시멘틱 웹 서비스의 조합

이용주*
*상주대학교 컴퓨터공학과
e-mail:yongju@sangju.ac.kr

Composition of Semantic Web Services

Yong-Ju Lee*
*Dept of Computer Engineering, Sangju National University

요 약

사용자가 현존하는 하나의 웹 서비스로는 요구사항들을 만족시켜줄 수 있는 기능이 없을 때, 그러한 요구를 만족시키기 위해 몇 개의 서비스들을 결합해야 하는 문제가 최근에 큰 이슈로 부각되고 있다. 본 논문에서는 비즈니스 분야에서 성공적으로 활용되고 있는 워크플로우 기법을 적용하여 웹 서비스 조합을 구현한다. 워크플로우의 웹 서비스 적용은 아직까지는 새로운 분야로써 다음과 같은 두 가지 문제 해결을 요구한다. (1) 인터넷상에 흩어져 있는 수많은 웹 서비스들 중에서 어떻게 원하는 서비스를 효율적으로 탐색할 수 있는가, (2) 탐색된 다양한 이기종 웹 서비스들 간의 상호운용성 극대화 문제에 관한 해결이 필요하다. 본 연구에서는 (1)을 수행하기 위해 웹 서비스 매칭 알고리즘이 제안되고, (2)를 해결하기 위해 매칭 알고리즘에 온톨로지 개념이 적용된다.

1. 서론

웹 서비스(web services)는 표준적인 웹 프로토콜을 통해 웹 환경 분산 컴퓨팅을 가능케 하며, CORBA나 DCOM과 같은 기존의 분산 컴퓨팅 모델을 대체할 수 있는 새로운 기술이므로 최근에 많은 관심을 받고 있다. 웹 서비스가 점차 현실화되고 보편화됨에 따라 향후 응용프로그램들은 웹 서비스들의 집합으로 구축될 수도 있다.

최근에 웹 서비스 조합(composition)에 대한 필요성이 점차 증가되고 있다[1]. 이는 단순히 하나의 웹 서비스 그 자체만으로는 다양하고 복잡한 사용자의 요구사항을 충분히 만족시켜줄 수 없기 때문에 서비스 조합을 통해 새롭고 유용한 솔루션을 얻기 위함이다.

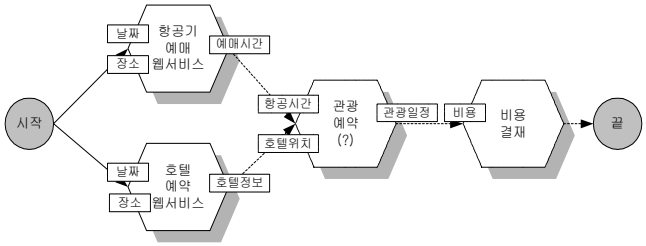
현재 웹 서비스 사용은 사용자들이 이미 알고 있는 몇 개의 서비스들을 이용하거나, 키워드 기반 검색엔진(예, 야후) 또는 웹 서비스 레지스트리(예, UDDI) 탐색에 의해 적합한 서비스들을 발견하고 있다. 또한 발견한 서비스들의 조합이나 이들 사이의 데이터 흐름은 주로 사용자 판단에 의한 수동으로 이루어지고 있다[2]. 이는 매우 불편한 작업이며 특히 복잡한 상황에서는 많은 시간이 소비되고 지루한 일이 될 수 있다.

본 논문은 웹 서비스 조합 자동화 기법에 초점을 맞추고 있다. 주된 아이디어는 시멘틱 웹을 실현하기 위한 웹 온톨로지(Web ontology)와 비즈니스 분야에서 성공적으로 활용되고 있는 워크플로우(workflow) 기법들을 적용하여 웹 서비스 조합 자동화 시스템을 구현하는 것이다. 워크플로우를 이용한 웹 서비스 조합에는 다음과 같은 두 가지 기술적인 문제 해결을 요구한다. (1) 웹 서비스의 발견(discovery): 인터넷 상에 흩어져 있는 수많은 웹 서비스들 중에서 어떻게 원하는 서비스를 효율적으로 탐색할 수 있겠는가, (2) 웹 서비스의 통합(integration): 앞 단계에서 발견된 다양한 이기종 웹 서비스들 간의 상호운용성 극대화 문제에 관한 해결이 필요하다. 기존 관련 논문들에서는 (1)에 관한 연구는 주로 이루어졌으나, (1)(2)가 결합된 연구는 거의 수행되지 않았다. 따라서 본 연구에서는 (1)을 위해 웹 서비스 매칭 알고리즘이 제안되고, (1)(2) 결합을 위해 웹 서비스 워크플로우가 새롭게 제안된다.

2. 시나리오

본 연구를 위해 먼저 현실적으로 실제 일어날 수 있는 하나의 가상 시나리오를 설정하고 여기에서 야기되는 새로운 이슈들을 살펴본다.

예를 들어 여행 날짜와 장소가 정해져 있을 때, 사용자는 "항공기 예매"와 "호텔예약" 웹 서비스는 기존의 UDDI를 통해 쉽게 발견할 수 있었다(그림 1 참조). 그렇지만, "관광예약" 웹 서비스는 적절한 웹 서비스를 찾지 못하였다. 이를 완성하기 위해 사용자는 적합한 관광예약 웹 서비스를 찾아서, 이 웹 서비스를 기존의 워크플로우에 결합시켜야만 한다.



(그림 1) 웹 서비스 탐색 및 조합

여기에서 워크플로우 사용자에게 두 가지 큰 문제가 발생하게 된다.

(1) 웹 서비스 탐색 문제: 전통적인 워크플로우 시스템과는 달리 인터넷상에는 수 없이 많은 웹 서비스들이 존재하므로 이들 중에서 가장 적당한 것을 찾는 일은 너무 시간이 많이 소비되고 지루한 일이 될 것이다. 따라서 수동으로 이러한 작업을 수행하는 것은 거의 불가능하게 보이며, 이를 효율적으로 지원할 수 있는 웹 서비스 탐색 메카니즘이 제공되어야만 한다.

(2) 웹 서비스 통합 문제: 일단 웹 서비스가 발견되었더라도 완벽하게 워크플로우에 일치되고 상호 운용 가능하리라고는 보장할 수 없다. 발견된 웹 서비스가 입출력 인터페이스 부분에서 구조적 또는 의미적으로 상이할 수 있으므로 이 작업을 효율적으로 지원할 수 있는 통합 메카니즘 제공이 필요하다.

본 논문에서는 차세대 웹 서비스 온톨로지 언어인 OWL-S[3] 및 워크플로우 기법을 이용하여 이 두 가지 문제를 해결하려고 한다.

3. 웹 서비스 탐색 및 통합 기법

3.1 질의 템플릿

사용자가 워크플로우에 첨가될 웹 서비스를 탐색할 필요가 있을 때 하나의 질의 템플릿을 작성한다. 질의 템플릿은 워크플로우가 최종적인 목표에 가깝게 도달될 수 있도록 그 단계에서 요구되는 웹 서비스의 특성을 표현한 하나의 청사진이다. 질의 템플릿 T는 다음과 같이 표현된다.

$T = <N, D, Is, Os, [Ps, Es, C, QR, SP]>$

여기서, N은 웹 서비스 이름, D는 텍스트 설명, Is는 입력 항목, Os는 출력 항목, Ps는 전제조건, Es는 효과, C는 카테고리, QR은 품질 등급, SP는 서비스 매개변수, [ ]는 선택사항을 표시한다.

3.2 매칭(matching) 알고리즘

매칭 알고리즘의 기본적인 원리는 질의 템플릿의 입력항목을 사용하여 원하는 출력을 산출해 낼 수 있는 웹 서비스들을 찾는 것이다. 이를 위해서 선택되는 웹 서비스는 반드시 템플릿의 출력항목을 포함하고 있어야만 하고, 이 서비스의 입력항목들은 템플릿의 입력항목에 포함되어 있어야만 한다.

> [정의 1] 질의 템플릿 T의 모든 출력항목들이 웹 서비스 S의 출력항목들과 매치가 되고, 이 S의 모든 입력항목들이 T의 입력항목들과 매치가 될 때, 이 S와 T는 매치된다고 할 수 있다.

[정의 1]은 매치되는 S가 T의 모든 요구사항을 만족할 수 있고, 이 S를 올바로 작동시키기 위해 필요한 모든 입력항목들은 T가 제공할 수 있다는 것을 보장한다. [정의 1]을 기반으로 한 웹 서비스 매칭 알고리즘은 (그림 2)와 같다. (그림 2)에서 main( ) 함수는 질의 템플릿 T를 서비스 저장소(service repository)에 있는 모든 웹 서비스들과 비교한다. 만일 매치가 발견되면 기록되고 우선순위에 의해 정렬된다. match( ) 함수는 먼저 템플릿 출력항목을 웹 서비스 출력항목과 비교하여 유사도를 계산하고, 매치가 실패하지 않는다면 반대로 웹 서비스 입력항목과 템플릿 입력항목을 비교한다.

```
Algorithm 1: Matching
  main(T) {
        record = empty;
        for each S ∈ serviceRepository) {
            if match(T, S) then record.append(S) }
        return sort(record)
  }
  match(T, S) {
        outputMatch(T{Os}, S{Os})
        inputMatch(S{Is}, T{Is})
  }
```

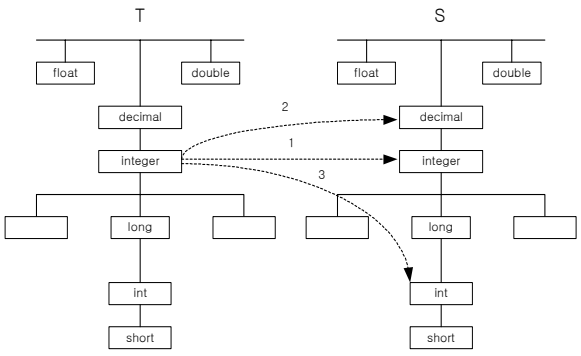
(그림 2) 웹 서비스 매칭 알고리즘

위 알고리즘에서 어떤 서비스들은 정확하게 매치되어 최종 결과로 선택되어 질 수도 있지만, 정확하지 않더라도 무조건 탈락되지 않고 상호 포함관계에 따라 우선순위가 정해질 수도 있다. 따라서 본 논문에서는 유사성 측정기법을 적용하여 우선순위별로 정렬하여 가장 높은 점수를 얻은 웹 서비스를 최종 결과로 선택하도록 한다.

OWL-S의 입출력은 내장 데이타 타입이나 온톨로지에서 정의된 개념으로 기술될 수 있다[3]. 따라서 두개의 입력항목(또는 출력항목)* 사이의 유사성은 두가지 경우, 즉 내장 데이타 타입과 온톨로지 개념으로 구분하여 측정할 수 있다.

(1) 내장 데이타 타입으로 선언된 경우

내장 데이타 타입과 관련된 유사도(Similarity) SM함수는 WfMS(Workflow Management System) 데이타 타입 변환 기능과 밀접한 연관이 있다. 일반적으로 WfMS에서는 태스크 사이에 데이타 전달을 할 때 데이타 타입 변환을 지원하는 능력을 가지고 있다. 예를 들면, 질의 템플릿 입력항목이 integer로 선언되어 있으나 웹 서비스 입력항목은 decimal인 경우, WfMS가 integer를 decimal로 데이타 타입 변환을 할 수 있다면 이의 유사도 SM 값은 1로 결정된다. 값 1은 유사도가 최대임을 의미한다. 그러나 WfMS에서 데이타 타입 변환이 불가능한 경우에는 웹 서비스 실행이 불가능하므로 SM 값은 0으로 되며, 이는 유사도가 전혀 없음을 의미한다. 본 논문에서 유사도 SM 측정값은 XML 스키마 데이타 타입 계층도를 기반으로 하고 있으며 통합 메카니즘을 위해 0과 1 사이의 유사도 값을 산출한다. <표 1>은 (그림 3)을 이용한 3가지 예(case1~3)에 대한 유사도를 계산한 것이다.



(그림 3) XML 스키마 데이타 타입 계층도

<표 1> 세가지 경우에 있어서 유사도 값

	Condition	SM
case 1	$T\{I\}$와$S\{I\}$가 같은 경우	1
case 2	$T\{I\}$가 $S\{I\}$의 하위 개념인 경우	1
case 3	$T\{I\}$가 $S\{I\}$의 상위 개념이거나 관계가 없는 경우	α

α값은 유사도 측정치를 관리자가 사전에 정의하여 검색항목이 내장 데이타 타입으로 선언된 경우 관련되는 SM 값을 산출한다.

(2) 온톨로지 개념으로 선언된 경우

질의 템플릿 입력항목이 온톨로지 개념으로 선언된 경우에는 객체지향 모델과 같은 계층 관계에 따라 유사도가 결정된다. 온톨로지 개념 간 유사성 관계는 다음과 같이 5가지 경우가 발생될 수 있다.

case 1: 같은 경우($T\{I\} = S\{I\}$)
case 2: $S\{I\}$가 $T\{I\}$의 상위 개념인 경우($S\{I\}$ subsumes $T\{I\}$)
case 3: $T\{I\}$가 $S\{I\}$의 상위 개념인 경우($T\{I\}$ subsumes $S\{I\}$)
case 4: $T\{I\}$와 $S\{I\}$ 사이에 일부분 공통된 속성이 있는 경우 ($T\{I\}$ intersect $S\{I\}$)
case 5: $T\{I\}$와 $S\{I\}$ 사이에 전혀 관계가 없는 경우 ($T\{I\} \neq S\{I\}$)

위의 5가지 경우를 고려한 유사성 측정 알고리즘은 다음과 같이 표현되고 각 경우에 따라 유사도 SM 값이 반환된다.

```
degreeOfMatch(T, S) {
        If T{I} = S{I} then return Exact
        If S{I} subsumes T{I} then return PlugIn
        If T{I} subsumes S{I} then return Subsumes
        If T{I} intersect S{I} then return Intersect
        otherwise Fail
}
```

여기서**,

$$SM = \begin{cases} 1 & \text{if } Exact \\ 1 & \text{if } PlugIn \\ \dfrac{|\,T.Ps\,|}{|\,S.Ps\,|} & \text{if } Subsumes \\ \dfrac{\partial}{\{\,|\,T.Ps\,|-\partial\}+\{\,|\,S.Ps\,|-\partial\}+\partial} & \text{if } Intersect \\ 0 & \text{if } Fail \end{cases}$$

만일 입출력항목이 여러 개 존재하는 경우에는 전체의 평균값을 유사도로 설정한다. 즉 전체 유사도 Similarity는 다음과 같이 표현된다.

$$Similarity = \frac{\Sigma\, SM_i}{n}$$

* 본 논문에서는 편의상 입력 항목만 예시한다.

** ∂ = $|\,T.Ps \cap S.Ps\,|$

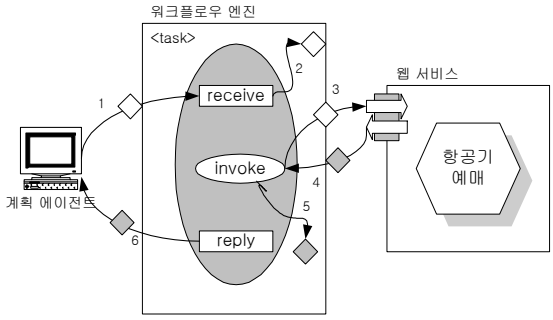
3.3 웹 서비스 워크플로우

워크플로우 시작에서부터 점차적으로 하나씩 새로운 서비스가 첨가되어 웹 서비스 워크플로우가 완성되고 나면 워크플로우 엔진에 의해 이들 프로세스들이 자동으로 수행되어야 한다.

OWL-S는 웹 서비스의 지능적 탐색 및 프로세스 실행은 지원하고 있으나 이질적 분산 환경에서 비지니스 프로세스의 조합을 위한 기능은 제공하지 못하고 있다. 이러한 필요성에 의해 BPEL4WS, WSFL, XLang, WSCI, WS-CDL 등 여러 가지 웹 서비스 조합언어가 발표되었으나 아직까지는 표준화가 정해져 있지 왔고, 이들 언어들은 시멘틱 개념을 지원해 주지 못하고 있다. 따라서 본 연구에서는 BPEL4WS 스펙을 기반으로 자체 웹 서비스 워크플로우 실행계획 SEP(Service Execution Plan)을 개발하였고 향후 OWL-S 등이 활용가능하면 이를 대체할 수 있도록 하였다.

SEP은 XML로 표현된 워크플로우 프로세스로써 태스크(task)와 제어 구성자(control construct)의 집합으로 구성된다. 태스크는 서비스 그라운딩 파일로부터 추출된 포트 타입과 작업(operation), 그리고 입력과 출력 매개변수를 가지고 있으며, 최종적으로 WSDL 파일과 바인딩하게 된다. 제어 구성자는 4개의 형태 즉, 순차(sequence), 반복(while), 병행(flow), 선택(switch)을 가지고 있다. 순차 블록에서는 단위 업무(activity)들이 하나씩 순차적으로 수행되고, 반복 블록은 리스트의 각 멤버들을 반복적으로 수행한다. 그리고 병행과 선택 블록에서는 단위 업무가 병렬로 수행되거나 여러 개 중 하나가 선택되어 수행된다.

워크플로우 엔진에서 웹 서비스가 실행되는 내부 과정을 살펴보면, 먼저 입력 매개변수를 받고(receive), 관련 웹 서비스를 호출(invoke)하여, 에이전트에게 응답(reply)하는 3단계로 이루어진다. 예를 들면, (그림 4)에서와 같이 워크플로우 클라이언트에서 날짜와 장소 입력 매개변수를 워크플로우 엔진에게 보내면 태스크 클래스가 생성되고 receive 메쏘드가 시작된다. 이 매개변수가 입력 컨테이너에게 넣어진 후, 곧바로 원격지에 있는 "항공기예매" 웹 서비스가 호출(invoke)된다. 웹 서비스가 실행된 후 그 결과가 출력 컨테이너에 넣어지고, 마지막으로 reply 메쏘드가 수행되어 계획 에이전트에게 그 결과가 보내진다. 한편, 태스크가 여러개 존재하는 복잡한 워크플로우인 경우에도 이러한 과정이 연속적으로 진행된다.



(그림 4) 웹 서비스 실행 내부 흐름

4. 결론

본 논문에서는 비즈니스 분야에서 성공적으로 활용되고 있는 워크플로우 기법을 적용하여 시멘틱 웹 서비스 조합을 구현하였다. 워크플로우의 웹 서비스 적용은 아직까지는 새로운 분야로써 웹 서비스 탐색 및 웹 서비스 통합 문제 두가지 문제 해결을 요구한다. 본 연구에서는 탐색 문제를 위해 웹 서비스 매칭 알고리즘이 제안되고, 통합 문제를 위해 시멘틱 웹 온톨로지 개념이 적용되었다.

참고문헌

[1] Sirin E., Hendler J., and Parsia B. "Semi-automatic Composition of Web Services using Semantic Description" Web Services: Modeling, Architecture and Infrastructure Workshop in Conjuction with ICEIS, 2003

[2] Arpinar I.B., Aleman-Meza B., Zhang R., and Maduko A. "Ontology-Driven Web Services Composition Platform" IEEE Conference on E-Commerce Technology(CEC 2004), San Diego, California, July 2004

[3] OWL Services Coalition "OWL-S: Semantic Markup for Web Services" OWL-S White Paper, http://www.daml.org/services/owl-s/1.0/owl-s.pdf, 2004