Tạp chí Tin học và Điều khiển học, T.23, S.2 (2007), 153–163

CÁC GIẢI THUẬT TÌM KIẾM VÀ LỰA CHỌN THÀNH PHẦN COTS TỐI ƯU THEO CÁC TIÊU CHÍ GIÁ THÀNH VÀ DƯ THỪA DỮ LIỆU

HUỲNH QUYẾT THẮNG[1], PHẠM THỊ QUỲNH[2]

[1]*Khoa CNTT, Trường ĐHBK Hà Nội*
[2]*Khoa CNTT, Trường ĐHSP Hà Nội*

Abstract. Component-based software development is gaining recognition as the key technology for the construction of high-quality, evolvable and large software systems in timely and affordable manners. Commercial components (COTS) is being used more and more in Component-Based Software Engineering for building software applications, and accordingly, some mechanisms are demanded by developers of software applications to describe, search, select and compose COTS components. In this paper we presented two algorithms applied to select COTS components witth optimization of cost and data ređunancy (size). The proposed algorithms are realized based on .NET platform and experimented on case study Geographic Translator Service. The comparison between proposed and original algorithm also is presented.

Tóm tắt. Phát triển phần mềm hướng thành phần (Component-Based Software Development - CBSD) là một trong những kỹ thuật tiêu biểu xây dựng các phần mềm lớn, phức tạp, giúp giảm thời gian, công sức và giá thành xây dựng phần mềm. Các thành phần thương mại (COTS) đang được sử dụng ngày càng nhiều trong công nghệ phần mềm dựa thành phần để xây dựng các ứng dụng phần mềm, và do đó các nhà phát triển ứng dụng đã yêu cầu một số cơ chế để mô tả, tìm kiếm, lựa chọn và xây dựng các thành phần COTS. Trong bài báo này chúng tôi trình bày giải thuật lựa chọn các thành phần COTS tối ưu theo dư thừa dữ liệu (kích thước) và giá thành, để tích hợp trong phần mềm cần xây dựng. Thuật toán đề xuất có độ phức tạp chấp nhận được. Các thử nghiệm được thực hiện trên một bài toán cụ thể: Thiết kế dịch vụ chuyển đổi khuôn dạng các ảnh không gian.

1. MỞ ĐẦU

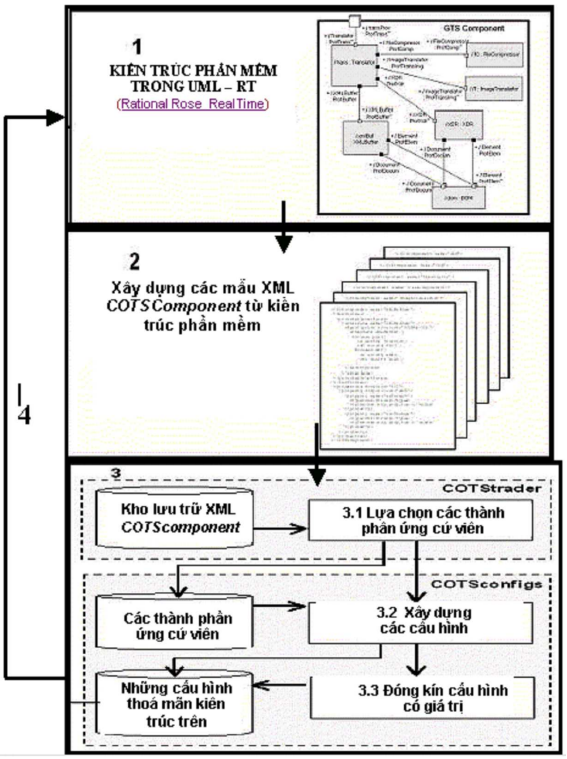
Phát triển phần mềm dựa thành phần (CBSD) cho phép người phát triển tạo ra những ứng dụng dùng một phần phần mềm được gọi là các thành phần. Thành phần phần mềm là một đơn vị cấu thành với giao diện được thỏa thuận trước và chỉ phụ thuộc dưới góc độ nội dung đã rõ ràng, chúng có khả năng liên kết, tương tác với nhau hình thành nên một hệ thống mới. Thành phần phần mềm có thể được triển khai một cách độc lập. COTS là một thành phần phần mềm mang tính thương mại (có thể được cấp giấy phép, hoặc được bán) cho phép đóng gói, phân phối, lưu giữ, sửa chữa và tuỳ biến theo ý người sử dùng, những thành phần này thường lớn và được lưu trữ trong các kho phần mềm.

Việc phát triển phần mềm dựa nền thành phần COTS ngày càng trở nên linh hoạt là do có sự gia tăng về chất lượng và sự đa dạng của các sản phẩm dựa nền thành phần COTS

như hệ điều hành, cơ sở dữ liệu, hệ thống tin nhắn, thư điện tử, GIS, GUI builders... Các sản phẩm này gồm các ứng dụng và các thành phần hoàn thiện và đã có mặt trên thị trường CNTT. Số lượng các thành phần COTS đang tiếp tục gia tăng, chất lượng cũng như khả năng ứng dụng của nó ngày càng được cải thiện. Hơn nữa, phát triển phần mềm dựa trên các thành phần COTS còn cung cấp khả năng mở rộng và biến đổi các ứng dụng phần mềm thông qua các hàm APIs, các ngôn ngữ plug- ins và script. Từ đó, nó có thể phù hợp với nhu cầu sử dụng của từng mục đích, ràng buộc khác nhau.

Trong phát triển phần mềm dựa thành phần COTS, việc lựa chọn thành phần COTS phù hợp là một trong những bài toán rất quan trọng. Trong bài báo này chúng tôi tập trung vào nghiên cứu mở rộng giải thuật lựa chọn các tổ hợp thành phần ứng cử viên COTS cho kiến trúc phần mềm cần xây dựng. Các mở rộng trong giải thuật lựa chọn này được xây dựng trên 2 tiêu chí: dư thừa dữ liệu của các thành phần và giá thành của thành phần. Ý tưởng chúng tôi áp dụng trong giải thuật xây dựng giải thuật lựa chọn là phương pháp nhánh cận.

Cấu trúc của bài báo được trình bày như sau. Mục 2 trình bày về tiến trình xây dựng phần mềm dựa thành phần COTS. Mục 3 sẽ trình bày giải thuật COTSConfigs và những đánh giá về các khả năng mở rộng và tối ưu thuật toán. Tiếp theo, Mục 4 là các mở rộng đề xuất cho thuật toán COTSConfigs. Trong Mục 5 là các mở rộng thông tin trong đặc tả COTS Document để lưu trữ giá thành thành phần. Mục 6 trình bày các đánh giá thử nghiệm đã thực hiện với các giải thuật đề xuất, cuối cùng là kết luận và hướng phát triển.



Hình 1. Tiến trình xây dựng phần mềm dựa trên các thành phần COTS

2. TIẾN TRÌNH PHÁT TRIỂN PHẦN MỀM DỰA TRÊN CÁC THÀNH PHẦN COTS

Mô hình tiến trình phát triển phần mềm hướng thành phần dựa trên các thành phần COTS, mô tả trong Hình 1, được chia thành ba giai đoạn ([1, 5, 6]):

Giai đoạn 1: Sử dụng các công cụ mô hình hóa (ví dụ UML-RT của bộ công cụ Rational Rose) để mô tả và thiết kế kiến trúc phần mềm. Mẫu đặc tả được xây dựng và trình bày trong [1, 5].

Giai đoạn 2: Sử dụng một tiến trình tự động để xuất các thông tin từ ký pháp UML - RT thành các mẫu đặc tả XML.

Giai đoạn 3: Tiến trình COTSTrader, thực hiện tìm kiếm danh sách các thành phần ứng cử

viên trong kho chứa mẫu XML. Cơ sở của tiến trình này là giải thuật COTSTrader. Giải thuật đã được trình bày chi tiết trong [1].

Tiếp theo tiến trình tìm kiếm là tiến trình COTSConfigs lựa chọn tổ hợp thành phần từ danh sách thành phần ứng cử viên để tạo ra các cấu hình thỏa mãn kiến trúc phần mềm yêu cầu. Trong [1, 2, 3] cũng trình bày chi tiết giải thuật COTSConfigs, áp dụng tư tưởng vét cạn các trường hợp thỏa mãn cấu hình. Một số cải tiến của giải thuật này cũng đã được trình bày trong [5, 6].

Bước cuối cùng trong giai đoạn này, khi tất cả các cấu hình đã được sinh, chúng ta thực hiện bước đóng gói các cấu hình để tạo ra một ứng dụng hoàn chỉnh.

Trọng tâm của bài báo tập trung vào bước thứ hai của giai đoạn 3 - tiến trình COTSConfigs lựa chọn tổ hợp thành phần.

3. THUẬT TOÁN COTSConfigs LỰA CHỌN TỔ HỢP CÁC THÀNH PHẦN COTS

3.1. Mô tả bài toán

Tiến trình sinh các cấu hình từ tập các thành phần ứng viên là một quá trình quan trọng trong toàn bộ quá trình tìm kiếm các thành phần COTS. Tiến trình này sẽ lựa chọn tập các thành phần phù hợp nhất với kiến trúc phần mềm cần xây dựng. Điều kiện phù hợp ở đây là phải lựa chọn các thành phần sao cho tập dịch vụ mà cấu hình đáp ứng được phải khớp với tập dịch vụ yêu cầu của kiến trúc phần mềm cần xây dựng.

Bài toán lựa chọn tổ hợp các thành phần COTS được phát biểu như sau ([1]): Cho một kiến trúc phần mềm A. Có một kho B chứa toàn bộ các thành phần ứng viên COTS phù hợp để lựa chọn tích hợp vào A. Cần phải sinh ra các cấu hình S là tập hợp của các thành phần ứng viên COTS có trong B, sao cho cấu hình S đáp ứng các dịch vụ/chức năng mà A yêu cầu. Như vậy cấu hình S cần đáp ứng hai điều kiện:

(i) Tập các dịch vụ được hỗ trợ bởi các thành phần của S phải trùng với tập các dịch vụ được hỗ trợ bởi A (tức là không có các lỗ hổng dịch vụ).

(ii) Không có hai thành phần nào của S cung cấp cùng một dịch vụ chung (tức là không có dịch vụ trùng)

Sau đây là thuật toán sẽ sinh ra các cấu hình S ([1]).

Giải thuật COTSConfigs

```
1   function COTSconfigs(i, Sol, S)
2   // 1 ⩽ i ⩽ size(CB(A)) khảo sát qua tất cả các khả năng có trong kho CB(A)
    (chứa danh sách các thành phần ứng cử viên)
3   // Sol là cấu hình tạm hiện thời đang được xây dựng trong giải thuật
4   // S chứa đựng tập hợp tất cả các cấu hình có giá trị đối với kiến trúc A
5   if i ⩽ size(CB(A)) then
6       for j := 1 to size(Ci.R) do//thực hiện với tất cả các dịch vụ trong C1
7           // thử gom dịch vụ Ci.Ri vào Sol
8   if {Ci.Ri} ∩ Sol.R = ∅ then // Ci.Ri ∉ Sol : R?
9       Sol := Sol ∪ {Ci.Ri}
10          if A.R ⊆ Sol.R then // Kiểm tra xem Sol có là một cấu hình không?
```

```
11          S := S ∪ {Sol} // Nếu Sol là một cấu hình, nó sẽ được gom nhóm vào
              trong S
12          else // nhưng nếu vẫn còn các dịch gối nhau và dịch vụ trùng nhau
13          configs(i, Sol, S) ; // tìm trong Ci...
14          endif
15              Sol := Sol − {Ci.Ri}
16          endif
17      endfor
18      configs(i + 1, Sol, S) // tiếp tục trong CB(A)
19  endif
20  endfunction
```

Các khởi tạo của giải thuật là $S =; Sol =$; configs $(1; Sol; S)$. Mỗi cấu hình (dòng 9) đều được sinh ra bằng cách thử tất cả các ứng cử viên, kết hợp dần các dịch vụ $Cj.Rj$ còn chưa chứa trong A, và bỏ đi những dịch vụ mà A đã có rồi (dòng 8 và dòng 10). Khi giải thuật kết thúc, biến S sẽ chứa tất cả các cấu hình cần thiết.

3.2. Đánh giá nhận xét về giải thuật

Thuật toán áp dụng tư tưởng giải thuật quay lui để thực hiện quá trình tìm kiếm lựa chọn các cấu hình hợp lệ. Nó sẽ sinh từ tập hợp các ứng cử viên trong kho $B : CB(A) = \{C_1, ..., C_k\}$, và từ kiến trúc phần mềm cần xây dựng A, một tập S các cấu hình hợp lệ (dòng 11). Không có trường hợp các cấu hình trong S còn tồn tại có lỗ hổng dịch vụ (một dịch vụ nào đó không được đáp ứng) hay các dịch vụ chồng chéo. Do đó, giải thuật sẽ chỉ sinh những cấu hình hợp lệ. Tuy nhiên, giải thuật này có nhược điểm là một giải thuật tìm kiếm vét cạn, với độ phức tạp là $O(2^n)$, trong đó n là số lượng các dịch vụ mà tất cả các thành phần trong $CB(A)$ có thể đáp ứng. Khi đó với một kho các thành phần ứng viên lớn, thì sẽ dẫn tới bùng nổ tổ hợp. Quá trình tìm kiếm như vậy sẽ không khả thi.

4. CÁC ĐỀ XUẤT CẢI TIẾN GIẢI THUẬT COTSCONFIGS LỰA CHỌN TỔ HỢP CÁC THÀNH PHẦN COTS

Để giảm độ phức tạp của thuật toán trên, chúng ta có thể đề xuất thay bằng giải thuật nhánh cận (brand and bound). Tư tưởng của thuật toán nhánh cận là nhờ vào một số các thông tin đã có để nhằm loại bỏ bớt một số các phương án chắc chắn không phải là tối ưu, tức là có thể lược bớt một số nút không cần thiết trên cây tìm kiếm. Vậy một số tiêu chí có thể đưa vào để tăng hiệu quả tìm kiếm như:

- Lựa chọn thành phần với dư thừa dữ liệu là tối thiểu.
- Lựa chọn thành phần với tổng giá thành là tối thiểu.

4.1. Cải tiến giải thuật lựa chọn thành phần với dư thừa dữ liệu tối thiểu

Khi xây dựng các cấu hình phù hợp với kiến trúc phần mềm yêu cầu, chúng ta có thể thu được nhiều kết quả cấu hình ([1, 2, 5, 6]). Trong đó, sẽ có những cấu hình là tổ hợp các thành phần mà ngoài những dịch vụ mà kiến trúc phần mềm đã yêu cầu, còn có những dịch vụ dư thừa không cần dùng. Nếu sử dụng các cấu hình này sẽ làm tăng độ phức tạp dẫn tới dư thừa dữ liệu và thành phần của hệ thống phần mềm được xây dựng. Đây là những cấu

hình không được lựa chọn. Vì vậy chúng ta cần phải xây dựng giải thuật tìm kiếm, lựa chọn COTSConfigs sao cho loại bỏ những cấu hình không cần thiết này.

Giải thuật nhánh cận tìm kiếm lựa chọn các thành phần với dư thừa dữ liệu là tối thiểu được đề xuất như sau.

Giải thuật COTSConfigs cải tiến với dư thừa dữ liệu tối thiểu

```
1    total = numOfInterfaces(CB(A)) //tổng số dịch vụ của một cấu hình
2    function brandAndBound(i, Sol, S) //giải thuật nhánh cận
3     if i ⩽ size(CB(A)) then
4          for j := 1 to size(Ci.R) do //thực hiện với tất cả các dịch vụ trong C1
             // thử gom dịch vụ Ci.Ri vào Sol
5              if {Ci.Ri} ∩ Sol.R = ∅ then // Ci.Ri ∉ Sol : R?
6                   Sol := Sol ∪ {Ci.Ri}
7              if A.R ⊆ Sol.R then //
8                   if numOfInterface(S) < total then
                      //số dịch vụ của cấu hình này còn ít hơn cả ngưỡng(mục tiêu) nên
                      //đây là một giá trị của ngưỡng về số dịch vụ của một cấu hình
                      mới
9                       total := numberOfInterface(S) // cập nhật ngưỡng mới
10                        S := // xóa các kết quả cũ
11                        S := S ∪ {Sol} // đưa Sol vào tập kết quả S
12                  else if numberOfInterface(S) = total then
                      //thỏa mãn ngưỡng về số dịch vụ
13                        S := S ∪ {Sol} // đưa Sol vào tập kết quả S
14                  endif
15             else // nhưng nếu vẫn còn các dịch vụ gối nhau hoặc dịch vụ trùng nhau
16                  configs(i, Sol, S) // tìm trong Ci...
17             endif
18             Sol := Sol − {Ci.Ri}
19             endif
20         endfor
21    if (numberOfInterface(S) + (n − m)× minOfInterface(CB(A))
        ⇐ total then
22         brandAndBound(i + 1, Sol, S)
23         endif
24         endif
25    endfunction
```

Ta có, một cấu hình tối đa sẽ là tập hợp của toàn bộ các các thành phần ứng viên. Như vậy, tức là số các dịch vụ tối đa của một cấu hình sẽ là tổng của các dịch vụ được cung cấp bởi các thành phần ứng viên. Vậy, trước tiên, ta gọi biến total là biến lưu tổng số dịch vụ của một cấu hình (dòng 1). Khởi tạo biến này bằng tổng số dịch vụ của toàn bộ các thành phần ứng viên. Đây chính là ngưỡng về tổng số dữ liệu của một cấu hình.

Tương tự như giải thuật chuẩn COTSConfigs chúng ta cũng lần lượt thử các cấu hình (dòng 5 và 6). Chúng ta thêm vào kiểm tra xem Sol có là một cấu hình không và nếu đúng, kiểm tra cấu hình này có tối thiểu về dữ liệu không (dòng 9). Giả sử tại một thời điểm trong quá trình tìm kiếm ta có, kiến trúc A cần n dịch vụ, mà lúc này Sol đã thỏa mãn được m dịch vụ của A. Như vậy, còn cần $n-m$ dịch vụ của A nữa thì Sol thỏa mãn là một cấu hình. Tồi nhất, là $m-n$ dịch vụ này nằm trên $n-m$ thành phần khác nhau. Như vậy, điều kiện để tiếp tục tìm kiếm là tổng: (số dịch vụ hiện có của Sol cộng với $(n-m)$ thành phần nhân với số dịch vụ nhỏ nhất có thể của một thành phần) phải nhỏ hơn ngưỡng tổng số dịch vụ. Nếu thỏa mãn thì tiếp tục tìm kiếm, còn không thì dừng quá trình tìm kiếm.

4.2. Cải tiến giải thuật lựa chọn thành phần với tổng giá thành tối thiểu

Như ta đã biết, các thành phần COTS là các thành phần được đem ra thương mại. Vì vậy, khi xây dựng hệ thống phần mềm sử dụng các thành phần COTS, ngoài yếu tố kĩ thuật là phải đáp ứng được kiến trúc phần mềm, thì yếu tố về mặt giá thành của COTS cũng là một yếu tố rất quan trọng. Thông tin về giá thành của thành phần COTS được đi kèm trong tài liệu đặc tả về thành phần, COTS Document. Vì vậy, thuật toán sau đây được đề xuất để cải tiến thuật toán tìm kiếm để đưa ra những cấu hình phù hợp đồng thời phải thỏa mãn một giá thành cho trước.

Giải thuật COTSConfigs cải tiến với giá thành tối thiểu

```
1   total = numOfInterfaces(CB(A))
2   function brandAndBound(i, Sol, S) // giải thuật nhánh cận
3       if i ⩽ size(CB(A)) then
4           for j := 1 to size(Ci.R) do // thực hiện với tất cả các dịch vụ
              // thử gom dịch vụ Ci.Ri vào Sol
5               if {Ci.Ri} ∩ Sol.R = ∅ then // Ci.Ri ∉ Sol : R?
6                   Sol := Sol ∪ {Ci.Ri}
7               if A.R ⊆ Sol.R then
8               if totalCost(S) ⩽ maxCost then
9                   S := S ∪ {Sol} // đưa Sol vào tập kết quả S
10              endif
11          else // nhưng nếu vẫn còn các dịch gối và dịch vụ tr ùng
12              configs(i, Sol, S) // tìm trong Ci...
13          endif
14          Sol := Sol − {Ci.Ri}
15      endif
16      endfor
17      if (totalCost(S) + (n − m)× minOfCost(CB(A)) ⩽ maxCost then
          // nếu thỏa mãn thì tiếp tục tìm kiếm, còn không thì thôi
18              brandAndBound(i + 1, Sol, S)
19          endif
20      endif
21  endfunction
```

Trước tiên, thuật toán yêu cầu nhập vào ngưỡng giá của một cấu hình. Thuật toán sẽ thực hiện tìm kiếm tất cả các cấu hình thỏa mãn mà tổng giá thành của các thành phần sẽ nhỏ hơn ngưỡng giá cho trước

$$\text{maxCost} := \langle \text{khởi tạo một giá trị ngưỡng giá} \rangle.$$

Tương tự như giải thuật chuẩn COTSConfigs chúng ta cũng lần lượt thử các cấu hình (dòng 5 và 6). Chúng ta thêm vào kiểm tra xem Sol có là một cấu hình không và nếu Sol đã là một cấu hình kiểm tra cấu hình này có thỏa mãn về giá hay không.

Giả sử tại một thời điểm trong quá trình tìm kiếm ta có, kiến trúc A cần n dịch vụ, mà lúc này Sol đã thỏa mãn được m dịch vụ của A. Như vậy, còn cần $n-m$ dịch vụ của A nữa thì Sol thỏa mãn là một cấu hình. Tồi nhất, là $n-m$ dịch vụ này nằm trên $m-n$ thành phần khác nhau. Như vậy, điều kiện để tiếp tục tìm kiếm là tổng: (giá thành hiện tại của Sol cộng với $(m-n)$ thành phần nhân với giá nhỏ nhất có thể của một thành phần) phải nhỏ hơn ngưỡng giá cho trước ở trên.

4.3. Nhận xét về hai giải thuật cải tiến

Thuật toán áp dụng tư tưởng giải thuật nhánh cận để thực hiện quá trình tìm kiếm lựa chọn cấu hình hợp lệ và thỏa mãn tiêu chí dư thừa dữ liệu tối thiểu và tổng giá thành tối thiểu. Do đó, khi so sánh với giải thuật COSTConfigs cổ điển, số lượng nút cần duyệt trong hai giải thuật sẽ giảm đi rất nhiều. Vì khi chúng ta kiểm tra ngưỡng mà thấy không thỏa mãn thì thuật toán sẽ không tiếp tục đi theo nhánh đó nữa. Tuy nhiên, trong trường hợp tồi nhất thì độ phức tạp tính toán của hai giải thuật đã cải tiến này vẫn là $O(2^n)$.

Ngoài hai thuật toán đã cải tiến, tác giả còn bổ sung thêm các tiêu chí về tối thiểu hóa dư thừa dữ liệu và tổng giá thành. Vì vậy, cấu hình tìm được sau khi thực hiện hai giải thuật này là duy nhất và tối ưu nhất theo tiêu chí cho trước. Đây là một kết quả khá quan trọng khi kết quả của giải thuật COSTConfigs cổ điển cho biết tất cả các cấu hình hợp lệ.

5. CẢI TIẾN MẪU TÀI LIỆU COTSDocument, ĐẶC TẢ THÀNH PHẦN COTS

Để phục vụ cho mục đích tìm kiếm và lựa chọn thành phần COTS thì việc đầu tiên là phải định nghĩa một mẫu XML đặc tả COTSDocument chuẩn cho kho dữ liệu COTS XML. Để thực hiện đăng ký các dịch vụ vào kho dữ liệu COTS XML, các nhà cung cấp (Exporter) cần mô tả thông tin về các dịch vụ của mình theo mẫu đặc tả này. Các thông tin chính trong một mẫu đặc tả gồm có ([1]):

```
    // Định nghĩa IDL của giao diện thành phần theBuffer
module theBuffer {
    // giao diện cung cấp
    interface Buffer {
        void write(in long x)
        long read()
        }
}
    // giao diện yêu cầu
    interface out {
```

```
        oneway void print(in long x)
        }
    }
```

```
⟨?xml-stylesheet type="text/xsl" href="http://www.cotstrader.com/
                                                COTS-XMLStyle.xsl"?⟩
⟨COTScomponent name="theBuffer"
xmlns="http://www.cotstrader.com/COTS-XMLSchema.xsd"
xmlns:types="http://www.w3.org/2000/10/XMLSchema"⟩
    ⟨functional⟩ ... ⟨/functional⟩
    ⟨properties⟩ ... ⟨/properties⟩
    ⟨packaging⟩ ... ⟨/packaging⟩
    ⟨marketing⟩ ... ⟨/marketing⟩
⟨/COTScomponent⟩
```

• Các mô tả mang tính chức năng (Functional description) ([1]): Phần đầu tiên mô tả tất cả các khía cạnh mang tính chất tính toán của thành phần, bao gồm tập các giao diện cung cấp và yêu cầu. Phần này có 3 phần chính được gọi là *providedInterfaces, requiredInterfaces,* và *serviceAccessProtocols*. ([1, 5, 6]).

• Các mô tả không mang tính chức năng (Non-functional description): sử dụng các kiểu W3C cho mô tả các loại thông tin này.

• Các ràng buộc về đóng gói/kiến trúc: Chú ý rằng, cũng giống như phần trên, các thông tin về gói và phát triển các thành phần có thể được mô tả trực tiếp theo một ký pháp đặc biệt và miêu tả trong thẻ ⟨description⟩ của mẫu XML, hoặc được lưu ở ngoài và sử dụng tham chiếu *href* đến nó.

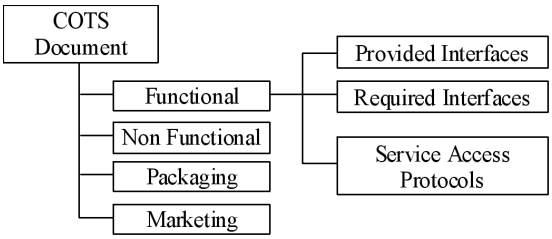
Đề xuất thành phần bổ sung trong COTS DOCUMENT

Các thông tin về marketing: Các loại thông tin điển hình ở đây cần mô tả bao gồm: bản quyền, chứng nhận, thông tin nhà cung cấp... Chú ý thông tin trong thẻ ⟨*expirydate*⟩ và thẻ ⟨*cost*⟩ là rất quan trọng, thúc đẩy các dịch vụ đã cũ được nâng cấp nhanh hơn và tăng tính cạnh tranh trong môi trường mở. Thông tin lưu trong hai thẻ này được các trader tìm kiếm và lựa chọn thành phần tối ưu theo giá thành.

```
⟨marketing⟩
⟨license href="http://www.cotstrader.com/samples/vendors/vendor3/theBuffer
                                                    /license.html"/⟩
⟨certificate href="http://www.cotstrader.com/samples/vendors/vendor3/theBuffer
                                                    /lcard.pgp"/⟩
⟨vendor⟩
⟨companyname⟩Vendor 3 Corp.⟨/companyname⟩
⟨webpage⟩http://www.cotstrader.com/samples/vendors/vendor3⟨/webpage⟩
⟨mailto⟩sales@cotstrader.vendor3.com⟨/mailto⟩
⟨address⟩
⟨zip⟩04120⟨/zip⟩
⟨street⟩Ctra Sacramento s/n⟨/street⟩
⟨city⟩Almeria⟨/city⟩
⟨country⟩Spain⟨/country⟩
```

⟨street⟩Campus de Teatinos⟨/street⟩
⟨city⟩Malaga⟨/city⟩
⟨country⟩Spain⟨/country⟩
⟨/address⟩
⟨/vendor⟩
⟨cost⟩500⟨/cost⟩
⟨currency⟩USD⟨/currency⟩
⟨/marketing⟩

Như vậy, những thông tin đặc tả trên được coi là tài liệu chuẩn mà các nhà cung cấp dùng đặc tả các dịch vụ. Từ đó, các trader mới có cơ chế tìm kiếm và lựa chọn thích hợp. Chức năng và giá thành của thành phần cũng có thể được đánh giá thông qua tài liệu đặc tả này. Tóm lại, thông tin trong mẫu tài liệu đặc tả XML COTSDocument được thể hiện qua Hình 2.



Hình 2. Các thông tin có trong mẫu đặc tả thành phần

6. LẬP TRÌNH THỰC HIỆN CÁC THUẬT TOÁN VÀ ĐÁNH GIÁ THỬ NGHIỆM

6.1. Xây dựng phần mềm thử nghiệm

Công cụ lựa chọn: Visual C#, .NET. Đây là môi trường, ngôn ngữ tiên tiến hiện nay. Điểm mạnh của môi trường này cho phép chúng ta dễ dàng tạo các ứng dụng nhanh trên nền hệ điều hành Windows, dễ dàng tạo giao diện một cách thân thiện. Điều đặc biệt quan trọng, là ngôn ngữ này hỗ trợ những thư viện xử lý các văn bản XML rất tốt. Để phục vụ thử nghiệm các giải thuật, hệ thống đã được xây dựng cùng KS. Nguyễn Thu Thuỷ và KS. Nguyễn Lê Minh, Khoa CNTT, Trường ĐHBK Hà Nội bao gồm các lớp cơ bản là: Interface, InterfaceSet, COTSComponent, Configure, Architecture, Repository. Đây là những lớp mô tả các khái niệm cơ bản phục vụ cho giải thuật như: khái niệm về tập giao diện của một thành phần, cấu hình, kho chứa các thành phần, kiến trúc phần mềm. Tiếp theo, hệ thống xây dựng các lớp ConfigGenerator, StandardConfigGenerator, MinCostConfigGenerator, MinDataConfigGenerator. Trong đó, lớp ConfigGenerator là một lớp trừu tượng mô tả quá trình tìm kiếm, sinh ra các cấu hình. Lớp StandardConfigGenerator được kế thừa từ ConfigGenerator sẽ lắp đặt cụ thể giải thuật sinh cấu hình, đây là giải thuật vét cạn đã đề cập ở trên. Tương tự MinCostConfigGenerator và MinDataConfigGenerator cũng kế thừa từ ConfigGenerator sẽ lắp đặt các thuật toán lựa chọn cấu hình tối ưu theo giá thành, và lựa chọn cấu hình giảm thiểu dư thừa dữ liệu.

6.2. Thử nghiệm các thuật toán

Dựa trên thư viện đã phát triển ta xây dựng chương trình phần mềm thử nghiệm các giải thuật đã trình bày. Chương trình cho phép nạp vào các tệp XML, một tệp chứa dữ liệu về kho chứa các thành phần ứng viên, và một tệp mô tả về kiến trúc phần mềm cần xây dựng. Thực nghiệm các thuật toán đã được tiến hành trên hai tệp XML:

- GTSCandidates.xml: Tệp chứa dữ liệu về kho chứa các thành phần ứng viên.
- GTSArchitecture.xml: Tệp chứa dữ liệu về kiến trúc phần mềm cần xây dựng.

Các tập XML mô tả các thành phần ứng viên cần xây dựng và các thành phần ứng viên có trong kho được lấy từ bài toán điển hình ([1, 2, 3]): Thiết kế dịch vụ chuyển đổi khuôn dạng các ảnh không gian, có tên gọi là GTS (Geographic Translator Service). Nếu tìm kiếm theo thuật toán vét cạn ban đầu thì số lượng nút trên cây tìm kiếm phải duyệt qua là 958 nút, và kết quả thu được là 24 cấu hình thỏa mãn. Nếu tìm kiếm theo giải thuật tối ưu về giá thành. Khi người dùng nhập vào giá thành là 2200 USD thì sẽ chỉ cho một kết quả cấu hình thỏa mãn, và số nút cần duyệt qua là 855. Nếu tìm kiếm theo giải thuật tối ưu về dữ liệu, tức tối ưu về không dư thừa các dịch vụ. Kết quả thu được chỉ là một cấu hình duy nhất, với số dịch vụ tổng cộng chỉ là 7, vừa đúng đối với số dịch vụ mà kiến trúc phần mềm yêu cầu. Kết quả so sánh thử nghiệm các giải thuật được tổng hợp trong Bảng 1.

Bảng 1. Kết quả thử nghiệm với các thuật toán

	(*)	(**)	(***)
Số nút cần duyệt	958	823	855
Số cấu hình thu được	24	1	1

(*) Giải thuật COTSConfigs chuẩn
(**) Giải thuật tìm kiếm thành phần COTS tối ưu theo dư thừa dữ liệu
(***) Giải thuật tìm kiếm thành phần COTS tối ưu theo giá thành

6.3. Đánh giá và kết luận

Kết quả thử nghiệm cho thấy áp dụng tư tưởng nhánh cận trong thuật toán lựa chọn và tìm kiếm các cấu hình hợp lệ cho giải thuật COTSConfigs đã đưa lại hiệu quả tốt. Số lượng các nút cần duyệt đã giảm đi là do ta không cần tìm kiếm toàn bộ các tổ hợp cấu hình trong bài toán duyệt cây mà chỉ cần lựa chọn nhánh phù hợp để đưa ra kết quả tìm kiếm cuối cùng.

Việc đưa vào các tiêu chí tìm kiếm cho các cấu hình cần lựa chọn - tối ưu theo dư thừa dữ liệu và tối ưu theo giá thành làm cho phương pháp xây dựng phần mềm dựa trên các thành phần COTS trở nên khả thi hơn trong thực tế. Trong bài báo đã trình bày các giải pháp nâng cao hiệu quả tiến trình tìm kiếm và lựa chọn thành phần COTS, phát triển giải thuật tìm kiếm và lựa chọn thành phần COTS tối ưu theo hướng giảm thiểu dư thừa dữ liệu và giá thành của thành phần COTS và thử nghiệm giải thuật vào hệ thống GTS (Geographic Translator Service).

Bài báo đã trình bày đánh giá thử nghiệm các giải thuật tìm kiếm, lựa chọn các thành phần phần mềm theo các tiêu chí. Đây là một vấn đề quan trọng trong phát triển phần mềm hướng thành phần cho phép lựa chọn thành phần phù hợp nhất với đặc tả kiến trúc cần xây dựng.

Một số hướng phát triển của bài toán này như kiểm tra tính đúng đắn của các đặc tả thành phần theo các đặc tả XML một cách tự động, mở rộng tiêu chí lựa chọn các thành phần phần mềm dựa trên đánh giá các tương tác của nó với các thành phần đã có sẵn trong

kiến trúc, v.v. Hiện tại chúng tôi đang phát triển các nghiên cứu theo các định hướng này.

TÀI LIỆU THAM KHẢO

[1] Luis Iribarne, Jose M. Trova, Antonio Vallecillio, A trading service for COTS components, *The Computer Journal* **47** (3) (2004).

[2] G. T. Heineman, W. T. Councill, *Component-Based Software Engineering* (Putting the pieces together) Addison Wesley, 2001. ISBN 0-201-70485-4.

[3] B. Meyer, *Object-Oriented Software Construction*, 2nd Ed. Series on Computer Science, Prentice Hall, 1997.

[4] N. Medvidovic, and R. N. Taylor, A classification and comparison framework for software architecture description languages, *IEEE Transactions on Software Engineering* (3) (2000) 70–93.

[5] Huỳnh Quyết Thắng, Phan Thế Đại, Phát triển phần mềm hướng thành phần - tiến trình tự động hóa khi xây dựng phần mềm phức tạp dựa nền thành phần COTS, *Kỷ yếu Hội thảo khoa học quốc gia về Nghiên cứu phát triển và ứng dụng Công nghệ thông tin và truyền thông ICT.RDA lần thứ II*, Hà Nội 24-25/9/2004 (165–174).

[6] Huỳnh Quyết Thắng, Đỗ Tuấn Dũng, Phát triển phần mềm hướng thành phần - lựa chọn và đánh giá các thành phần phần mềm hỗ trợ đa giao diện, *Kỷ yếu Hội thảo Khoa học Quốc gia lần thứ nhất - Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin FAIR 2003*, Hà Nội 4-5/10/2003 (423– 433).

[7] Huỳnh Quyết Thắng, Phạm Thị Quỳnh, Mở rộng các phép đo khả năng tích hợp và khả năng tái sử dụng và xây dựng phép đo tổng hợp cho phần mềm hướng thành phần, *Kỷ yếu Hội thảo khoa học quốc gia về Nghiên cứu phát triển và ứng dụng Công nghệ thông tin và truyền thông ICT.RDA lần thứ III*, Hà Nội 22-23/5/2006 (257–267).

Nhận bài ngày 19 - 10 - 2006

Nhận lại sau sửa ngày 29 - 6 -2007