồng bộ chọn ngẫu nhiên nếu có nhiều khả năng kết hợp xảy ra.

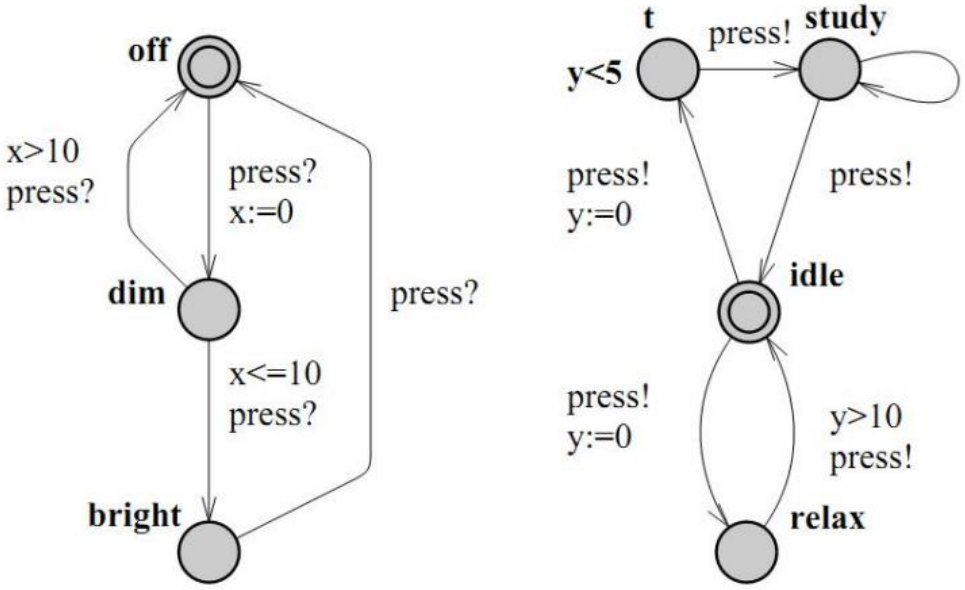
- Broadcast Channel khai báo là broadcast chan c. Trong broadcast synchronisation một người gửi c! có thể đồng bộ với số lượng tùy ý các người nhận c?
- Urgent Synchronisation kênh được khai báo khi thêm vào khai báo kênh cú pháp urgent. Nếu có kênh có nhãn urgent thì sẽ không có trễ khi đồng bộ chuyển tiếp.
- Urgent Location tương đương với một đồng hồ thêm x về mặt ngữ nghĩa, reset ở mọi cạnh đến và có bất biến. Do đó, thời gian không được thay đổi khi hệ thống đang ở Urgent location.
- Committed Location điều kiện còn chặt chẽ hơn urgent. Một trạng thái là committed nếu mọi location trong trạng thái đó là committed.
- Arrays dùng được cho đông hồ, kênh, hằng và biến nguyên.
- Initialisers dùng để khởi tạo biến nguyên và mảng của nó.
- Record types khai báo với struct như trong C.
- Custom types tương tự như typedef trong C
- User functions có thể khai báo toàn cục hay địa phương trong các templates. Cách sử dụng tương tự như C nhưng không có con trỏ.
- Expressions trong Uppaal quy định phạm vi của đồng hồ và các biến nguyên. Expression được dùng với các nhãn sau: Select một danh sách diễn đạt name : type trong đó name là tên biến và type là kiểu biến. Những biến này chỉ có thể truy nhập khi ở cạnh tương ứng và nhận giá trị không định trước trong kiểu của nó; Guard diễn đạt thỏa mãn những điều kiện sau: không hiệu ứng phụ , trả về dạng boolean, chỉ liên quan đến đồng hồ, biến nguyên và hằng, chênh lệch giữa các đồng hồ chỉ so sánh với diễn tả nguyên; Synchronisation diễn tả bằng Expression! hay Expression? hay nhãn trống. Biểu diễn này phải không có hiệu ứng phụ, đánh giá kênh, chỉ đến số nguyên, hằng hoặc các kênh; Update danh sách ngăn cách bởi dấu phẩy với hiệu ứng phụ, chỉ có thể chỉ đến đồng hồ, biến nguyên và hằng, có thể gọi được hàm; Invariant thỏa mãn những điều kiện sau: không hiệu ứng phụ, chỉ đồng hồ, biến nguyên và hằng là liên quan.

3.2.2 Mô hình mạng các ô-tô-mát thời gian trong Uppaal

Một mạng các ô-tô-mát thời gian là một sự ghép nối song song của một tập các ô-tô-mát thời gian (được goi là các tiến trình). Việc ghép nối giữa các tiến trình trong mạng thông qua giao tiếp đồng bộ và không đồng bộ: Giao tiếp đồng bộ giữa

các tiến trình thông qua việc bắt tay đồng bộ giữa chúng sử dụng kênh đồng bộ; Giao tiếp bất đồng bộ được thực hiện bởi các biến chia sẻ.

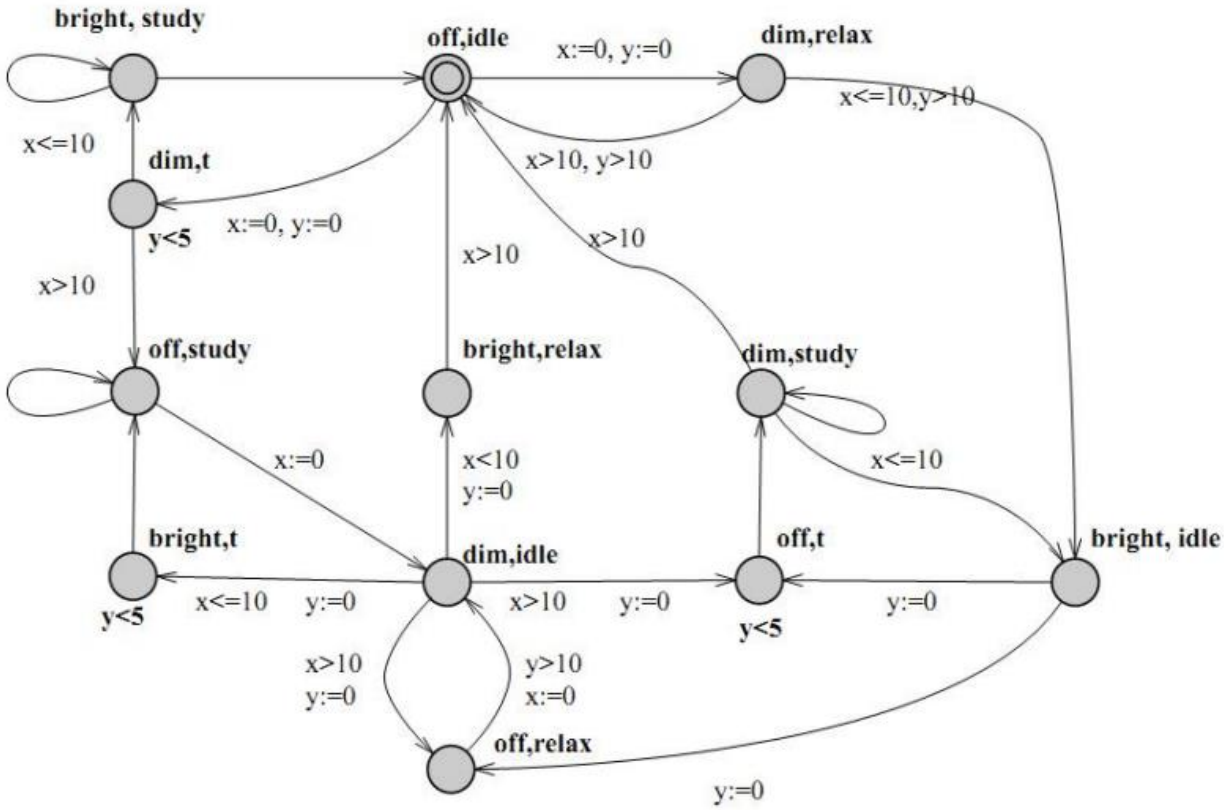
Để mô hình hóa việc đồng bộ giữa các tiến trình, Uppaal đề xuất bảng kí hiệu hành động bao gồm các kí hiệu hành động đầu vào kí hiệu là *a?* và đầu ra, kí hiệu là *a!* và các hành động bên trong biểu diễn bởi kí hiệu phân biệt τ.



Hình 3.5 Mạng các ô-tô-mát thời gian của hệ thống điều khiển đèn

Để minh họa cho việc ghép nối các tiến trình trong Uppaal ta xét một mạng ô-tô-mát thời gian của mô hình một hệ thống điều khiển đèn gồm công tắc bật đèn phụ thuộc thời gian (bên trái) và người dùng (bên phải). Người dùng và công tắc giao tiếp với nhau sử dụng nhãn press (nhấn). Người dùng có thể nhấn công tắc (press!) và công tắc đợi được nhấn (press?). Ô-tô-mát tích, tức là ô tô mát mô tả hệ thống được kết hợp (xem hình 3.6)

Ngữ nghĩa của các mạng được cho bởi một ô-tô-mát thời gian đơn theo khía cạnh các hệ dịch chuyển. Một trạng thái của mạng là một bộ *(l,u)* với *l* là véc tơ các vị trí hiện tại của mạng, và *u* là một phép gán giá trị đồng hồ thông thường theo định nghĩa. Một mạng có thể thực hiện hai kiểu dịch chuyển là dịch chuyển tại chỗ (delay) và các dịch chuyển rời rạc. Luật cho dịch chuyển tại chỗ (giữ nguyên vị trí) là tương tự với trường hợp ô tô mát đơn lẻ với bất biến của một véc tơ vị trí là giao của các bất biến của các tiến trình.



Hình 3.6 Ô-tô-mát tích của ô-tô-mát công tắc đèn và người dùng (hình 3.5)

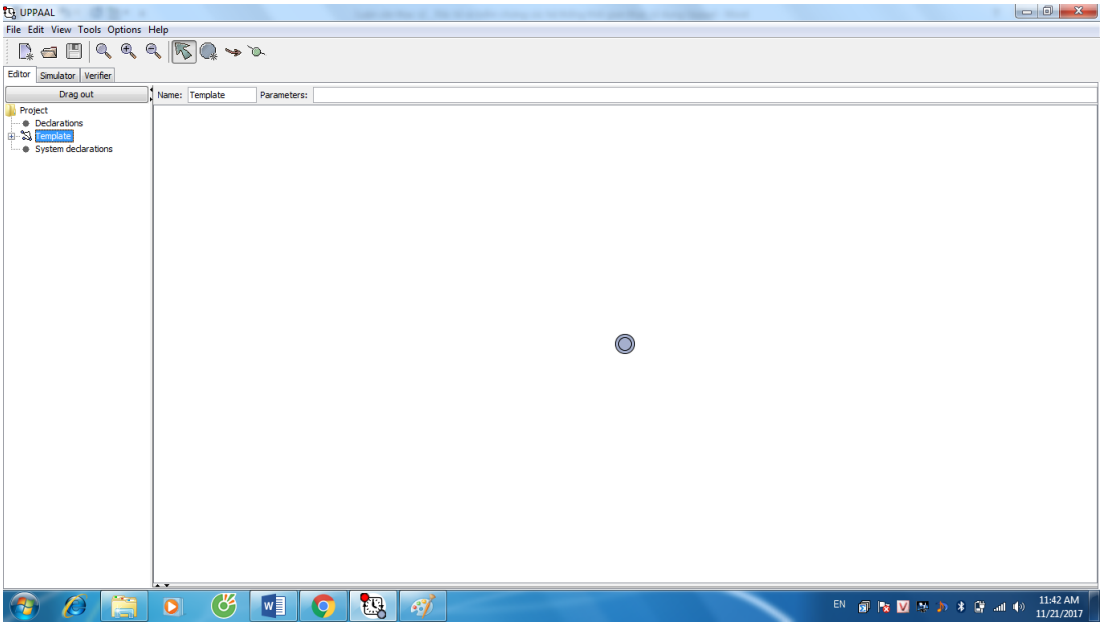
3.3 Đặc tả trong Uppaal

Việc đặc tả hệ thống thời gian thực trong Uppaal là một đặc tả hình thức. Trong công cụ Uppaal một hệ thống có yếu tố thời gian được mô hình hóa thành một mạng ô-tô-mát thời gian gồm các ô-tô-mát thời gian xếp song song có thể hoạt động độc lập hoặc đồng bộ với nhau qua các kênh đồng bộ.

Việc biểu diễn một mạng ô-tô-mát thời gian trong Uppaal được thực hiện thông qua khung soạn thảo. Các mô tả của mô hình bao gồm ba phần: khai báo cục bộ và toàn cục; các khuôn mẫu ô-tô-mát và định nghĩa hệ thống. Để biểu diễn mạng ô-tô-mát thời gian trong Uppaal trên khung soạn thảo ta cần tiến hành các bước sau [3, 8, 10, 12]:

Bước 1:Phân tích và nhận diện các khuôn mẫu có trong hệ thống: Cần xác định trong hệ thống có những khuôn mẫu nào, mỗi khuôn mẫu sẽ ứng với một quá trình và được biểu diễn là một ô-tô-mát thời gian trong khung soạn thảo.

Bước 2: Mô hình hóa các khuôn mẫu: Mỗi khuôn mẫu cần xác định rõ: Có những trạng thái nào? Bước chuyển trạng thái ra sao? Có cần truyền tham số gì không? Từ đó xác định các biến toàn cục và biến địa phương trong hệ thống.

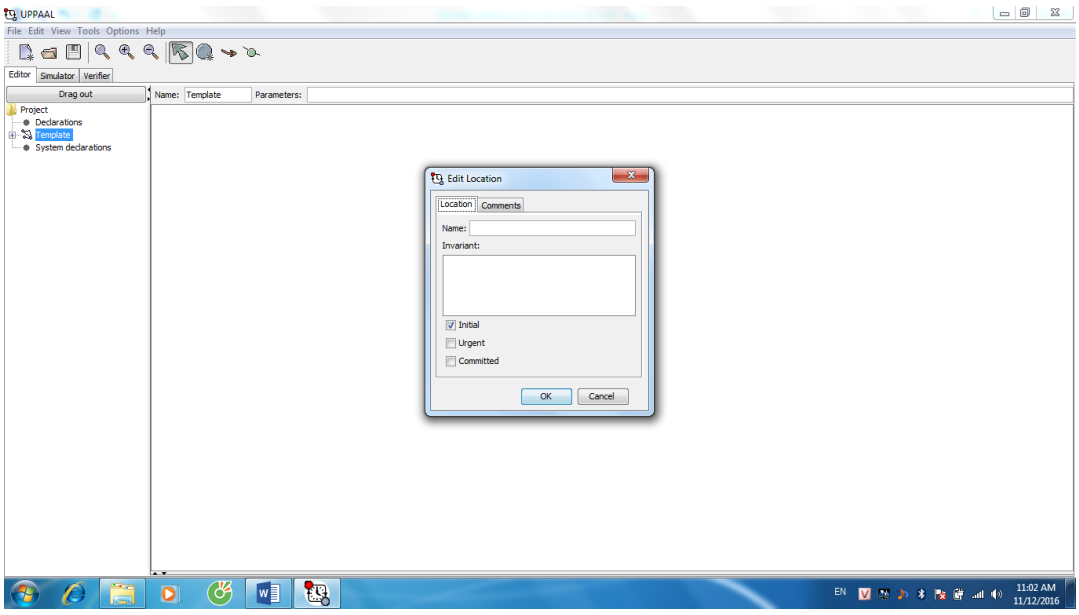


Hình 3.7 Màn hình thể hiện việc dùng nút Add location vẽ các trạng thái

Bước 3: Vẽ ô-tô-mát thời gian: Để vẽ các ô-tô-mát thời gian trong Uppaal ta cần tiến hành như sau:

- Vẽ nút: Mỗi nút thể hiện một trạng thái trong ô-tô-mát. Để vẽ nút ta dùng chuột nhấp vào biểu tượng Add location trên thanh công cụ (xem hình vẽ), sau đó nhấp chuột vào vị trí muốn vẽ trong khung vẽ, mỗi lần nhấp chuột ta được một nút. (xem hình 3.7)

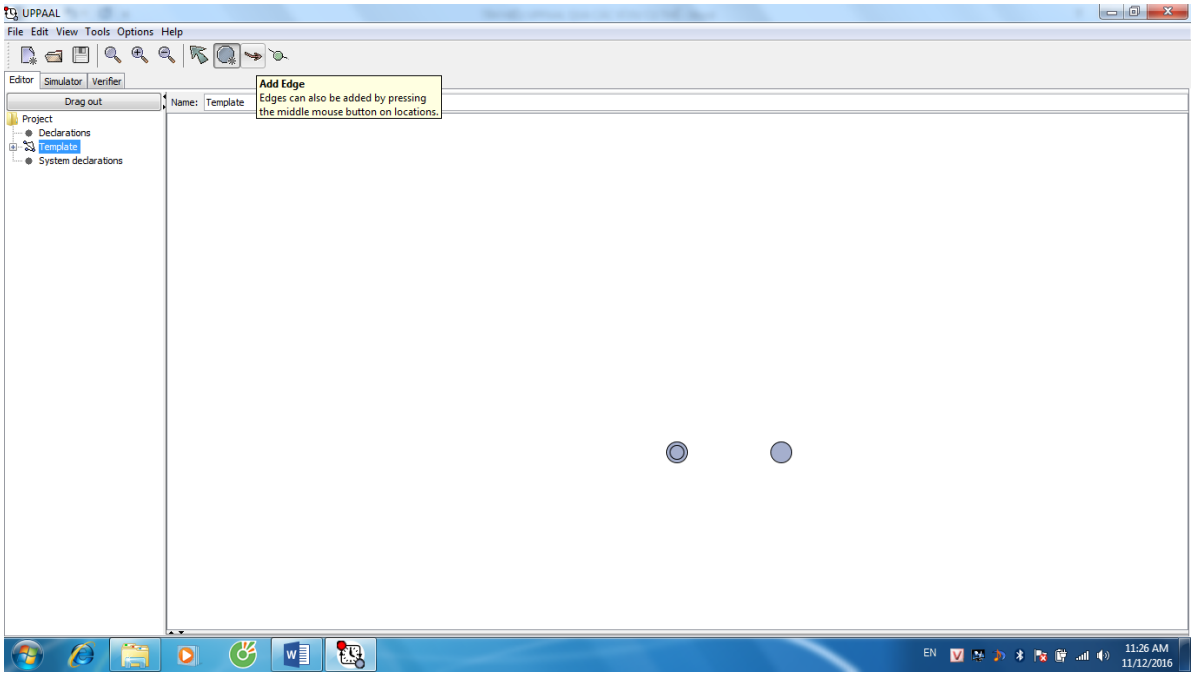
Để khai báo cho nút đó ta nhấp đúp chuột vào vị trí của nút, khi đó trên màn



Hình 3.8 Màn hình dùng chức năng Edit để khai báo cho nút:

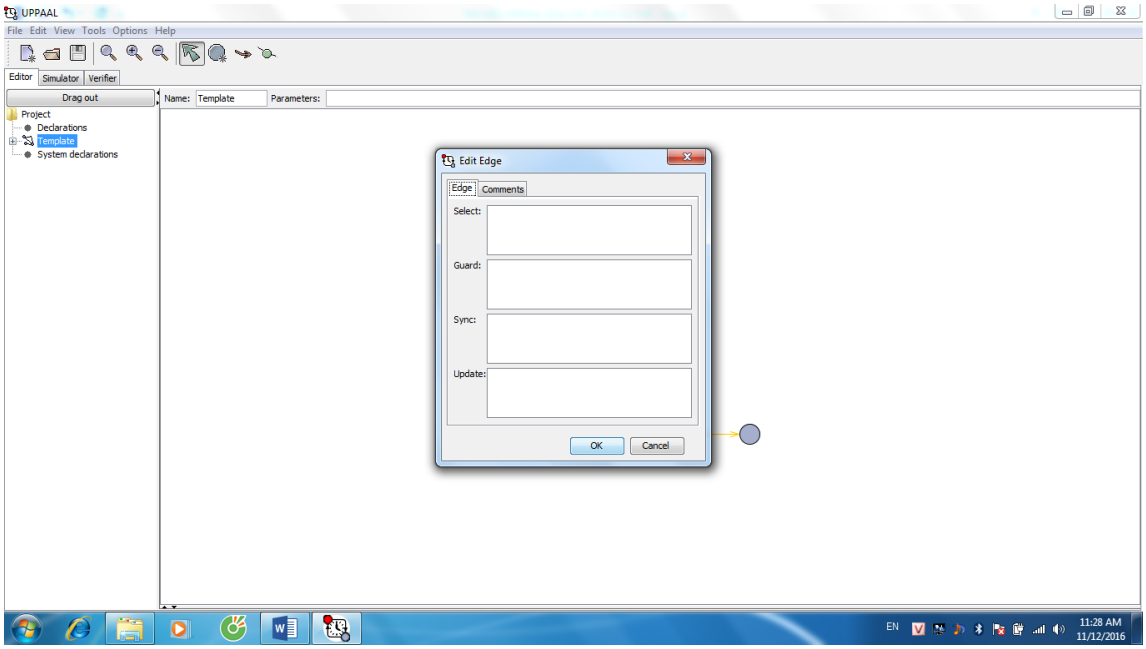
hình hiện ra một hộp hội thoại cho phép ta khai báo các đặc tính của nút gồm: Tên trạng thái (Name); điều kiện của biến-đồng hồ (Invariant). Trạng thái của nút, ở đây các trạng thái được đề xuất gồm: trạng thái Instant; trạng thái Urgent: tương đương với 1đồng hồ thêm, reset ở mọi cạnh đến và có bất biến (thời gian tại U là ko trôi); Trạng thái Committed: Điều kiện chặt chẽ hơn Urgent, không cho phép trễ, giữ nguyên ở trạng thái tiếp theo hoặc 1 Commited khác (xem hình 3.8).

- Vẽ cạnh: Mỗi cạnh trong ô-tô-mát thời gian trình bày trong Uppaal thể hiện một quá trình của ô-tô-mát thời gian đó. Để vẽ một cạnh ta nhấp chuột vào biểu tượng Add Edge trên thanh công cụ, sau đó trên khung vẽ chọn đểm đầu và cuối của cạnh bằng cách nhấp chuột vào nút tương ứng (xem hình 3.9).



Hình 3.9 Màn hình dùng lệnh Add Edge

Để khai báo các điều kiện cho một cạnh nào đó, ta nhấp đúp chuột vào vị trí của cạnh, khi đó trên màn hình hiện ra hộp hội thoại và cho phép ta khai báo các đặc tính của cạnh gồm các danh mục sau: Select cho phép khai báo danh sách biến có thể truy nhập khi ở cạnh tương ứng. Để khai báo trong mục này ta sử dụng kiểu khai báo là *name : type* trong đó name là tên biến; type kiểu biến; Guard cho phép



Hình 3.10 Màn hình dùng chức năng Edit Edge để khai báo cho cạnh

ta khai báo điều kiện cần thỏa mãn (điều kiện của đồng hồ, biến nguyên, hằng; trả về dạng boolean); Synchronisation cho phép ta khai báo tín hiệu nhận và gửi (đồng bộ giữa các ô-tô-mát) dạng Expression! (gửi); Expression? (nhận); Update cho phép ta khai báo giá trị của đồng hồ; biến nguyên; hằng khi thực hiện chuyển trạng thái. (xem hình 3.10)

3.4 Kiểm chứng trong Uppaal

Sau khi mô tả một hệ thống thời gian thực dưới dạng mô hình mạng ô-tô-mát thời gian, công cụ Uppaal cho phép mô phỏng và xác minh mô hình. Kỹ thuật kiểm chứng trong Uppaal được thực hiện bằng phương pháp kiểm tra mô hình, trong đó để kiểm chứng một tính chất nào đó của hệ thống bằng cách duyệt trong mô hình tất cả các trạng thái mà trong thời gian chạy hệ thống có thể rơi vào trạng thái đó.

3.4.1 Mô phỏng sự hoạt động của hệ thống

Sau khi biên tập hệ thống trong phần soạn thảo, Uppaal cho phép kiểm chứng hoạt động của hệ thống qua chức năng Simulator [7, 8, 13, 16].

Ở chức năng này, bước đầu người dùng phát hiện những lỗi trong thiết kế Ô-tô-mát, lỗi mã nguồn. Nếu ở bước Editor gặp các lỗi này khi chạy Simulator máy sẽ báo lỗi và chỉ rõ lỗi mắc phải qua hộp hội thoại hiện trên màn hình cũng như đánh dấu lỗi trên bản Editor bằng chuyển phông chữ sang màu đỏ và gạch chân.

Khi đó, người dùng sẽ biết lỗi ở đâu và sửa lỗi. Chỉ đến khi nào phần biên tập không còn các lỗi soạn thảo thì mới chạy được Simulator [4, 7].

Trong quá trình chạy Simulator người dùng sẽ bước đầu kiểm tra được sự vận hành của hệ thống qua sự dịch chuyển trạng thái được mô phỏng ngẫu nhiên và các trạng thái của các Ô-tô-mát song song theo thời gian [4, 7, 8]. Ở bước này nếu hệ thống bị deadlock cũng sẽ được phát hiện ra sau khi chạy thử các bước, nếu bị deadlock quá trình sẽ bị dừng và không chuyển sang bước kế tiếp được.

3.4.2 Kiểm chứng bằng dòng lệnh

Uppaal cho phép ta kiểm chứng khả năng hoạt động của ô-tô-mát qua mọi trạng thái bởi chức năng Verifier . Chức năng này cho phép ta kiểm chứng các tính năng của hệ thống thông qua các dòng lệnh (ngôn ngữ C++) [4, 8, 9, 10, 13, 15] như: Tính có thể đạt được; Tính an toàn; Tính sống; Khả năng rơi vào trạng thái deadlock (không chịu chuyển trạng thái).

Chức năng kiểm chứng trong Uppaal được thiết kế để kiểm tra một tập con của công thức TCTL (Time Computation Tree Logic) cho các mạng các ô-tô-mát thời gian. Các câu lệnh để kiểm chứng có cấu trúc như sau

A[]φ – đảm bảo rằng φ luôn luôn đúng

E[]φ − đảm bảo rằng φ luôn có thể đạt được

E<>φ− Kiểm tra tính đạt được của trạng thái φ

A <> φ − đảm bảo φ trước sau gì cũng sẽ xảy ra

φ −−> ψ – đảm bảo φ xảy ra thì trước sau gì ψ cũng sẽ xảy ra.

Với *φ, ψ* là các các thuộc tính cục bộ mà có thể được kiểm tra một cách cục bộ tại một trạng thái. Hệ dịch chuyển trạng thái của một mạng có thể được mở ra trong một cây vô hạn chứa các trạng thái và các dịch chuyển. Các ngữ nghĩa của các công thức được định nghĩa trên cây này. Ở đây các ký tự A và E được sử dụng để định lượng trên các đường dẫn: Kí tự A (tất cả - All) được sử dụng để biểu diễn rằng thuộc tính được đưa ra cần được giữ trên tất cả các đường đi trên cây trong khi đó E (tồn tại – Exist) biểu diễn rằng có ít nhất một đường đi trên cây mà thuộc tính giữ (hold). Trong khi đó ký hiệu [] và <> được sử dụng để định lượng trên các trạng thái trong đường đi: Kí hiệu [] biểu diễn rằng tất cả các trạng thái trên đường đi thỏa thuộc tính, trong khi kí hiệu <> để chỉ rằng có ít nhất một trạng thái trên đường thỏa thuộc tính, cụ thể các tính chất của hệ thống được kiểm chứng trong Uppaal và cú pháp được cho như sau:

Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định). Để kiểm chứng tính năng này ta dùng cú pháp: E<>φ (trong đó φ là công thức trạng thái).

Ví dụ: Trong hệ thống Train-Gate, để kiểm chứng tính đạt được của các trạng thái ta có thể dùng các câu lệnh như: E<> Gate.Occ để kiểm tra xem bộ điều khiển có đạt được trạng thái Occ không (có thể nhận và lưu được id của các tàu đến hay ko); E<> Train1.cross để kiểm tra xem Train1 có qua được cầu hay ko; E<> Train1.cross and Train2.Stop để kiểm xem Train1 có vượt qua cầu khi Train2 đang dừng hay ko; E<> Train0.cross and (forall (i:id_t) i!=0 imply Train(i).stop) để kiểm tra xem Train0 có vượt cầu khi tất cả các tàu khác đang dừng không.

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng). Để kiểm tra tính năng này trong Uppaal ta dùng cú pháp: A[] và E[].

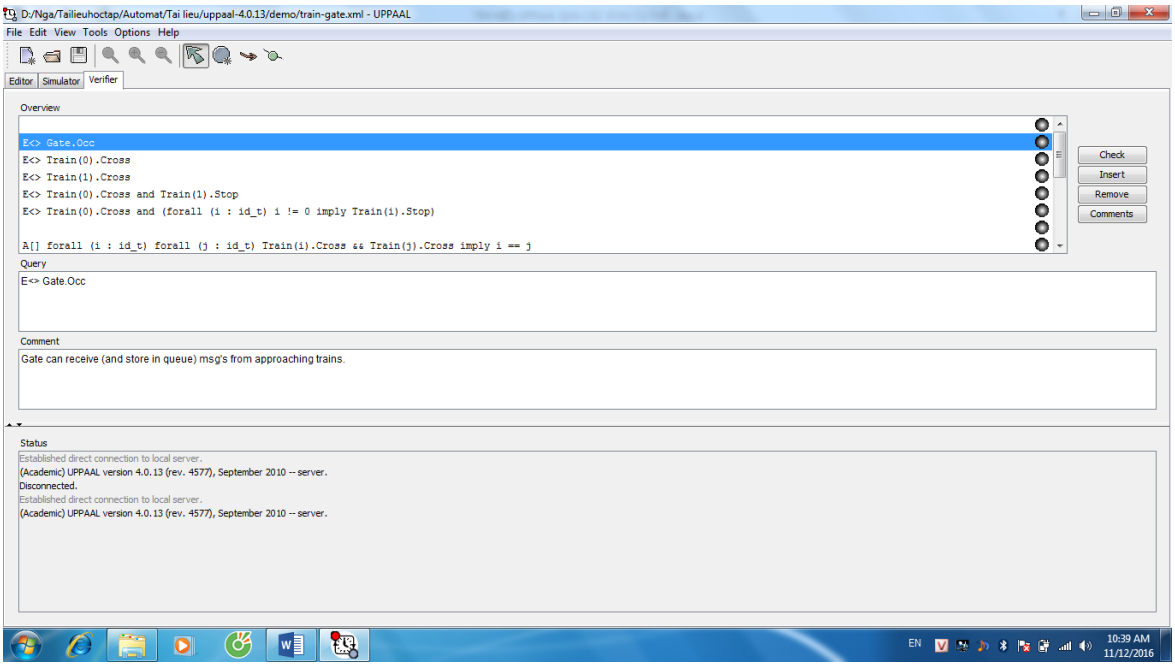
Ví dụ: Trong hệ thống Train-Gate, để kiểm chứng tính năng này ta có thể dùng các câu lệnh như: A[] Gate.list(N)==0 để chắc chắn rằng không bao giờ có N tàu trong hàng đợi; A[] forall(i:id_t) forall(j:id_t) Train(i).cross and Train(j).cross imply i==j để chắc chắn rằng hai tàu khác nhau không bao giờ cùng qua cầu tại cùng một thời điểm.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra). Để kiểm chứng tính năng này của hệ thống trong Uppaal ta dùng cú pháp: A<> φ: Chỉ ra rằng một tính chất φ luôn được thỏa mãn; cú pháp φ-->ψ : chỉ ra rằng khi φ thỏa mãn thì ψ cũng thỏa mãn.

Ví dụ: Trong hệ thống Train-Gate, để kiểm chứng tính năng này cho hệ thống ta có thể dùng các câu lệnh như: Train(0).Appr-->Train(0).Cross để đảm bảo tàu 0 tiến vào thì tàu 0 sẽ được qua cầu.

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không. Để kiểm chứng tính năng này của hệ thống trong Uppaal ta sử dụng cú pháp A[] not deadlock.

Để sử dụng chức năng kiểm chứng trong Uppaal ta chọn mục Verifier, khi đó



Hình 3.11 Màn hình thể hiện chức năng kiểm chứng

trên màn hình sẽ hiện ra ba vùng chức năng gồm: Overview (hiện ra các câu lệnh kiểm chứng); Query (vùng soạn thảo câu lệnh); Coment (chú thích nội dung của câu lệnh). Thao tác thực hiện trong phần này ta tiến hành như sau: Soạn cú pháp cần kiểm tra trong Query và bấm Check khi đó cú pháp vừa được kiểm tra hiện ra trong Overview; Chú thích nội dung coment trong hộp coment (nếu cần). Lúc này kết quả sẽ hiện ra trong hộp status. Nếu một cú pháp kiểm tra là thỏa mãn thì trong ststus sẽ cho kết luận màu xanh, nếu không thỏa mãn thì cho màu đỏ, còn không xác định được thì cho màu vàng (xem hình 3.11).

CHƯƠNG 4. ÁP DỤNG ĐẶC TẢ VÀ KIỂM CHỨNG MỘT SỐ HỆ THỐNG THỜI GIAN THỰC BẰNG CÔNG CỤ UPPAAL

Để tìm hiểu cách đặc tả và kiểm chứng trong Uppaal, tác giả đã tiến hành xây dựng một số ví dụ nhằm thông qua đó tác giả hiểu rõ hơn về cách biểu diễn một hệ thống phần mềm dưới dạng các ô-tô-mát thời gian có liên hệ với nhau qua các kênh, hiểu được sự vận hành của hệ thống và từ đó kiểm chứng các tính năng của hệ thống.

Trong chương này tác giả đưa ra bốn ví dụ để minh họa cho hai loại hệ thống phần mềm, mỗi hệ thống tác giả đưa ra hai ví dụ có thể hiểu như hai phiên bản khác nhau của hệ thống đó.

4.1 Hệ thống phân loại

Đây là một hệ thống giả định mà tác giả đưa ra nhằm tìm hiểu cách biểu diễn đặc tả hệ thống qua các ô-tô-mát thời gian cũng như cách kiểm chứng sự vận hành của hệ thống và các tính năng của hệ thống. Với hệ thống này tác giả đưa ra hai ví dụ như hai phiên bản khác nhau của hệ thống.

4.1.1 Ví dụ1. Hệ thống phân loại bóng theo màu sắc (Hệ thống Bong7mau).

Một hệ thống có đầu vào là các loại bóng với nhiều màu sắc khác nhau (demo là 7 màu), các quả bóng lần lượt cho chạy qua một sensor nhận biết màu sắc, sensor sẽ thông báo tín hiệu cho các cửa đẩy tương ứng. Bóng sẽ chạy trên một băng chuyền với vận tốc đảm bảo để di chuyển từ cửa này đến cửa kế tiếp là một khoảng thời gian cố định và gặp đúng cửa nó sẽ được đẩy ra khỏi băng chuyền.

*Đặc tả: Một quả bóng có màu sắc i (gán mỗi màu bởi một mã màu i - $i=\overline{1;7}$) khi đi qua sensor sẽ được sensor phát hiện màu sắc, lập tức sensor sẽ phát tín hiệu tới cửa thứ I tương ứng. quả bóng qua sensor sẽ được chạy trên băng chuyền với tốc độ đảm bảo đến cửa thứ i sẽ mất đúng i*5s. Sau đúng i*5s kể từ khi nhận được tín hiệu từ sensor cửa thứ i sẽ đẩy ra, quả bóng sẽ bị đẩy xuống rãnh tương ứng.*

Để tiến hành kiểm chứng sự vận hành của hệ thống trên qua Uppaal ta lần lượt thực hiện các bước sau:

Bước 1. Phân tích và nhận diện đối tượng trong hệ thống: Dựa vào đặc tả của hệ thống thì có thể chia hệ thống thành hai hệ khuôn mẫu (Template) liên hệ nhau qua

màu sắc của quả bóng. Cụ thể là: Hệ 1 có 1 mắt cảm ứng nhận diện màu sắc (Sensor); Hệ 2 gồm 7 cửa đẩy (PushDoor(i), i=1 ÷ 7).

Bước 2: Soạn thảo (thiết kế hệ thống dưới dạng các ô-tô-mát thời gian)

Trong phần này ta tạo ra hai khuôn mẫu gồm Sensor và Pushdoor.

Khai báo biến toàn cục: với giả định mô hình phân loại bảy màu sắc khác nhau, mỗi màu bóng sẽ được mã hóa bởi một số nguyên để nhận diện đồng bộ nên trong phần này ta khai báo biến toàn cục (trong mục dedaraions) như sau:

const int N = 7;//Số lượng màu bóng, trong ví dụ này demo là có 7 màu

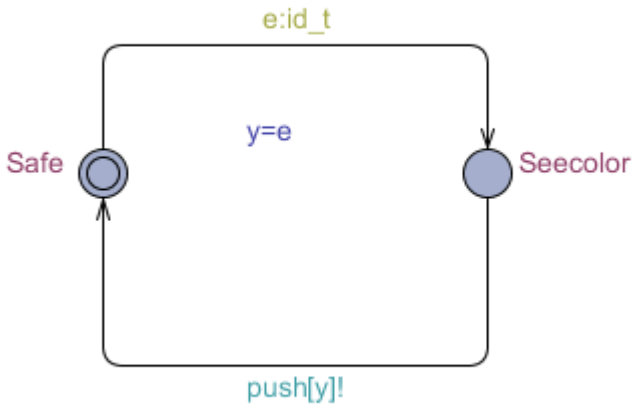
typedef int[0,N-1] id_t;// mã hóa màu sắc của quả bóng

chan appr[N], push[N];// Kênh đồng bộ giữa 2 hệ template

Phân tích và thiết kế khuôn mẫu Sensor:

Đối với khuôn mẫu này tồn tại hai trạng thái là : Safe (trạng thái an toàn, khi không có bóng nào đi qua) và SeeColor (nhìn thấy bóng đi qua). Mỗi trạng thái sẽ thể hiện bởi một nút trên ô-tô-mát.

Ban đầu khi chưa có bóng đi qua, Sensor sẽ ở trạng thái Safe. Khi có một bóng (gán bởi một số thứ tự tương ứng với màu sắc) đi qua, sensor phát hiện ra màu và nhớ màu đó (gán y=chỉ số màu tương ứng) rồi chuyển sang trạng thái Seecolor. Sau đó ngay lập tức gửi tín hiệu đến cửa thứ i (i=y) tương ứng (lệnh push[y]!) và chuyển về trạng thái Safe.



Hình 4.1 Ô-tô-mát Sensorcủa hệ thống Bong7mau

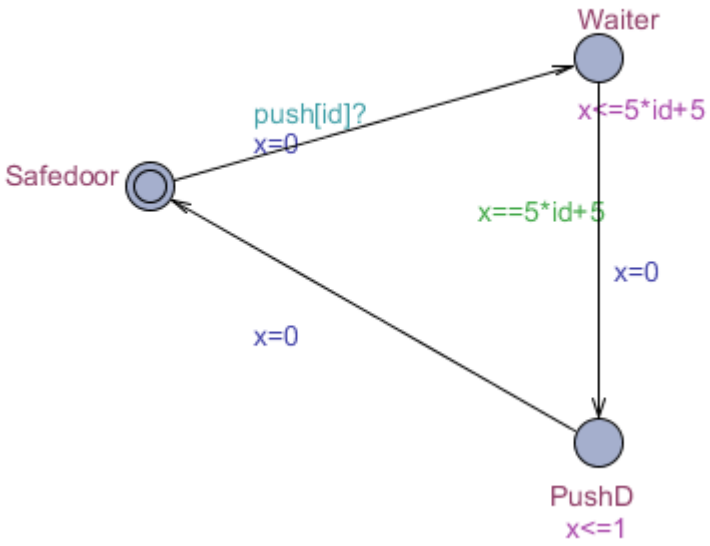
Ô-tô-mát Sensor: Trong mục soạn thảo ta tiến hành vẽ ô-tô-mát cho khuôn mẫu này. Cụ thể ta vẽ hai nút Safe và Seecolor để thể hiện hai trạng thái của khuôn mẫu: Từ Safe đến Seecolor vẽ một cạnh thể hiện bước chuyển trạng thái (gán e là chỉ số màu và biến y để nhớ màu sắc mà Sensor nhìn thấy); Từ Seecolor đến Safe vẽ một cạnh thể hiện bước chuyển trạng thái, đồng thời ở bước này được đồng bộ với ô-tô-mát Pushdoor qua kênh push bằng cách báo lệnh push[y] đến cửa ra của màu có gán nhãn y. (xem hình 4.1)

Phân tích và thiết kế hệ khuôn mẫu Pushdoor

Một khuôn trong hệ này có 3 trạng thái: Trạng thái chưa có yêu cầu báo mở (Safedoor); Trạng thái có nhận được yêu cầu báo mở nhưng chờ đến thời gian mở (Waiter); Trạng thái đẩy cửa (PushD). Mỗi trạng thái này sẽ được thể hiện bằng một nút trên ô-tô-mát thời gian.

Đặc tả của khuôn mẫu: Khi cửa thứ i đang ở trạng thái Safedoor nếu nhận được lệnh có bóng màu thứ i đang đi tới, lập tức chuyển sang trạng thái Waiter và ở trạng thái Waiter đó đúng 5*id+5 giây và lập tức chuyển sang trạng thái PushD. Sau đúng 1 giây thì trở về trạng thái an toàn.

Ô-tô-mát PushDoor (xem hình 4.2):

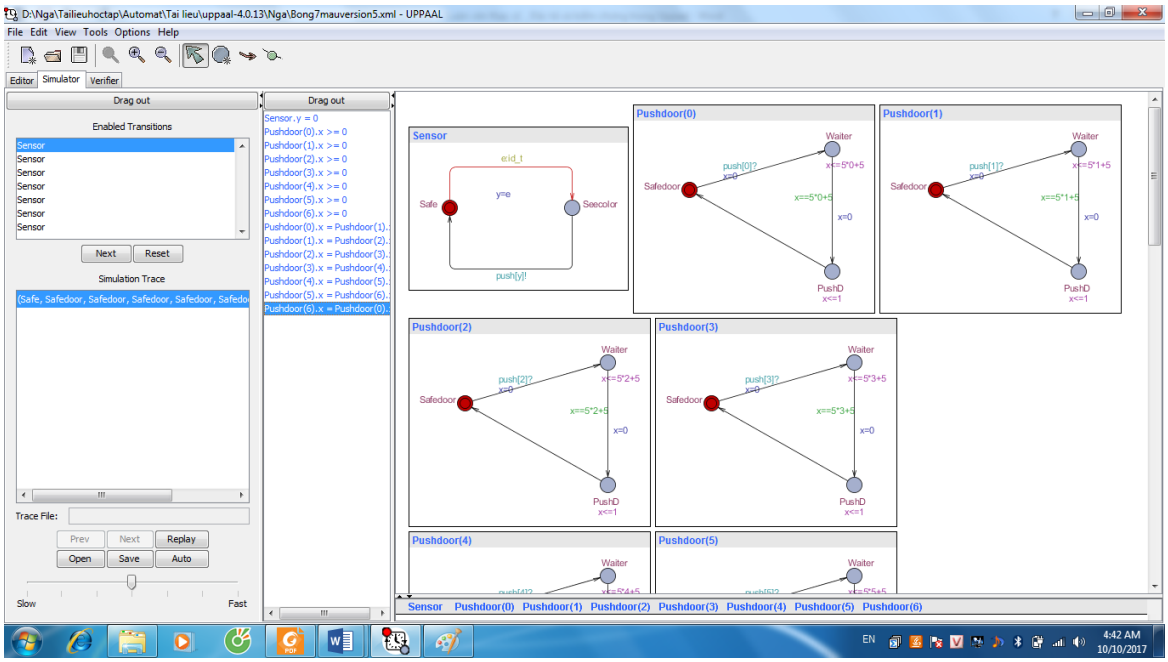


Hình 4.2 Ô-tô-mát PushDoor của hệ thống Bong7mau

- Khai báo các template có trong hệ thống: Trong System declaration ta tiến hành khai báo: **system** Sensor, Pushdoor;

Bước 3. Mô phỏng sự vận hành của hệ thống: Sau khi đã soạn thảo xong trong editor, nếu không có lỗi soạn thảo, chức năng Simulation cho phép ta mô phỏng sự vận hành của hệ thống. Cụ thể chức năng này cho phép ta mô phỏng theo hai cách sau:

- Mô phỏng sự thay đổi trạng thái của các đối tượng, màn hình chức năng mô phỏng cho ta thấy rõ các bước chuyển trạng thái của các quá trình theo thời gian (xem hình 4.3).

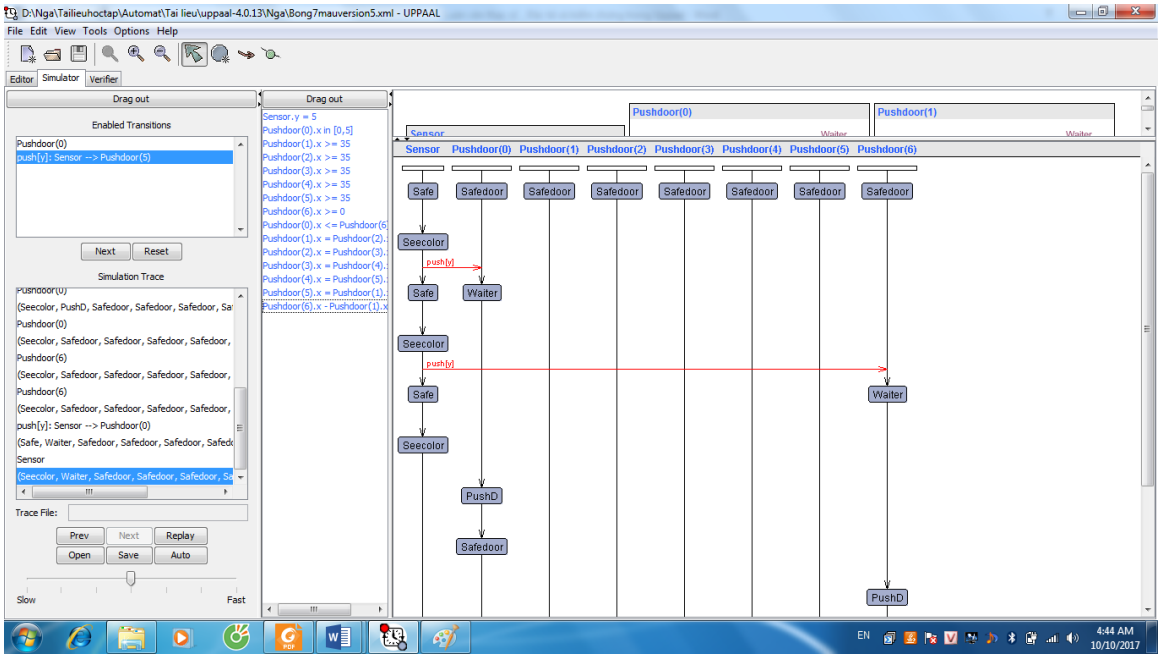


Hình 4.3 Màn hình chức năng mô phỏng Simulation của hệ thống Bong7mau

- Mô phỏng sự đồng bộ theo thời gian của các đối tượng, ngoài việc mô phỏng trên thì trên màn hình chức năng mô phỏng còn cho ta theo dõi được sự vận hành của cả hệ thống theo thời gian của các ô-tô-mát song song và sự tác động giữa các ô-tô- mát thông qua các kênh đã cài đặt (xem hình 4.4).

Bước 4. Kiểm chứng hoạt động của hệ thống: Chức năng Verifier cho phép ta kiểm chứng các tính chất của hệ thống thông qua các câu lệnh, cụ thể như:

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định). Để kiểm chứng tính chất này ta sử dụng cú pháp: E<>φ (trong đó φ là trạng thái mà ta cần kiểm tra của một quá trình cụ thể).



Hình 4.4 Màn hình chức năng mô phỏng Simulation của hệ thống Bong7mau

Trong ví dụ này ta kiểm chứng tính đạt được của các trạng thái như sau:

E<> Pushdoor(0).PushD: kiểm tra xem các cửa có đạt được trạng thái PushD hay không. Tương tự ta kiểm tra được cho các cửa còn lại: E<> Pushdoor(1).PushD; E<> Pushdoor(2).PushD; E<> Pushdoor(3).PushD; E<> Pushdoor(4).PushD; E<> Pushdoor(5).PushD; E<> Pushdoor(6).PushD.

E<>Senor.Seecolor Kiển tra xem Sensor có chuyển sang trạng thái nhìn thấy màu không và có lưu lại màu sắc vừa nhìn không.

E<> Pushdoor(0).PushD and (forall (i:id_t) i!=0 imply Pushdoor(i).Safedoor): Kiểm tra xem khi một cửa đẩy thì các cửa khác có ở trạng thái an toàn không (đảm bảo chỉ có một cửa đẩyr a trong một lúc)

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng). Để kiểm chứng tính chất này ta sử dụng cú pháp: A[] và E[]

A[] forall(i:id_t) forall(j:id_t) Pushdoor(i).PushD and Pushdoor(j).PushD imply i==j. Tính chất 2 cửa khác nhau không bao giờ cùng đẩy.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra). Để kiểm chứng tính chất này ta sử dụng cú pháp: A<> φ: với mục đích chỉ ra rằng φ luôn được thỏa mãn và cú pháp φ-->ψ đề đảm bảo rằng khi φ thỏa mãn thì ψ cũng thỏa mãn.

Trong ví dụ này ta tiến hành kiểm chứng tính chất này của hệ thống bằng các câu lệnh sau:

Pushdoor(0).PushD --> Pushdoor(0).Safedoor. Kiểm tra nếu cửa thứ i đẩy thì cửa thứ i sẽ trở về trạng thái an toàn. Tương tự cho các cửa còn lại:

Pushdoor(1).PushD-->Pushdoor(1).Safedoor;

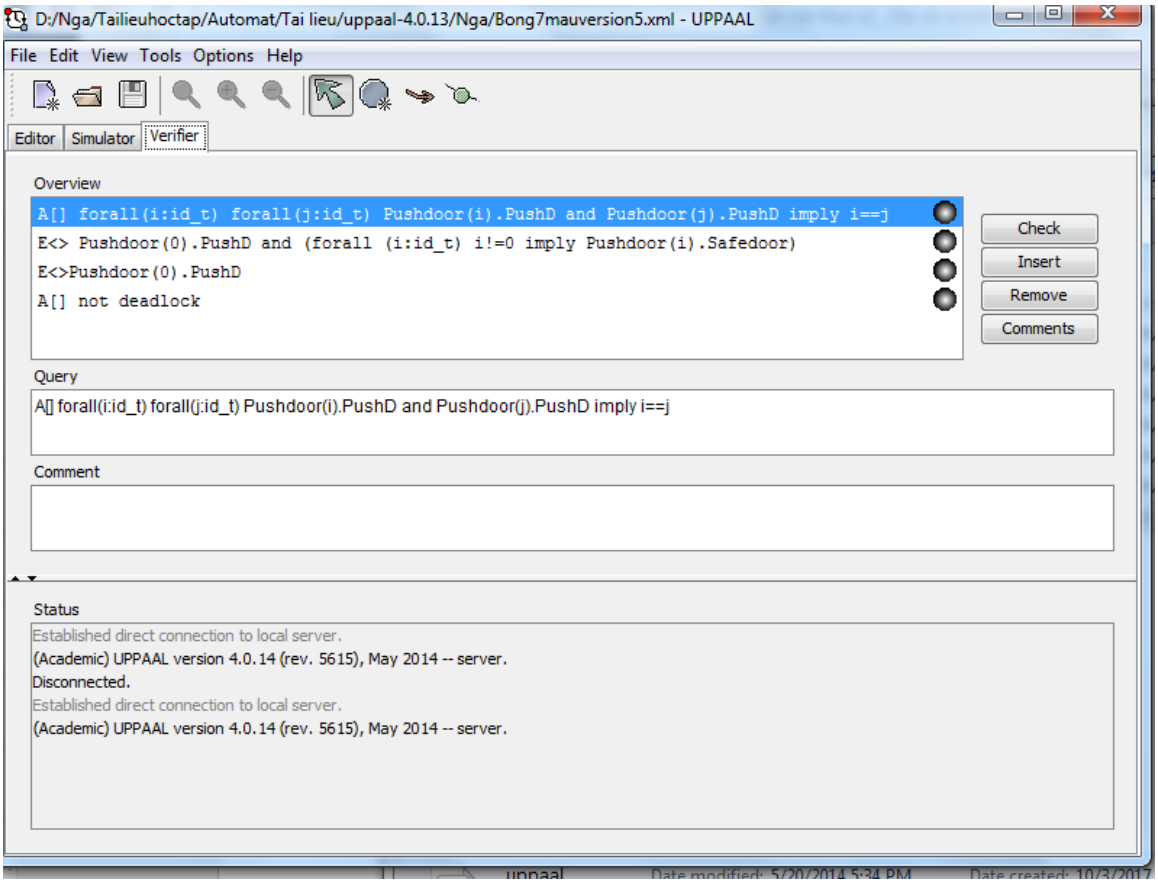
Pushdoor(2).PushD-->Pushdoor(2).Safedoor;

Pushdoor(3).PushD-->Pushdoor(3).Safedoor;

Pushdoor(4).PushD-->Pushdoor(4).Safedoor

Pushdoor(5).PushD --> Pushdoor(5).Safedoor

Pushdoor(6).PushD --> Pushdoor(6).Safedoor



Hình 4.5 Màn hình chức năng kiểm chứng Verifier của hệ thống Bong7mau

- Kiểm chứng ô-tô-mát có rơi vào deadlock hay không. Để kiểm chứng tính chất này ta dùng cú pháp: A[] not deadlock (xem hình 4.5).

4.1.2 Ví dụ 2. Hệ thống phân loại sản phẩm (sản phẩm đạt chất lượng hay chưa).

Tương tự như ví dụ 1, ta xét một hệ thống phân loại khoai tây, có đầu vào là các củ khoai tây sau khi thu hoạch, các củ khoai tây được cho qua một phễu để chạy qua một sensor kiểm tra chất lượng (kích thước, cân nặng, màu sắc), sensor sẽ thông báo tín hiệu cho các cửa đẩy tương ứng. Củ khoai tây sẽ chạy trên một băng chuyền với vận tốc đảm bảo để di chuyển từ cửa này đến cửa kế tiếp là một khoảng thời gian cố định và gặp đúng cửa nó sẽ được đẩy vào đúng rãnh phân loại.

Đặc tả: Một củ khoai tây khi đi qua sensor sẽ được sensor phát hiện chất lượng thông qua kích thước, cân nặng và màu sắc, Nếu đảm bảo kích thước, cân nặng và màu sắc tốt thì củ khoai đó được xếp hạng A và lập tức sensor sẽ phát tín hiệu tới cửa hạng A, còn lại sẽ xếp hạng B và được báo tín hiệu đến của hạng B. Củ khoai tây sau khi qua sensor sẽ được chạy trên băng chuyền gặp đúng cửa mở nó sẽ rơi xuống rãnh của cửa đó và được phân loại.

Để kiểm chứng sự vận hành của hệ thống này bằng công cụ Uppaal ta tiến hành các bước sau:

Bước 1. Phân tích và nhận diện đối tượng trong hệ thống

Hệ thống gồm đèn cảm ứng chất lượng (Sensor) và 2 cửa mở (ADoor và BDoor), các củ khoai được xem là đối tượng tham gia trong hệ thống (Potato).

Bước 2. Soạn thảo (thiết kế hệ thống dưới dạng các ô-tô-mát thời gian)

Trước hết ta xây dựng các khuôn mẫu tương ứng với các đối tượng đã phân tích ở trên. Tiếp đến trong phần khai báo biến toàn cục, ta khai báo các kênh đồng bộ trong hệ thống trong Declarations như sau:

chan Astyle, Bstyle, OpenA, OpenB, cross;

Sau đó tiến hành vẽ ô-tô-mát cho từng khuôn mẫu. Cụ thể với từng khuôn mẫu ta phân tích và thiết kế như sau:

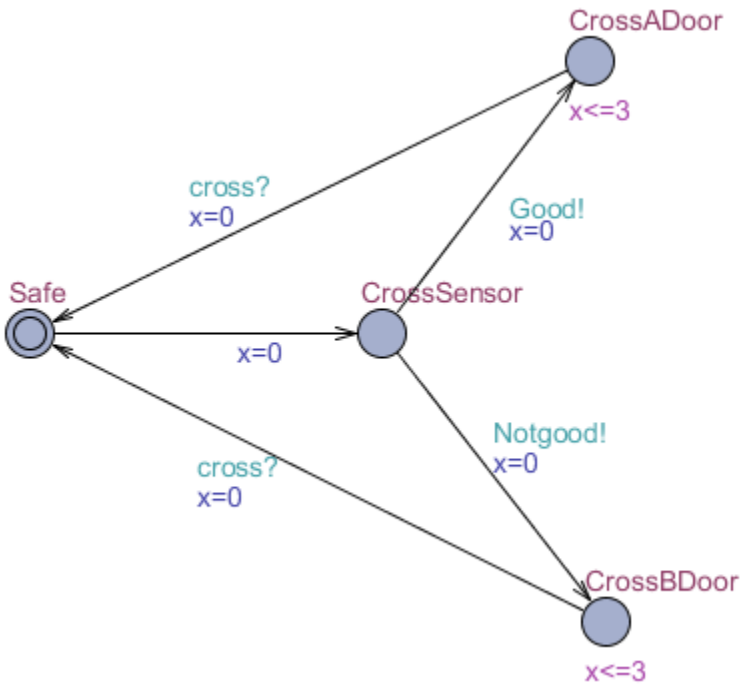
- Với khuôn mẫu Potato:

Trước hết, phân tích những trạng thái của khuôn mẫu này ta thấy khuôn mẫu có bốn trạng thái bao gồm Safe, CrossSensor, CrossA, CrossB. Trong đó trạng thái Safe là trạng thái không ở trong đường truyền; trạng thái CrossSensor trạng thái

đi qua sensor; trạng thái CrossA trạng thái đi vào rãnh A; trạng thái CrossB là trạng thái đi vào rãnh B.

Tiếp đến ta phân tích bước chuyển trạng thái trong khuôn mẫu. Từ trạng thái safe củ khoai tây được chạy qua phễu và đến vị trí của sensor, bắt đầu kích hoạt đồng hồ, reset về 0. Nếu là củ khoai đạt chất lượng thì báo tín hiệu Good! Và đi đến cửa A, ngược lại, báo tín hiệu Notgood! và đi đến của B. Sau đúng 3 giây thoát khỏi cửa và trở về trạng thái an toàn (đã phân loại).

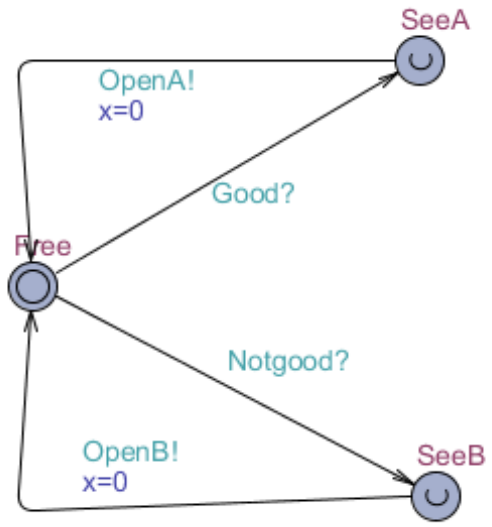
Cuối cùng tiến hành vẽ ô-tô-mát Potato (xem hình 4.6)



Hình 4.6 Ô-tô-mát Potato của hệ thống Potato

- Với khuôn mẫu Sensor: khuôn mẫu này gồm ba trạng thái: Free, SeeA và SeeB. Trong đó, trạng thái Free là trạng thái an toàn; trạng thái SeeA là trạng thái phát hiện củ khoai đạt chất lượng loại A; trạng thái SeeB là trạng thái phát hiện củ khoai đạt chất lượng loại B.

Việc thực hiện chuyển trạng thái của khuôn mẫu này được mô tả như sau: Khi có một củ khoai đi qua, sensor phát hiện ra chất lượng nhờ vào kích thước, cân nặng và màu (được hiểu như củ khoai phát tín hiệu), nếu nhận được tín hiệu Good! Thì chuyển sang SeeA, ngược lại chuyển sang SeeB. Sau đó ngay lập tức gửi tín hiệu đến cửa tương ứng (OpenA! Hoặc OpenB!) và chuyển sang trạng thái Free.



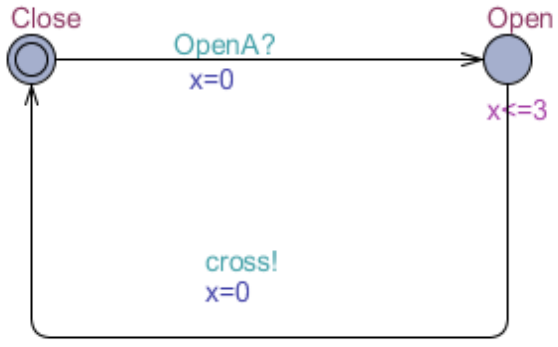
Hình 4.7 Ô-tô-mát Sensor

Vẽ ô-tô-mát Sensor dựa trên phân tích rạng thái và bươc chuyển trạng thái ở trên (xem hình 4.7)

- Khuôn mẫu ADoor: khuôn mẫu này gồm hai trạng thái là Close và Open, trong đó trạng thái Close là trạng thái chưa có yêu cầu báo mở; trạng thái Open là trạng thái mở cửa.

Việc thực hiện chuyển trạng thái được mô tả như sau: Khi cửa đang ở trạng thái Close nếu nhận được lệnh có củ khoai đạt chất lượng đang đi tới (OpenA?), lập tức reset đồng hồ về 0 và chuyển sang trạng thái Open và ở trạng thái khoảng 3s và sau đó chuyển sang trạng thái Close.

Theo phân tích trạng thái và chuyển trạng thái như trên ta tiến hành vẽ ô-tô-mát ADoor như sau (xem hình 4.8).

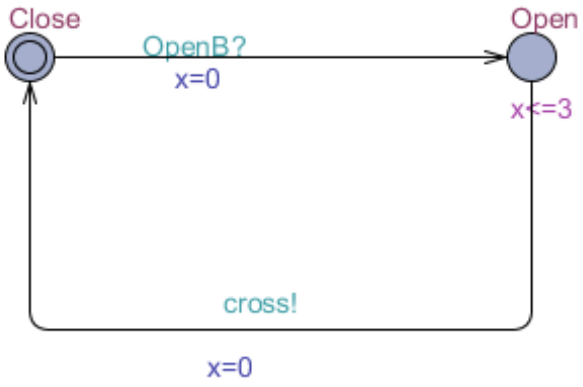


Hình 4.8 Ô-tô-mát Adoor của hệ thống Potato

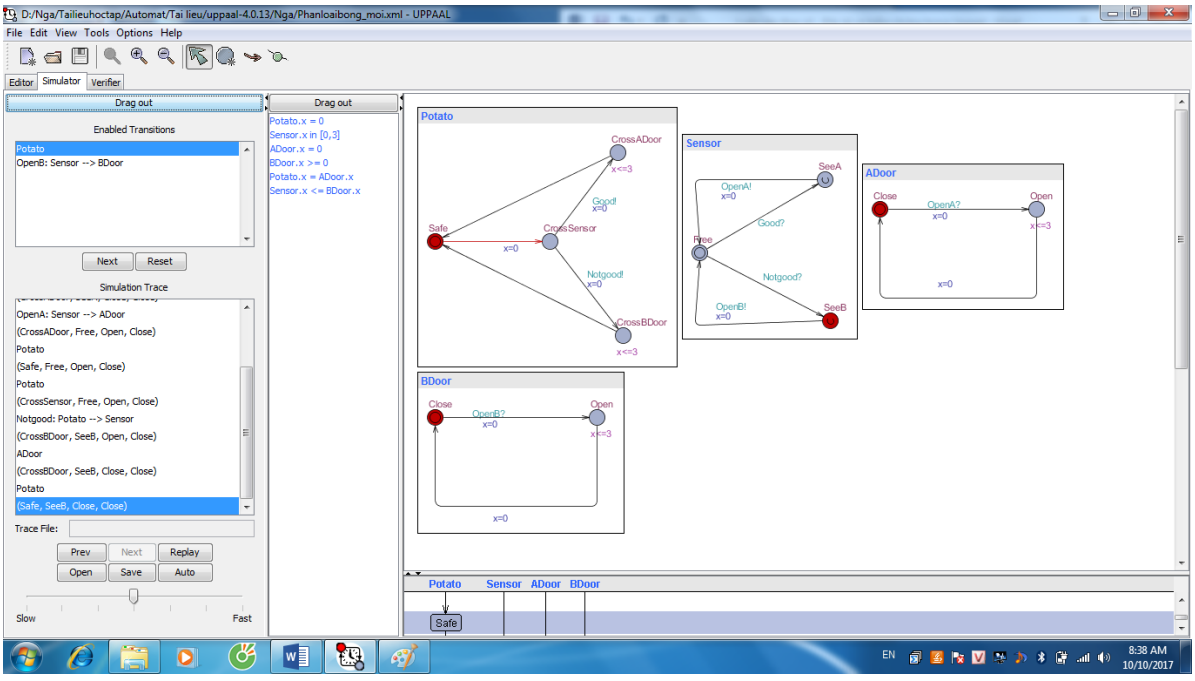
- Khuôn mẫu BDoor: khuôn mẫu này gồm hai trạng thái là Close và Open, trong đó trạng thái Close là trạng thái chưa có yêu cầu báo mở; trạng thái Open là trạng thái mở cửa.

Việc thực hiện chuyển trạng thái được mô tả như sau: Khi cửa đang ở trạng thái Close nếu nhận được lệnh có củ khoai đạt chất lượng đang đi tới (OpenB?), lập tức reset đồng hồ về 0 và chuyển sang trạng thái Open và ở trạng thái khoảng 3s và sau đó chuyển sang trạng thái Close.

Theo phân tích trạng thái và chuyển trạng thái như trên ta tiến hành vẽ ô-tô-mát BDoor như sau (xem hình 4.9).



Hình 4.9 Ô-tô-mát Bdoor của hệ thống Potato



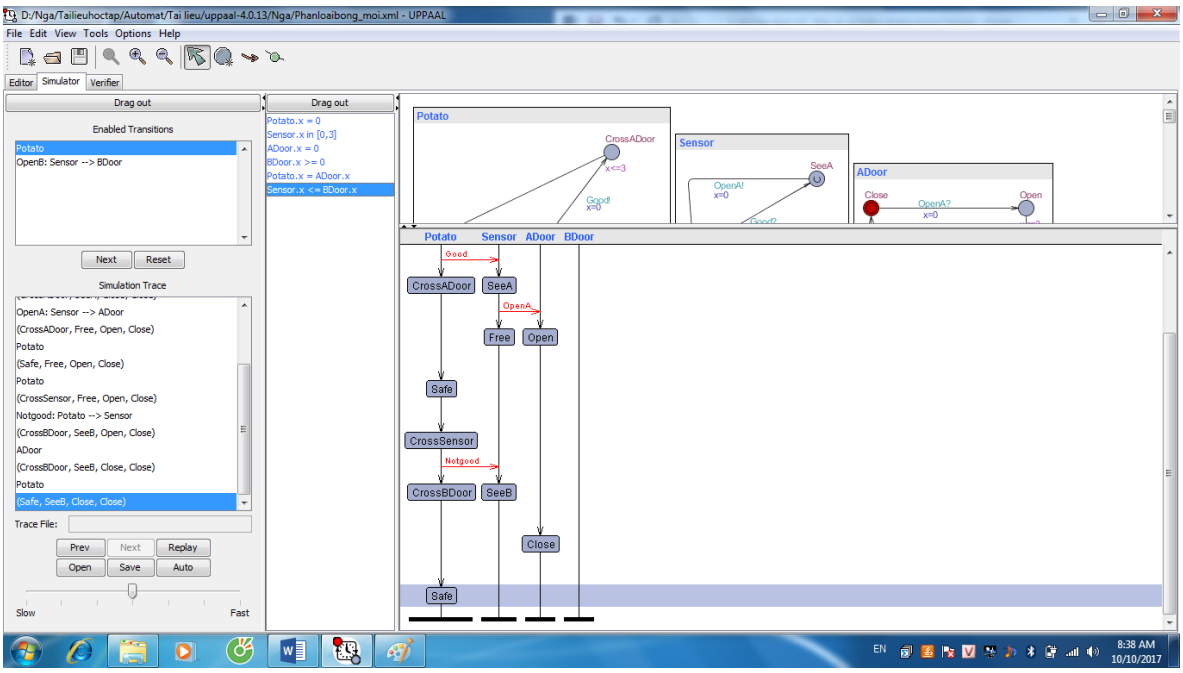
Hình 4.10 Màn hình chức năng mô phỏng Simulation của hệ thống Potato

Sau khi đã thiết kế xong các template ta tiến hành khai báo các template có trong hệ thống trong System declaration:

System Potato, Sensor, ADoor, Bdoor

Bước 3. Mô phỏng sự vận hành của hệ thống: Tương tự như trong ví dụ 1, sau khi đã soạn thảo xong hệ thống mà không có lỗi thì công cụ cho phép ta chạy mô phỏng hệ thống dưới hai hình thức là: Mô phỏng sự thay đổi trạng thái của các đối tượng (xem hình 4.10)

Và mô phỏng sự đồng bộ theo thời gian của các đối tượng (xem hình 4.11).



Hình 4.11 Màn hình chức năng mô phỏng Simulation của hệ thống Potato

Bước 4. Kiểm chứng các tính chất của hệ thống: Chức năng Verifier cho phép ta kiểm chứng các tính chất của hệ thống, cụ thể như:

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định). Để kiểm chứng tính chất này ta sử dụng các câu lệnh sau:

E<>Potato.CrossSensor kiểm tra xem một củ khoai tây có chuyển sang trạng thái CrossSensor được không.

E<>Potato.CrossADoor or Potato.CrossBDoor để kiểm tra xem một củ khoai hoặc là phải ra được cửa A hoặc cửa B.

E<>Senor.SeeA hoặc E<>Sensor.SeeB để kiểm tra xem Sensor có chuyển sang trạng thái phát hiện ra củ khoai tây đạt loại A hay đạt loại B không.

- Kiểm chứng tính an toàn (một điều gì đó luôn luôn đúng). Để kiểm chứng tính chất này của hệ thống ta sử dụng cú pháp sau:

E[]Potato.CrossADoor and Potato.CrossBDoor đảm bảo một củ khoai tây không bao giờ qua cả hai cửa.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra). Để kiểm chứng tính chất này của hệ thống ta sử dụng cú pháp sau:

Sensor.SeeA-->ADoor.Open: Đảm bảo nếu sensor phát hiện là là củ đạt chất lượng A thì cửa A phải mở.

Sensor.SeeB-->BDoor.Open: Đảm bảo nếu sensor phát hiện là là củ đạt chất lượng B thì cửa B phải mở.

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không, ta sử dụng cú pháp: A[] not deadlock

4.2 Hệ thống điều khiển sử dụng vùng tài nguyên

Dựa trên mô hình Train-Gate [6][15], trong bài toán này thời gian để đi qua cầu của các tàu mặc định là như nhau. Tác giả đã đề xuất mô hình hệ thống điều khiển vùng tài nguyên có mở rộng về thời gian sử dụng vùng tài nguyên chung là khác nhau cho các quá trình. Cụ thể tác giả đề xuất hai ví dụ tương ứng với hai mô hình, trong đó ví dụ thứ nhất tác giả đề xuất là mô hình gồm hai quá trình (có thể mở rộng ra nhiều quá trình) cùng có nhu cầu sử dụng một vùng tài nguyên chung và thời gian để sử dụng vùng tài nguyên đó ở hai quá trình là khác nhau, ở ví dụ thứ hai tác giả đề xuất có nhiều quá trình cùng tham gia sử dụng chung một vùng tài nguyên nhưng các quá trình đó chia thành hai nhóm (có thể mở rộng thành nhiều nhóm quá trình) với thời gian sử dụng vùng tài nguyên ở hai nhóm là khác nhau.

4.2.1 Ví dụ 3. Hệ thống điều khiển việc sử dụng chung vùng tài nguyên Process ResourceV1 (có ràng buộc về thời gian sử dụng nguồn tài nguyên).

Có n quá trình (demo là 2) cùng có nhu cầu sử dụng 1 vùng tài nguyên, hệ thống đảm bảo việc điều khiển sao cho tại một thời điểm chỉ có một quá trình được sử

dụng vùng tài nguyên và các quá trình gửi yêu cầu trước sẽ được bố trí sử dụng trước (giả thiết là các quá trình cần sử dụng vùng tài nguyên với thời gian khác nhau).

Đặc tả: Có n quá trình –Process (demo là n=2) đều có nhu cầu sử dụng một nguồn tài nguyên – Resource. Khi một quá trình có nhu cầu sử dụng nó sẽ gửi tín hiệu thông báo cho bộ điều khiển, bộ điều khiển tiếp nhận tín hiệu và tiến hành xử lí tín hiệu đó, nếu trong thời điểm đó nguồn tài nguyên đang rảnh nó sẽ báo lại tín hiệu cho quá trình được phép sử dụng nguồn tài nguyên, nếu hiện đang có quá trình đang sử dụng nó, bộ điều khiển sẽ lưu thứ tự các quá trình có nhu cầu vào một hàng đợi và đến khi nguồn tài nguyên rảnh nó sẽ gọi quá trình đầu tiên ra sử dụng trước. Khi quá trình sử dụng xong vùng tài nguyên (mặc định là sau một khoảng thời gian cho trước tương ứng với quá trình đó) nó sẽ báo lại cho bộ điều khiển biết và rời khỏi cùng tài nguyên.

Yêu cầu: Quá trình nào có nhu cầu đều được bố trí sử dụng nguồn tài nguyên, không có sự xung đột, đảm bảo tại một thời điểm chỉ có một quá trình được sử dụng.

Để tiến hành kiểm chứng cho hệ thống này bằng công cụ Uppaal, tác giả đã thực hiện các bước sau:

Bước 1. Phân tích và nhận diện đối tượng trong hệ thống

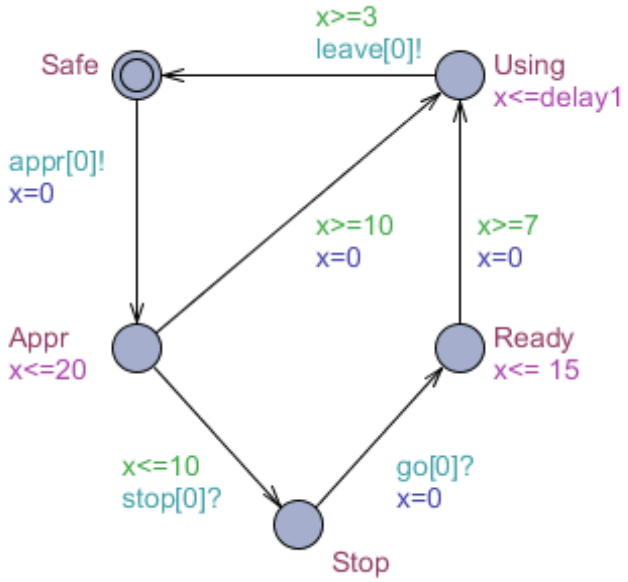
Hệ thống có hai quá trình (Process1 và Process2) và một nguồn tài nguyên (Resource). Các Process1 và Process2 hoạt động song song, và được đồng bộ với Resource thông qua các kênh: báo tín hiệu yêu cầu sử dụng (appr[i]), dừng (stop[i]), được phép sử dụng(go[i]) và rời khỏi vùng tài nguyên (leave[i]).

Bước 2. Soạn thảo và mô hình hóa các đối tượng

Trong vùng soạn thảo tạo các khuôn mẫu có trong hệ thống gồm Process1, Process2 và Resource. Tiến hành khai báo biến toàn cục trong Declaration như sau:

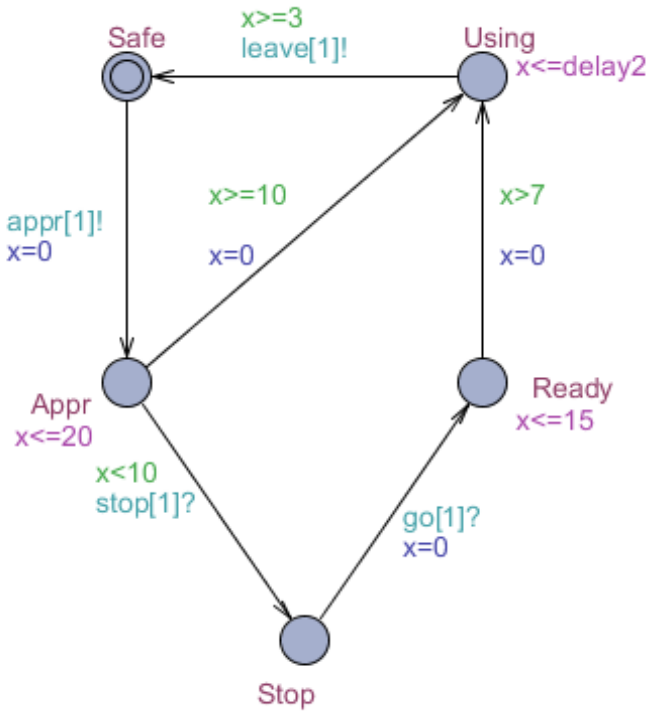
```
const int N = 2;        // # Process
typedef int[0,N-1] id_t; // Mã quá trình
chan appr[N], stop[N], leave[N];  //các tín hiệu đồng bộ giữa các mô hình
urgent chan go[N];  // không có trễ khi đồng bộ chuyển tiếp
```

Tiếp theo đó ta tiến hành soạn thảo cho các khuôn mẫu có trong hệ thống. Với khuôn mẫu Process1, khuôn mẫu này được truyền tham số là thời gian mà quá trình này sẽ sử dụng vùng tài nguyên (delay1). Biến địa phương được khai báo cho khuôn mẫu nàylà đồng hồ x. Khuôn mẫu này có năm trạng thái: Safe, Appr, Stop, Ready, Using. Trong đó: Safe là trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên; Appr là trạng thái đăng kí sử dụng nguồn tài nguyên; Stop là trạng thái chờ đến lượt sử dụng; Using: sử dụng nguồn tài nguyên. Bước chuyển trạng thái của khuôn mẫu này được thể hiện như sau: Process1 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr[0]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong khoảng thời gian delay1 (là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop[0]?) từ bộ điều khiển thì lập tức chuyển sang trạng thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go[0]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có tín hiệu (stop[0]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[0]!) thì chuyển sang trạng thái Safe. Dựa trên phân tích các trạng thái và bước chuyển trạng thái trên, ô-tô-mát của Process1 được thiết kế như sau (xem hình 4.12).



Hình 4.12 Ô-tô-mát của Process1 trong hệ thống Process ResourceV1

Khuôn mẫu Process2 dược truyền tham số là thời gian mà quá trình này sẽ sử dụng vùng tài nguyên (delay2). Biến địa phương được khai báo trong khuôn mẫu này là đồng hồ x. Khuôn mẫu gồm năm trạng thái: Safe; Appr; Stop; Ready; Using. Trong đó: Safe là trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên; Appr là trạng thái đăng kí sử dụng nguồn tài nguyên; Stop là trạng thái chờ đến lượt sử dụng vùng tài nguyên; Using là trạng thái sử dụng nguồn tài nguyên. Các bước chuyển trạng thái của khuôn mẫu này được thiết kế như sau: Process 2 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr[1]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong khoảng thời gian delay2 (là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop[1]?) từ bộ điều khiển thì lập tức chuyển sang trạng thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go[1]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có tín hiệu (stop[1]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[1]!) thì chuyển sang trạng thái Safe. Dựa trên phân tích trạng thái và các bước chuyển trạng thái trên, ô-tô-mát cho Process2 được thiết kế như sau (xem hình 4.13).



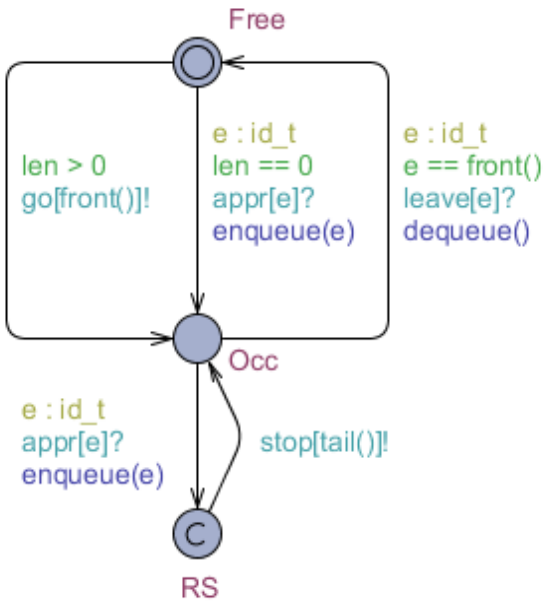
Hình 4.13 Ô-tô-mát Process2 trong hệ thống Process ResourceV1

Khuôn mẫu Resource được khai báo biến và hàm sẽ sử dụng trong khuôn mẫu như sau:

```
id_t list[N+1];
int[0,N] len;
// Put an element at the end of the queue
void enqueue(id_t element)
{
    list[len++] = element;
}
// Remove the front element of the queue
void dequeue()
{
    int i = 0;
    len -= 1;
```

```
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 0;
}
// Returns the front element of the queue
id_t front()
{
  return list[0];
}
// Returns the last element of the queue
id_t tail()
{
  return list[len - 1];
}
```

Khuôn mẫu này gồm có ba trạng thái là Free; Occ; RS. Trong đó: Free là trạng thái nguồn tài nguyên rảnh; Occ là trạng thái tiếp nhận thông tin đăng kí xếp hàng và RS là trạng thái xếp hàng cho các quá trình (trạng thái là trạng thái Committed). Các bước chuyển trạng thái của khuôn mẫu này được diễn tả như sau: Nếu nguồn tài nguyên đang ở trạng thái rảnh mà số quá trình đang có nhu cầu xử dụng trong hàng đợi lớn hơn không thì gọi ra quá trình đầu tiên cho sử dụng trước. Nếu không có quá trình nào trong hàng đợi, đồng thời nhận được tín hiệu báo có nhu cầu sử dụng thì xếp nó vào hàng đợi và chuyển sang trạng thái Occ. Từ trạng thái Occ nếu



Hình 4.14 Ô-tô-mát Resource trong hệ thống Process ResourceV1

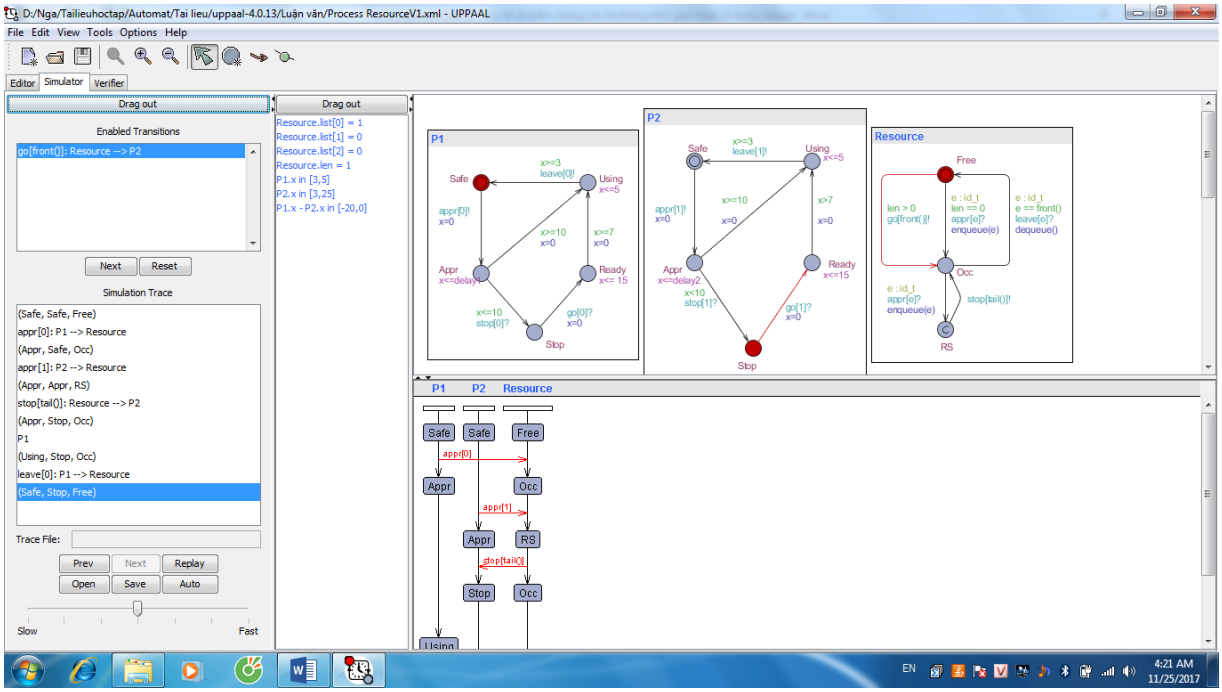
vẫn tiếp tục nhận được tín hiệu từ quá trình khác có nhu cầu sử dụng thì xếp quá trình đó vào hàng đợi chuyển qua trạng thái RS- Trạng thái Commited không cho phép trễ, đồng thời gửi tín hiệu stop! cho quá trình đó rồi lập tức trở về trạng thái Occ. Từ trạng thái Occ nếu nhận được tín hiệu leave[i]? lập tức xóa quá trình đó khỏi hàng đợi và chuyển về trạng thái Free. Dựa vào phân tích trạng thái và các bước chuyển trạng thái, ô-tô-mát Resource được thiết kế như sau (xem hình 4.14).

- Khai báo các quá trình và các hằng được sử dụng trong hệ thống trong mục System declaration như sau:

const int delay1=20; // thời gian quá trình 1 sử dụng vùng tài nguyên;
const int delay2=30;// thời gian quá trình 1 sử dụng vùng tài nguyên;
P1=Process1(delay1);
P2=Process2(delay2);
system P1, P2, Resource;

Bước 3. Mô phỏng sự vận hành của hệ thống

Khi kết thúc bước 2 mà không có lỗi thì hệ thống cho phép chạy mô phỏng sự vận hành của hệ thống dưới hai hình thức khác nhau là thay đổi trạng thái của các đối tượng và mô phỏng sự đồng bộ theo thời gian của các đối tượng (xem hình 4.15)



Hình 4.15 Màn hình mô phỏng sự vận hành của hệ thống Process ResourceV1

Bước 4. Kiểm chứng hoạt động của hệ thống.

Chức năng Verifier của Uppaal cho phép kiểm chứng các tính chất của hệ thống thông qua các câu lệnh cụ thể như:

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định)

E<> P1.Using kiểm tra xem quá trình 1 có chuyển sang trạng thái Using được không. Tương tự cho quá trình 2, ta sử dụng câu lệnh: E<> P2.Using.

E<> Resource.Occ kiểm tra xem quá trình Resource có chuyển sang trạng thái Occ được không.

- Kiểm chứng tính an toàn (một điều gì đó luôn luôn đúng)

A[]P1.Using && P2.Using: đảm bảo tại một thời điểm chỉ có nhiều nhất một quá trình được sử dụng vùng tài nguyên (tính không xung đột).

A[] Resource.list[N] == 0 đảm bảo không có quá n quá trình trong hàng đợi.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra)

P1.Appr --> P1.Using: Đảm bảo một quá trình 1 khi có nhu cầu sử dụng thì sẽ được sử dụng. Tương tự cho quá trình 2 ta sử dụng câu lệnh: P2.Appr --> P2.Using

E<> P1.Using and P2.Using: Đảm bảo 2 quá trình khác nhau sẽ không cùng được sử dụng. Hoặc câu lệnh: E<> P1.Using and P2.Stop hay E<> P2.Using and P1.Stop để đảm bảo một quá trình đang sử dụng thì tất cả các quá trình khác đều trong trạng thái phải chờ.

- Kiểm chứng ô-tô-mát có rơi vào deadlock hay không ta sử dụng cú pháp A[] not deadlock.

4.2.2 Ví dụ 4. Hệ thống điều khiển việc sử dụng chung vùng tài nguyên Process Resource V2(có nhiều nhóm quá trình có ràng buộc về thời gian sử dụng nguồn tài nguyên).

Ví dụ này là mở rộng của ví dụ 3, ta giả thiết là lúc này hệ thống có n quá trình (demo là 4) cùng có nhu cầu sử dụng chung một vùng tài nguyên, trong đó có nhiều nhóm quá trình muốn sử dụng vùng tài nguyên với thời gian khác nhau (demo là 2 nhóm: Nhóm 1 có N1 quá trình muốn dùng vùng tài nguyên trong thời gian delay1, nhóm 2 có N2 quá trình muốn sử dụng vùng tài nguyên với thời gian delay2), hệ thống đảm bảo việc điều khiển sao cho tại một thời điểm chỉ có một quá trình được sử dụng vùng tài nguyên và các quá trình gửi yêu cầu trước sẽ được bố trí sử dụng trước.

Đặc tả: Hệ thống có 2 nhóm quá trình Process1 và Process2 (với các mã quá trình được đánh số theo cách: nhóm 1 từ 0 đến N1-1; nhóm 2 từ N1 đến N2-1) đều có nhu cầu sử dụng một nguồn tài nguyên Resource. Khi một quá trình có nhu cầu sử dụng nó sẽ gửi tín hiệu thông báo cho bộ điều khiển, bộ điều khiển tiếp nhận tín hiệu và tiến hành xử lí tín hiệu đó, nếu trong thời điểm đó nguồn tài nguyên đang rảnh nó sẽ báo lại tín hiệu cho quá trình được phép sử dụng nguồn tài nguyên, nếu hiện đang có quá trình đang sử dụng nó, bộ điều khiển sẽ lưu thứ tự các quá trình có nhu cầu vào một hàng đợi và đến khi nguồn tài nguyên rảnh nó sẽ gọi quá trình đầu tiên ra sử dụng trước. Khi quá trình sử dụng xong vùng tài nguyên (mặc định là sau một khoảng thời gian cho trước tương ứng với quá trình đó) nó sẽ báo lại cho bộ điều khiển biết và rời khỏi cùng tài nguyên.

Yêu cầu: Quá trình nào có nhu cầu đều được bố trí sử dụng nguồn tài nguyên, không có sự xung đột, đảm bảo tại một thời điểm chỉ có một quá trình được sử dụng.

Phân tích và nhận diện đối tượng trong hệ thống

Hệ thống có 2 nhóm quá trình Process1 và Process2 (trong đó nhóm 1 có N1 quá trình, nhóm 2 có N2 quá trrình) và 1 nguồn tài nguyên (Resource). Các Process1 và Process2 hoạt động song song, và được đồng bộ với Resource thông qua các kênh: báo tín hiệu yêu cầu sử dụng (appr1[i] hoặc appr2[i]), dừng (stop1[i] hoặc stop2[i]), được phép sử dụng(go1[i] hoặc go2[i]) và rời khỏi vùng tài nguyên (leave[i]).

Mô hình hóa các đối tượng

Khai báo biến toàn cục

const int N1 = 2; // # Process1

const int N2 = 2; // # Process2

typedef int[0,N1-1] id_t1; // Mã quá trình

typedef int[N1,N1+N2-1] id_t2; // Mã quá trình

typedef int[0,N1+N2-1] id_t3;

chan appr1[N1], stop1[N1], leave[N1+N2], appr2[N1+N2], stop2[N1+N2]; //các tín hiệu đồng bộ giữa các mô hình

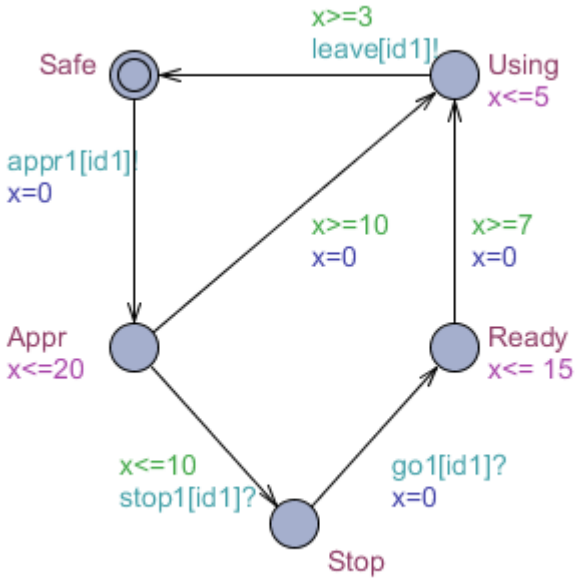
urgent chan go1[N1],go2[N1+N2]; // không có trễ khi đồng bộ chuyển tiếp

Khuôn mẫu Process1

Được truyền tham số là mã quá trình (đánh số từ 0 đến N1-1), biến địa phương là đồng hồ x. Quá trình này gồm năm trạng thái: Safe; Appr; Stop; Ready; Using trong đó: Safe là trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên; Appr là trạng thái đăng kí sử dụng nguồn tài nguyên; Stop là trạng thái chờ đến lượt sử dụng; Using là trạng thái sử dụng nguồn tài nguyên.

Các bước chuyển trạng thái của quá trình này được diễn tả như sau: Process1 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr1[i]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong khoảng thời gian delay1(demo là 20) (là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop1[i]?) từ bộ điều khiển thì lập tức chuyển sang trạng thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go1[i]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có

tín hiệu (stop1[i]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[i]!) thì chuyển sang trạng thái Safe.



Hình 4.16 Ô-tô-mát Process1 của hệ thống Process-Resource V2

Dựa vào phân tích trạng thái và các bước chuyển trạng thái ô-tô-mát Process1 được thiết kế như sau (xem hình 4.16):

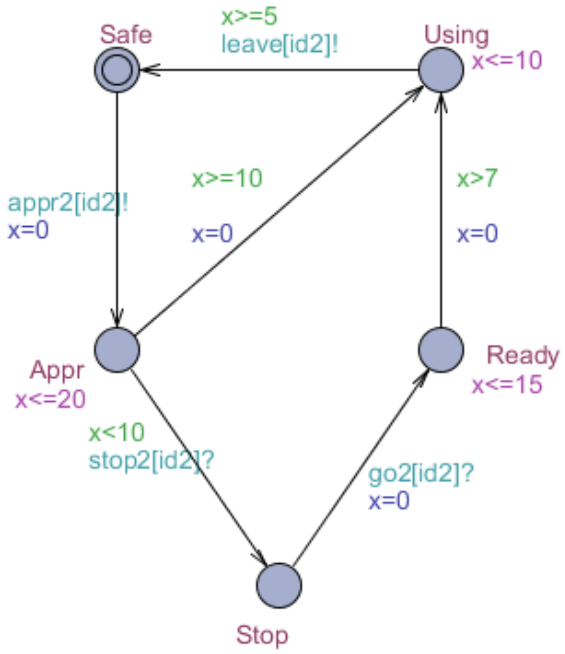
Khuôn mẫu Process2

Được truyền tham số là mã quá trình (được đánh số thứ tự từ N1 đến N2-1), biến địa phương là đồng hồ x. Khuôn mẫu này gồm 5 trạng thái: Safe; Appr; Stop; Ready; Using trong đó: Safe là trạng thái chưa có nhu cầu sử dụng nguồn tài nguyên; Appr là trạng thái đăng kí sử dụng nguồn tài nguyên; Stop là trạng thái chờ đến lượt sử dụng; Using là trạng thái sử dụng nguồn tài nguyên.

Các bước chuyển trạng thái của khuôn mẫu được diễn tả như sau: Process2 ở trạng thái Safe, nếu có nhu cầu sử dụng nguồn tài nguyên nó sẽ gửi tín hiệu (appr2[i]!) đến bộ điều khiển và chuyển sang trạng thái Appr, tại đây đồng hồ x sẽ được giới hạn trong khoảng thời gian delay2 (demo là 30s, là thời gian mà quá trình này sẽ dùng để sử dụng vùng tài nguyên +10s chờ tín hiệu). Nếu trong vòng 10s nó nhận được tín hiệu yêu cầu dừng (stop2[i]?) từ bộ điều khiển thì lập tức chuyển sang trạng thái Stop, và chờ ở đó đến khi nhận được tín hiệu cho phép sử dụng (go2[i]?) thì chuyển sang trạng thái Ready và ở đó sau 7s sẽ được sử dụng

vùng tài nguyên và chuyển sang trạng thái Using. Nếu sau 10s mà không thấy có tín hiệu (stop2[i]) thì nó chuyển sang trạng thái được sử dụng Using. Ở trạng thái Using đúng 5s (thời gian để ra khỏi vùng tài nguyên) thì báo cho bộ điều khiển tín hiệu đã sử dụng xong (leave[i]!) và chuyển sang trạng thái Safe.

Với phân tích trạng thái và các bước chuyển trạng thái như trên, ô-tô-mát Process2 được thiết kế như sau (xem hình 4.17):



Hình 4.17 Ô-tô-mát Process2 của hệ thống Process-Resource V2

Khuôn mẫu Resource

Khuôn mẫu này được khai báo biến và hàm như sau:

```
id_t3 list[N1+N2+1];
int[0,N1+N2] len;
// Put an element at the end of the queue
void enqueue(id_t3 element)
{
    list[len++] = element;
}
// Remove the front element of the queue
void dequeue()
```

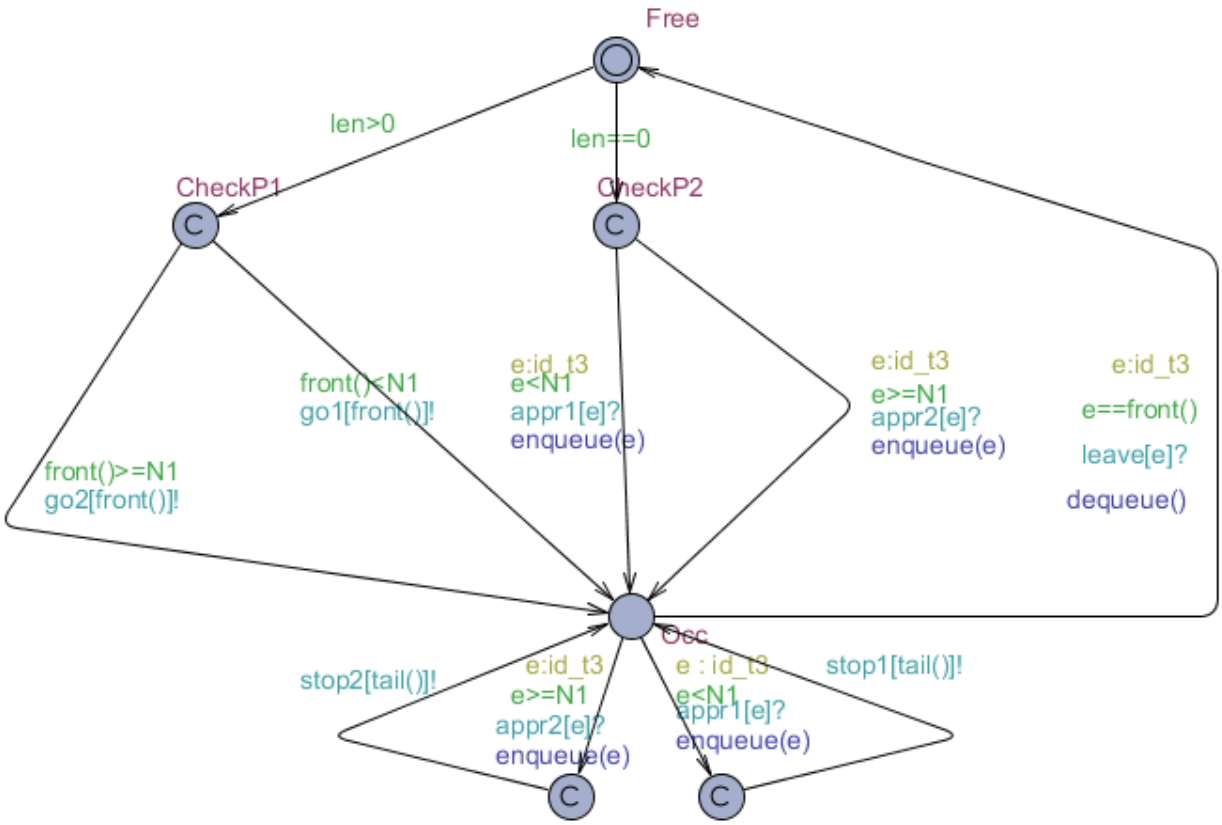
```
{
    int i = 0;
    len -= 1;
    while (i < len)
    {
        list[i] = list[i + 1];
        i++;
    }
    list[i] = 0;
}
// Returns the front element of the queue
id_t3 front()
{
  return list[0];
}
// Returns the last element of the queue
id_t3 tail()
{
  return list[len - 1];
}
```

Khuôn mẫu này gồm có sáu trạng thái: Free; CheckP1; CheckP2; Occ; RS1; RS2 trong đó: Free là trạng thái nguồn tài nguyên rảnh; CheckP1, CheckP2 là các trạng thái kiểm tra xem quá trình xếp hàng đầu tiên là thuộc nhóm nào; Occ là trạng thái tiếp nhận thông tin đăng kí xếp hàng; RS1, RS2 là các trạng thái xếp hàng cho các quá trình thuộc nhóm 1 và nhóm 2.

Các bước chuyển trạng thái được diễn tả như sau: Nếu nguồn tài nguyên đang ở trạng thái rảnh mà số quá trình đang có nhu cầu xử dụng trong hàng đợi lớn hơn không thì chuyển sang trạng thái kiểm tra xem quá trình đầu tiên thuộc nhóm 1

hay nhóm 2, rồi gọi ra quá trình đầu tiên đó cho sử dụng trước và chuyển sang trạng thái Occ. Nếu không có quá trình nào trong hàng đợi thì chuyển sang trạng thái kiểm tra xem tín hiệu báo nhu cầu sử dụng là thuộc nhóm 1 hay nhóm 2, đồng thời nhận được tín hiệu báo có nhu cầu sử dụng thì xếp nó vào hàng đợi và chuyển sang trạng thái Occ. Từ trạng thái Occ nếu vẫn tiếp tục nhận được tín hiệu có nhu cầu sử dụng thì xếp vào hàng đợi chuyển qua trạng thái RS1- nếu là thuộc nhóm 1 hoặc RS2-nếu thuộc nhóm 2, đồng thời đây là trạng thái Commited không cho phép trễ, đồng thời gửi tín hiệu stop1[i]! - nếu thuộc nhóm 1 hoặc stop2[i]! - nếu thuộc nhóm 2 rồi lập tức trở về trạng thái Occ. Từ trạng thái Occ nếu nhận được tín hiệu leave[i]? lập tức xóa quá trình đó khỏi hàng đợi và chuyển về trạng thái Free

Ô-tô-mát Resource được thiết kế như sau (xem hình 4.18):



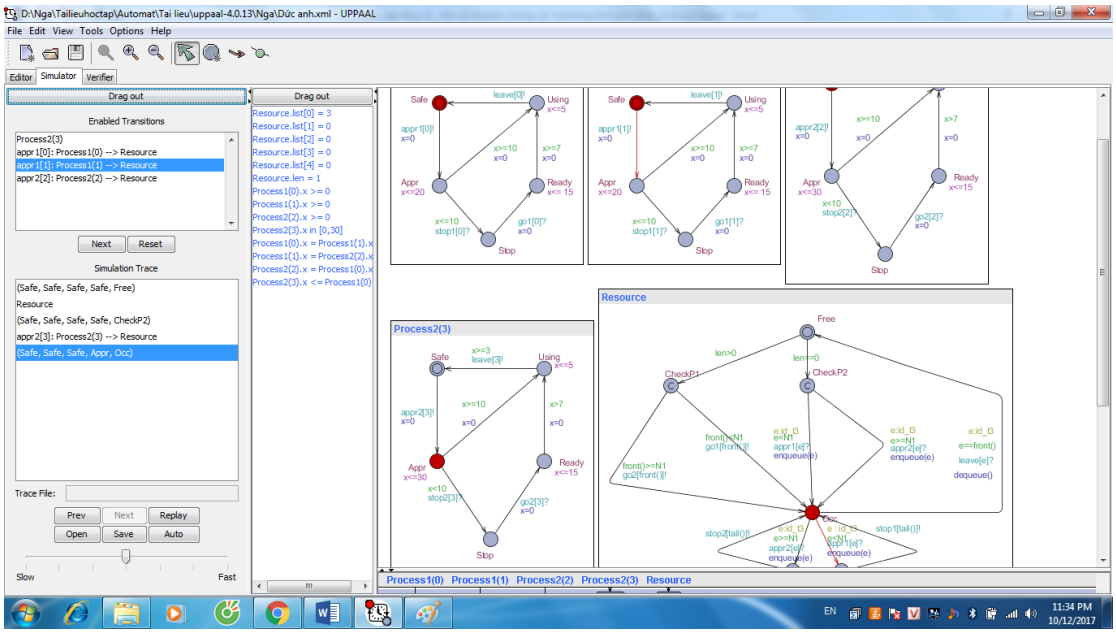
Hình 4.18 Ô-tô-mát Resource của hệ thống Process Resource V2

.

Việc khai báo các quá trình và các hằng sử dụng trong hệ thống này được thực hiện bằng câu lệnh sau: *system Process1, Process2, Resource;*

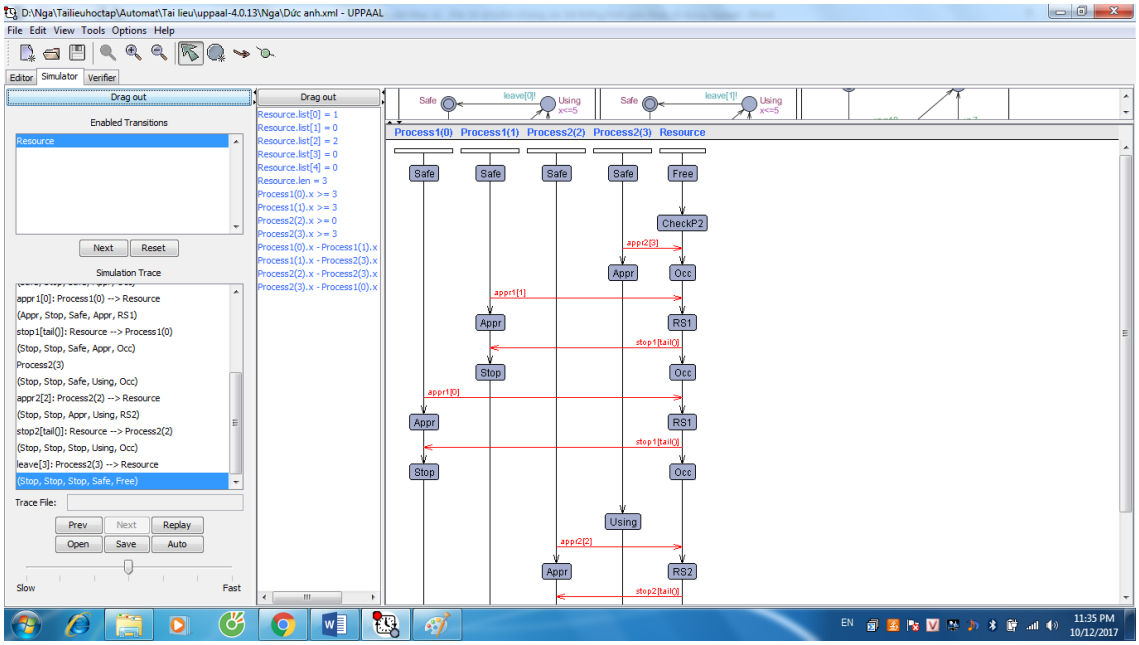
Mô phỏng sự vận hành của hệ thống

- Mô phỏng sự thay đổi trạng thái của các đối tượng (xem hình 4.19).



Hình 4.19 Màn hình mô phỏng sự vận hành của hệ thống Process ResourceV2

- Mô phỏng sự đồng bộ theo thời gian của các đối tượng (xem hình 4.20)



Hình 4.20 Màn hình mô phỏng sự vận hành trong hệ thống Process ResourceV2

Kiểm chứng hoạt động của hệ thống

- Kiểm chứng tính có thể đạt được (khả năng tới được 1 trạng thái nhất định)

Cú pháp: E<>φ (trong đó φ là công thức trạng thái)ψ

E<>Process1(0).Using kiểm tra xem một quá trình có chuyển sang trạng thái Using được không.

E<>Process1(2).Using.

E<> Resource.Occ: kiểm tra xem một quá trình có chuyển sang trạng thái Occ được không?

- Kiểm tra tính an toàn (một điều gì đó luôn luôn đúng)

Cú pháp: A[] và E[]

A[]Process1(0).Using && Process2(2).Using: đảm bảo tại một thời điểm chỉ có nhiều nhất một quá trình được sử dụng vùng tài nguyên (tính không xung đột).

A[] Resource.list[N1+N2] == 0 đảm bảo không có quá n quá trình trong hàng đợi.

- Kiểm tra tính liveness của hệ thống (tính chất này đảm bảo một điều gì đó trước sau gì cũng xảy ra)

Cú pháp: A<> φ: Chỉ ra rằng φ luôn được thỏa mãn

φ-->ψ : Khi φ thỏa mãn thì ψ cũng thỏa mãn.

Process1(0).Appr --> Process1(0).Using: Đảm bảo một quá trình 1 khi có nhu cầu sử dụng thì sẽ được sử dụng.

Process2(2).Appr --> Process2(2).Using

E<>Process1(0).Using imply Process2(3).Stop: đảm bảo một quá trình đang sử dụng thì tất cả các quá trình khác đều trong trạng thái phải chờ.

E<>Process1(0).Using and Process2(2).Using: Đảm bảo 2 quá trình khác nhau sẽ không cùng được sử dụng.

- Kiểm tra ô-tô-mát có rơi vào deadlock hay không

Cú pháp A[] not deadlock

KẾT LUẬN

Ngày nay, với sự phát triển ngày càng mạnh trong khoa học, kỹ thuật, quân sự và y tế. Các hệ thống có yếu tố thời gian ngày càng trở nên phổ biến và khẳng định vị trí quan trọng của nó trong mọi lĩnh vực của đời sống xã hội. Việc đặc tả và kiểm chứng một hệ thống thời gian thực trở nên cấp thiết hơn bao giờ hết và việc đưa công cụ để đặc tả và kiểm chứng tự động cho hệ thống thời gian thực là một xu thế tất yếu phù hợp với sự phát triển vũ bão của khoa học công nghệ.

Công cụ kiểm chứng Uppaal với giao diện thân thiện, khả năng kiểm chứng tối ưu dựa trên cơ sở mô phỏng sự vận hành của hệ thống theo thời gian cũng như kiểm chứng được các đặc tính quan trọng của hệ thống như tính an toàn, khả năng đến được, tính not deadlock thông qua các dòng lệnh đơn giản đã khiến Uppaal trở thành một công cụ kiểm chứng tốt nhất hiện nay đối với các hệ thống có yếu tố thời gian.

Việc nắm bắt và sử dụng được một công cụ tốt như Uppaal có ý nghĩa rất quan trọng, hơn nữa việc xây dựng nên các hệ thống đặc trưng có giá trị ứng dụng thực tế, đặc tả và kiểm chứng nó trên công cụ Uppaal là một nhiệm vụ rất cần thiết.

Tác giả đã tìm hiểu và sử dụng thành thạo bộ công cụ kiểm chứng Uppaal đồng thời nghiên cứu và xây dựng được 4 ví dụ về hệ thống thời gian giả định có tính ứng dụng trong thực tế (hệ thống tự động phân loại sản phẩm và hệ thống điều khiển việc sử dụng chung nguồn tài nguyên),vận dụng đặc tả và kiểm chứng các hệ thống đó bằng công cụ Uppaal. Đây là những bước đầu, các ví dụ về các hệ thống còn khá đơn giản. Hơn nữa điểm hạn chế của công cụ này là phải mô phỏng được cá hệ thống thời gian thành hệ ô-tô-mát thời gian (một nhiệm vụ không phải là dễ dàng đối với người sử dụng).

Trong tương lai, tác giả mong muốn sẽ mở rộng ra đặc tả và kiểm chứng cho các hệ thống phức tạp hơn với các ràng buộc thời gian chặt chẽ hơn, đồng thời tìm hiểu để có thể tự động hóa nhiều hơn ngay từ khâu đặc tả.

TÀI LIỆU THAM KHẢO

TIẾNG VIỆT

[1] Đỗ Đức Giáo (2000), *Toán rời rạc*, NXB Đại học Quốc Gia Hà Nội, Hà Nội.

[2] Phan Đình Diệu (1997), *Lý thuyết ô-tô-mát và thuật toán*, NXB Đại học và Trung học chuyên nghiệp, Hà nội.

[3] Vũ Đức Thi (1999), *Thuật toán trong tin học*, NXB Khoa học và Kỹ thuật, Hà Nội.

TIẾNG ANH

[4] A. Belinfante, J. Feenstra, R. d. Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink (1999), "Formal test automation: A simple experiment". *In 12th Int. Workshop on Testing of Communicating Systems*, pp 179–196.

[5] Christel Baier, Joost - Pieter Katoen (2008), *Principles of Model Checking*, The MIT Press Cambridge, Massachusetts London, England, chepter 1,9

[6] Edmund M. Clarke (2008), "The Birth of Model Checking", *Book 25 Years of Model Checking*, pp 1-26.

[7] Gerd Behrmann, Alexandre David, and Kim G. Larsen (2006), "A Tutorial on Uppaal 4.0", *Department of Computer Science, Aalborg University, Denmark.*

[8] Johan Bengtsson Kim Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi (1996), "UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems", *Proceedings of the DIMACS/SYCON workshop on Hybird systems III: verification and control*, pp 232-243.

[9] Kim G. Larsen, Paul Pettersson, Wang Yi. 1991. Model - Checking for Real - Time Systems. Proceedings of the 18th international colloquium on Automata, languages and programming. Page 115-126.

[10] Kim G. Larsen, Paul Pettersson, and Wang Yi (1996), "Diagnostic Model-Checking for Real-Time Systems". Appears in Alur, Henzinger and Sontag, editors, *DIMACS Workshop on Verification and Control of Hybrid Systems*, *HYBRID '96 Proceedings*, pp 575–586.

[11] Le Vo Hue Quan (2012). “Model Checking Real-Time Systems with Schedulers”. *Japan Adv anced Institute of Science and Technology*.

[12] Luca de Alfaro and Thomas A. Henzinger (2001), “Interface Automata”, *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pp 109-120.

[13] Marius Mikucionis Kim G, Larsen Brian Nielsen (2004), “T-UPPAAL: Online Model-based Testing of Real-time Systems”. *Proceeding ASE '04 Proceedings of the 19th IEEE international conference on Automated software engineering*, pp 396-397.

[14] Rajeev Ah and David L. Dill (1994), “A theory of timed automata”, *Theoretical Computer Science 126 (1994)*, pp 183-235.

[15] Wang Yi, Paul Pettersson, and Mats Daniels (1995), “Automatic Verification of Real-Time Communicating Systems by Constraint-Solving”, *Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques VII*, pp 243-258.

[16] UPPAAL. W. S. www.uppaal.com.