잉여 없는 웹 서비스 조합을 위한 2단계 탐색 알고리즘

(Two-phase Search Algorithm for Web Services Composition Redundanty)

김 현 지 [†] 권 준 호 [††] 이 대 욱 [†††] 이 석 호 [††††]
(Hyeonji Kim) (Joonho Kwon) (Daewook Lee) (Sukho Lee)

요 약 최근 웹 서비스 관련 기술들이 많은 관심을 받고 있다. 그 중 하나인 웹 서비스 컴포지션에 대한 연구도 활발히 진행되고 있다. 웹 서비스 컴포지션은 기존에 존재하던 웹 서비스들을 조합하여 마치 하나의 서비스처럼 보여 주는 것이다. 웹 서비스 컴포지션을 사용하면 기존에 존재하지 않던 웹 서비스의 기능도 제공할 수 있으며 사용자에게 더 많은 질의 결과를 줄 수도 있다. 기존의 많은 웹 서비스 컴포지션 관련 연구들은 전향 혹은 후향 체인 방식을 사용하였다. 그러나 전향 체인 방식이나 후향 체인 방식은 질의와 상관없는 방향을 탐색하는 경우가 많기 때문에 많은 시간이 걸린다는 단점이 있다. 전향과 후향을 모두 사용하여 2단계로 컴포지션을 찾는 방식도 있으나, 이 방식은 컴포지션에 포함되지 않아도 되는 웹 서비스를 포함한 결과를 낸다는 문제점이 있다. 본 논문에서는 전향 단계와 후향 단계를 거쳐 웹 서비스 컴포지션을 찾는 2단계 웹 서비스 컴포지션 탐색 방법을 제안하였다. 전향 단계에서는 미리 구축한 연결 인덱스를 사용하여 좀 더 빠르게 후보 컴포지션을 찾는다. 후향 단계에서는 토큰을 사용하여 후보 컴포지션을 잉여 웹 서비스가 포함되지 않은 컴포지션으로 분해한다. 실험을 통해 2단계 웹 서비스 컴포지션이 기존의 한 방향으로 진행하는 방식보다 더 효율적이라는 것을 보였다. 또한 기존의 2단계 방식보다 더 많은 컴포지션 결과를 사용자에게 돌려주면서도 실행 시간 면에서 기존의 2단계 방식에 필적한다는 것을 보였다.

키워드 : 웹 서비스 컴포지션, 2단계 알고리즘, 잉여 없는 컴포지션

Abstract In recent years, the web services composition search has become an issue of great interest. The web services composition search is the process of integrating individual web services to yield desired behavior. Through the web services composition search, more sophisticated functionalities can be provided. Current solutions can be classified into three main classes: forward chaining approach, backward chaining approach and two-phase approach. However one-way chaining approaches, such as forward chaining approach and backward chaining approach have limitations of searching irrelevant web services. And two-phase approach has limitations of including redundant web services. In this paper, we propose an unredundant web services composition search based on the two-phase algorithm. The algorithm consists of a forward phase and a backward phase. In the forward phase, the candidate web services participating composition will be found efficiently by searching the Link Index. In the backward phase, unredundant web services compositions will be generated from candidate web services by using the Token Manager. The experimental results show that our proposed algorithm is more efficient than one-way chaining approaches. The experimental results also show that our algorithm

· 이 논문은 Korean DataBase Conference 2008에서 "잉여 없는 웹 서비스 컴포지션을 위한 2단계 알고리즘"의 제목으로 발표한 논문을 확장하였음

† 학생회원 : 티맥스소프트 연구원
hyeonji_kim@tmax.co.kr
†† 정 회 원 : 서울대학교 차세대융합기술원 연구원
joonhokwon@gmail.com
(Corresponding author임)
††† 학생회원 : 연세대학교 컴퓨터과학과 BK21 박사후연구원
daewook@gmail.com
†††† 종신회원 : 서울대학교 전기컴퓨터공학부 명예교수
shlee@snu.ac.kr

논문접수 : 2008년 6월 23일
심사완료 : 2009년 1월 2일

정보과학회논문지: 데이타베이스 제36권 제2호(2009.4)

can provide more solutions than previous two-phase approach and is comparable to previous one in execution time.

Key words : Web services composition, Two-phase algorithm, Unredundant composition

1. 서 론

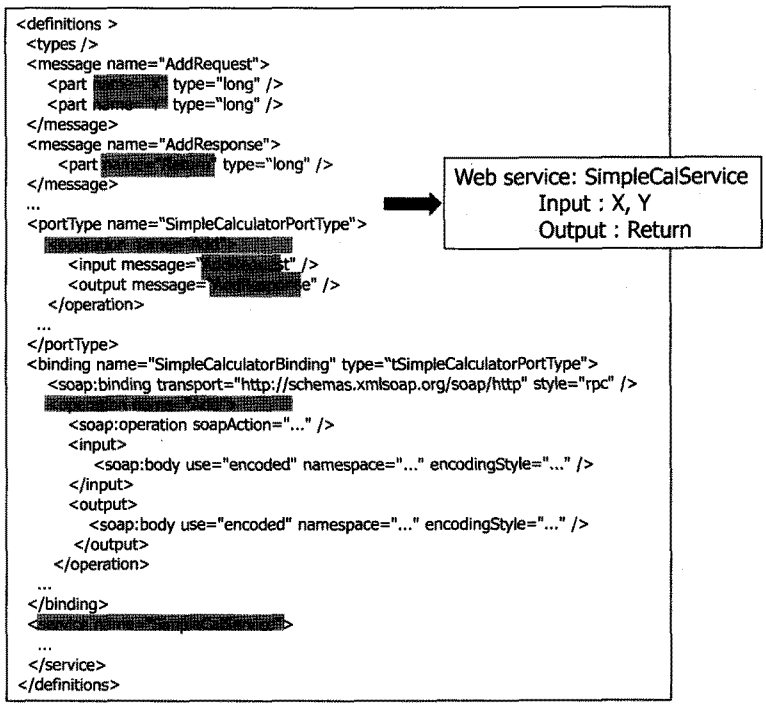
네트워크 기술이 발전하고 인터넷에 대한 수요가 증가함에 따라 웹과 관련된 기술 역시 빠르게 발전하고 있다. 초기의 웹은 사람과 컴퓨터 간의 상호작용을 위한 시스템으로, 원격 컴퓨터에 있는 웹 페이지를 사람이 볼 수 있는 형태로 변환하여 보여줄 뿐이었다. 하지만 이제 웹은 컴퓨터와 컴퓨터 사이의 상호작용을 지원하는 형태로 진화하고 있다. 이러한 대표적인 예가 웹 서비스(Web Service)이다.

웹 서비스(web service)[1]는 네트워크상에서 서로 다른 종류의 컴퓨터들 간에 상호작용을 하기 위한 소프트웨어 시스템이다. 웹 서비스는 WSDL(Web Service Description Language)[2]로 기술된다. WSDL에는 웹 서비스의 위치, 제공하는 연산과 입력 및 출력 파라미터와 같은 웹 서비스에 대한 정보가 들어있다.

그림 1은 간단한 계산기 웹 서비스를 정의한 WSDL 문서의 예이다. WSDL의 service 엘리먼트에서 웹 서비스의 이름 "SimpleCalService"를 추출할 수 있다. WSDL의 operation와 binding 엘리먼트에서 SimpleCalService가 수행하는 연산 "Add"를 추출할 수 있다. 다시 portType 엘리먼트를 거쳐 message 엘리먼트에서 해당 연산의 입/출력 파라미터도 추출할 수 있다. 위 WSDL에는 입력 파리미터로 X, Y값을 갖고 출력파라미터로 Return을 갖는 웹 서비스 SimpleCalService의 연산 Add가 정의되어 있다.

서비스 제공자는 웹 서비스의 정보를 UDDI(Universal Description, Discovery, and Integration)[3] 레지스트리에 등록한다. 웹 서비스 검색을 원하는 사용자는 입력과 출력을 명세한 질의를 UDDI에 요청한다. 요청을 받은 UDDI는 등록된 웹 서비스들을 검색하여 질의에 해당하는 서비스의 WSDL을 사용자에게 돌려준다. 그러면 사용자는 서비스 제공자와 SOAP(Simple Object Access Protocol)[4]을 통해 서로 메시지를 교환한다. 그림 2는 이 과정을 그림으로 표현한 것이다.

사용자가 UDDI에 질의를 요청했을 때 해당하는 웹 서비스가 등록 되어 있지 않다면 UDDI는 결과를 돌려줄 수 없다. 그러나 웹 서비스 컴포지션을 이용하면 사




```
<definitions >
 <types />
 <message name="AddRequest">
    <part [redacted] type="long" />
    <part [redacted] type="long" />
 </message>
 <message name="AddResponse">
     <part [redacted] type="long" />
 </message>
 ...
 <portType name="SimpleCalculatorPortType">
    [redacted]
      <input message="[redacted]t" />
      <output message="[redacted]e" />
    </operation>
  ...
 </portType>
 <binding name="SimpleCalculatorBinding" type="tSimpleCalculatorPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
    [redacted]
      <soap:operation soapAction="..." />
      <input>
        <soap:body use="encoded" namespace="..." encodingStyle="..." />
      </input>
      <output>
        <soap:body use="encoded" namespace="..." encodingStyle="..." />
      </output>
    </operation>
  ...
 </binding>
 <[redacted]>
  ...
 </service>
</definitions>
```

그림 1 웹 서비스의 WSDL 정의 예제

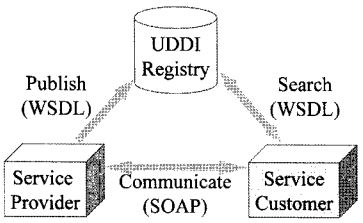


그림 2 웹 서비스 모델

용자가 명세한 입력과 출력을 가진 웹 서비스가 UDDI에 등록되어 있지 않더라도 다른 웹 서비스들을 조합하여 사용자가 원하는 기능을 수행하는 서비스를 돌려줄 수 있다[5-17]. 예를 들어 사용자가 프랑스어-독일어 사전을 찾고 있다고 하자. 그러면 사용자는 UDDI에 입력은 프랑스어, 출력은 독일어인 서비스를 찾아달라는 질의를 요청할 것이다. 컴포지션을 지원하지 않는 레지스트리라면 사용자 질의를 만족하는 단일 웹 서비스를 찾지 못해 서비스가 없다는 결과를 돌려 줄 것이다. 하지만 컴포지션을 지원하는 레지스트리라면, 프랑스어-독일어가 아니더라도 그림 3과 같이 프랑스어-영어 사전과 영어-독일어 사전을 연결한 컴포지션을 결과로 줄 수도 있다.

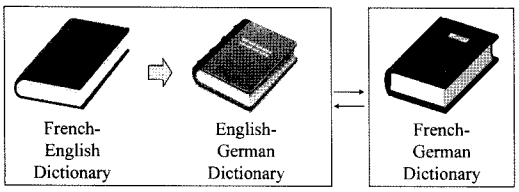


그림 3 웹 서비스 컴포지션의 예

본 논문에서는 웹 서비스 컴포지션 탐색을 위한 2단계 알고리즘을 제안한다. 2단계 웹 서비스 컴포지션은 전향단계와 후향 단계로 나뉜다. 전향 단계에서는 연결 인덱스를 사용하여 컴포지션에 포함될 후보 웹 서비스를 빠르게 검색한다. 후향 단계에서는 전향 단계에서 걸러낸 웹 서비스들만을 이용하여 컴포지션을 하기 때문에 문제 공간 크기가 줄어든다. 또한 토큰 관리자를 사용하여 잉여 웹 서비스가 발생하는지 여부를 빠르게 검사할 수 있다.

이 논문의 중요한 특징은 다음과 같이 요약할 수 있다.

첫째, 이 논문에서 잉여 없는 웹 서비스 컴포지션을 정형적으로 정의(formal definition)하였다.

둘째, 잉여 없는 웹서비스 컴포지션을 찾는 효율적은 2단계 알고리즘을 제안하였고, 제안한 알고리즘이 잉여 없는 웹 서비스 컴포지션을 반환한다는 것을 증명하였다.

셋째, 실험을 통하여 2단계 알고리즘과 기존의 컴포지션 검색 기법들을 비교 평가하였으며, 제안한 2단계 알고리즘의 효율성을 보였다.

이 논문의 구성은 다음과 같다. 2장에서는 웹 서비스 컴포지션과 관련된 배경 지식을 설명하고 연구 동기를 소개한다. 3장에서 웹 서비스 컴포지션 문제를 정의하고 4장에서 이를 해결하기 위한 구체적인 알고리즘을 제시한다. 5장에서는 실험을 통해서 제안한 시스템을 기존의 컴포지션 방법과 비교한다. 마지막으로 6장에서는 결론과 향후 연구를 제시한다.

2. 배경 지식 및 연구 동기

웹 서비스의 정보 그림 1처럼 WSDL 파일에 기술된다. 본 논문에서는 WSDL 파일에 기술된 정보 중에서 핵심 요소들인 서비스 이름과 입력 파라미터, 출력 파라미터로 웹 서비스를 추상화한다. 또한 사용자 질의는 사용자가 찾기를 원하는 웹 서비스의 입력 파라미터와 출력 파라미터를 명세한 것 표현한다. 이 논문에서는 웹 서비스 WS의 입력 파라미터를 WS_{in}, 출력 파라미터를 WS_{out}으로 표기한다. 또한 질의 q의 입력 파라미터는 q_{in}, 출력 파라미터는 q_{out}으로 표기한다.

Service	in	out
ws_1	BookName,Author	ISDN, Cost
ws_2	CCN, Cost	Authorize
ws_3	SSN	UserID
ws_4	SSN	UserID
ws_5	Authorize,ISDN,UserID	OrderID
ws_6	Authorize,ISDN,Cost,UserID	OrderID
ws_7	ISDN	Cost

Query	in	out
q	BookName,Author,CCN,SSN	OrderID

그림 4 예제 웹 서비스와 질의

그림 4는 예제 웹 서비스와 질의를 나타낸 것이다. 책을 주문하는 서비스를 원하는 사용자가 있다고 하자. 이 사용자는 책 이름(BookName), 작가(Author), 카드 번호(CCN), 주민등록번호(SSN)를 넣으면 주문 번호(OrderID)를 돌려주는 서비스를 찾으려 한다. 레지스트리에는 웹 서비스 $WS_1,\cdots,WS_7$이 있으나 혼자서 질의를 만족하는 웹 서비스는 없다. 하지만 웹 서비스들을 연결한다면 $WS_1 \rightarrow WS_2 \rightarrow WS_3 \rightarrow WS_5$와 같이 질의를 만족하는 컴포지션을 찾을 수 있다.

웹 서비스 컴포지션은 크게 수동(manual), 반자동

(semi-automated), 자동(automated) 컴포지션으로 나눌 수 있다[5]. 수동 컴포지션[6]은 문서 편집기와 같은 도구를 사용하여 사용자가 직접 컴포지션을 찾아서 웹 서비스의 작업 흐름을 지정해 주는 방식이다. 반자동 컴포지션[7]은 시스템이 웹 서비스를 추천해주면 사용자가 그 중에서 골라서 컴포지션을 찾는 방식이다. 자동 컴포지션[8-13]은 사용자가 질의 q를 입력하면 q_{in}으로 실행시킬 수 있으면서 q_{out}을 출력하는 컴포지션을 찾아준다.

이 장에서는 자동 컴포지션에 관한 기존 연구들 중 전향 체인 방식과 후향 체인 방식, 그리고 전후향 방식을 예제를 통해 설명한다.

2.1 전향 체인 방식

전향 체인 방식은 질의의 입력을 기준으로 컴포지션을 수행한다[8,9]. 처음에는 질의의 입력을 입력으로 갖는 웹 서비스를 찾고, 그 후에는 찾은 웹 서비스들의 출력을 입력으로 갖는 웹 서비스를 찾는다. [8]은 깊이 우선 탐색을 이용한 전향 체인 방식으로 최소인 컴포지션을 찾는다. [8]에서 정의하는 최소 컴포지션은 컴포지션에 속하는 웹 서비스가 하나라도 빠진다면 질의를 만족하지 않는 컴포지션이다.

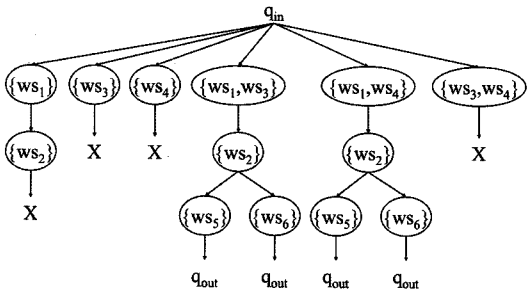


그림 5 전향 체인 방식의 컴포지션

예제 1. 그림 5는 그림 3의 예제를 가지고 [8]의 전향 체인 방식으로 컴포지션을 찾는 과정을 나타낸 것이다. q_{in}을 입력으로 갖는 웹 서비스를 찾으면 ws_1, ws_3, ws_4가 나온다. 이들의 모든 부분집합을 구해 노드를 만들고 각 노드를 탐색한다. $\{ws_1,ws_3\}$을 탐색하는 경우를 예로 든다. $(ws_{1out} \cup ws_{3out}) \cup q_{in}$을 입력으로 갖는 웹 서비스 ws_2,ws_7를 찾는다. ws_7의 출력은 ws_1의 출력에 완전히 포함되므로 고려하지 않는다. $\{ws_2\}$를 $\{ws_1,ws_3\}$의 자식 노드로 만들고, ws_1,ws_3,ws_2의 출력과 q_{in}을 입력으로 갖는 웹 서비스인 ws_5와 ws_6을 찾는다. ws_5와 ws_6의 모든 부분집합을 구하는데, $\{ws_5,ws_6\}$은 최소가 아니므로 $\{ws_5\}$와 $\{ws_6\}$만 노드로 만든다. 먼저 $\{ws_5\}$를 살펴본다. ws_5는 q_{out}인 OrderID를 출력하므로 루트에서 $\{ws_5\}$까지의 경로인 $\{ws_1,ws_3\}\rightarrow\{ws_2\}\rightarrow\{ws_5\}$는 질의를 만족하는 컴포지션이 된다. 탐색을 완료하면 $\{ws_1,ws_3\}\rightarrow\{ws_2\}\rightarrow\{ws_6\}$, $\{ws_1,ws_4\}\rightarrow\{ws_2\}\rightarrow\{ws_5\}$, $\{ws_1,ws_4\}\rightarrow\{ws_2\}\rightarrow\{ws_6\}$도 결과로 얻을 수 있다. □

그림 5에서 X로 표시된 노드는 더 이상 컴포지션을 진행할 수 없음을 나타낸다. 이처럼 전향 체인 방식은 질의의 출력과는 상관없는 방향도 탐색하는 경향이 있다.

2.2 후향 체인 방식

후향 체인은 전향 체인과 달리 q_{out}을 출력으로 갖는 웹 서비스를 찾는 것부터 시작한다[9-11]. 그리고 찾은 웹 서비스들의 입력 파라미터를 출력으로 갖는 웹 서비스를 찾는 일을 반복한다. 웹 서비스들의 입력 파라미터가 q_{in}에 있는 파라미터만으로 구성되어 있다면 종료한다.

예제 2. 그림 6은 [10]의 후향 체인 방식으로 컴포지션을 찾는 과정을 나타낸 것이다. 처음에는 질의의 출력인 OrderID를 출력으로 갖는 ws_5와 ws_6을 찾아서 노드로 만든다. 그 후 $\{ws_5\}$와 $\{ws_6\}$의 입력을 출력하는 웹 서비스들을 찾는다. 우선 $\{ws_5\}$를 살펴보자. ws_5의 입력인 {Authorize, ISDN, UserID}를 출력해 주는 $\{ws_1, ws_2,ws_3\}$과 $\{ws_1,ws_2,ws_4\}$를 찾을 수 있다. 그 후 $\{ws_1, ws_2, ws_3\}$의 입력을 출력으로 갖는 웹 서비스를 찾는다. 이 때 ws_1, ws_3의 모든 입력과 ws_2의 CCN은 q_{in}에 있기 때문에 ws_2의 Cost를 출력으로 갖는 웹 서비스만 찾으면 된다. Cost를 출력으로 갖는 웹 서비스는 ws_1과 ws_7이다. ws_1의 입력은 q_{in}에 모두 있으므로 $\{ws_1\}\rightarrow\{ws_1,ws_2,ws_3\}\rightarrow\{ws_5\}$는 컴포지션이 된다. 컴포지션 탐색을 완료하면 $\{ws_1\}\rightarrow\{ws_1,ws_2,ws_4\}\rightarrow\{ws_5\}$, $\{ws_1\}\rightarrow\{ws_1,ws_2,ws_3\}\rightarrow\{ws_6\}$, $\{ws_1\}\rightarrow\{ws_1,ws_2,ws_4\}\rightarrow\{ws_6\}$도 결과로 얻을 수 있다. □

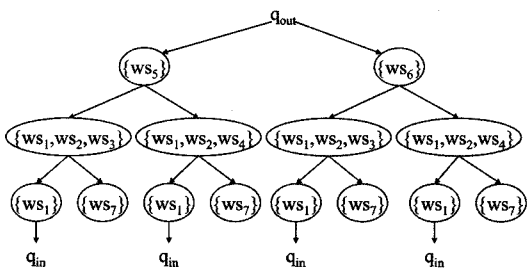


그림 6 후향 체인 방식의 컴포지션

후향 체인 방식 역시 계속 웹 서비스를 연결해도 q_{in}만 입력으로 갖는 웹 서비스를 찾지 못하는 경우가 생길 수 있다. 또한 후향 체인은 $\{ws_1\}\rightarrow\{ws_1,ws_2,ws_3\}\rightarrow\{ws_5\}$처럼 웹 서비스가 중복돼서 나타나는 경우도 있기 때문에 중복을 제거해줘야 한다.

전향 체인 방식과 후향 체인 방식은 너비 우선 탐색[10]이나 깊이 우선 탐색[8]으로 컴포지션을 찾는다. 너

비 우선 탐색은 체인의 길이가 가장 짧은 컴포지션을 찾아 준다. 하지만 컴포지션 결과를 출력해 주기 위해서는 트리 전체를 유지하고 있어야 한다. 깊이 우선 탐색은 현재 있는 노드에서 루트 노드에 이르는 경로만을 유지하고 있으면 결과를 출력해 줄 수 있지만, 해가 없는 경로에 깊이 빠질 수도 있다.

2.3 전후향 방식

전향 체인 방식이나 후향 체인 방식은 가능한 경우를 모두 고려하면서 탐색을 진행하기 때문에 느리다. 그에 비해 전후향 방식[12]은 빠른 시간에 컴포지션 결과를 돌려준다.

전향 단계는 전향 체인과 같이 질의의 입력을 기준으로 수행한다. 그러나 [8]에서는 매 단계에서 찾은 웹 서비스들을 부분집합으로 나눈데 반해 [12]에서는 나누지 않는다. 후향 단계에서는 q_{out}에 직접 혹은 간접적으로 기여하지 않는 웹 서비스를 제거한다.

예제 3. 처음에 입력 파라미터가 q_{in}의 부분 집합인 모든 웹 서비스들을 찾는다. ws_1, ws_3, ws_4를 찾을 수 있다. 그러면 q_{in}과 ws_1, ws_3, ws_4의 출력 파라미터들이 이용 가능하게 된다. 이용 가능하게 된 파라미터들을 이용하여 ws_2를 찾는다. 이를 q_{out}이 나올 때까지 반복하면 그림 6과 같은 결과를 얻을 수 있다. 후향 단계에서는 q_{out}에 기여하지 않는 웹 서비스를 제거한다. ws_5와 ws_6은 q_{out}인 OrderID를 출력하므로 제거되지 않는다. ws_2는 ws_5와 ws_6의 입력 파라미터인 Authorize를 출력하므로 역시 제거되지 않는다. ws_1, ws_3, ws_4도 마찬가지 이유로 제거되지 않는다. 그림 7은 전후향 방식으로 컴포지션을 수행한 결과이다. □

[12]에서 제안한 방식의 후향 단계는 시간은 적게 걸리지만 잉여 웹 서비스를 완전히 제거하지는 못한다는 문제가 있다. 그림 6의 컴포지션 결과는 q_{in}만으로 실행시킬 수 있는 웹 서비스들로 이루어져 있고, q_{out}을 출력하므로 질의를 만족하는 컴포지션이 된다. 하지만 ws_6과 ws_5 모두 OrderID를 출력하기 때문에 둘 중 하나를 제거해도 질의를 만족하는 데에는 아무런 문제가 없다. 즉 질의를 만족하는 데에 없어도 무방한 웹 서비스가 포함되어 있는 것이다.

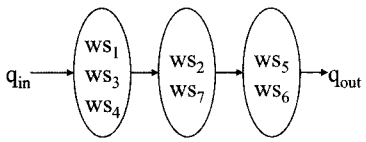


그림 7 전후향 방식 컴포지션의 결과

2.4 연구 동기

전향 체인 방식과 후향 체인 방식은 컴포지션을 찾는데 많은 시간이 걸린다는 문제가 있다. 또한 전후향 방식은 잉여 웹 서비스가 포함된 컴포지션을 결과로 낸다는 문제가 있다.

잉여 웹 서비스가 포함되어 있으면 다음과 같은 문제가 발생할 수 있다. 우선 컴포지션 된 웹 서비스를 실행시키는 시스템 관점에서 살펴보자. 컴포지션을 실행시키려면 컴포지션에 포함된 웹 서비스에 입력 파라미터로 넘겨 줄 값들을 준비하고 웹 서비스를 호출해야 한다. 만약 불필요한 웹 서비스가 포함되어 있다면 그만큼 시스템은 필요 없는 작업을 하는데 시간을 들이게 된다. 또한 잉여 웹 서비스의 실행시간이 컴포지션의 전체 수행 시간을 늦추는 악영향을 끼칠 수도 있다.

또 다른 문제는 사용자가 의도한 것과는 다른 결과를 돌려줄 수도 있다는 것이다. 그림 7과 같은 컴포지션은 주문을 체결하는 웹 서비스인 ws_5와 ws_6이 하나의 컴포지션에 포함 되어 있어 한 번의 결제로 주문이 두 번 될 수 있다. 이는 사용자가 의도했던 것과는 다른 결과일 것이다. 또한 ws_5와 ws_6는 별개의 웹 서비스이기 때문에 OrderID 값으로 다른 값을 출력할 수도 있다. 그러면 둘 중 어느 값을 선택해야 하는가 하는 문제가 생기게 된다. 따라서 그림 8과 같이 같은 파라미터를 출력하는 웹 서비스의 수를 최소화하고, 질의를 만족하는데 꼭 필요한 웹 서비스들만으로 컴포지션 한 결과를 사용자에게 돌려주어야 할 것이다. 또한 그림 8처럼 여러 개의 컴포지션을 돌려줄 수 있다면, 보다 더 적합하다고

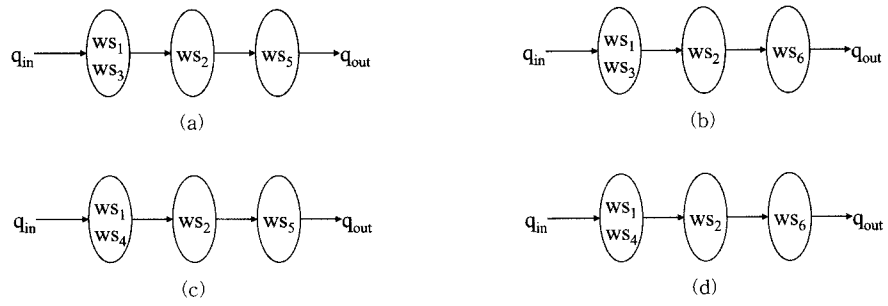


그림 8 제안하는 컴포지션 결과

생각하는 컴포지션을 사용자가 선택할 수 있게 제공해 줄 수 있을 것이다.

본 논문에서는 기존 전후향 방식의 단점을 보완하고 그림 8과 같이 잉여 웹 서비스가 없고, 되도록 많은 결과를 사용자에게 빠르게 돌려줄 수 있는 2단계 웹 서비스 컴포지션 방식을 제안한다. 2단계 웹 서비스 컴포지션은 전후향 방식처럼 전향 단계와 후향 단계를 거쳐 컴포지션을 찾는다. 결과로는 기존의 전후향 방식과는 달리 잉여 웹 서비스가 포함되지 않은 컴포지션을 1개 이상 돌려준다.

우선 전후향 방식의 전향 단계처럼 각 단계에서 실행이 가능한 웹 서비스들만을 골라내고 컴포지션의 길이를 결정한다. 그러면 후향 단계에서는 전향 단계에서 걸러진 웹 서비스들만을 대상으로 컴포지션을 할 수 있고, 깊이 우선 탐색을 하더라도 정해진 깊이 이상은 탐색하지 않아도 될 것이다. 잉여 웹 서비스가 포함되지 않은 컴포지션을 생성하기 위해 토큰 관리자를 사용한다.

3. 2단계 웹 서비스 컴포지션 시스템

이 장에서는 잉여 없는 웹 서비스 컴포지션을 정의하고, 2단계 웹 서비스 컴포지션 시스템의 구조를 설명한다.

3.1 문제 정의

웹 서비스 컴포지션은 그림 8과 같이 웹 서비스들의 집합을 순서 관계에 따라 나열한 시퀀스(sequence)로 나타낼 수 있는데, 같은 집합에 속한 웹 서비스들은 병렬(parallel)적으로 수행할 수 있다.

2.4절에서 언급했듯이 컴포지션 결과에 잉여 웹 서비스가 포함된다면 여러 문제들이 발생할 수 있다. 따라서 본 논문에서는 잉여가 없는 웹 서비스 컴포지션을 정의하고 잉여가 없는 컴포지션만을 결과로 돌려주려고 한다.

표 1은 웹 서비스 컴포지션을 정의하기 위해서, 본 논문에서 사용한 기호들과 그 의미를 보여준다.

잉여 없는 웹 서비스 컴포지션을 정형적으로 정의(formal definition)하면 다음과 같다.

정의 1. 질의를 만족하는 시퀀스

사용자 질의를 q라고 하고, 시퀀스 C=(W_1,W_2,W_3,…, W_l), $W_i(1 \le i \le l)$는 웹 서비스의 집합이라고 하자. 아래의 조건 i, ii, iii을 만족하는 시퀀스 C는 사용자 질의 q를 만족한다.

i. for all $ws \in W_1$, $ws_{in} \subseteq q_{in}$,

ii. for $2 \le i \le l$, $\bigcup_{ws \in W_i} ws_{in} \subseteq \left(\bigcup_{j=1}^{i-1} Out(W_j) \cup q_{in} \right)$,

iii. $q_{out} \subseteq \left(\bigcup_{i=1}^{l} Out(W_i) \cup q_{in} \right)$

정의 2. 최소 웹 서비스 집합

사용자 질의를 q라고 하고, 시퀀스 C=(W_1,W_2,W_3,…, W_l), $W_i(1 \le i \le l)$는 웹 서비스의 집합이라고 하자. 아래의 조건을 만족하는 경우, 웹 서비스 집합 W_i를 최소 웹 서비스 집합이라고 한다. 모든 $ws \in W_i$에 대해, 시퀀스(W_1,…,W_{i-1},W_i-{ws},W_{i+1},…,W_l)가 q를 만족하지 않는 경우 W_i를 최소 웹 서비스 집합이라고 한다.

정의 3. 잉여가 없는 컴포지션

사용자 질의를 q라고 하고, 시퀀스 C=(W_1,W_2,W_3,…, W_l), $W_i(1 \le i \le l)$는 웹 서비스의 집합이라고 하자.

모든 $W_i(1 \le i \le l)$가 최소 웹 서비스 집합이면서 q를 만족하는 시퀀스 C를 잉여 없는 컴포지션(Unredundant Composition, URC)이라고 한다.

3.2 시스템 구조

웹 서비스 컴포지션 및 검색을 수행하는 시스템의 구조는 그림 9와 같다. 서비스 제공자(Service Provider)는 UDDI레지스트리에 웹 서비스를 등록한다. 시스템은 레지스트리에서 웹 서비스 컴포지션에 필요한 각 연산과 그 입·출력과 같은 정보를 얻어 내부에 저장한다.

표 1 논문에서 사용한 기호들

기호	의미
q	사용자 질의
q_{in}, q_{out}	사용자 질의 q의 입력과 출력 파라미터들
ws	웹 서비스
ws_{in}, ws_{out}	웹 서비스 ws의 입력과 출력 파라미터들
WS	모든 웹 서비스를 포함하는 집합
W_i	병렬적으로 수행 가능한 웹 서비스들의 집합
C	웹 서비스 집합 W_i 이루어진 시퀀스
$In(W_i)$	집합 W_i에 포함된 웹 서비스들의 모든 입력 파라미터들 ($\bigcup_{ws \in W_i} ws_{in}$)
$Out(W_i)$	집합 W_i에 포함된 웹 서비스들의 모든 입력 파라미터들 ($\bigcup_{ws \in W_i} ws_{out}$)

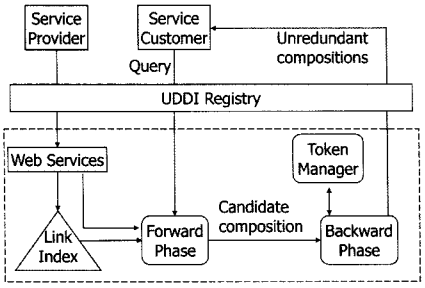


그림 9 시스템 구조

Key	Value
WS_1	WS_2, WS_5, WS_6, WS_7
WS_2	WS_5, WS_6
WS_3	WS_5, WS_6
WS_4	WS_5, WS_6
WS_7	WS_2, WS_6

그림 10 연결 인덱스 구축 예

그 후 저장된 웹 서비스 정보를 이용하여 연결 가능한 웹 서비스들을 인덱스에 삽입한다. 이 작업들은 사용자 질의가 오기 전에 수행된다.

서비스 사용자(Service Customer)가 웹 서비스의 입력과 출력을 명세한 질의(Query)를 요청하면, 전향 단계(Forward Phase)에서는 미리 생성한 연결 인덱스를 이용하여 후보 컴포지션을 만든다. 후향 단계(Backward Phase)에서는 토큰 관리자에 토큰에 대한 연산을 요청하면서 후보 컴포지션을 잉여 없는 컴포지션(Unredundant Composition)으로 분해한다. 잉여가 없는 컴포지션을 사용자에게 결과로서 돌려준다.

4. 2단계 웹 서비스 컴포지션 탐색 알고리즘

이 장에서는 연결 인덱스를 구축하는 방법과 2단계 웹 서비스 컴포지션 탐색 알고리즘을 설명한다.

4.1 연결 인덱스

웹 서비스의 수가 많을 것이라고 기대되므로, 전향 단계를 수행할 때 탐색 공간의 크기를 줄일 수 있는 자료구조(data structure)가 필요하다. 웹 서비스들 사이의 연결성(connectivity)를 바탕으로 연결 인덱스(Link Index)를 사용자 질의가 있기 전에 미리 구축한다.

웹 서비스들 사이의 연결성을 정형적으로 정리하면 정의 4와 같다.

정의 4. 연결 가능 웹 서비스

레지스트리에 존재하는 모든 웹 서비스를 WS라고 할 때, $\{(ws_i, ws_j) | ws_i, ws_j \in WS, (ws_{i,out} \cap ws_{j,in} \neq \emptyset)\}$를 만족하는 두 웹 서비스를 연결 가능하다고 한다.

연결 가능한 모든 웹 서비스 쌍을 찾아서 ws_i를 키(key)로 하고, ws_j를 값(value)으로 하는 해시 인덱스를 생성한다.

예제 4. 그림 10은 그림 4의 예제 웹 서비스를 대상으로 구축한 연결 인덱스를 나타낸 것이다. ws_1의 출력은 ISDN, Cost이고 ws_2의 입력은 CCN, Cost이므로 ws_1과 ws_2는 연결 가능하다. ws_1을 키로 해서 ws_2를 인덱스에 삽입한다.

4.2 전향 단계(Forward Phase)

사용자 질의 q를 받으면 입력 파라미터가 q_{in}의 부분집합이 되는 웹 서비스를 찾는다. 이 때 찾은 웹 서비스들의 출력 파라미터들은 다른 웹 서비스의 입력 파라미터로 사용될 수도 있다. 따라서 다음 단계에서는 q_{in} 뿐만 아니라 지금까지 찾은 웹 서비스들의 출력 파라미터들도 실행 가능한 웹 서비스들을 찾는 기준이 될 수 있다. 이 과정을 반복하다가 질의에 주어진 출력 파라미터를 생성하는 웹 서비스가 발견되면 전향 단계를 종료한다. 2단계 컴포지션 알고리즘의 전향 단계와 전향 체인의 다른 점은 전향 체인은 각 단계에서 찾은 웹 서비스들을 분해하여 모든 가능한 경우를 고려하면서 진행하지만 전향 단계에서는 분해를 하지 않는다는 것이다.

알고리즘 1은 전향 단계를 알고리즘으로 표현한 것으로 후보 컴포지션 $C=(W_1,\cdots,W_l)$을 구한다. 1-6은 초기화를 하는 과정이다. 우선 레지스트리에 등록된 모든 웹 서비스들을 검사해서 q_{in}에 속한 파라미터만을 입력으로 갖는 웹 서비스들을 찾아 W_1에 저장한다. 그리고 q_{in}과 W_1의 웹 서비스의 출력 파라미터들의 합집합을 P_2에 넣는다. 8-18은 $W_2,\cdots,W_l$을 구하는 과정으로 질의의 출력 파라미터를 찾을 때까지 P_i에 있는 파라미터들만을 입력 파라미터로 갖는 웹 서비스들을 찾는다. 9-10은 질의를 만족하는 컴포지션이 존재하지 않는 경우로, 더 이상 진행하지 않고 종료한다. 13-14에서는 4.1절에서 제시한 연결 인덱스를 사용해서 W_{i-1}에 속한 웹 서비스들과 연결된 웹 서비스들만을 후보로 골라낸다. 그 후 15-18에서 후보로 골라낸 웹 서비스들의 입력 파라미터가 P_i의 부분집합인지 봐서 실행 가능한지 확인한다. 실행이 가능한 웹 서비스를 W_i에 추가한다. 그리고 추가한 웹 서비스의 출력 파라미터를 P_{i+1}에 넣어 준다.

예제 5. 그림 11은 전향 단계를 수행하는 과정을 나타낸 것이다. 원은 W_i를, 사각형은 P_i를 나타낸다. P_1은 q_{in}이다. 우선 P_1을 입력으로 갖는 웹 서비스들을 찾아 W_1에 넣는다. 그 다음에 P_1의 파라미터들과 W_1에 있는 웹 서비스들의 출력 파라미터를 P_2에 넣는다. ws_1, ws_3, ws_4와 연결 가능하면서 P_2를 입력으로 갖는 웹 서비스

```
    Input : {q, WS}
    q : 사용자 질의, WS : 레지스트리에 등록된 웹 서비스들의 집합
    Output : {C,l,P}
    C : 웹 서비스 집합의 시퀀스, l: 후보 컴포지션의 길이
    P : 파라미터 집합의 시퀀스 (P1,...,Pl+1)
    procedure Forward(q,WS)
1   P1 ← qin; P2 ← P1;
2   W1←∅;
3   for each ws in WS do
4       if wsin ⊆ P1 then
5           W1 ← W1∪{ws};
6           P2 ← P2 ∪ wsout;
        end if
    end for
7   i←1;
8   while qout ⊈ Pi+1 do
9       if Wi = ∅ then
10          l←0; return;
        end if
11      i←i+1;
12      Wi←∅; Pi+1←Pi;
13      for each ws in Wi-1 do
14          CW ← CW ∪ LinkIndexLookup(ws)
        end for
15      for each ws in CW do
16        if wsin ⊆ Pi then
17            Wi← Wi ∪ {ws};
18            Pi+1 ← Pi+1 ∪ wsout;
          end if
        end for
    end while
19  l←i;
20  insert W1,...,Wl to sequence C;
    return;
```

알고리즘 1 전향 단계 알고리즘

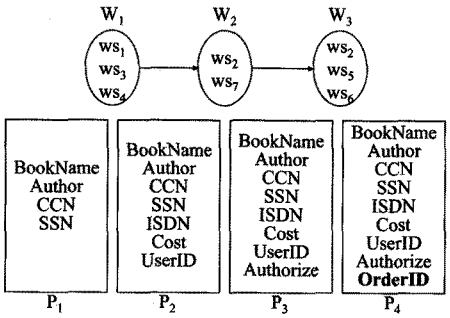


그림 11 전향 단계 수행 과정

들을 W_2에 넣는다. 이를 반복하다가 P_4에 q_{out}인 OrderID가 나오면 종료한다.

4.3 후향 단계(Backward Phase)

전향 단계는 레지스트리의 웹 서비스들로 이루어진 최대 길이가 1인 후보 컴포지션을 반환한다. 잉여 없는 컴포지션을 결과로 얻기 위해서는 이 후보 컴포지션을 분해하는 작업이 필요하다. 후향 단계는 전향 단계에서 구한 후보 컴포지션들로부터 잉여 없는 컴포지션을 추출하는 작업을 수행한다.

이 절에서는 후향 단계에서 사용하는 토큰 관리자와 알고리즘에 대하여 설명한다. 4.3.1절에서는 커버되지 않은 파라미터의 개념에 대하여 설명하고, 커버되지 않은 파라미터와 토큰을 이용한 알고리즘이 잉여 없는 컴포지션을 반환한다는 것을 보인다. 4.3.2절에서는 후향 단계 알고리즘의 절차에 대하여 설명한다.

4.3.1 토큰 관리자(Token Manager)

전향 단계의 결과로 나온 후보 컴포지션은 각 단계에서 실행이 가능한 웹 서비스들만을 포함하고 있으므로, 이 후보 컴포지션을 잉여 없는 컴포지션으로 만들어야 한다. 일단 후보 컴포지션을 구한 후 컴포지션 내의 모든 웹 서비스들을 하나씩 제거해 보면서 사용자의 질의를 만족하는지, 만족하지 않는지를 검사하면 잉여 없는 컴포지션(URC)인지 알 수 있다. 그러나 웹 서비스의 개수가 증가할수록 이런 방식은 시간이 걸릴 것이다. 따라서 후향 단계에서 효율적으로 잉여 웹서비스의 발생 여부를 판단해야 하는데, 이를 위하여 토큰 매니저에서는 토큰과 커버되지 않은 파라미터의 개념을 사용한다.

커버되지 않은 파라미터는 정의 5와 정의 6을 통하여 설명할 수 있다.

정의 5. 커버(cover)

웹 서비스를 ws, 파라미터를 p라고 할 때, $p \in ws_{out}$이면 ws는 p를 커버한다고 한다.

정의 6. 커버되지 않은 파라미터(uncovered parameters)

질의 q를 만족하는 시퀀스 $C=(W_1,\cdots,W_l)$,$W_i(1 \le i \le l)$는 웹 서비스의 집합이라고 하자. 시퀀스 $C'=(W_{i+1},\cdots,W_l)$라고 하자.

$$\{\bigcup_{j=i+2}^{l+1}(In(W_j)-\bigcup_{k=i+1}^{j-1}Out(W_k))\cup In(W_{i+1})\}-q_{in}$$

을 C'의 커버되지 않은 파라미터라고 한다.

예제 6. 그림 7(a)에 있는 시퀀스 $C=(\{ws_1, ws_3\}, \{ws_2\}, \{ws_5\})$에서 $C'=(\{ws_2\}, \{ws_5\})$라 하자. q_{out}은 OrderID이고, ws_5의 입력은 {Authorize, ISDN, MemberID}, ws_2의 입력은 {CCN, Cost}이다. 이 중 OrderID는 ws_5에 의해, Authorize는 ws_2에 의해 커버되고, CCN은 q에 의해 커버된다. 따라서 C'의 커버되지 않은 파라미터는 ISDN, MemberID, Cost이다. □

정의 6에 기반을 하여, 아래와 같이 정리 1을 기술할 수 있다.

정리 1. 질의 q를 만족하는 시퀀스 $C=(W_1,\cdots,W_l)$, W_i $(1 \le i \le l)$는 웹 서비스의 집합이라고 하자. 시퀀스 $C'=(W_{i+1},\cdots,W_l)$의 커버되지 않은 파라미터를 UP라고 하자.

모든 ws∈W_i에 대해 $UP \not\subseteq \left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - \{ws\})\right)$이면 W_i는 최소 웹 서비스 집합이다.

증명.

$$UP \not\subseteq \left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - \{ws\})\right)$$

$$\left(\left(\bigcup_{i=i+2}^{l+1} \left(In(W_i) - \bigcup_{k=i+1}^{l-1} Out(W_k)\right) \cup In(W_{i+1})\right) - q_{in}\right)$$

$$\not\subseteq \left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - \{ws\})\right)$$

그러면

$$\left(\left(In(W_x) - \bigcup_{x=i+1}^{x-1} Out(W_x)\right) - q_{in}\right) \not\subseteq \left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - \{ws\})\right)$$

인 x(i+2≤x≤l+1)가 존재하거나(경우 1),

$In(W_{i+1}) - q_{in} \not\subseteq \left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - ws)\right)$이다(경우 2).

i) 경우 1,

$$In(W_x) \not\subseteq \left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - \{ws\}) \cup q_{in} \cup \left(\bigcup_{k=i+1}^{x-1} Out(W_k)\right)\right)$$

$$\not\subseteq \left(\bigcup_{i=1}^{x-1} Out(W_i) - Out(W_i)\right) \cup Out(W_i - \{ws\}) \cup q_{in}\Big)$$

$(W_1,\cdots W_i \ \{ws\},\cdots,W_l)$을 $(W^1,\cdots,W^i,\cdots,W^l)$로 치환한다.

$$In(W_x) \not\subseteq \left(\bigcup_{i=1}^{x-1} Out(W^i) \cup q_{in}\right)$$

ii) 경우 2,

$$In(W_{i+1}) \not\subseteq \left(\left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - ws)\right) \cup q_{in}\right)$$

$$In(W_{i+1}) \not\subseteq \left(\bigcup_{i=1}^{x-1} Out(W^i) \cup q_{in}\right)$$

두 경우 모두 정의 1의 조건 ii에 위배되므로 $(W_1,\cdots W_i \{ws\},\cdots,W_l)$은 q를 만족하지 않는다. 따라서 정의 2에 의해 $UP \not\subseteq \left(\bigcup_{i=1}^{i-1} Out(W_i) \cup Out(W_i - \{ws\})\right)$인 W_i는 최소 웹 서비스 집합이다. □

정리 1이 의미하는 바는 다음과 같다. W_i가 최소 웹 서비스 집합이라면 시퀀스 C가 질의 q를 만족하기 위해서는 모든 ws∈W_i가 UP의 파라미터 중 일부를 커버해야 한다. 그리고 ws가 커버한 파라미터 중 적어도 하나는 시퀀스$(W_1,\cdots,W_i-\{ws\})$에 있는 웹 서비스에 의해 커버되지 않아야 한다. 즉 모든 웹 서비스가 각각 UP의 어떤 파라미터를 커버하는 유일한 웹 서비스라는 것이다. 따라서 각 웹 서비스마다 유일하게 커버하는 파라미터들의 리스트를 기록해 두고 모든 웹 서비스들이 하나 이상의 파라미터를 유일하게 커버하고 있는지를 검사한다면 URC인지 여부를 쉽게 판단할 수 있을 것이다.

토큰 관리자는 토큰을 통해 웹 서비스가 유일하게 커버하는 파라미터에 대한 정보를 제공한다. 토큰 관리자가 제공하는 연산은 다음과 같다.

i. p에 대한 토큰 생성(create a token t_p)

파라미터 p에 대한 토큰 t_p를 생성한다. p에 대한 토큰은 하나만 있어야 하므로 만약 t_p가 이미 존재한다면 생성하지 않는다. UP에 속한 파라미터 외의 파라미터를 유일하게 출력한다는 것은 아무런 의미가 없기 때문에 토큰을 미리 생성해 두고, 생성된 토큰만 할당하도록 한다.

ii. p에 대한 토큰 제거(remove a token t_p)

t_p를 찾아서 제거한다.

iii. ws에게 p에 대한 토큰 할당(ws obtains a token t_p.)

웹 서비스 ws에게 t_p를 할당한다. 만약 토큰 t_p를 다른 웹 서비스 ws_r이 갖고 있다면 ws_r로부터 t_p를 빼앗아서 ws에게 준다. ws_r이 토큰 t_p만 갖고 있는 경우, ws를 컴포지션에 삽입하면 ws_r이 잉여 웹 서비스가 되므로 토큰을 빼앗을 수 없다. 토큰 t_p가 존재하지 않거나, ws에게 토큰 t_p를 할당했다면 잉여가 없는 것이므로 성공을 리턴하고, ws가 토큰을 빼앗지 못했다면 잉여가 있는 것이므로 실패를 리턴한다.

주목할 것은 토큰을 얻는데 실패한 ws가 아닌 ws_r이 잉여 웹 서비스가 된다는 것이다. 후향 단계에서 후보 컴포지션을 분해할 때는 W_l부터 역순으로 진행하지만 컴포지션을 실행시킬 때는 W_1에 있는 웹 서비스들부터 실행한다. 따라서 후향 단계 알고리즘에서 ws_r를 먼저 결과에 삽입하더라도 실행할 때는 ws를 먼저 실행하므로 ws_r이 잉여 웹서비스가 된다.

iv. p에 대한 토큰 반환(ws restores a token t_p)

현재 t_p를 갖고 있는 웹서비스 ws가 토큰을 반환한다. 그러면 토큰 관리자는 ws 바로 전에 t_p를 갖고 있던 웹 서비스에게 토큰을 돌려준다.

토큰 관리자는 토큰 테이블(Token table)에 각 웹 서비스들이 할당 받은 토큰에 대한 정보를 유지하고, 자유 토큰(Free tokens)에는 생성은 되었으나 아직 웹 서비스에 할당 되지 않은 토큰들에 대한 리스트를 저장한다.

토큰 관리자가 후향 단계에서 어떻게 사용되는지 간략히 설명하면 다음과 같다. 후향 단계에서는 후보 컴포지션 $C=(W_1,\cdots,W_l)$을 W_l부터 분해하면서 URC를 찾는다. $(W_{i+1},\cdots,W_l)$를 분해한 것을 $C'=(W'_{i+1},\cdots,W'_l)$라 하자. W'_i를 구하기 전에 우선 C'의 커버되지 않은 파라미터에 대한 토큰을 생성한다. 그리고 웹 서비스 ws를 W'_i에 넣을 때 ws의 출력 파라미터에 대한 토큰을 ws에게 준다. 만약 ws가 삽입되어 잉여 웹서비스가 생긴다면, ws가 토큰을 받으려 할 때 가지고 있던 모든 토큰을 빼앗기는 웹 서비스가 생긴다. 따라서 모든 토큰을 빼앗기는 웹 서비스가 있는지만 확인하면 잉여 웹 서비스의 존재 여부를 확인할 수 있다.

예제 7. 그림 12는 웹 서비스가 삽입 되었을 때 잉여 웹 서비스가 생기는 경우를 나타낸 것이다. $W'_2=\{ws_2, ws_7\}$, $W'_3=\{ws_6\}$일 때 토큰 할당 상황은 그림 11(a)와

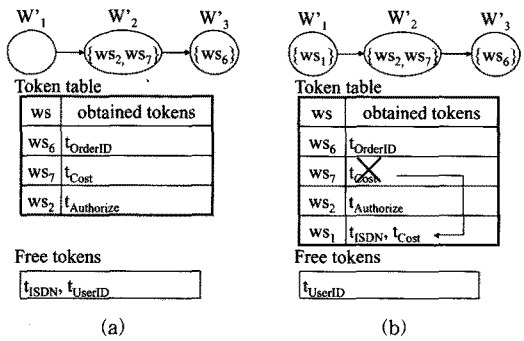


그림 12 잉여 웹 서비스가 생기는 예

같다. 이 때 ws_1을 W'_1에 삽입해 보려 한다. ws_1이 tCost를 얻으면 그림 11(b)에서처럼 ws_7이 아무런 토큰을 갖지 않게 되므로 ws_7은 잉여 웹 서비스가 된다.

4.3.2 후향 단계 알고리즘

후향 단계는 전향 단계에서 구한 후보 컴포지션 $C=(W_1,\cdots,W_l)$을 분해하여 잉여 없는 컴포지션 $C'=(W'_1,\cdots,W'_l)$를 구하는 단계로 전향 단계와는 반대 방향으로 진행된다. W_l에 속한 웹 서비스들 중에서 q_{out}을 출력하는 웹 서비스를 선택하고, 그들의 부분 집합을 구해서 W'_l을 구한다. 이 때 토큰 매니저를 사용하여 최소 웹 서비스 집합만을 W'_l로 한다. 또한 전향 단계에서 구한 P_l을 이용하여 P_l과 W'_l의 출력 파라미터가 qout을 만족하는지 확인한다. 그 후 W'_l에 속한 웹 서비스들의 입력 파라미터와 W'_l에 의해 커버 되지 않은 q_{out}의 파라미터들을 새로운 질의의 출력으로 하여 W'_{l-1}을 구한다. 이를 반복하여 W'_1까지 구하면 잉여 없는 컴포지션(URC)를 구할 수 있다.

알고리즘 2는 후향 단계 알고리즘을 표현한 것이다. 전향 단계의 후보 컴포지션이 후향 단계로 넘어오면 q_{out}에 대한 토큰을 생성한 후 Backward(l,C,C',P,q_{out},q)를 호출한다. l,C,P는 전향 단계로부터 얻는다. 아직 아무것도 분해하지 않았으므로 C'는 비어 있다. 처음에 커버되지 않은 파라미터는 q_{out}뿐이다. 1-2는 재귀 함수의 종료 조건으로, 만약 i가 0이라면 URC를 찾은 것이므로, 결과를 출력하고 리턴한다. 3에서는 C의 W_i에 있는 웹 서비스들 중에서 UP에 속한 파라미터를 출력 파라미터로 갖는 서비스를 찾는다. 4에서는 3에서 찾은 서비스의 멱집합을 구한다. 이 때 모든 부분집합을 찾는 것이 아니라 토큰관리자를 통해 잉여를 발생시키지 않는 부분 집합만을 찾는다. 부분 집합을 찾는 방법은 알고리즘 3에서 설명한다. 6-13은 4에서 구한 각 부분집합을 각각 W'_i에 넣어보면서 깊이 우선 탐색을 하는 과정이다. 4에서 잉여 웹 서비스가 생기지 않은 웹 서비스 집합을 얻기 때문에 6에서 토큰을 얻어을 때는 무조건 성공하게 된다. 7-9는 다음 재귀 호출을 위한 인자들을 준비하는 단계이다. 7은 시퀀스 $(S,W'_{i+1},\cdots,W'_l)$의 커버되지 않은 파라미터를 계산하는 부분이다. 9에서는 S에 속한 웹 서비스들의 입력 파라미터에 대한 토큰을 생성한다. 10에서 W'_{i-1}을 구하기 위한 재귀호출을 한다. 10이 종료되면 $(S,W'_{i+1},\cdots,W'_l)$가 포함된 시퀀스는 모두 살펴본 것이다. W'_i에 S를 넣었을 때 변경한 것은 다른 부분 집합을 넣어 볼 때는 반영되지 않아야 하므로 11-13에서 이전 상태로 돌아가기 위한 작업을 한다.

```
     Input : (i,C,C',P,UP,q)
     i:진행 단계, C:후보 컴포지션, C':분해된 컴포지션
     P : 파라미터 집합의 시퀀스(P1,...,Pl-1)
     UP : 커버되지 않은 파라미터, q : 사용자 질의
     procedure Backward(i,C,C',P,UP,q)
1    if i=0 then
2        print C'; return;
     end if
3    CW ← {ws|(ws∈Wi, Wi∈C)∧(wsout ∩ UP ≠ ∅)};
4    PS ← getSubset(CW, UP, Pi,∅);
5    for each set S in PS do
6        all ws in S obtain tokens for their output parameters.
7        UP'←(⋃ws∈S wsin ∪ (UP − ⋃ws∈S wsout)) − qin ;
8        insert S as W'i into sequence C';
9        createToken(S);
10       Backward(i-1,C,C',P,UP',q);
11       restore tokens for all ws in S;
12       remove newly created tokens at line 10;
13       delete W'i from C';
     end for

     procedure createToken(S)
14   for each ws in S do
15       for each p in wsin do
16           if token for p does not exist then
17               create a token for p;
             end if
         end for
     end for
```

알고리즘 2 후향 단계 알고리즘

예제 8. 그림 13은 후향 단계에서 그림 11의 후보 컴포지션을 분해해서 잉여 없는 컴포지션(URC)로 나누는 과정을 보여주고 있다. 그림의 왼쪽은 전향 단계에서 구한 후보 컴포지션을 표시한다. 오른쪽 탐색 트리에서 노드의 아래에는 커버되지 않은 파라미터(UP)를 표시하였다.

그림 13의 오른쪽 있는 탐색 트리에서 점선 부분을 탐색하는 과정을 예로 설명하면 다음과 같다. 처음에 토큰 $t_{OrderID}$를 만들고 후향 단계를 시작한다. 후보 컴포지

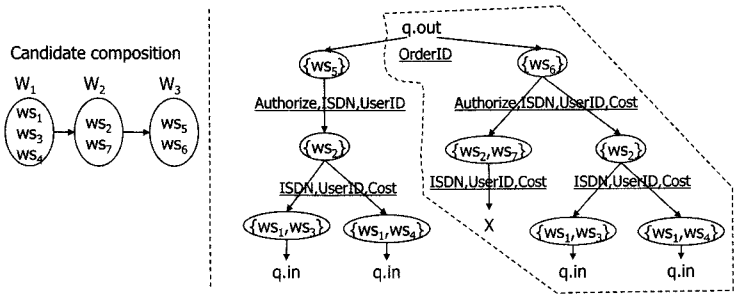


그림 13 후향 탐색 경로

션의 W_3에 있는 웹 서비스 중에서 OrderID를 출력하는 웹 서비스 ws_5와 ws_6를 찾는다. {ws_5,ws_6}의 잉여가 발생하지 않는 부분집합 {ws_5}, {ws_6}을 W'_3에 넣어본다. 여기서는 {ws_6}을 넣은 경우만 예로 든다. {ws_6}을 W'_3에 넣은 후, ws_6에게 토큰 $t_{OrderID}$를 할당하고 토큰 $t_{Authorize}$, t_{ISDN}, t_{UserID}, t_{Cost}를 생성한다. 후보 컴포지션의 W_2에서 시퀀스(W'_3)의 UP를 출력하는 웹 서비스 ws_2와 ws_7을 찾고, 잉여가 발생하지 않는 부분집합 {ws_2,ws_7}, {ws_2}을 구한다. 예제 4에서처럼 {ws_2,ws_7}를 W'_2에 넣은 후 ws_1을 포함한 부분집합을 W'_1에 넣으면 잉여가 발생한다. 또한 ws_1이 포함되지 않은 부분집합은 질의를 만족하지 않으므로 더 이상 진행할 수 없다. W'_2에서 {ws_2,ws_7}을 제거하고 {ws_2}를 넣는다. {ws_2}의 출력에 대한 토큰을 얻고, 입력에 대한 토큰을 생성한다. 후보 컴포지션의 W_1에서 시퀀스 (W'_2,W'_3)의 UP를 출력하는 웹 서비스 ws_1,ws_2,ws_3을 찾는다. W_1의 웹 서비스들은 모두 q_{in}만을 입력으로 가진다. 그러므로 {ws_1,ws_3,ws_4}의 잉여를 발생시키지 않는 부분집합을 구하면 컴포지션을 찾을 수 있다. □

부분집합을 찾을 때도 트리를 탐색한다. 트리를 생성하는 방법은 다음과 같다. 우선 CW에 속한 웹 서비스 각각을 노드로 하여 최상위 레벨을 구성한다. 각 노드는 자신의 왼쪽 형제 노드들과 조상 노드에 없는 웹 서비스를 자식으로 갖는다. 트리의 가장 위 노드는 웹 서비스가 각각 하나씩 존재하는 부분집합을 의미한다. 레벨 2에 위치한 노드들은 자신과 부모노드를 합해서 2개의 웹 서비스로 이루어진 부분집합을 의미한다. 이렇게 트리를 만들면 가능한 모든 부분집합을 확인할 수 있다.

알고리즘 3은 부분집합을 구하기 위한 알고리즘이다. 3번째 줄에서는 잉여 웹 서비스가 생기는지 검사한다. $(ws_{out} \cap UP) \subseteq \bigcup_{ws_j \in S} ws_{j,out}$이면 ws는 잉여 웹 서비스가 된다. 왜냐하면 ws의 출력이 같은 부분집합에 있는 다른 웹 서비스들의 출력에 포함되는 것이기 때문이다. ws의 출력에 대한 토큰을 얻는 연산이 성공한다는 것은 ws를 컴포지션에 포함시켜도 잉여 웹 서비스가 발생하지 않는다는 것을 의미한다. 6에서는 이 부분 집합이 질의를 만족할 수 있을지 검사하는 것이다. 부분 집합의 출력들에 의해 커버되지 않더라도 P_i에 있다면 나중에 다른 웹 서비스에 의해 커버될 수도 있다. 하지만 부분집합의 출력과 P_i로도 UP가 커버되지 않으면 이 부분집합은 질의를 만족할 수 없다. 8은 재귀 호출을 통해 자식 노드를 방문하는 부분이다.

```
    Input : {CW, UP, Pi, S}
    CW : 전체집합, UP : 커버되지 않은 파라미터
    Pi : 파라미터의 집합 S : 이전에 선택된 웹 서비스의 집합
    Output : {R}
    R : 잉여를 발생시키지 않는 CW의 부분집합
    procedure getSubset(CW,UP,Pi,S)
1   R←∅;
2   for each ws in CW do
3       if (wsout ∩ UC) ⊈ ⋃ wsout AND ws succeeds in
        obtaining tokens for wsout then
4           S' ← {ws}∪S;
5           CW ← CW-{ws};
6           if UP ⊆ ⋃ wsj,out ∪ Pi then
7               R ← R ∪ {S'};
            end if
8           R←R∪getSubset(CW, UP, Pi, S');
9           restore tokens owned by ws;
        end if
    end for
10  return R;
```

알고리즘 3 부분 집합 알고리즘

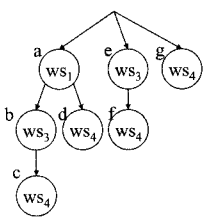


그림 14 부분 집합 생성을 위한 트리

예제 9. W'_2={ws_2}, W'_3={ws_5}이라면 커버되지 않은 파라미터(UP)는 ISDN, UserID, Cost이다. W_1에서 UP를 출력하는 웹 서비스들을 찾으면 ws_1, ws_3, ws_4가 나온다. 그림 14는 ws_1, ws_3, ws_4의 부분집합을 구하기 위한 트리이다. 노드 위의 알파벳은 노드를 식별하기 위해 적은 것이다. 우선 노드a를 본다. ws_2와 ws_5는 ws_1의 출력인 ISDN과 Cost를 출력하지 않으므로 ws_1은 토큰 t_{ISDN}과 t_{Cost}를 얻는데 성공한다. 그 후 ws_1이 질의를 만족할 수 있는지 확인한다. P_1과 ws_1의 출력을 합해도 UserID가 커버되지 않는다. 따라서 ws_1은 질의를 만족할 수 없다. 노드b를 방문한다. UserID를 출력하는 웹 서비스가 없으므로 ws_3은 토큰 t_{UserID}를 얻는데 성공한다. {ws_1, ws_3}은 UP를 커버하므로 {ws_1, ws_3}을 결과에 넣는다. 그 후 노드c를 본다. ws_4의 출력은 ws_3의 출력과 같다. 따라서 ws_4 또는 ws_3이 잉여 웹 서비스가 된다. 말단 노드까지 탐색을 하였으므로 노드c와 노드b에서 얻은 토큰은 반환하고 노드d를 방문한다. 노드 g까지 방문을 마치면 {ws_1, ws_3}과 {ws_1, ws_4}를 결과로 얻을 수 있다. □

5. 실험 결과

5.1 실험 환경

모든 실험은 펜티엄4 2.40GHz와 2GB의 메인 메모리가 장착되고 CentOS release 5 운영체제가 설치된 시스템에서 이루어졌다. 본 논문에서 제안한 알고리즘과 성능 비교를 위한 기존 알고리즘 모두 C++로 구현하였다. 연결 인덱스는 STL의 map 컨테이너를 이용하였다.

5.2 실험 데이타 및 알고리즘

웹 서비스 컴포지션 시스템의 성능 평가를 위한 표준 데이타 세트가 아직 없기 때문에 기존의 연구들은 [11,16] 가상 데이타를 생성하여 성능을 평가하였다. 본 논문에서도 가상적으로 생성한 데이타셋을 사용하여 성능을 평가하였다. 생성한 데이타셋의 특성은 표 2와 같다. 입력 또는 출력으로 1~4, 5~8, 9~12개의 파라미터를 갖는 웹 서비스들을 100,000 개씩 생성하였다. 이 웹 서비스들의 입력과 출력 파라미터는 미리 생성한 100,000개의 파라미터 도메인에서 균등 분포(uniform distribution)가 되도록 랜덤하게 뽑았다. 각 데이타셋에 대한 사용자 질의를 1000개씩 생성하였다. 이 질의들은 길이가 3인 컴포지션을 결과로 갖는다.

표 2 실험 데이타 세트

	D_1	D_2	D_3
웹 서비스 수	100,000		
파라미터 도메인 크기	100,000		
입·출력 파라미터 수	1~4	5~8	9~12

실험의 대상이 된 알고리즘은 너비 우선 탐색을 이용한 후향 체인 방식[10](BC)과 깊이 우선 탐색을 이용한 전향 체인 방식[8](FC), 기존의 전후향 방식[12](2P), 그리고 본 논문에서 제안하는 알고리즘(2PUR)이다.

5.3 후향 체인 방식과의 비교

그림 15는 데이타셋 D1을 사용하여 2PUR과 BC의 실행 시간을 비교한 것이다. 생성한 100,000개의 웹 서비스를 20,000개씩 삽입하면서 웹 서비스의 수의 증가에 따른 실행 시간 변화를 측정하였다. BC과 2PUR의 실행시간 차이가 많이 나서 y축은 로그 눈금으로 나타내었다.

그림 15에서 볼 수 있듯이 BC를 사용했을 경우 웹 서비스의 수가 많아짐에 따라 성능이 급격히 저하되었다. 파라미터 도메인의 크기를 고정시킨 상태에서 웹 서비스의 수를 늘렸기 때문에, 웹 서비스의 수가 늘어날수록 같은 파라미터를 출력하는 웹 서비스가 많아지게 된다. 예를 들어 사용자가 질의의 출력으로 지정한 파라미터가 q_{out1}, q_{out2}, …, q_{outn}이라고 하자. q_{outi}를 출력하는 웹 서비스의 개수를 $n(q_{outi})$라고 하면 최대 $n(q_{out1})\times\cdots\times n(q_{outn})$ 개의 노드가 탐색의 첫 단계에서 고려될 것이다. 따라서 한 파라미터를 출력하는 웹 서비스가 많아질수록 고려해야 할 노드의 수가 증가하게 되므로 실행 시간도 증가하게 된다.

D_1에서 웹 서비스의 수가 100,000개인 경우와 D_2, D_3의 경우 같은 출력을 갖는 웹 서비스의 수가 너무 많아져서 실험을 진행할 수 없었다.

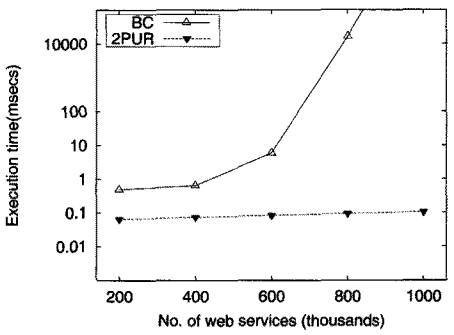
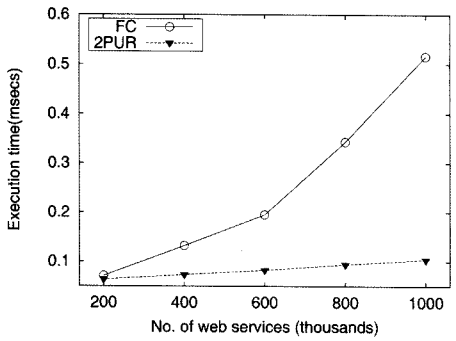


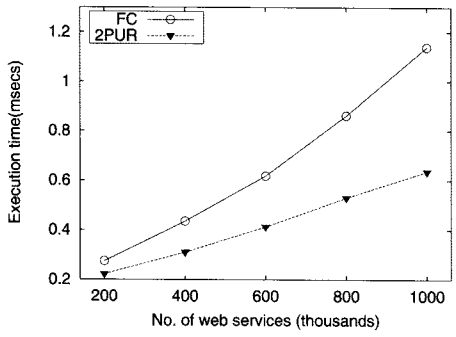
그림 15 2PUR과 BC의 실행 시간 비교

5.4 전향 체인 방식과의 비교

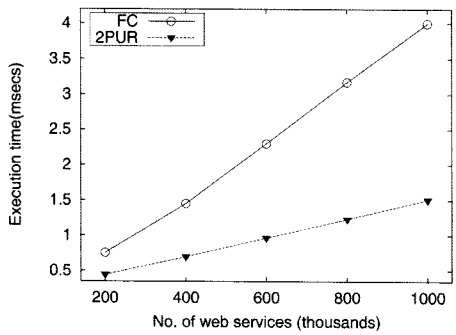
그림 16은 데이타셋 D_1, D_2, D_3을 사용하여 2PUR과 FC의 실행 시간을 비교한 것이다. 마찬가지로 생성된 100,000개의 웹 서비스를 20,000개씩 삽입하면서 웹 서비스의 수의 증가에 따른 실행 시간 변화를 측정하였다. FC는 깊이 우선 탐색을 하기 때문에 길이가 무척 긴 컴포지션을 찾을 수도 있다. 2P, 2PUR, BC는 길이가



(a) 데이타셋 D_1



(b) 데이타셋 D_2



(c) 데이타셋 D_3

그림 16 2UR과 FC의 실행 시간 비교

가장 짧은 컴포지션만을 구하므로 공정한 비교를 위해 임계값(threshold)을 두어 길이가 4 이상이 되는 경로는 찾지 않게 하였다.

FC는 q_{in}으로 실행시킬 수 있는 웹 서비스들을 찾아서 그들의 모든 부분집합을 노드로 만든다. 크기가 n인 집합의 멱집합의 수는 2^n이므로 전체 웹 서비스의 수가 증가할수록 실행시킬 수 있는 웹 서비스의 수도 증가할 것이다. 따라서 웹 서비스 수가 증가함에 따라 실행 시간도 크게 증가한다. 반면에 2PUR은 q_{in}으로 실행시킬 수 있는 웹 서비스들을 찾아 모든 부분집합을 노드로 만드는 것이 아니라 일단 모든 웹 서비스들을 찾아 둔 다음에, q_{out}부터 시작하여 반대방향으로 진행하면서 분해를 하기 때문에 FC보다 빠르다.

5.5 전후향 방식과의 비교

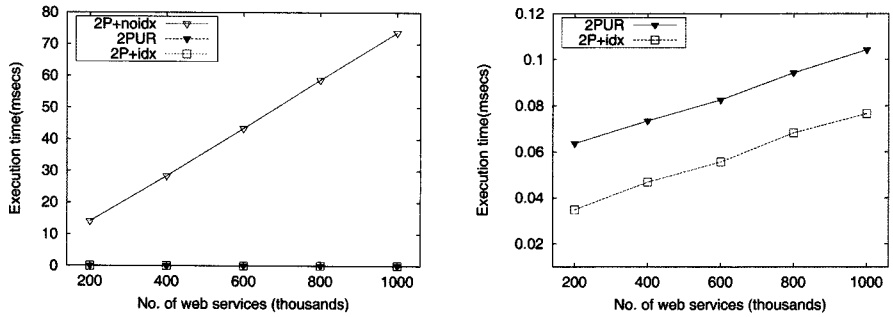
그림 17은 데이타셋 D_1, D_2, D_3을 사용하여 2PUR과 2P의 실행 시간을 비교한 것이다. [12]에서는 인덱스의 사용에 대해서 언급하지 않았기 때문에, 본 논문에서는 연결 인덱스를 사용한 경우(2P+idx)와 사용하지 않은 경우(2P+noidx) 모두를 실험 해 보았다. 2PUR는 2P+idx보다는 약간 느렸지만 큰 차이는 나지 않았고, 2P+noidx보다는 훨씬 더 빨랐다. 2P+noidx의 실행 시간이 다른 두 방법의 실행보다 현저하게 느리기 때문에, 정확한 비교를 하기 위해서 2PUR과 2P+nodix, 2P+idx를 따로 비교하여 그래프를 그렸다. 그림 17의 왼쪽은 2PUR과 2P+noidx, 2P+idx의 실행 시간을 비교한 그래프이고, 오른쪽은 2PUR과 2P+idx의 실행 시간만을 비교한 그래프이다.

2P+nodix는 전향 단계에서 다음 단계에 실행 시킬 수 있는 웹 서비스들을 찾을 때 레지스트리에 등록된 모든 웹 서비스들을 검사하므로 웹 서비스의 수가 늘어남에 따라 그 실행 시간이 급격히 증가한다. 하지만 인덱스를 사용하면 연결이 가능한 웹 서비스들만을 빠르게 찾을 수 있으므로 2P+idx의 실행 시간의 줄어드는 것이다.

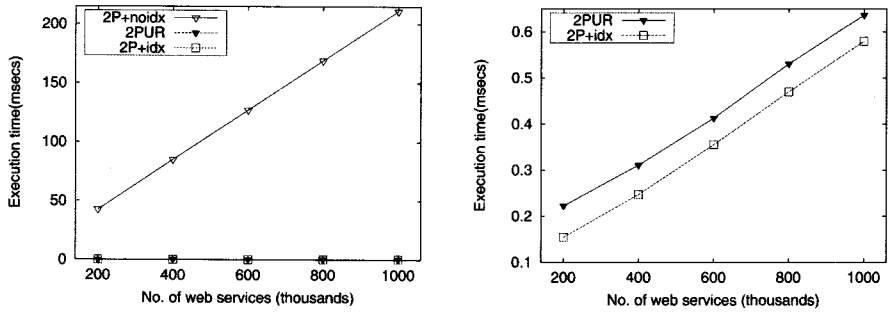
그림 17에서 웹 서비스의 입력 출력 파라미터 수가 많을수록 2P+idx와 2PUR의 실행시간 차이가 적어지는 것을 관찰할 수 있다. 모든 입력 파라미터가 만족되어야 웹 서비스가 실행될 수 있기 때문에 필요한 입력 파라미터가 많으면 실행될 수 있는 웹 서비스의 수가 줄어들게 된다. 전향 단계에서 찾은 웹 서비스의 수가 줄어들면 후향 단계에서 분해할 때 고려할 웹 서비스의 수도 줄어들게 된다. 따라서 입력 파라미터의 수가 많을수록 2PUR의 실행 시간은 2P+idx와 별 차이가 나지 않게 된다.

표 3은 2PUR가 분해한 컴포지션의 개수를 나타낸 것이다. 2P는 항상 1개의 결과를 돌려주지만, 2PUR는 웹 서비스가 100,000개일 때 평균 1.621개, 최대 6개의 컴포지션으로 분해한 결과를 돌려주었다. 2P보다 더 많은 결과를 돌려주므로 사용자에게 다양한 선택의 기회를 제공할 수 있다.

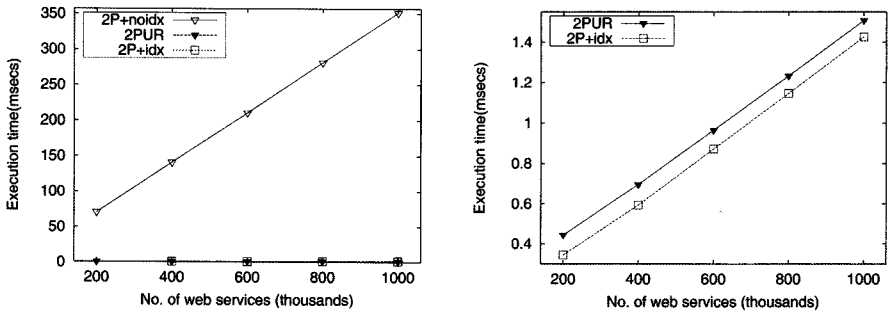
표 4는 하나의 컴포지션 결과에 포함된 웹 서비스의 수를 나타낸 것이다. 데이타셋 D_1에서는 차이가 없었으나, 데이타셋 D_2와 D_3의 경우에는 2P가 구한 컴포지션에 더 많은 웹 서비스들이 포함되어 있었다. 2PUR은 컴포지션을 실행하는데 없어도 무방한 웹 서비스들을 제거하지만, 2P는 단순히 질의의 출력이나 다음 단계에



(a) 데이타셋 D_1



(b) 데이타셋 D_2



(c) 데이타셋 D_3

그림 17 2PUR과 2P+noidex(왼쪽), 2P+idx(오른쪽)의 실행 시간 비교

표 3 2PUR을 사용했을 때 나온 URC의 개수

웹 서비스 수	D_1		D_2		D_3	
	평균	최대	평균	최대	평균	최대
20000	1	1	1.015	2	1.621	6
40000	1	1	1.015	2	1.621	6
60000	1	1	1.015	2	1.621	6
80000	1	1	1.015	2	1.621	6
100000	1	1	1.015	2	1.621	6

필요한 파라미터를 출력하지 않는 웹 서비스를 제거하는 작업만 하기 때문에 필요 없는 웹 서비스들이 남아 있을 수 있다.

표 4 컴포지션에 포함된 평균 웹 서비스의 수

웹 서비스 수	D1		D2		D3	
	2PUR	2P	2PUR	2P	2PUR	2P
20000	3	3	3	3.026	3.20815	4.855
40000	3	3	3	3.026	3.20815	4.855
60000	3	3	3	3.026	3.20815	4.855
80000	3	3	3	3.026	3.20815	4.855
100000	3	3	3	3.026	3.20815	4.855

6. 결론 및 향후연구

본 논문에서는 전향과 후향의 2단계를 거쳐서 잉여가 없는 웹 서비스 컴포지션을 검색하는 방법을 제안하였

다. 전향 단계에서는 좀 더 빠르게 후보 컴포지션을 찾기 위하여 연결 가능한 웹 서비스들을 미리 인덱스로 구축 해 두었다. 후향 단계에서는 토큰을 사용하여 후보 컴포지션을 잉여 웹 서비스가 포함되지 않은 컴포지션으로 분해하였다. 실험을 통해 2단계 웹 서비스 컴포지션이 기존의 한 방향으로 진행하는 방식보다 시간 면에서 더 효율적이라는 것을 보였다. 또한 전후향 방식보다 더 많은 컴포지션 결과를 사용자에게 제공하면서도 시간 면에서는 기존의 방식에 필적한다는 것을 보였다.

현재 웹 서비스 컴포지션에 대한 연구가 많이 이루어지고 있으나 웹 서비스 컴포지션 시스템을 평가하기 위한 표준화된 벤치마크는 아직 나오지 않았다. 또한 웹 서비스를 위한 표준화된 온톨로지가 없기 때문에 서비스 제공자들은 각자의 온톨로지를 사용하여 웹 서비스를 기술하였다. 따라서 현재 제공 되고 있는 웹 서비스들을 수집하여 웹 서비스 컴포지션 시스템의 성능을 평가하는 것은 무의미할 것이다. 때문에 본 논문에서도 임의로 생성한 웹 서비스들을 사용하여 실험한 결과를 제시하였다. 향후 웹 서비스와 웹 서비스 컴포지션에 대한 표준화가 좀 더 진행되고 신뢰할 수 있는 벤치마크가 나온다면 본 논문에서 제안한 방식과 기존 방식의 성능을 비교할 계획이다.

참 고 문 헌

[1] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard, "Web services architecture," February 2004. http://www.w3.org/ TR/ws-arch/

[2] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1," June 2007. http://www.w3.org/ TR/wsdl20.

[3] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers, "UDDI Version 3.0.2," October 2004. http://www. uddi.org/pubs/uddi_v3.htm

[4] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, "Simple Object Access Protocol (SOAP) 1.1," May 2000. http://www.w3c.org/TR/2000/ NOTE-SOAP-20000508.

[5] Shalil Majithia, David W. Walker and W. A. Gray, "A framework for automated service composition in service-oriented architecture," At 1st European Semantic Web Symposium (ESWS 2004), pages 269-283, May 2004.

[6] Benatallah, B., Sheng, Q.Z., and Dumas, M, "The self-serv environment for web services composition," IEEE Internet Computing, Vol.7, No.1, pages 40-48, Jan/Feb, 2003.

[7] Liming Chen, Nigel Shadbolt, Carole A. Goble, Feng Tao, Simon J. Cox, Colin Puleston and Paul R. Smart, "Towards a knowledge-based approach to semantic service composition," In Proceedings of International Semantic Web Conference 2003, pages 319-334, Sanibel Island, Florida, USA, 2003.

[8] Hafedh Mili, Guy Tremblay, Anne-Elisabeth Caillot, Radouane Ben Tamrout, and Abdel Obaid, "Web service composition as a function cover problem," In Proceedings of The Montreal Conference on eTechnologies 2005(MCeTech 2005), pages 61-71, Montreal, Canada, 2005.

[9] on Constantinescu, and Boi Faltings, "Large scale, type-compatible service composition," In Proceedings of the IEEE International Conference on Web Services(ICWS '04), pages 506-513, San Diego, California, USA, 2004.

[10] Lerina Aversano, Gerardo Canfora, and Anna Ciampi, "An algorithm for web service discovery through Their composition," In Proceedings of the IEEE International Conference on Web Services (ICWS '04), pages 332-339, San Diego, California, USA, 2004.

[11] Li Kuang, Ying Li, Jian Wu, ShuiGuang Deng, and Zhaohui Wu, "Inverted indexing for composition-oriented service discovery," In Proceedings of the International Conference on Web Services (ICWS'07), pages 257-264, Salt Lake City, Utah, USA, 2007.

[12] Srividya Knoa, Ajay Bansal, and Gopal Gupta, "Automatic composition of semantic web services," In Proceedings of the International Conference on Web Services (ICWS'07), pages 150-158, Salt Lake City, Utah, USA, 2007.

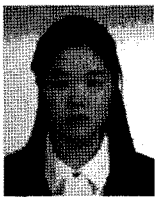
[13] Joonho Kwon, Kyuho Park, Daewook Lee, and Sukho Lee, "PSR : Pre-computing solutions in RDBMS for fast web services composition search," In Proceedings of the International Conference on Web Services(ICWS'07), pages 808-815, Salt Lake City, Utah, USA, 2007.

[14] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella, "Automatic composition of transition-based semantic web services with messaging," In Proceedings of the 31st VLDB Conference, Pages 613-624, Trondheim, Norway, August 2005.

[15] John Gekas and Maria Fasli, "Automatic web service composition using web connectivity analysis techniques," In International Conference on Ontologies, Databases and Applications of Semantics, pages 1571-1587, Agia Napa, Cyprus, 2005.

[16] Zhen Liu, Anand Ranganathan, Anton Riabov, "Modeling web services using semantic graph transformations to aid automatic composition," In Proceedings of the International Conference on Web Services(ICWS'07), pages 78-85, Salt Lake City, Utah, USA, 2007.

[17] Ulrich Kuster, Mirco Stern, and Birgitta Konig-Ries, "A classification of issues and approaches in automatic service composition," In Proceedings of the First International Workshop on Engineering Service Compositions (WESC'05), pages 25-33, December 2005.



김 현 지

2006년 한양대학교 컴퓨터교육과 졸업 2008년 서울대학교 전기컴퓨터공학부(석사). 2009년~현재 티맥스소프트 연구원 관심분야는 웹 서비스

권 준 호

1999년 서울대학교 컴퓨터공학과 졸업 2001년 서울대학교 전기컴퓨터공학부(석사). 2009년 서울대학교 전기컴퓨터공학부(박사). 2009년~현재 서울대학교 차세대융합기술연구원 연구원. 관심분야는 XML 문서 필터링, XML 인덱싱, 웹 서비스등

이 대 욱

1995년 경북대학교 컴퓨터공학과(학사) 1997년 경북대학교 대학원 컴퓨터공학과(석사). 1997년~2003년 ㈜퓨쳐인포넷부설 연구소 연구원. 2009년 서울대학교 전기컴퓨터공학부(박사). 2009년~현재 연세대학교 컴퓨터과학과 BK21 박사후연구원. 관심분야는 XML 인덱싱, XML 문서 필터링, 웹 서비스 등

이 석 호

1964년 연세대학교 정치외교학과 졸업 1975년, 1979년 미국 텍사스대학교 전산학 석사와 박사학위 취득. 1979년~1982년 한국과학원 전산학과 조교수. 1982년~1986년 한국정보과학회 논문 편집위원장. 1986년~1989년 미국 IBM T.J. Watson 연구소 객원교수. 1988년~1990년 데이타베이스연구회 운영위원장. 1989~1991년 서울대학교 중앙교육연구전산원 원장. 1994년 한국정보과학회 회장. 1997년~1999년 한국학술진흥재단부설 첨단학술정보센터 소장. 1982년~현재 서울대학교 컴퓨터공학부 교수. 1982년~2009년 서울대학교 컴퓨터공학부 교수. 2009년~현재 서울대학교 컴퓨터공학부 명예교수