웹 서비스와 OpenAPI를 사용한 SOA 기반 동적 서비스 합성 프레임워크

(A SOA-based Dynamic Service Composition Framework using Web Services and OpenAPIs)

김 진 한 † 이 병 정 ††
(Jinhan Kim) (Byungjeong Lee)

요 약 최근 웹 2.0의 등장과 함께 플랫폼으로서의 웹이 강조되어 OpenAPI가 급격히 증가하고 있다. OpenAPI는 서비스를 결합한 새로운 서비스를 만들기 위하여 사용된다. 하지만 OpenAPI는 표준 문서가 존재하지 않아 사용에 한계가 있다. 그래서 기존 매쉬업 연구는 동적 서비스 합성보다는 주로 도구 설계나 서비스 결합을 위한 언어 정의에 국한되고 있다. 반면 SOA 구현 기술 중의 하나인 웹서비스는 서비스 설명을 위한 WSDL, 서비스 등록을 위한 UDDI, 메시지 전송을 위한 SOAP 등의 표준 문서를 제공한다. 이러한 기술들을 이용하여 웹 애플리케이션이 서비스를 해석하여 실행시킬 수 있다. 그리고 최근 SOA 연구에서는 서비스의 동적 합성과 의미적 특성을 지원한다. 만약 웹 서비스와 OpenAPI를 결합하는 동적이고 체계적인 방법이 제공된다면 웹 애플리케이션은 다양한 서비스를 사용자에게 제공할 수 있다.

본 연구에서는 OpenAPI와 웹 서비스의 매쉬업을 위한 SOA 기반 프레임워크를 제시한다. 본 프레임워크는 합성된 서비스의 프로세스는 OWL-S로 표현하여 OpenAPI와 웹 서비스의 동적 합성을 지원한다. 그리고 프로토타입을 통하여 본 프레임워크의 유효성을 보인다. 본 프레임워크는 기존 웹 서비스에 다양성을 부여할 것으로 기대된다.

키워드 : 서비스 합성, 웹 서비스, 오픈API, 온톨로지, 매쉬업

Abstract With the advent of Web 2.0, OpenAPIs are becoming an increasing trend to emphasize Web as platform recently. OpenAPIs are used to combine services and generate new services by mashup. However because the standard documents for OpenAPIs do not exist, it may restrict the use of OpenAPIs. Previous studies of OpenAPIs mashup have been limited to tool design or language definition for service combination rather than dynamic composition. On the other hand, Web services that are a software technology implementing SOA provide standard documents such as WSDL to explain each service, UDDI to register it, and SOAP to transfer messages. Thus Web applications can interpret and execute services by using these technologies. Recent works have also been performed to provide semantic features and dynamic composition for SOA. If a dynamic and systematic approach is provided to combine Web services and OpenAPIs, Web applications can provide users with diverse services.

In this study, we present a SOA based framework for mashup of OpenAPIs and Web services. The framework supports dynamic composition of OpenAPIs and Web services, where the process of composite services is described in OWL-S. A prototype is provided to validate our framework. The framework is expected to add diversity to typical Web services.

Key words : Service Composition, Web Service, OpenAPI, Ontology, Mashup

· 이 논문은 2008년도 서울시립대학교 교내학술연구비에 의하여 지원되었음
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '매쉬업을 사용한 SOA 기반 동적 서비스 합성 프레임워크'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 쌍용정보통신 개발지원팀
mail@kimjinhan.com
†† 종신회원 : 서울시립대학교 컴퓨터과학부 교수
bjlee@uos.ac.kr
(Corresponding author임)

논문접수 : 2008년 9월 3일
심사완료 : 2009년 1월 15일

정보과학회논문지: 소프트웨어 및 응용 제36권 제3호(2009.3)

1. 서 론

SOA(Service Oriented Architecture)는 전통적인 소프트웨어 개발에 새로운 패러다임과 기술 혁신을 가져다주었다[1]. SOA는 서비스 제공자와, 서비스 소비자 그리고 서비스 중개자와 그들의 관계로 구성되며 이종의 플랫폼간의 상호운용성을 제공하고, 느슨한 결합과 위치투명성과 같은 특징들을 가진다[2]. 이러한 특징들은 빠르게 변화하는 비즈니스 시장에서 시장 적시성을 만족시키고, 서비스의 재사용에 따른 비용감소등과 같은 효과를 가진다. 또한 SOA는 전통적인 공급자 중심을 소비자 중심의 서비스 조립(Composition)으로 소프트웨어 개발 접근법을 바꾸었다. 하지만 SOA는 BPEL (Business Process Execution Language), WSDL (Web Service Description Language), SCA(Service Component Architecture)와 같은 많은 SOA 기술의 학습을 요구하며, 이들 기술은 이미 존재하는 전통적인 (legacy) 애플리케이션과 웹 애플리케이션을 잘 지원하지 못하는 단점을 가진다[1].

웹 2.0[3] 기술은 인터넷상에서 혁신적인 서비스를 구현하기 위한 방법을 제공하고 사용자 중심의 비즈니스 모델을 지향한다. 웹 2.0 기술 중 하나인 OpenAPI는 플랫폼으로서의 웹을 실현하기 위해 데이타 또는 서비스를 널리 분산시키는 목적을 가지며, 이들 OpenAPI들의 매쉬업은 실제 프로세스를 가지지 않은 새로운 서비스를 재창조 해낸다. 하지만 OpenAPI는 단순함을 유지하기 위해 자신을 설명할 수 있는 표준적인 문서를 가지지 않으며 REST(REpresentational State Transfer), XML-RPC(Remote Procedure Call), SOAP(Simple Object Access Protocol) 등의 다양한 통신 프로토콜로 구현된다. 매쉬업 애플리케이션은 구조적으로 API 제공자, 매쉬업 호스팅 사이트, 소비자로 나누어진다[4]. 이것은 SOA의 구성요소와 같으며 그 역할도 유사하다.

본 논문에서는 매쉬업을 SOA 환경으로 통합하고 매쉬업의 단순한 특성으로 인한 문제점을 SOA의 정형적인 문서 및 특성으로 변경한다. 또한 기존 SOA 환경에서 사용되는 제한적인 서비스에 외부의 수많은 Open-API로부터 서비스를 가져와 그들 사이에서 동적 서비스 합성을 가능케 하고 이를 이용하는 애플리케이션에 빠르게 적응하는 프레임워크를 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 매쉬업 기반의 서비스 합성과 시맨틱 정보기반의 서비스 합성에 관한 기존 연구를 소개한다. 3장에서는 SOA 기반 동적 서비스 합성 프레임워크에 대하여 기술하고, 4장에서는 사례 연구에 대해 설명하고, 5장에서 토의를 기술한다. 마지막으로 6장에서는 결론과 향후 연구를 기술한다.

2. 관련 연구

2.1 매쉬업 기반 서비스 합성

매쉬업은 그 특징에 따라 UI 컴포넌트 모델, 서비스 컴포넌트 모델, 액션 컴포넌트 모델로 나누어 각각 모델을 결합하여 하나의 매쉬업을 만들었다[1]. 이들 모델은 클라이언트의 웹 페이지를 통해 매쉬업된 결과를 확인한다. WMSL(Web Mashup Scripting Language)[5]은 HTML, 메타데이타, 일부분의 코드를 결합하여 사용자가 비주얼한 도구에 의해 서비스들을 쉽게 매쉬업할 수 있도록 도와준다. 하지만 WMSL은 매쉬업을 위한 편의는 도모하나 자동화된 매쉬업을 제공하지 않는다. 또한 클라이언트의 웹 브라우저에서 표현되거나 실행되는 UI 컴포넌트 모델과 액션 컴포넌트 모델은 SOA 서비스들과 상호작용하기 위하여 적합하지 않다.

2.2 시맨틱스 기반 서비스 합성

서비스 합성은 다양한 합성을 통해 컴포넌트들의 재사용을 높여 새로운 애플리케이션을 빠르게 개발할 수 있도록 돕는다[6]. 서비스 합성 기술은 두 개의 타입으로 특징지어지는데 정적 서비스 합성과 동적 서비스 합성이다[7]. 정적 서비스 합성은 컴포넌트들 사이에서 메시지 교환을 상태 차트로 나타내거나 또는 워크플로우를 설계하여 수동으로 새로운 애플리케이션을 설계하는 접근법이다. BPEL4WS[8]와 WSCI[9]은 정적 서비스 합성을 위해 사용되는 언어이다. 정적 서비스 합성은 분기나 상호작용과 같은 복잡한 메시지 교환을 하는 애플리케이션을 지원한다. 그러나 이들의 애플리케이션들은 배포되기 이전에 수동으로 설계하는 단점이 있다. 따라서 정적 서비스 합성은 상호작용은 아주 복잡하지만, B2B와 같은 정적인 상황에 적합하다. 동적 서비스 합성은 애플리케이션의 배포이후에도 사용자의 요청에 의해 동적으로 애플리케이션을 합성한다. eFlow[10]와 SWORD [11]은 동적 서비스 합성 시스템의 예이다. 동적 서비스 합성은 복잡한 상호작용을 가지는 애플리케이션들을 합성하기 어려운 면이 아직 있으나, 사용자의 요청을 기반으로 컴포넌트들을 적절하게 선택하고 합성하는 유연한 애플리케이션을 만들 수 있는 방법을 제공한다. 하지만 지금까지 살펴온 연구들은 시맨틱스 개념을 지원하지 않고 단지 키워드 중심 탐색만이 가능하므로 그 활용도가 떨어진다. 시맨틱스 기반 동적 웹 서비스 합성에 관한 연구는 WS-Composer[12]에서 수행되었는데, 이 시스템은 동적 서비스 합성에 OWL-S(Web Ontology Language for WebService)를 이용한다. OWL-S는 서비스를 위한 웹 온톨로지 언어이다. 서비스를 기술하기 위해 OWL-S에는 세 가지 구성 요소들이 있다. 서비스 프로파일(ServiceProfile)은 사용자들로부터 요구하는 것

과 사용자들에게 제공할 수 있는 것을 기술한다. 서비스 모델(ServiceMedel)은 어떻게 서비스가 작동하는지를 명세 한다. 그리고 마지막으로 서비스 그라운딩(Service Grounding)은 어떻게 서비스를 사용하는지에 대한 정보를 가지고 있다. 서비스 모델의 하위클래스 중 프로세스 모델(ProcessModel)은 합성의 프로세스들과 그들의 연관 관계, 그리고 상호작용을 기술할 수 있다[13]. 그리고 ESB(Enterprice Service Bus) 아키텍처를 통해 이종의 서비스 메시지를 통합하는 연구[14]는 서비스에 대한 단일의 WSDL로 표현이 가능하나 시맨틱 정보를 활용할 수 없다. 이와 같은 연구들은 SOA 환경을 기반으로 하고 표준적인 문서를 가지지 않은 OpenAPI들의 매쉬업 환경을 통합하려는 노력이 없었다.

3. 동적 서비스 합성 프레임워크

본 논문에서는 매쉬업을 사용한 SOA 기반의 동적 서비스 합성 프레임워크를 제안한다. 매쉬업은 웹 애플리케이션에서 OpenAPI들의 서비스 합성을 의미한다. 그리고 OWL-S는 단일 서비스 또는 합성된 서비스들 사이의 프로세스를 나타내며 또한 물리적 호출을 위한 정보도 담고 있다. 본 프레임워크는 매쉬업을 SOA환경으로 통합하며, 이로 인해 기존 웹서비스들과 OpenAPI들 사이에서 서비스 합성을 제공하고 합성된 서비스의 프로세스를 OWL-S를 이용하여 기술한다. OpenAPI들과 기존 웹서비스들 간의 서비스 합성을 위해 단일 형태의 서비스 설명이 필요하다. 그리므로 본 논문은 단일 형태의 서비스 설명을 정의하고 이들을 기반으로 매쉬업을 SOA환경에 통합하여 동적 서비스 합성을 제공한다.

그림 1은 본 논문에서 제안하는 SOA 환경 기반의 서비스 합성 프레임워크를 보인다. 이들 구성요소는 크게 서비스 제공자(Service Provider), 서비스 중개자(Service Broker) 그리고 서비스 소비자(Service Consumer)와 같이 크게 세 개로 나뉜다. 첫 번째는 존재하는 서비스들을 프레임워크에 알맞게 등록하는 서비스 제공자이고, 두 번째는 프레임워크를 기반으로 애플리케이션을 사용하는 서비스 소비자이다. 세 번째는 서비스 제공자의 서비스 등록 요청과 등록된 서비스들의 서비스 합성 그리고 서비스 소비자의 서비스 검색 및 OWL-S를 제공하는 서비스 중개자이다.

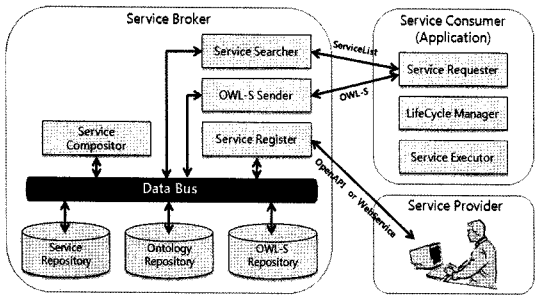


그림 1 SOA 기반 서비스 합성 프레임워크

서비스 제공자가 서비스 중개자의 서비스 등록기(Service Register)를 통해 서비스 등록 설명(Service Registration Description)을 등록하면, 서비스 중개자의 서비스 합성기(Service Compositor)는 서비스 저장소(Service Repository)에 등록된 모든 서비스들과 연관하여 자동으로 서비스들을 합성하고 이로 인해 산출되는 서비스 설명(Service Description)과 OWL-S를 서비스 저장소와 OWL-S 저장소(OWL-S Repository)에 각각 저장한다. 서비스 소비자는 프레임워크를 기반으로 작성된 소프트웨어 또는 파생되는 애플리케이션들이다. 서비스 소비자는 생명주기 관리자(LifeCycle Manager)를 통해 서비스 중개자로부터 가져온 서비스들을 배포(Deploy) 및 제거(UnDeploy)할 수 있고 서비스 실행기(Service Executor)를 통해 배포된 서비스들을 실행할 수 있다. 또한 서비스 소비자가 애플리케이션에 새로운 서비스를 적용하고자 한다면, 서비스 요청 설명(Service Request Description)을 작성하여 서비스 중개자의 서비스 검색기에 요청한다. 서비스 검색기는 서비스 저장소에 저장된 서비스들과 서비스 요청 설명을 의미적으로 대조하여 적절한 서비스들의 목록을 만들어 서비스 소비자의 서비스 요청기(Service Requester)에게 서비스 목록을 돌려준다. 서비스 요청기는 가져온 서비스 목록 내에서 서비스들의 순위와 서비스 합성도를 평가하여 선택한 서비스의 OWL-S 참조 주소를 서비스 중개자의 OWL-S 제공자(OWL-S Sender)에게 요청한다. 이후 가져온 OWL-S를 생명주기 관리자를 통해 프레임워크에 서비스를 배포한다. 서비스 중개자의 온톨로지 저장소(Ontology Repository)는 프레임워크 내에서 사용되는 온톨로지가 저장된다. 이 온톨로지는 서비스 등록 설명을 만들 때, 서비스를 조합할 때, 서비스 요청 설명을 만들 때 그리고 서비스를 검색할 때 참조되며 언제든지 온톨로지 내부에 개념(Concept)을 추가 또는 수정이 가능하다.

3.1 서비스 등록

기존 웹 서비스에서는 각각의 서비스를 설명할 수 있는 WSDL 문서를 포함한다. 웹 서비스를 호출하는 클라이언트는 WSDL 문서를 기계적으로 해석하여 그 웹 서비스가 가지고 있는 기능을 식별하고 이 기능을 호출하기 위한 입력 파라미터와 그 결과로 얻어질 결과 파라미터에 대한 정보를 미리 알 수 있다. 하지만 이 웹 서비스를 사용하기 위해 WSDL 문서와 SOAP 메시지에 대한 지식을 필요로 한다. 이와 대조적으로 Open-

API은 그 사용에 있어 매우 단순하며, 스스로 서비스를 설명하는 그 어떤 문서도 요구하지 않는다. 때문에 기계적으로 그 서비스를 해석하기는 그 만큼 어려움이 있다. OpenAPI를 사용하는 자동화된 서비스 합성을 구현하기 위해 본 논문에서는 서비스 제공자가 서비스를 등록할 때 OpenAPI 서비스 특징을 설명하도록 요구한다. 이 서비스 등록 설명이 포함하는 입력, 출력, 선 조건 그리고 효과 파라미터 타입을 문자 이외에 의미가 없는 꼬리표(tag) 보다는 의미와 관계가 표현된 온톨로지의 클래스로 명시해준다. 또한 웹 서비스도 WSDL에 그들의 입력, 출력 파라미터 타입만을 가지고 있기 때문에 선 조건과 효과 파라미터를 추가하고 이들 모두에 해당하는 적절한 온톨로지를 선택하여 서비스 등록 설명 정보를 만들어 준다.

그림 2는 서비스 등록 설명을 위한 서비스 등록 설명 스키마이다. 이 스키마는 XML 스키마 표준을 따르며, 등록되는 서비스 등록 설명의 유효성을 검증할 수 있다. <service> 요소 내부에는 유일한 서비스 이름인 <name> 요소와 이 서비스를 접근하기 위한 <accessURI> 요소를 포함한다. 또한 이 논문에서는 SOAP과 REST 프로토콜을 모두 지원하기 때문에 이 둘을 구분하기 위한 <accessMethod> 요소를 포함한다. 이 요소의 값은 SOAP과 REST 중 하나의 값만 올 수 있다. 만약 요소 값이 SOAP이면 기존 웹 서비스 호출방식으로 간주하여 이 웹 서비스의 메서드들 중 호출을 원하는 메서드 이름을 <SOAP_Operation>에 명시하고 WSDL에 명세된 이름공간(namespace)명을 <SOAP_Namespace>에

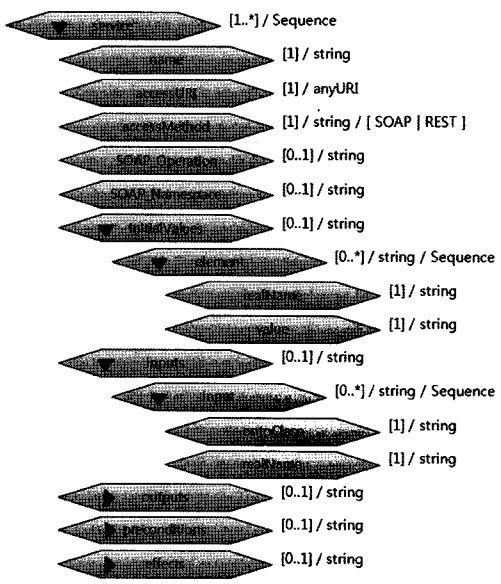


그림 2 서비스 등록 설명 스키마

기술한다. 반면에 REST가 명시되면 OpenAPI기반의 서비스들로 간주하고 SOAP와 관련된 요소의 값들은 무시된다.

<initialValues> 요소는 서비스를 호출할 때마다 고정적으로 요구되는 초기 값들을 설정할 수 있다. OpenAPI 호출을 위한 회원 아이디 또는 인증키의 요구는 서비스 합성의 입출력 파라미터로 받아드리기 어렵기 때문에 미리 초기 변수명과 초기 값을 <initialValues> 자식 요소인 <realName>과 <value> 요소에 각각 명시해 준다. 서비스를 호출하기 위한 서비스 입력과 그 결과로 얻어지는 결과를 명세하기 위해 <inputs> 요소와 <outputs> 요소를 포함하고 있다. 이들 요소는 기계적으로 해석되어 다른 서비스들과 합성되기 위해 온톨로지를 사용하며 온톨로지내의 개념을 <ontoClass> 요소에 값으로 참조 URI를 지정한다. 또한 물리적 호출을 위해 실제 입력 변수 명을 <realName>에 명세 한다. <preconditions> 요소는 호출하고자 하는 서비스에 선 조건이 있을 경우 그에 해당하는 온톨로지 개념을 지정하고 호출했을 때 기대하는 효과가 있을 경우 <effects> 요소에 효과들을 지정한다.

3.2 서비스 합성

3.2.1 서비스 합성 알고리즘

서비스 합성은 하나 이상의 독립적 실행 가능한 서비스들이 서로 메시지를 교환하는 합성된 서비스를 의미한다. 즉, 합성된 서비스는 처리 논리를 가진 단일 서비스들 또는 다른 합성된 서비스들의 메시지 프로세스가 하나로 합성된 서비스이다. 이러한 서비스 합성에 대한 연구는 [15]에서 서비스간 의미적 매칭의 수치와 QoS에 해당하는 최단 실행시간을 합산하는 인터페이스 기반 자동 합성을 제안하였으며, [16]에서는 어휘적 이름 매칭, 의미적 이름 매칭, 타입 매칭 그리고 구조적 매칭의 수치를 각각 구하여 가중치를 적용한 서비스 합성 유사도를 구했으며, [17]에서는 온톨로지 클래스들 사이의 상속관계 뿐만 아니라 포함 관계(Part-of)까지 수용한 서비스 합성을 보여준다.

본 논문에서는 합성된 서비스를 자동으로 생성하기위해 단일 서비스간의 메시지 프로세스에 대해 순차적 호출(Sequence Call)만을 대상으로 하며 아래와 같이 합성 서비스를 정의한다.

정의 1. 합성 서비스

전체 시스템은 서비스 집합 $S=\{s_1,s_2,...,s_i,...,s_n\}$으로 구성되고, 서비스 s_i는 단일 서비스 또는 다른 합성 서비스이다. 임의의 완전 순서화된 서비스 집합 $S'=\{s_1,s_2,...,s_i,...,s_m\}$, $1 \le m < n$ 그리고 $S \supset S'$인 서비스 집합 S'을 합성 서비스 c로 정의한다.

정의 1에서 합성 서비스는 여러 단일 서비스를 포함

하며, 그들은 하나의 열로 연결된다. 이 사실은 단일 서비스들은 순차적으로 호출이 발생되며, 어떠한 단일 서비스도 한번만 그 열에 소속됨으로서 순환조건을 배제했다. 정의 2와 정의 3은 합성 서비스를 구성하는 단일 서비스들의 매칭과 합성 서비스의 IOPE(Input, Output, Precondition and Effect)에 대한 정의이다.

정의 2. 합성 서비스를 구성하는 단일 서비스들의 매칭

합성 서비스를 구성하는 단일 서비스 집합 $S=\{s_1,s_1,...,s_i,...,s_n\}$일 때, s_i와 s_{i+1}의 연결 조건은 s_i의 하나이상의 출력 클래스와 s_{i+1}의 하나이상의 입력 클래스에 매칭 된다. s_i와 s_{i+1}에 각각 효과와 선 조건이 모두 존재하는 경우, 이들은 서로 하나이상 매칭된다.

정의 3. 합성 서비스의 IOPE

합성 서비스 c와 합성 서비스를 구성하는 단일 서비스 집합 $S=\{s_1,s_2,...,s_i,...,s_n\}$일 때, c의 입력은 s_1의 입력과 s_1을 제외한 모든 s_i의 입력이 s_{i-1}의 출력과 연결되지 못한 입력들의 합이다. c의 선 조건은 s_1의 선 조건과 s_1을 제외한 모든 s_i의 선 조건이 s_{i-1}의 효과와 연결되지 못한 선 조건들의 합이다. c의 출력은 s_n의 출력과 s_n을 제외한 모든 s_i의 출력이 s_{i+1}의 입력과 연결되지 못한 출력들의 합이다. c의 효과는 s_n의 효과와 s_n을 제외한 모든 s_i의 효과가 s_{i+1}의 선 조건과 연결되지 못한 효과들의 합이다.

서비스들 사이에서 메시지를 서로 교환하기 위해서는 메시지의 타입이 서비스들 간에 매칭이 되어야 서비스 합성이 가능하다. 특히 온톨로지를 이용한 의미적 매칭은 텍스트 매칭의 한계를 극복하고 의미적 접근을 가능케 해준다.

그림 3은 책을 모델링한 온톨로지 예제를 보여준다. thing 클래스로부터 person, title 그리고 ISBN 클래스가 상속을 받는다. 그리고 author, publisher 클래스는 person 클래스의 자식 클래스들이다. 이 온톨로지를 기반으로 만든 단일 서비스가 표 1에 세 개(S_1, S_2, S_3)의 예제 서비스들로 정의하였다.

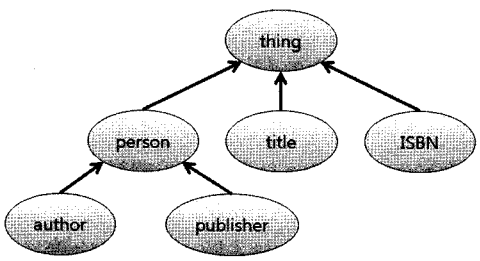
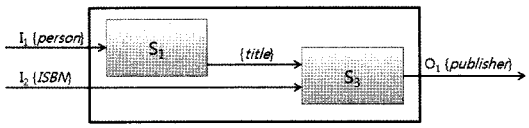


그림 3 서비스 합성을 위한 온톨로지 예제

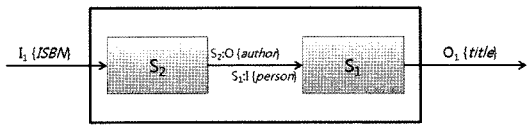
표 1 온톨로지 기반의 서비스 정의

S_1 (in:{*person*}, out:{*title*})
S_2 (in:{*ISBN*}, out:{*author*})
S_3 (in:{*title, ISBN*}, out:{*publisher*})

각 서비스 S_n은 서비스 호출을 위한 선 조건과 효과가 생략된 입력(in)과 출력(out)을 가지며, 입출력 타입은 온톨로지의 클래스로 지정하였다. 그림 4는 표 1의 세 개의 서비스에서 합성 가능한 최대 합성 서비스를 자동으로 산출한 결과를 보여준다. (a)의 S_3는 S_1의 결과인 {*title*} 뿐만 아니라 {*ISBN*}를 필요로 한다. 이때 합성된 서비스는 입력 I_1 {*person*}, I_2 {*ISBN*} 각각을 다른 시점에서 내부의 단일 서비스들에 대입해줌으로서 S_1과 S_3을 합성할 수 있다. (b)의 S_2출력과 S_1의 입력은 다른 타입으로 정의되어있으나, 그림 3의 온톨로지 모델을 참조하여 {*author*} 클래스가 {*person*}의 자식 클래스임을 추론하여 의미적 매칭을 가능케 해준다.



(a) 합성된 S_1-S_3 서비스



(b) 합성된 S_2-S_1 서비스

그림 4 자동적으로 산출된 합성된 서비스들

온톨로지를 이용한 의미적 매칭은 [18] 연구에서 제안된 방식으로 매치의 정도를 아래와 같이 분류한다.

- *Exact* : 요청 클래스와 대상 클래스가 일치
- *Subsume* : 요청 클래스가 대상 클래스의 하위 클래스일 때(자동 타입 변환 가능)
- *Relaxed* : 요청 클래스가 대상 클래스의 상위 클래스일 때(수동 타입 변환 가능)
- *Fail* : 요청 클래스와 대상 클래스가 불일치

두 개의 파라미터의 매칭은 그들 파라미터가 가지는 온톨로지 클래스를 대상으로 하며, 첫 번째 호출되는 서비스의 출력 또는 효과 클래스를 요청 클래스라 하고, 요청 클래스와 매치되어야 할 클래스를 대상 클래스라 한다. *Fail*를 제외한 모든 매치는 두 클래스가 연결 가능한 매칭이고 서비스들의 합성 가능 여부를 판단할 수 있다. 또한 Trversky의 특징기반 유사도 모델[19]을 변

형하여 두 클래스의 매치 정도가 *Relaxed*인 경우 클래스가 지니는 속성의 개수에 따라 온톨로지 클래스의 의미적 유사도를 다음과 같이 정의한다.

정의 4. 온톨로지 입·출력 클래스 유사도

$$Similarity(x,y)=\begin{cases}1 & if\ \ Exact\\ 1 & if\ \ Subsume\\ \dfrac{P(x)}{P(y)} & if\ \ Relaxed\\ 0 & if\ \ fail\end{cases}$$

- $Similarity(x,y)$: $Similarity$ 함수는 요청 온톨로지 클래스 x, 대상 온톨로지 클래스 y의 클래스 유사도를 반환한다.
- $P(x)$: P 함수는 x 온톨로지 클래스의 속성 개수를 반환한다.

온톨로지에 표현되는 클래스는 상위 또는 하위 클래스와 상속 관계를 가질 수 있으며, 또한 다른 클래스와 연관 관계를 가질 수 있다. 하나의 클래스에 속성들을 선언할 수 있으며 이 클래스 모델로부터 인스턴스가 만들어지면 속성들에 값을 할당할 수 있다. 상속 관계에서 하위 클래스는 상위 클래스의 모든 속성을 그대로 유지하면서 하위 클래스만의 속성을 추가할 수 있다. 두 서비스가 합성되었다는 것은 전자의 서비스 출력이 후자의 서비스 입력과 연결되어 데이타를 전송하며 최종 결과 값을 얻을 수 있다는 것을 의미한다. 이때 후자 서비스 입력의 타입이 전자 서비스 출력 타입의 상위 클래스 일 때, 후자 타입의 모든 속성을 만족하는 하위 클래스가 전자의 타입이다. 그래서 Exact 관계의 클래스 유사도는 최대값인 1이다. 또한 Subsume 관계인 경우 서비스 출력 타입이 적어도 서비스 입력 타입의 동일 속성 집합을 가지고 있기 때문에 1이 된다[19]. 반대로 *Relaxed* 관계에서는 후자 서비스의 입력 타입이 전자 서비스의 출력 타입의 하위 클래스이다. 즉, 전자의 출력보다 더 많은 속성을 지닌 하위 클래스를 후자 서비스가 입력으로 요구한다. 이때, 전자의 출력 타입의 모든 속성은 후자의 입력 타입의 모든 속성의 부분 집합이다. 그러므로 하위 클래스가 상위 클래스 보다 얼마나 속성을 추가하였는지에 따라 유사도는 점점 더 0으로 향하게 된다. 즉, 속성의 개수 $P(y)$의 변화에 따른 클래스 유사도가 계산된다. 합성 서비스를 구성할 때 온톨로지에 모델로 저장된 클래스의 이름을 레퍼런스로 사용하며 속성은 *Relaxed* 관계일 때 정량적으로 유사도를 평가할 수 있는 근거로 사용된다.

정의 5. 합성 서비스 합성도

$$Composability(c)=\frac{\sum_{i=1}^{l-1}\left\lfloor\left\{\sum_{\substack{x\in O_i\\ y\in I_{i+1}\\ \exists x\rightarrow y}}Similarity(x,y)\right\}/m+\left\{\sum_{\substack{x\in I_i\\ y\in P_{i+1}\\ \exists x\rightarrow y}}Similarity(x,y)\right\}/n\right\rfloor/2}{l-1}$$

- $Composability(c)$: $Composability$ 함수는 합성 서비스 c의 합성도를 반환한다.
- l : 합성 서비스 c를 구성하는 단일 서비스들의 순서화된 집합 S'의 $n(S')$
- i : S'의 원소 s_i의 색인
- m : i 서비스 출력과 $i+1$ 서비스 입력 사이의 연결된 매칭 집합
- n : i 서비스 효과와 $i+1$ 서비스 선조건 사이의 연결된 매칭 집합
- $x\rightarrow y$: 온톨로지 클래스 x로부터 y로의 링크

정의 5는 합성 서비스를 구성하는 단일 서비스들 사이의 합성도를 구하는 함수에 대한 정의이다. 합성 서비스의 합성도는 단일 서비스들 사이의 IOPE에 대하여 연결을 가지는 온톨로지 클래스 유사도의 평균값이다. 단일 서비스들의 합성은 그들의 온톨로지 클래스가 연결 가능한 경우 모두 합성이 되며 합성도는 합성 서비스의 품질에 대한 측정치이다. 표 2는 자동화된 서비스 합성을 위한 알고리즘의 의사 코드를 보여준다. 서비스

표 2 서비스 합성 의사 코드

```
function setCompositeList(Composite)
    Composite.outputs =
Composite.lastAtomicService.outputs
    Composite.effects =
Composite.lastAtomicService.effects
    CompositeList.add(Composite)
end #function

function compatible(Service1, Service2)
    foreach item1 in Service1
        foreach item2 in Service2
            if item1 equivalence item2 then
                return true;
            else if item1 subClassOf item2 then
                rethrn ture;
            else if item1 superClassOf item2 then
                return ture;
            end #if
        end #foreach
    end #foreach
    return false;
end #function

function match(Composite)
    S1 = Composite.lastAtomicService
    foreach S2 in serviceRepository
      if (compatible(S1.outputs, S2.inputs)) and
        (compatible(S1.effects, S2.preconditions)) then
          if S2 exist in Composite then
              return
          end #if
          Composite.addAtomicService(S2)
          setCompositeList(Composite)
          match(Composite)
      end #if
   end #foreach
end #function

foreach service in serviceRepository
    Composite.inputs = service.inputs
    Composite.preconditions = service.preconditions
    Composite.addAtomicService(service)
    match(Composite)
end
```

제공자가 서비스를 등록할 때, 등록하려는 서비스와 현재까지 서비스 저장소에 등록된 모든 서비스들과 결합 가능한 합성 서비스들을 동적으로 찾는다. 이때, 코드의 ***compatible***은 두 서비스에 해당하는 온톨로지 클래스들의 연결 조건을 [18]에서의 매치 정도를 사용하여 연결 가능을 판단한다. 즉, *Fail*를 제외한 *Exact, Subsume, Relaxed*는 ***compatible*** 가능하다. 이것은 본 논문이 서비스의 출력과 입력이 정확히 매치되는 것뿐만 아니라 온톨로지를 이용하여 그 유사성을 고려하여 서비스를 합성시키고 있는 것을 보여주고 있다.

3.2.2 서비스 합성의 프로세스

서비스 합성의 결과물은 합성된 서비스의 프로세스이다. 본 프레임워크는 이 프로세스를 나타내기 위해 OWL-S의 서비스 모델을 사용한다. 표 3은 합성된 서비스의 프로세스를 OWL-S로 표현한 예제이다. 이 프로세스는 2개의 단일(Atomic)서비스가 합성되어 있으며 이 프로세스에 참가하는 입력은 feeling, category이고, 출력은 item이다. 하지만 category는 첫 번째 호출되는 단일 서비스의 출력 값이기 때문에 실제적인 입력은 feeling이고, 최종 얻어지는 결과는 item이다. 본 논문은 자동화된 합성을 생성하기 때문에 프로세스에 관하여 단순한 순차(Sequence) 프로세스만 생성한다. 표 3에서 <process:Perform rdf:nodeID="Perform2"> 요소에는 Perform1의 결과가 Perform2의 입력이며, 결과는 <process:hasResult>를 통해 최종 item을 얻을 수 있다.

3.3 서비스 검색

본 논문에서 제안하는 프레임워크는 서비스 소비자의 요청을 따르는 서비스를 제공하기 위해 [20]에서 제안된 서비스 발견 기법을 확장하여 사용한다. [20] 연구에서는 검색하고자 하는 대상이 단일 서비스이었으나, 본 논문은 합성 서비스를 대상으로 하고 서비스 요청 결과에 합성 서비스의 합성도를 포함하도록 확장하였다. 또한 [20] 연구에서는 서비스 요청 설명을 OWL-S의 서비스 프로파일을 사용하였으나, 본 논문에서는 새롭게 서비스 요청 설명을 정의하여 IOPE에 대한 가중치가 아닌 모든 IOPE가 가지는 요소마다 가중치를 적용한다. 서비스 중개자 내부에는 온톨로지 저장소, 서비스 저장소 그리고 OWL-S 저장소가 있다. 온톨로지 저장소는 서비스를 등록할 때 서비스들의 합성을 위해 참조되며 또한 서비스 검색시점에도 참조가 된다. 서비스 저장소는 서비스 중개자가 가진 단일 서비스들과 이 단일 서비스로부터 파생된 합성 서비스들을 저장하고 있다. 서비스 소비자가 서비스 검색을 요청하면 이 저장소로부터 서비스들을 검색한다. OWL-S 저장소는 서비스 소비자가 검색된 서비스 중 하나를 선택하여 OWL-S를 요청하면 직접 OWL-S를 제공한다.

표 3 OWL-S의 합성 프로세스 서비스 모델

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY list "http://www.daml.org/services/owl-s/1.1/generic/ObjectList.ow
]>
<process:CompositeProcess rdf:about="Process">

<process:hasInput rdf:resource="ontology.owl#feeling"/>
<process:hasInput rdf:resource="ontology.owl#category"/>
<process:hasOutput rdf:resource="ontology.owl#item"/>

<process:composedOf>
   <process:Sequence>
      <process:components>
         <process:ControlConstructList>
            <list:first>
               <process:Perform rdf:nodeID="Perform1"/>
            </list:first>
            <list:rest>
               <process:ControlConstructList>
                  <list:first>
                     <process:Perform rdf:nodeID="Perform2"/>
                  </list:first>
                  <list:rest/>
               </process:ControlConstructList>
            </list:rest>
         </process:ControlConstructList>
      </process:components>
   </process:Sequence>
</process:composedOf>
               </process:ValueOf>
            </process:valueSource>
         </process:OutputBinding>
   </process:withOutput>
</process:Result>
</process:hasResult>

</process:CompositeProcess>

<process:Perform rdf:nodeID="Perform1">
   <process:hasDataFrom>
      <process:InputBinding>
         <process:valueSource>
            <process:ValueOf>
               <process:theVar rdf:resource="ontology.owl#feeling"/>
            </process:ValueOf>
         </process:valueSource>
      </process:InputBinding>
   </process:hasDataFrom>
</process:Perform>

<process:Perform rdf:nodeID="Perform2">
   <process:hasDataFrom>
      <process:InputBinding>
         <process:valueSource>
            <process:ValueOf>
               <process:fromProcess rdf:nodeID="Perform1"/>
               <process:theVar rdf:resource="ontology.owl#category"/>
            </process:ValueOf>
         </process:valueSource>
      </process:InputBinding>
   </process:hasDataFrom>
</process:Perform>
```

서비스 소비자는 이 프레임워크를 기반으로 작성된 애플리케이션에 새로운 서비스를 추가하거나 다른 서비스로 교체함으로써 서비스들을 재구성할 수 있다. 재구성 이전에 필요로 하는 서비스에 대한 IOPE를 추출하여 서비스 요청 설명을 작성한다. 표 4는 서비스 요청 설명의 예를 보여준다. 입력으로 온톨로지의 feeling(기분 상태) 클래스를 지정하였고 결과로 price(상품 가격)와 image(상품 이미지)를 지정하였다. 또한 서비스의 선 조건으로 유효한 기분 상태 값을 입력받기 위해 validFeeling을 지정하였다. 두 개의 결과 파라미터가

존재하는데 각각 가중치(weight)를 달리 주었다. 서비스에서 제공하는 결과에는 꼭 필요한 결과 파라미터가 있고 덜 중요한 결과 파라미터가 있다. 또는 가져온 서비스를 서비스 소비자의 애플리케이션에 적응시킬 때 결과 파라미터들 중 쉽게 적응 되는 파라미터가 있고 그렇지 못한 파라미터들이 있다. 이때 가중치의 역할은 중요한 결과 파라미터에 가중치를 높여주거나 또는 쉽게 적응되기 어려운 곳에 가중치를 높여줌으로써 서비스 중개자의 서비스 검색기가 서비스들의 순위를 조절할 수 있다.

표 4 서비스 요청 설명

```
<requestService>
    <hasInputs>
        <input>
            <ontoClass>ontology.owl#feeling</ontoClass>
            <weight>1</weight>
        </input>
    </hasInputs>
    <hasOutputs>
        <output>
            <ontoClass>ontology.owl#price</ontoClass>
            <weight>2</weight>
        </output>
        <output>
            <ontoClass>ontology.owl#image</ontoClass>
            <weight>1</weight>
        </output>
    </hasOutputs>
    <hasPreconditions>
            <ontoClass>ontology.owl#validFeeling</ontoClass>
            <weight>1</weight>
    </hasPreconditions>
    <hasEffects>
    </hasEffects>
</requestService>
```

그림 5는 서비스 소비자로부터 만들어지는 서비스 요청 설명을 서비스 중개자의 서비스 검색기에 전달하는 시점에서 최종 OWL-S를 얻는 과정을 순차도로 보여주고 있다.

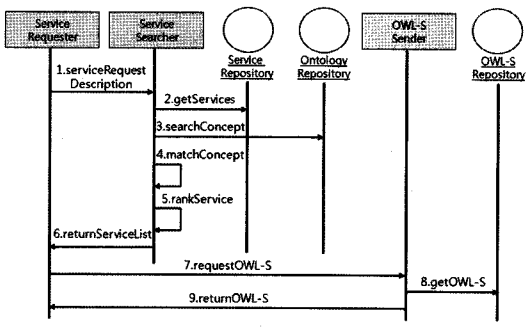


그림 5 서비스 검색 순차도

매칭의 순서는 (1) 서비스 검색기가 서비스 소비자의 서비스 요청기로부터 서비스 요청 설명을 전달받고, (2) 서비스 저장소로부터 모든 서비스들을 가져온다. (3) 서비스 요청 설명과 서비스 저장소로부터 가져온 서비스들에 존재하는 온톨로지 개념들을 추출하고 온톨로지 저장소로부터 해당하는 개념들을 가져온다. (4) 매칭 알고리즘을 사용하여 전체 서비스 중에서 서비스 요청 설명에 해당하는 서비스들의 매칭 점수를 계산한다. (5) 점수를 기반으로 서비스들을 랭킹 한다. (6) 최종 서비스 리스트를 서비스 소비자에게 반환한다. (7) 서비스 소비자는 리스트 중에서 최고 순위의 서비스를 선택하여 서비스 합성도가 일정 품질 이상을 경우 그 서비스의 OWL-S URI를 서비스 중개자의 OWL-S 제공자에게 요청한다. (8) OWL-S 제공자는 OWL-S 저장소에서 해당 OWL-S를 추출하여 (9) 서비스 소비자에게 돌려준다. 표 5는 표 4의 서비스 요청 설명에 대한 서비스 응답 설명의 예를 보인다. 발견된 서비스의 이름은 RecommendCategory_11stCategoryInfo_11stProductInfo이며, 매칭 되는 입출력이 feeling, price, image이고 추가적으로 product와 satisfaction을 출력으로 가지고 있다. 합성 서비스로 구성되어 있으며 합성도는 0.875이며, 매치의 정도와 가중치에 따른 총점 5점을 볼 수 있다.

표 5 서비스 응답 설명

```
<services>
    <service>
        <name>RecommendCategory_11stCategoryInfo_11stProductInfo</name>
        <inputs>
            <input>
                <ontoClass>ontology.owl#feeling</ontoClass>
            </input>
        </inputs>
        <outputs>
            <output>
                <ontoClass>ontology.owl#product</ontoClass>
            </output>
            <output>
                <ontoClass>ontology.owl#price</ontoClass>
            </output>
            <output>
                <ontoClass>ontology.owl#image</ontoClass>
            </output>
            <output>
                <ontoClass>ontology.owl#satisfaction</ontoClass>
            </output>
        </outputs>
        <preconditions>
            <precondition>
                <ontoClass>ontology.owl#validFeeling</ontoClass>
            </precondition>
        </preconditions>
        <effects>
            <effect>
                <ontoClass>ontology.owl#validProduct</ontoClass>
            </effect>
        </effects>
        <composability>0.875</composability>
        <score>5</score>
    </service>
</services>
```

3.4 서비스 재구성

본 프레임워크의 서비스 소비자는 서비스 요청기, 생명주기 관리자 그리고 서비스 실행기로 구성된다. 프레임워크를 기반으로 작성된 애플리케이션에서 사용자로부터 또는 애플리케이션 스스로 서비스를 재구성하기 위해 서비스 요청 설명을 작성하면, 서비스 요청기는 이를 사용하여 매칭된 서비스 리스트를 가져와 적합한 서비스를 선택하여 OWL-S를 제공받는다. OWL-S의 서비스 모델은 서비스의 프로세스를 의미하며 이를 통해 메시지 전달 순서 및 입출력 인자를 매칭할 수 있다. 하지만 합성된 프로세스에 참여하는 개별 서비스들의 물리적인 호출방법과 결과를 가져오는 방법에 대한 명세는 존재하지 않는다. 이를 위해 OWL-S의 서비스 그라운딩을 사용할 수 있다. 하지만 아직까지 서비스 그라운딩은 WSDL 1.1 만을 지원하며 본 논문에서 제시하는 OpenAPI의 프로토콜인 REST에 대하여 통합된 명세를 지원하지 않는다. 그래서 본 논문은 OWL-S의 서비스 그라운딩을 확장한다.

그림 6은 확장된 서비스 그라운딩을 보여주는 온톨로지 개념도이다. AccessMethod를 두어 REST에 대한 표현이 가능하고 InitialValue를 두어 그 서비스를 호출하기 위한 초기 값 설정도 가능하다. Value 개념은 기존 Parameter 개념이 가지는 하위 개념들은 서비스를 호출하기위한 변수 명과 타입일 뿐 초기 값을 설정할 수 있는 값에 대한 정의가 없기 때문에 정의하였다. 이 서비스 그라운딩은 서비스 중개자의 최종 결과 값으로서 OWL-S에 포함되어 전달받는다.

그림 7은 서비스 소비자의 아키텍처를 보여준다. 서비스 소비자의 서비스 요청기는 가져온 OWL-S의 서비스 모델과 서비스 그라운딩 문서를 기반으로 실행 가능한 서비스 컴포넌트를 생성한다. 이 컴포넌트는 생명주기 관리자에 의해 생명주기 저장소(LifeCycle Storage)에 위치한다. 사용자 또는 애플리케이션은 프레임워크에 배포 명령을 보내 생명주기 기억장치(LifeCycle Memory)로 컴포넌트를 위치시킬 수 있으며 반대로 삭제 명령을 보내 생명주기 기억장치로부터 생명주기 저장소로 컴포넌트를 내릴 수 있다. C_1과 C_2는 생명주기 메모리상에 있고, C_3은 생명주기 저장소로 내려지고 있는 컴포넌트이며 C_4는 생명주기 메모리로 올라가는 컴포넌트이다. 생명주기 관리자는 내부에 컴포넌트를 위한 공용 기억장치(Public Memory)와 개별 기억장치(Private Memory)를 두어 공유 데이타와 개별 데이타를 각각에 저장하여 사용한다. 서비스 실행기는 생명주기 기억장치에 존재하는 컴포넌트의 참조를 얻는다. 그리고 사용자 또는 애플리케이션으로부터 입력 값을 얻어 참조한 컴포넌트의 개별 기억장치에 보관한다. 또한 실행 명령을 받아 컴포넌트를 실행 후 결과를 개별 기억장치에 보관하여 사용자 또는 애플리케이션이 가져갈 수 있도록 한다.

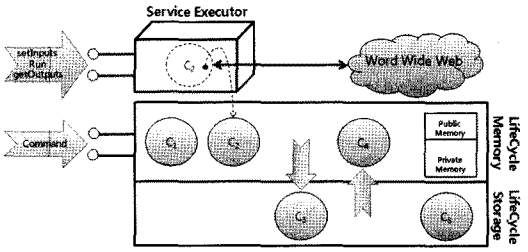


그림 7 서비스 소비자 아키텍처

4. 사례 연구

본 논문에서 제안하는 매쉬업을 사용한 SOA기반 동적 서비스 합성 프레임워크의 프로토타입을 구현하였다. 이 프로토타입은 프레임워크 부분과 애플리케이션 부분으로 나누어진다. 프레임워크는 서비스 중개자와 서비스 소비자로 구분되며 1:N의 연결로 구성된다. 서비스 중개자는 서비스 제공자로부터 얻어온 서비스를 등록하고

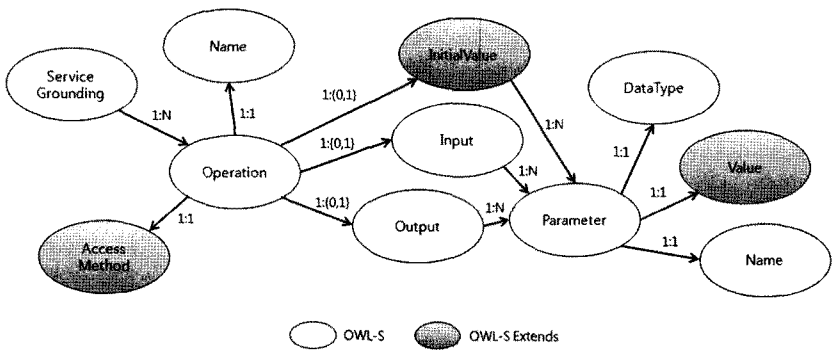


그림 6 확장된 서비스 그라운딩의 온톨로지 개념도

자동화된 서비스 합성을 실시하며 서비스 소비자의 서비스 검색 요구에 적합한 서비스를 제공한다. 이러한 역할을 담당하기위해 서비스 중개자는 서버위에 설치되고 웹을 통해 서비스를 제공한다. GlassFish 애플리케이션 서버에 서비스 중개자를 배포하고 JSP 페이지를 통해 서비스 등록과 서비스 검색을 제공하고 OpenAPI를 통해 서비스 검색과 OWL-S를 제공한다. 즉, 두 가지 접근방법을 제공함으로서 사용자뿐만 아니라 프로그래밍 가능한 애플리케이션의 접근을 허락한다. 서비스 소비자는 실행 중에 서비스들을 재구성 할 수 있으며 효율적인 서비스관리를 위해 생명주기 명령어를 수용한다. 애플리케이션 개발자 혹은 사용자는 서비스 소비자 프레임워크를 연결하여 생명주기 관리자와 서비스 요청기의 참조를 얻어 위의 목적을 달성할 수 있다. 전체 프레임워크에서 사용되어지는 온톨로지 관리는 SOFA(Simple Ontology Framework API)[21]를 사용하였고 OWL-S 변환은 OWL-S API[22]를 사용하였다. 애플리케이션 부분은 시나리오 기반으로 작성되었고 흐름은 아래와 같다.

1. 사용자는 애플리케이션에서 현재 기분 상태에 따라 다양한 상품의 카테고리를 추천해주는 서비스를 통해 카테고리 정보를 제공받는다.
2. 서비스 제공자는 쇼핑 OpenAPI를 서비스 중개자에게 등록한다.
3. 사용자는 애플리케이션에게 서비스 요청 설명을 주어 새로운 서비스를 서비스 중개자를 통해 가져온 후 서비스를 재구성한다.
4. 사용자가 현재 기분 상태를 입력하였을 때, 그 기분 상태에 해당하는 상품의 카테고리를 가져와 그 카테고리에 해당하는 상품을 쇼핑 OpenAPI를 통해 구매 정보를 얻어온다.

이 시나리오에 참여하는 단일 서비스는 표 6과 같다. 세 개의 서비스가 모두 등록되는 시점에서 Recommend Category Service -> 11st Open Market Category Info Service -> 11st Open Market Product Info Service 서비스들이 순서대로 합성된다. 이때 11st Open Market Category Info Service의 출력이 item이고, 11st Open Market Product Info Service의 입력이 product이다. 온톨로지 저장소에는 Item 개념이 product 개념의 슈퍼타입으로 명시되어있기 때문에 의미적으로 매칭이 된다.

그림 8은 서비스 제공자를 위한 서비스 중개자의 서비스 등록 모습이며 11st Open Market에서 제공하는 상품정보 서비스에 대한 서비스 등록을 보여주고 있다. 서비스 이름과 접근 URI 그리고 REST 프로토콜을 명시하였고, 이 OpenAPI를 호출하기 위한 인증된 키 값을 InitialValue 항목의 key 변수 값에 할당하였다. 또한 입력, 출력 항목에는 이 서비스의 각각 파라미터에 해당하는 온톨로지의 개념과 실제 호출을 위한 입력 파라미터명과 결과물인 XML문서를 파싱하기 위한 출력 요소를 명시하였다. 또한 이 서비스를 호출하기 위한 선조건으로서 유효한 상품 코드로 제한하였다. 등록(Submit) 버튼을 누르면 입력 폼을 기반으로 서비스 등록 설명을 만들어 서버 내부에서 등록 절차가 일어난다. 등록과 동시에 등록된 서비스들과의 가능한 합성을 찾아 OWL-S 저장소와 서비스 저장소에 각각 OWL-S와 서비스가 등록된다.

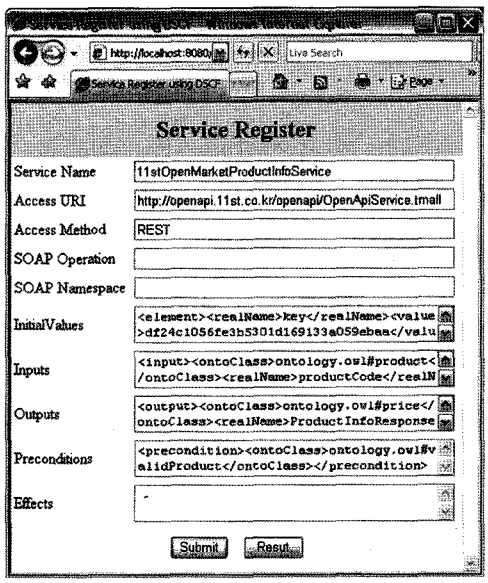
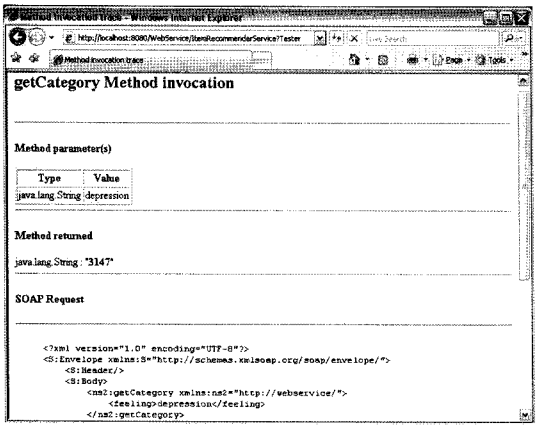


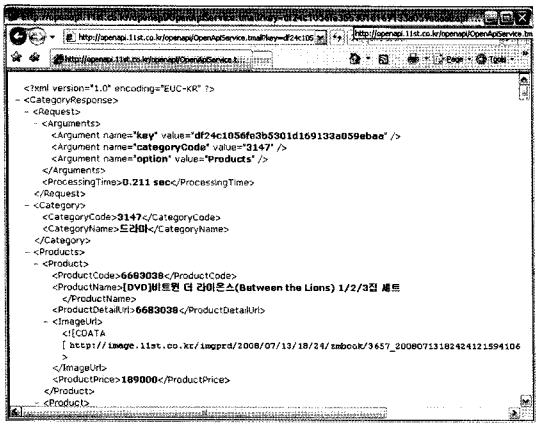
그림 8 서비스 중개자의 서비스 등록 화면

표 6 서비스 정의 및 입출력 명세

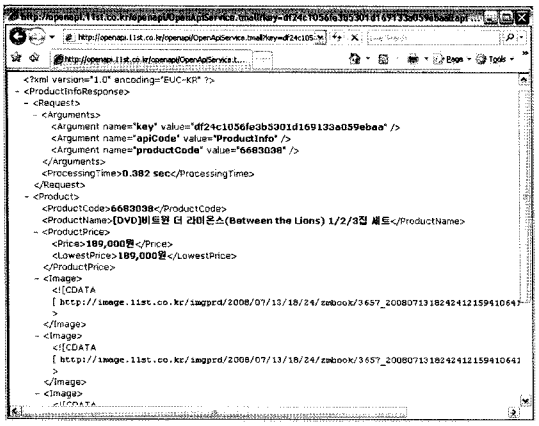
서비스 이름	입력	출력	선 조건	효과
Recommend Category Service	#feeling	#category	#validFeeling	#validCategory
11st Open Market CategoryInfoService	#category	#item	#validCategory	#validitem
11st Open Market ProductInfo Service	#product	#product #price #image #satisfaction	#validProduct	



(a) Recommend Category Service



(b) getItem Web service

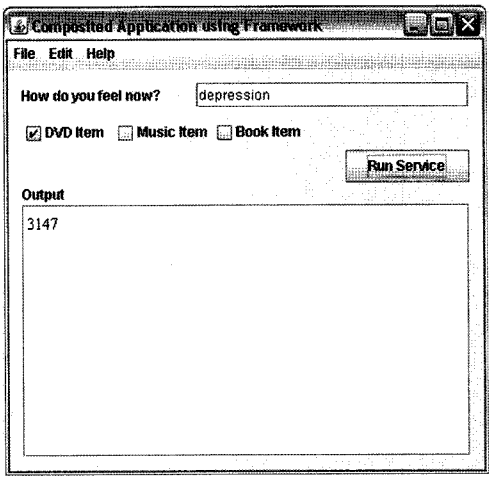


(c) 11st ProductSearch Service

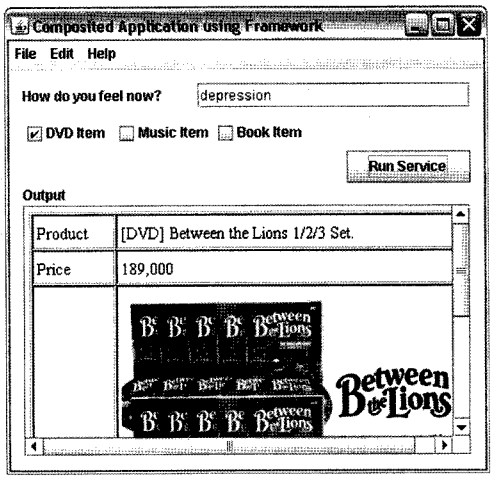
그림 9 웹 브라우저를 이용한 각각의 서비스 호출

그림 9는 프로토타입을 위해 외부에 존재하는 (a) 기분상태에 따른 카테고리 추천 웹 서비스, 오픈 마켓인 11번가 OpenAPI[1] 중에 (b) 카테고리에 따른 상품 코드 리스트를 출력하는 서비스와 (c) 상품 코드에 해당하는 상품 정보를 출력하는 서비스를 각각 웹 브라우저에서 보인다. 이들 서비스 모두가 서비스 등록 설명으로 표현되고 프레임워크 내부로 등록이 되면 표 2의 알고리즘에 의해 결합 가능한 합성 서비스를 만들어낸다.

그림 10은 본 프레임워크를 기반으로 서비스 중개자와 상호작용하는 애플리케이션 부분을 보여준다. (a)는 현재 기분상태에 따른 추천 카테고리를 출력해주는 단일 서비스만으로 구성된 상태이다. 이 후 서비스 중개자의 서비스 검색기에 입력을 기분 상태(Feeling), 출력을 상품 정보(Product, Price, Image)로 구성된 합성 서비



(a) 서비스 합성 이전



(b) 서비스 합성 후

그림 10 프레임워크의 동적 서비스 합성

1) http://openapi.11st.co.kr/

스를 요청하고 검색된 합성 서비스를 서비스 소비자 내부로 배포하고 수행시킨다. 그림 10의 (b)는 그림 9의 (a), (b), (c) 서비스가 합성되어 순서대로 호출된 후 최종 결과를 보여준다.

5. 토 의

매쉬업은 OpenAPI들의 구성으로 최근 비전문가들 사이에서 쉽게 만들어지고 있으며 그 응용범위는 대단히 넓다. 그러나 매쉬업은 비정형적인 형태로 만들어지고 검증에 대한 단서도 충분하지 못하다. 반면 SOA는 소프트웨어 개발의 새로운 패러다임으로서 정형적인 표현들과 유연한 확장 및 이기종간의 연결성이 좋고, 비용절감은 물론 재사용성을 극대화시킨다. 하지만 대부분 기업 내부의 자원에 국한되며 비즈니스 로직에 대한 서비스들을 대상으로 한다. 그래서 OpenAPI들의 매쉬업을 SOA에 적용한다면 양자의 장점을 모두 취할 수 있다. 또한 이러한 적용이 동적으로 수행된다면 개발자의 개입 없이 각각의 서비스 특징 또는 규칙만으로 다수의 합성된 서비스를 생산해야 하거나 또는 동적으로 적응해야 하는 소프트웨어를 위한 방법을 제공한다.

본 연구는 OpenAPI들의 매쉬업을 SOA환경에 통합하고 서비스 등록, 서비스 합성, 서비스 발견 그리고 동적인 서비스 재구성을 다룬다. OpenAPI를 정형화된 서비스로 포함시켜 이전 연구에서 시도되지 않은 이종 서비스에 대한 합성을 제공한다. 따라서 OpenAPI와 웹서비스의 동적 합성에 의해 기민한 애플리케이션 개발을 가능케 하고, 조직 내부에서 얻을 수 없는 정보를 활용할 수 있는 방법을 제공한다.

6. 결 론

본 논문은 OpenAPI들의 매쉬업을 SOA환경에 통합하고 서비스 등록, 서비스 합성, 서비스 발견 그리고 서비스 재구성을 다루는 프레임워크를 제안하였다. 본 연구에서는 동적 서비스 합성을 위하여 온톨로지를 활용함으로써 유연한 자동화 매칭을 제공한다. 또한 이 프레임워크에 대한 프로토타입을 만들고 사례 애플리케이션을 구축하여 유효성을 보였다. 본 연구는 비전문적인 OpenAPI를 정형적인 표현으로 통합하여 동적인 서비스 합성을 가능케 함으로써 빠른 서비스의 재창조를 위한 기회를 제공한다.

본 논문에서는 합성된 서비스 생성을 자동화된 코드에 의존하기 때문에 OWL-S의 Split, Split+Join, Any-Order, Choice, If-Then-Else, Iterate, Repeat-While 등과 같은 풍부한 프로세스 표현들을 만들어 내는데 한계가 있다. 향후 연구는 좀 더 유용한 서비스 합성을 생성하기 위해 수많은 서비스 소비자로부터 얻어지는 집단지성을 이용하는 서비스 합성과 합성된 서비스의 검증에 대한 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] X. Liu, Y. Hui, W. Sun and H. Liang, "Towards Service Composition Based on Mashup," *Proc. Of IEEE International Conference on Services Computing*, pp. 332-339, 2007.

[2] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo and T. Newling, *Patterns: Service-Oriented Architecture and Web-Services*, International Business Machines Corporation, Apr. 2004.

[3] T. O'Reilly, What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software, Sep. 2005.

[4] D. Merrill, Mashups: The new breed of Web app: An introduction to mashups, http://128.ibm.com/developerworks/xml/library/x-mashups.html.

[5] M. Sabbouh, J. Higginson, S. Semy and D. Gagne, "Web Mashup Scripting Language," *Proc. Of the 16th international conference on World Wide Web*, pp. 1305-1306, 2007.

[6] B. Raman and R. H. Katz, An architecture for highly available wide-area service composition, Computer Communications Journal, special issue on Recent Advances in Communication Networking, May 2003.

[7] D. Chakraborty and A. Joshi, Dynamic Service Composition: State-of-the-Art and Research Directions, Technical Report TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, 2001.

[8] T. andrews, F. Curbera, H. Dholakia, Y. Goland and F. Leymann, Business Process Execution Language for Web Services Version 1.1, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf, May 2003.

[9] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard and S. Pogliani, Web Service Choreography Interface (WSCI) 1.0, http://www.w3.org/TR/2002/NOTE-wsci-20020808/, Aug. 2002.

[10] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy and M. Shan, "Adaptive and Dynamic Service Composition in eFlow," *Proc. Of the International Conference on Advanced Information Systems Engineering*, 2000.

[11] P. Shankar and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition," *Proc. Of the Eleventh International World Wide Web Conference*, 2002.

[12] E. Sirin, B. Parsia and J. Hendler, "Composition-driven Filtering and Selection of Semantic Web Service," In AAAI Spring Symposium on Semantic

Web Services, 2004.
[13] N. Milanovic, M. Malek, "Current solutions for Web service composition," *Proc. Of IEEE International conference on Internet Computing*, Vol.8, Iss. 6, pp. 51-59, 2004.
[14] J. Ji-chen and G. Ming, "Enterprise Service Bus and an Open Source Implementation," *Proc. Of the International Conference on Management Science and Engineering*, 2006.
[15] I. B. Arpinar, B. Aleman-Meza, R. Zhang and A. Maduko, "Ontology-Driven Web Services Composition Platform," *Proc. Of the IEEE International Conference on E-Commerce Technology*, 2004.
[16] D. Caragea and T. Syeda-Mahmood, "Semantic API Matching for Automatic Service Composition," *Proc. Of the 13th international World Wide Web conference*, 2004.
[17] J. Cui, J. Liu, Y. Wu and N. Gu, "An Ontology Modeling Method in Semantic Composition of Web Services," *Proc. Of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, 2004.
[18] R. Gue, J. Le and X. Xia, "Capability Matching of Web Services Based on OWL-S," *Proc. Of the 16th International Workshop on Database and Expert Systems Applications*, 2005.
[19] J. Cardoso and A. Sheth, "Semantic e-Workflow Composition," *Journal of Intelligent Information Systems*, Vol.21, No.3, pp. 191-225, Nov. 2003.
[20] J. Kim, J. Lee and B. Lee, "Runtime Service Discovery and Reconfiguration Using OWL-S Based Semantic Web Service," *Proc. Of the 7th IEEE International conference on Computer and Information Technology*, pp. 891-896, Oct. 2007.
[21] A. Alishevskikh and G. Subbiah, SOFA: Simple Ontology Framework API, http://projects.semweb-central. org/projects/sofa.
[22] OWL-S API, maryland information and network dynamics lab semantic web agents project, http://www.mindswap.org/2004/owl-s/api/index.shtml.

김 진 한
2006년 서울시립대학교 졸업(학사). 2008년 서울시립대학교 졸업(석사). 2008년~현재 쌍용정보통신 개발지원팀. 관심분야는 소프트웨어 진화, 시맨틱 웹 서비스

이 병 정
1990년 서울대학교 계산통계학과(학사) 1998년 서울대학교 전산과학과(석사). 2002년 서울대학교 컴퓨터공학부(박사). 1990년~1998년 현대전자 SW연구소. 2002년~현재 서울시립대학교 컴퓨터과학부 부교수. 관심분야는 소프트웨어 진화, 개발 방법론, 소프트웨어 품질 등