Kỷ yếu Hội nghị Khoa học Quốc gia lần thứ IX "Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR'9)"; Cần Thơ, ngày 4-5/8/2016
DOI: 10.15625/vap.2016.00076

PHƯƠNG PHÁP SINH MÔ HÌNH TỰ ĐỘNG CHO CÁC BIỂU ĐỒ UML 2.0

Lê Chí Luận [1,2], Phạm Ngọc Hùng[2]

[1] Khoa Công nghệ Thông tin, Đại học Công nghệ Giao thông Vận tải

[2] Khoa Công nghệ Thông tin, Đại học Công nghệ - Đại học Quốc gia Hà Nội

luanlc@utt.edu.vn, hungpn@vnu.edu.vn

TÓM TẮT— *Báo cáo này giới thiệu một số cải tiến cho phương pháp sinh mô hình cho các đối tượng trong biểu đồ tuần tự UML 2.0. Ý tưởng chính của một số cải tiến là phân tích biểu đồ tuần tự đầu vào có cấu trúc phức tạp chứa hầu hết các phân đoạn lồng ghép với nhau để xác định các thông điệp vào ra của từng đối tượng và thứ tự thực hiện của chúng nhằm xây dựng mô hình cho từng đối tượng. Các mô hình này sẽ được đặc tả bằng ôtômát vào/ra (IO automata). Các mô hình sinh ra bởi phương pháp đề xuất sẽ được sử dụng để kiểm chứng tính đúng đắn của thiết kế đã cho cũng như được sử dụng để sinh các ca kiểm thử cho phương pháp kiểm thử dựa trên mô hình.*

Từ khóa— *Sinh mô hình tự động, biểu đồ tuần tự, ôtômát vào/ra.*

I. GIỚI THIỆU

Đảm bảo chất lượng là một vấn đề quan trọng và tốn chi phí cao trong quá trình phát triển phần mềm. Tự động hóa một số bước trong quá trình đảm bảo chất lượng là mục tiêu hướng tới của các doanh nghiệp nhằm giảm chi phí phát triển. Ngoài ra, đối với những sản phẩm có yêu cầu chất lượng cao như hệ thống điều khiển máy bay, tàu ga, kỹ thuật quân sự, y tế, v.v., các phương pháp hình thức sẽ được yêu cầu áp dụng nhằm đảm bảo tính đúng đắn của thiết kế trước khi triển khai tại pha thiết kế và chứng minh tính đúng đắn của cài đặt so với thiết kế. Giải pháp phổ biến nhất hiện nay để chứng minh tính đúng đắn của thiết kế là phương pháp kiểm chứng mô hình [1], [6], [5]. Để áp dụng những phương pháp này, ta cần phải xây dựng các mô hình đặc tả chính xác hành vi của hệ thống cần kiểm chứng [3], [2], [7]. Hơn nữa, các mô hình này còn được sử dụng để áp dụng các kỹ thuật kiểm thử dựa trên mô hình nhằm tự động phát hiện các lỗi lập trình so với thiết kế. Tuy nhiên, việc xây dựng các mô hình này là một công việc khó khăn và tiềm ẩn nhiều lỗi. Các nghiên cứu hiện tại hầu hết giả sử các mô hình này đã có và đúng đắn. Trong thực tế, giả định này rất khó để hiện thực, nhất là từ phía các công ty phát triển phần mềm. Hạn chế trên là một trong những nguyên nhân chính dẫn đến các phương pháp kiểm chứng mô hình và kiểm thử dựa trên mô hình khó áp dụng trong thực tế.

Để giải quyết vấn đề nêu trên, một trong những hướng tiếp cận quan trọng là sinh mô hình đặc tả chính xác hành vi của hệ thống từ biểu đồ tuần tự UML có sẵn trong các công ty. Để giải quyết vấn đề này, nghiên cứu được đề cập trong [4] và hầu hết các nghiên cứu liên quan chỉ xây dựng một mô hình, thường được đặc tả bằng ôtômát đơn định hữu hạn trạng thái cho toàn bộ biểu đồ tuần tự. Các mô hình này chỉ mô tả được các thông điệp tuần tự theo thời gian như thứ tự vốn có của chúng. Cách tiếp cận này không thể hiện được tính hướng đối tượng vốn có của biểu đồ tuần tự là sự tương tác giữa các đối tượng với nhau, gửi và nhận các loại thông điệp, các điểm xuất phát và điểm đến của chúng, đặc biệt đối với các hệ thống tương tranh. Vì vậy, các phương pháp này chỉ phù hợp với các biểu đồ tuần tự đơn giản UML 1.x và các mô hình được sinh ra không mô tả được các cấu trúc điển hình của UML 2.0 như *Parallel, Loop* kết hợp với *Break*, v.v. Hơn nữa, nếu áp dụng các phương pháp kiểm chứng trên những mô hình này, chúng ta chỉ kiểm chứng được các thuộc tính an toàn (safety properties).

Một cách tiếp cận để giải quyết vấn đề trên được đề xuất trong [8]. Ý tưởng chính của phương pháp này là xây dựng một ôtômát vào/ra (I/O Automata) [6] cho mỗi đối tượng của biểu đồ tuần tự. Tuy nhiên, phương pháp này mới chỉ áp dụng cho các biểu đồ tuần tự UML 2.0 đơn giản. Cụ thể, phương pháp này chỉ áp dụng cho biểu đồ tuần tự chỉ chứa duy nhất một khối đơn, khối chỉ chứa duy nhất một phân đoạn và mới áp dụng được cho bảy khối là: *Option, Alternative, Parallel, Loop, Strict, Critical, Break.* Vì vậy, phương pháp này chưa áp dụng được trong trường hợp biểu đồ tuần tự có cấu trúc phức tạp, các khối lồng ghép, đan xen vào nhau.

Báo cáo này hoàn thiện phương pháp xây dựng ôtômát vào/ra cho mỗi đối tượng của biểu đồ tuần tự phức tạp, có đầy đủ các khối hay dùng hơn trong UML 2.0 nhằm giải quyết các vấn đề nêu trên. Phương pháp đề xuất tiến hành phân tích đối tượng của biểu đồ tuần tự có cấu trúc phức tạp chứa các phân đoạn lồng ghép với nhau thành các khối đơn. Tiếp đến, phương pháp xây dựng mô hình tương ứng từ các khối đơn đã được phân tích. Cuối cùng, các mô hình được sinh ra từ các khối đơn của đối tượng sẽ được ghép nối với nhau theo thứ tự từ trong ra ngoài và từ trên xuống dưới để được mô hình duy nhất cho mỗi đối tượng tương ứng trong biểu đồ tuần tự.

Phần còn lại của báo cáo sẽ được trình bày theo cấu trúc sau. Phương pháp phân tích biểu đồ tuần tự thành các khối đơn được trình bày ở phần II. Phương pháp sinh mô hình từ các khối đơn tương ứng của đối tượng trong biểu đồ tuần tự trình bày ở phần III. Tiếp đến phần IV, trình bày phương pháp xây dựng ôtômát vào/ra cho đối tượng từ biểu đồ tuần tự. Phần V trình bày kết quả của thực nghiệm và so sánh với các nghiên cứu [8]. Cuối cùng, báo cáo được tổng kết ở Phần VI.

II. PHƯƠNG PHÁP PHÂN TÍCH BIỂU ĐỒ TUẦN TỰ THÀNH CÁC KHỐI ĐƠN

Một biểu đồ tuần tự bao gồm nhiều đối tượng và mỗi đối tượng có nhiều khối đơn lồng ghép với nhau. Việc bóc tách một đối tượng của biểu đồ tuần tự thành các khối đơn là rất cần thiết. Các khối đơn sau khi được bóc tách sẽ là đầu vào cho việc chuyển đổi sang ôtômát vào/ra sẽ được trình bày ở phần III. Phần này của báo cáo đề xuất thiết kế được biểu diễn bởi biểu đồ tuần tự của các đối tượng dưới dạng tệp xmi. Một công cụ được báo cáo phát triển để phân tích tệp xmi và tách thành các khối đơn của biểu đồ tuần tự.

Thuật toán 1. Phân tích biểu đồ tuần tự thành các khối đơn

Input: Biểu đồ tuần tự biểu diễn dưới dạng tệp xmi
Output: Danh sách các đối tượng của biểu đồ tuần tự được biểu diễn dưới dạng danh sách các khối đơn.

```
1 : create stack, singleFragmentStack
2 : create array sdObjectList, operandList, eventList and singleFragmentList
3 : For all element in xmi file do
4 :       If meet open tag then Switch element
5 :               case Object:
6 :                   create object; create singleFragment, push to singleFragmentStack; break
7 :               case Fragment:
8 :                    if stack is not null then
9 :                      create new event with fragment id and add to eventList; belong to the Operand on the top of
                         stack if stack is not null
10:                     create new singleFragment and push to singleFragmentStack
11:                    end if
12:                   create new fragment with singleFragment on top of singleFragmentStack and push to stack; break
13:               case Operand:
14:               create new operand and push to stack; break
15:               case Event:
16:         create new event and add to eventList; belong to the Operand on the top of stack if stack is not null; break
17:               case Constraint:
18:                 create new constraint and add to the fragment on the top of stack; break
19:               end switch
20:       else if meet close tag then
21:               if element is Operand then
22:               op = stack.pop();set op ∈ the Fragment on the top of stack; insert op to operandList
23:               else if element is Fragment then
24:               fragment = stack.pop()
25:              add fragment to singleFragment on top of singleFragmentStack
26:                        if singleFragmentStack has more than 1 item then
27:                           singleFragment = singleFragmentStack.pop;
28:                           add singleFragment to singleFragmentList
29:                       end if
30:               else if element is Object then
31:                        singleFragment = singleFragmentStack.pop;
32:                        add singleFragment to singleFragmentList
33:                        add singleFragmentList to object
34:                        add object to sdObjectList
35:           end if
36:       end if
37: end for
```

Thuật toán 1 mô tả quá trình phân tích tệp xmi đầu vào, phân tích biểu đồ tuần tự thành các đối tượng bao gồm các khối đơn. Trước tiên, thuật toán khởi tạo các đối tượng *stack, singleFragmentStack* (dòng 1) và các danh sách *sdObjectList, operandList, eventList* và *singleFragmentList* (dòng 2). Sau đó lần lượt đọc các *element* từ tệp xmi đầu vào. Nếu gặp *element* là thẻ mở, thuật toán kiểm tra xem đó là thẻ nào và tạo dữ liệu cho phù hợp. Có 5 loại thẻ có thể gặp là *Object* (dòng 5), *Fragment* (dòng 7), *Operand* (dòng 13), *Event* (dòng 15) và *Constraint* (dòng 23). Ứng với thẻ *Object*, thuật toán tạo đối tượng *singleFragment* và đẩy vào *singleFragmentStack* đồng thời tạo đối tượng *object* (dòng 6). Nếu gặp thẻ mở *Fragment*, stack sẽ được kiểm tra xem có phần tử hay không (dòng 8). Nếu có, thuật toán tạo một event giả với id giống như của fragment, thuộc *Operand* ở đỉnh stack và đưa vào eventList (dòng 9), đồng thời tạo đối tượng *singleFragment* và đẩy vào *singleFragmentStack* (dòng 10). Sau đó, thuật toán khởi tạo một *fragment* thuộc *singleFragment* ở đỉnh *singleFragmentStack* và đưa vào *stack* (dòng 12). Nếu gặp thẻ mở *Operand*, thuật toán khởi tạo đối tượng *Operand* và đưa vào *stack* (dòng 14). Nếu gặp thẻ mở *Event*, thuật toán tạo đối tượng *event* và đưa vào

eventList, *event* này sẽ thuộc *Operand* trên đỉnh *stack* nếu *stack* không rỗng, hoặc không thuộc *Operand* nào nếu ngược lại (dòng 16). Nếu gặp thẻ mở *Constraint*, thuật toán tạo *Constraint* cho *fragment* đầu tiên trên đỉnh *stack* (dòng 17). Trong trường hợp gặp thẻ đóng, thuật toán sẽ kiểm tra thẻ đóng đó là gì để xử lý. Có 3 trường hợp thẻ đóng có thể gặp là *Operand*, *Fragment* và *Object*. Trường hợp gặp thẻ đóng *Operand*, *Opeand* được đưa khỏi đỉnh *stack*, đánh dấu thuộc *fragment* ở trên đỉnh *stack* lúc này và đưa vào *opeandList* (dòng 22). Trường hợp gặp thẻ đóng *Fragment*, *Fragment* được lấy ra khỏi đỉnh *stack* và được đưa vào *singleFragment* ở đỉnh *singleFragmentStack* (dòng 24, 25). Sau đó, *singleFragmentStack* được kiểm tra xem có nhiều hơn một phần tử hay không (dòng 26). Nếu đúng, *singleFragment* được lấy ra khỏi *singleFragmentStack* và đưa vào *singleFragmentList* (dòng 27, 28). Trường hợp gặp thẻ đóng *Object*, *singleFragment* được lấy ra khỏi *singleFragmentStack* và đưa vào *singleFragmentList* (dòng 31, 32), sau đó *singleFragmentList* được đưa vào *Object* và *Object* được đưa vào *objectList* (dòng 33, 34). Sau khi kết thúc đọc tệp xmi, ta được *objectList* tương ứng với các đối tượng của biểu đồ tuần tự ban đầu. Mỗi phần tử trong *objectList* là một danh sách các *singleFragment* tương ứng được bóc tách từ đối tượng đó.

III. PHƯƠNG PHÁP SINH MÔ HÌNH TỪ CÁC KHỐI ĐƠN CỦA BIỂU ĐỒ TUẦN TỰ

Sau khi đã bóc tách mỗi đối tượng của biểu đồ tuần tự thành các khối đơn tương ứng, phần này trình bày phương pháp sinh ôtômát vào/ra từ các khối đơn, khối chỉ chứa nhiều nhất một phân đoạn của biểu đồ tuần tự [8]. Báo cáo này nghiên cứu xây dựng thuật toán chuyển đổi sang ôtômát vào/ra từ các khối đơn chứa các phân đoạn *Consider* và *Ignore*. Thuật toán chuyển đổi một trong bảy loại phân đoạn: *Option, Alternative, Loop, Break, Parallel, Strict, Critical* và trường hợp khối không chứa bất kì phân đoạn nào đã được trình bày trong [8].

Đầu vào của thuật toán là các khối đơn của biểu đồ tuần tự được mô tả bằng một bộ sáu SD = (E, FG, OP, C, num, frag), trong đó:

- E là tập các sự kiện E = $E^I \cup E^O$,
- E^I là tập các sự kiện nhận,
- E^O là tập các sự kiện gửi,
- FG là tập các phân đoạn, trường hợp khối đơn, FG chứa nhiều nhất một phần tử,
- OP là tập các Operand,
- C là tập các điều kiện C = {c_1, c_2, c_3.. c_k},
- *num* là danh sách các số thứ tự của event từ 0 đến n, và
- *frag* là một hàm chuyển từ E đến F.

Đầu ra của thuật toán là một ôtômát vào/ra tương ứng được mô tả bằng một bộ sáu O = (Q, $\sum^I$, $\sum^O$, δ, q_0, F), trong đó:

- Q = {q_0, q_1, q_2,.. , q_n} là tập các trạng thái,
- $\sum^I$ = {(c, e)| $c \in C, e \in E^I$ } là tập các kí tự vào,
- $\sum^O$ = {(c, e)| $c \in C, e \in E^O$ } là tập các kí tự ra,
- δ = Q x (C x E) → Q là tập các luật chuyển $\delta(q_i, < c,e >) = q_j$,
- q_0 là trạng thái khởi đầu, và
- F = {f_1, f_2,.. , f_m} là tập các trạng thái kết thúc.

Ôtômát vào/ra đầu ra được xây dựng bởi các quy tắc như sau [8]:

- Số lượng các trạng thái trong ôtômát chính bằng số lượng sự kiện trong SD: |Q| = |E| và Q = {q_0, q_1, …, q_n}
- Tập các điều kiện của ôtômát tương ứng là tập điều kiện của SD: C = {c| c ∈ C}
- Tập các sự kiện của ôtômát tương ứng là tập sự kiện của SD: E = {e| e ∈ E}
- Tập các kí tự vào của ôtômát được xác định: $\sum^I$ = {(c, e)| c ∈ C, e ∈ E^I}
- Tập các kí tự ra của ôtômát được xác định: $\sum^O$ = {(c, e)| c ∈ C, e ∈ E^O}
- Tập các luật chuyển δ và F được xác định bởi các trường hợp sau

Thuật toán 2: Thuật toán xác định tập các luật chuyển δ cho ôtômát vào/ra từ khối đơn chỉ chứa phân đoạn *Consider*

Trường hợp thứ hai được xét đến là khối đơn của biểu đồ tuần tự chứa duy nhất một phân đoạn *Consider*. Khi đó, FG của khối đơn sẽ có thêm tập các ràng buộc (constraint – CT). Sau khi có E = {e| e ∈ E} và C = {c| c ∈ C}, tập các trạng thái kết thúc F được xác định

F = {q_n} ∪ {q_i | e_i ∈ FG and e_n ∈ FG and e_i ∈ CT and $e_{i+1..n}$ ∉ CT}

Tập các luật chuyển δ được xác định bởi thuật toán.

1 : set $k = 0$
2 : For i from 1 to $|E|$ do
3 : If $e_i \in FG$
4 : If e_i in constraint CT set $\delta_k^{k+1} = <e_i>$; set k = k+1 end if
5 : end if

6 : else set δ_k^{k+1} =<e_i>; set k = k+1 end else
7 : end for

Thuật toán 2 xác định tập các luật chuyển δ cho ôtômát vào/ra từ khối đơn chỉ chứa phân đoạn *Consider*. Tập các quy tắc chuyển trạng thái δ của ôtômát vào/ra được xác định theo quy tắc: Trước tiên khởi tạo biến chỉ số trạng thái k = 0 (dòng 1). Với mỗi i từ 1 đến n (dòng 2), ta xét 2 trường hợp, trường hợp 1, nếu e_i thuộc FG (dòng 3), thuật toán kiểm tra xem e_i có thuộc constraint CT của FG không. Nếu đúng, trạng thái q_k và q_{k+1} có quy tắc chuyển trạng thái e_i hay δ_k^{k+1}= <e_i> và k được tăng lên 1 (dòng 4). Trường hợp 2, nếu e_i không thuộc FG, ta luôn có có quy tắc chuyển trạng thái giữa q_k và q_{k+1} hay δ_k^{k+1}= <e_i> (dòng 6). Sau vòng lặp với i, ta được δ là biểu diễn của tập các quy tắc chuyển trạng thái trong ôtômát vào/ra của khối đơn chỉ chứa một phân đoạn *Consider*.

Thuật toán 3: Thuật toán xác định tập các luật chuyển δ cho ôtômát vào/ra từ khối đơn chỉ chứa phân đoạn *Ignore*

Trường hợp thứ ba được xét đến là khối đơn của biểu đồ tuần tự chứa duy nhất một phân đoạn *Ignore*. Khi đó, FG của khối đơn sẽ có thêm tập các ràng buộc constraint CT. Sau khi có E = {e| e ∈ E} và C = {c| c ∈ C}, tập các trạng thái kết thúc F được xác định

F = {q_n} ∪ {q_i | e_i ∈ FG and e_n ∈ FG and e_i ∉ CT and $e_{i+1..n}$ ∈ CT}
Tập các luật chuyển δ được xác định bởi thuật toán.

1 : set *k= 0*
2 : For *i* from 1 to |*E*| do
3 : If e_i ∈ *FG*
4 : If e_i not in constraint CT set δ_k^{k+1} = <e_i>; set k=k+1 end if
5 : end if
6 : else set δ_k^{k+1} =<e_i>; set k=k+1 end else
7: end for

Thuật toán 3 xác định tập các luật chuyển δ cho ôtômát vào/ra từ khối đơn chỉ chứa phân đoạn *Ignore*. Tập các quy tắc chuyển trạng thái δ của ôtômát vào/ra được xác định theo quy tắc: Trước tiên khởi tạo biến chỉ số trạng thái k = 0 (dòng 1). Với mỗi i từ 1 đến n (dòng 2), ta xét 2 trường hợp. Trường hợp 1, nếu e_i thuộc FG (dòng 3), thuật toán kiểm tra xem e_i có thuộc constraint CT của FG không (dòng 4). Nếu không, trạng thái q_k và q_{k+1} có quy tắc chuyển trạng thái e_i hay δ_k^{k+1}= <e_i> và k được tăng lên 1 (dòng 4). Trường hợp 2, nếu e_i không thuộc FG, ta luôn có có quy tắc chuyển trạng thái giữa q_k và q_{k+1} hay δ_k^{k+1}= <e_i> (dòng 6). Sau vòng lặp với i, ta được δ là biểu diễn của tập các quy tắc chuyển trạng thái trong ôtômát vào/ra của khối đơn chỉ chứa một phân đoạn *Ignore.*

IV. PHƯƠNG PHÁP XÂY DỰNG ÔTÔMÁT VÀO/RA CHO ĐỐI TƯỢNG TỪ BIỂU ĐỒ TUẦN TỰ

Sau khi có được các ôtômát vào/ra từ các khối đơn của các đối tượng trong biểu đồ tuần tự, vấn đề tiếp theo được xét tới là phương pháp ghép nối các ôtômát vào/ra đó thành một ôtômát vào/ra tương ứng với mỗi đối tượng. Dựa trên thuật toán bóc tách có đầu ra là danh sách các khối đơn theo thứ tự từ trong ra ngoài ta có thể ghép được ôtômát vào/ra cho cả đối tượng dựa trên việc ghép lần lượt hai ôtômát vào/ra theo thứ tự ưu tiên từ trong ra ngoài và từ trên xuống dưới.

Phương pháp ghép nối 2 ôtômát vào/ra được mô tả trong thuật toán 4.

Đầu vào của thuật toán là hai ôtômát $O_1(Q_1, \sum_1^I, \sum_1^O, \delta_1, q_{01}, F_1)$, $O_2(Q_2, \sum_2^I, \sum_2^O, \delta_2, q_{02}, F_2)$ và vị trí *index, trong đó:*

- O_1 là ôtômát vào/ra gốc bao hàm ôtômát O_2 tại vị trí *index*
- O_2 là ôtômát vào/ra được ghép vào O_1

Đầu ra của thuật toán là ôtômát O (Q, $\sum^I$, $\sum^O$, δ, q_0, F), *trong đó*:

- Q = {q_0, q_1, q_2,.. ,q_n} là tập các trạng thái,
- $\sum^I$ = {(c, e)| $c \in C, e \in E^I$ } là tập các kí tự vào,
- $\sum^O$ = {(c, e)| $c \in C, e \in E^O$ } là tập các kí tự ra,
- δ = Q x (C x E) → Q là tập các luật chuyển,
- C là tập các điều kiện C = {c_1, c_2, c_3.. c_k},
- E là tập các sự kiện E = $E^I \cup E^O$,
- E^I là tập các sự kiện nhận,
- E^O là tập các sự kiện gửi,
- q_0 là trạng thái khởi đầu, và
- F = {f_1, f_2,.. , f_p} là tập các trạng thái kết thúc.

Thuật toán 4: Ghép nối hai ôtômát vào/ra

1 : |Q| = |Q_1| + |Q_2| - 1; $\sum = \sum_1 \cup \sum_2$; $\delta = \delta_1$

//Thay đổi chỉ số các trạng thái O trong mảng Q
2 : For *i* from $|Q_1|$ to *index* do
3 : For *j* from $|Q_1|$ to *0* do set $\delta^{j}_{i+|Q2|-1} = \delta^{j}_{i}$; δ^{j}_{i} = null end for
4 : end for
8 : For *j* from $|Q_1|$ to *index* do
9 : For *i* from $|Q_1|$ to *0* do set $\delta^{j+|Q2|-1}_{i} = \delta^{j}_{i}$; δ^{j}_{i} = null end for
10: end for
//Ghép trạng thái khởi tạo
11: For *i* from 0 to |Q| do
12: if isset δ^{index}_{i} *// xét với mọi* q_i *có luật chuyển tới* q_{index}
13: δ^{index}_{i} = null;
14: For *j* from 0 to $|Q_2|$ do
15: if is set $\delta 2^{j}_{0}$ do $\delta^{index+j-1}_{i} = \delta 2^{j}_{0}$ endif *// xét với luật chuyển từ* q_0 *của* O_2
16: if $\delta 2^{j}_{0}$is Break do $\delta^{index+|Q2|-1}_{i} = \delta^{index+|Q2|}_{i+1}$ endif
//nếu gặp khối Break thì thêm luật chuyển khi không chạy vào Break
17: endfor
18: endif
19:endfor
//Ghép luật trạng thái kết thúc của O_2 *tới mọi trạng thái kế tiếp nó trong* O_1
20: For *j* from 0 to |Q| do
21: If isset δ^{j}_{index} do *// xét với luật chuyển từ* q_{index} *của O*
22: For all F_2 as f *// xét tất cả trạng thái kết thúc của* O_2 *trừ kết thúc của Break*
23: If *f* is not Break finite do set $\delta^{j}_{index+f-1} = \delta^{j}_{index}$ end If
24: endFor
25: if all F_2 *as f* and $f \neq 1$ do set δ^{j}_{index} = null endIf *//xóa luật cũ sau khi chuyển*
26: endIf
27:endFor
// đưa tất cả luật chuyển còn lại của O_2 *vào O*
28:For i from 1 to $|Q_2|$
29: For j from 0 to $|Q_2|$
30: If $q_i \notin F_2$ and $\delta 2^{j}_{i} \neq$ null do $\delta^{index+j-1}_{index+i-1} = \delta 2^{j}_{i}$ endIf
31: endFor
32:endFor
//ghép trạng thái kết thúc
33: For all F_1 *as* f_1
34: if $f_1 \geq$ index do set $f_1 = f_1 + |Q_2| - 1$ endIf *//gán lại chỉ số trạng thái cho* F_1
35: endFor
//nếu index là một trong trạng thái kết thúc của O_1*, đưa toàn bộ trạng thái kết thúc của* O_2 *vào* O_1
36:If *index* is $\in F_1$ do
37: For all F_2 as f_2 do set $f_2 = f_2 + index - 1$; insert f_2 to F_1 endFor
38:endIf
//nếu O_2 *là Break, đưa trạng thái kết thúc của khối Break vào* O_1
39:For all F_2 as f_2
40: If f_2 is Break finite do insert $f_2 = f_2 + index - 1$ to F_1 endIf
41:endFor
//kết thúc, trạng thái kết thúc của O chính là trạng thái kết thúc của O_1 *hiện tại*

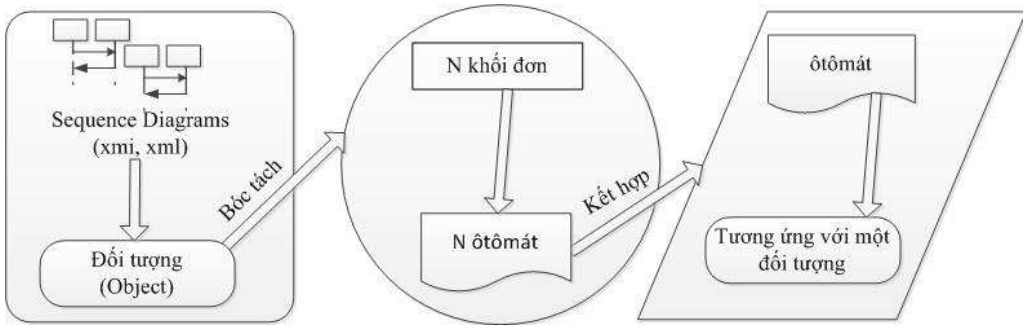
42: $F = F_1$

Thuật toán 4 mô tả phương pháp ghép nối ôtômát O_2 vào ôtômát O_1 tại vị trí *index*. Ôtômát O là kết quả của thuật toán được xác định bởi: Tập các trạng thái Q có độ lớn bằng độ lớn của Q_1 hợp với Q_2 loại trừ trạng thái q_{02} (dòng 1). Tập các kí tự (vào và ra) là hợp của các tập kí tự tương ứng (dòng 1). Đồng thời, thuật toán khởi tạo tập các luật chuyển $\delta = \delta_1$. Tiếp đó, thuật toán thay đổi chỉ số của các trạng thái có chỉ số lớn hơn *index* thêm một khoảng bằng $|Q_2| - 1$ (dòng 2 → dòng 10). Để ghép nối luật chuyển δ_2 vào δ, thuật toán triển khai 3 bước. Bước 1, với mỗi trạng thái q_i trong Q (dòng 11), thuật toán kiểm tra xem có luật chuyển từ q_i tới q_{index} hay không (dòng 12). Nếu có, δ^{index}_{i} sẽ được thay thế bằng các luật chuyển từ q_{02} tới các trạng thái trong O_2 (dòng 13, 14, 15). Trường hợp O_2 là khối *Break*, luật chuyển khi không chạy vào *Break* được thêm vào trực tiếp $\delta^{index+|Q2|-1}_{i} = \delta^{index+|Q2|}_{i+1}$ (dòng 16). Bước 2, với mỗi trạng thái q_j trong Q (dòng 20), thuật toán kiểm tra xem có luật chuyển từ q_{index} tới q_j hay không (dòng 21). Nếu có, với mỗi trạng thái kết thúc không thuộc khối *Break* của O_2, ta đều có luật chuyển tới q_j (dòng 22, 23). Luật chuyển từ q_{index} tới q_j được bỏ sau khi thay thế (dòng 25) trong trường hợp không có trạng thái kết thúc nào của O_2 ở vị trí đầu tiên.

Bước 3, đưa tất cả những luật chuyển còn lại của δ_2 vào δ (dòng 28 → dòng 32). Tiếp theo, thuật toán ghép nối trạng thái kết thúc của O từ trạng thái kết thúc của O_1 và O_2. Trước tiên, thuật toán tăng chỉ số cho các trạng thái kết thúc của O_1 có chỉ số lớn hơn *index* một khoảng bằng $|Q_2|$ - 1 (dòng 34). Sau đó, thuật toán xét xem *index* có trùng với một trạng thái kết thúc nào của O_1 hay không (dòng 36). Nếu có, đưa toàn bộ trạng thái kết thúc của O_2 vào O_1 sau khi tăng chỉ số thêm một khoảng index – 1 (dòng 37). Sau đó, nếu O_2 là khối *Break*, trạng thái kết thúc của khối *Break* sẽ được thêm trực tiếp vào O_1 (dòng 40). Cuối cùng, trạng thái kết thúc của O chính là trạng thái kết thúc của O_1 lúc này (dòng 42).

V. THỰC NGHIỆM

Để minh chứng cho tính đúng đắn và tính hiệu quả của phương pháp đề xuất chúng tôi đã cài đặt công cụ hỗ trợ phương pháp đề xuất bằng ngôn ngữ Java. Công cụ này có tên là SD2IOATool[1]. Kiến trúc phương pháp sinh mô hình cho biểu đồ tuần tự UML 2.0 được mô tả như Hình 1. Áp dụng công cụ này cho các biểu đồ tuần tự trong các trường hợp chỉ chứa duy nhất một phân đoạn hoặc chứa nhiều phân đoạn lồng ghép đan xen nhau, từ đó so sánh kết quả thu được với kết quả phương pháp đề xuất trong [8] như Bảng 1. Với biểu đồ tuần tự đầu vào có các khối đơn chứa *Consider*, *Ignore* và trong trường hợp các khối lồng nhau thì phương pháp đề xuất trong [8] chưa giải quyết được. Phương pháp đề xuất đã giải quyết được trường hợp biểu đồ tuần tự có các khối lồng ghép, đan xen nhau và giải quyết thêm được hai khối là *Consider* và *Ignore*.



Hình 1. Kiến trúc phương pháp sinh mô hình cho các đối tượng trong biểu đồ tuần tự UML 2.0.

Bảng 1. So sánh kết quả đề xuất và kết quả nghiên cứu trong [8]

Thứ tự	Khối đơn	Phương pháp đề xuất trong [8]	Phương pháp đề xuất
1	Không có khối nào	Yes	Yes
2	Alternative	Yes	Yes
3	Loop	Yes	Yes
4	Option	Yes	Yes
5	Break	Yes	Yes
6	Parallel	Yes	Yes
7	Critical	Yes	Yes
8	Strict	Yes	Yes
9	Consider	No	Yes
10	Ignore	No	Yes
11	Sequencing	No	No
12	Nagative	No	No
13	Assertion	No	No
14	Các khối lồng nhau	No	Yes

VI. KẾT LUẬN

Báo cáo này đã đề xuất một phương pháp sinh mô hình tự động cho biểu đồ tuần tự UML 2.0 nhằm cung cấp một giải pháp đầy đủ cho việc sinh mô hình tương ứng cho các đối tượng trong biểu đồ tuần tự có cấu trúc phức tạp. Phương pháp đã phân tích biểu đồ tuần tự đầu vào, bóc tách mỗi đối tượng trong biểu đồ tuần tự thành các khối đơn. Ứng với mỗi khối đơn sẽ sinh mô hình tương ứng, sau đó ghép nối các mô hình của các khối đơn theo thứ tự từ trong ra ngoài và từ trên xuống dưới để thu được một mô hình duy nhất cho mỗi một đối tượng trong biểu đồ tuần tự. Báo cáo đã xây dựng và cài đặt công cụ SD2IOATool để minh chứng cho phương pháp. Các mô hình sinh ra rất quan trọng được sử dụng để kiểm chứng tính đúng đắn của thiết kế và sinh ra các test case để áp dụng cho kiểm thử dựa trên mô hình. Việc sinh mô hình khắc phục được nhược điểm rất lớn trong việc áp dụng các phương pháp hình thức là xây dựng mô hình. Công cụ được xây dựng và cài đặt SD2IOATool được áp dụng cho đầu vào là hai biểu đồ có cấu trúc phức tạp và cho kết quả là các mô hình tương ứng của các đối tượng ứng với các biểu đồ tuần tự đầu vào.

[1] http://www.coltech.vnu.edu.vn/~hungpn/SD2IOATool/

Tuy nhiên, việc sử dụng các mô hình sau khi sinh ra để làm gì để việc kiểm chứng và kiểm thử hiệu quả như thế nào chưa được đề cập. Hơn nữa, chúng tôi đang tích hợp các mô hình được sinh ra này cho các công cụ kiểm chứng và kiểm thử dựa trên mô hình tương ứng để minh chứng cho tính hiệu quả và ý nghĩa của mô hình được sinh ra.

Lời cảm ơn

Nghiên cứu trong báo cáo này được thực hiện dưới sự tài trợ của đề tài mã số QG.16.31 do Đại học Quốc gia Hà Nội tài trợ.

TÀI LIỆU THAM KHẢO

[1] E. M. Clarke, O. Grumberg, and D. Peled, "Model Checking", MIT Press. Cambridge, Mass., 1999.

[2] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby and Hongjun Zheng, "Bandera: extracting finite-state models from Java source code", In Proceedings of the 22nd International Conference on Software Engineering, ICSE 2000, limerick Ireland, june 4-11, 2000, pages 439-448, 2000.

[3] L. B. Cuong and P. N. Hung, "A Method for Generating Models of Black-box Components", 4th International Conference on Knowledge and Systems Engineering, KSE 2012, Da Nang, Viet Nam, August 17-19, 2012, pages 177-222, 2012.

[4] H. M. Duong, L. K. Trinh, and P. N. Hung, "An Assume-Guarantee Model Checker for Component-Based Systems", In 2013 IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation and Vision for the Future, RIVF 2013, Ha Noi, Viet Nam, November 10-13, 2013, pages 22-26, 2013.

[5] D. Lorenzoli, L. Mariani and M. Pezzè, "Automatic generation of software behavioral models". In 30th International conference on Software engineering(ICSE 2008), Leipzing, Germany, may 10-18, 2008, pages 501-510, 2008.

[6] Lynch, N. A. & Tuttle, M. R., "An introduction to input/output automata", CWI Quarterly 2, 219-246, 1989.

[7] O. Tkachuk, M. B. Dwyer and C.S. Pasareanu, "Automated environment generation for software model checking", In 18th IEEE International Conference on Automated Software Engineering (ASE 2003), 6-10 October 2003, Montreal, Canada, pages 116-129, 2003.

[8] Zhang, C. & Duan, Z., Specification and Verification of UML 2.0 Sequence Diagrams Using Event Deterministic Finite Automata., in 'SSIRI (Companion)', IEEE Computer Society, pp. 41-46, 2011.

ON IMPROVEMENTS OF THE MODEL GENERATION METHOD FROM UML 2.0 SEQUENCE DIAGRAMS

Le Chi Luan, Pham Ngoc Hung

ABSTRACT*—This paper presents improvements for the model generation method from UML 2.0 sequence diagrams. The key idea of the improvements is to analyzing the given UML 2.0 sequence diagram with combined fragments in order to identify the incoming and outgoing messages for each object in the diagram. In this research, the generated models are represented by IO automata. The models generated by this method will be used for model checking of the corresponding system design. They are also useful for applying the model-based testing techniques.*