ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

다양한 Open API 타입들을 지원하는 시맨틱 기반 매쉬업 개발 툴☆

A Semantic-Based Mashup Development Tool Supporting Various Open API Types

이 용 주*
Yong-Ju Lee

요 약

최근에 매쉬업은 미래 IT 융합 서비스의 효과적인 구현 방법으로써 그 관심도가 점점 높아지고 있으며 그들의 활용도 매우 다양하다. 그렇지만 이러한 높은 관심에도 불구하고 Open API들을 매쉬업 속으로 결합할 때 여러 가지 이슈들이 있을 수 있다. 첫째, 포털 사이트들은 매쉬업에서 사용 가능한 수많은 API들을 제공하고 있는데, 이들에 대한 적합한 API들을 수동으로 탐색하고 발견하는 것은 매우 힘들고 많은 시간이 소비되는 작업이다. 둘째, 현존하는 어떠한 매쉬업 포털 사이트들도 전통적인 SOAP 기반 웹 서비스 분야에서 보였던 것처럼 API들을 찾고 통합하는데 시맨틱 기법을 활용하는 사이트는 없다. 세째, 적합한 API들을 발견하였더라도 특별한 기술적 훈련 없이 값어치 있는 매쉬업을 생성하기란 현실적으로 어려운 일이다. 본 논문에서는 위와 같은 이슈들을 해결하기 위해 먼저 기존의 SOAP 기반 웹 서비스 분야에서 사용된 시맨틱 기반 기술 및 알고리즘들을 최소의 수정만으로 재사용할 수 있음을 보인다. 다음으로, 조합 가능한 API들을 발견하기 위해 어떻게 API 특성들이 신택틱하게 정의되고 시맨틱하게 묘사될 수 있는지 보인다. 그러고 이러한 신택틱/시맨틱 정보들이 어떻게 Open API들의 발견과 조합에 도움을 줄 수 있는지 보인다. 마지막으로, 동적 Open API 조합을 위한 대화형 목표 지향 접근 방법을 제안한다. 여기서 최종 매쉬업은 API들의 순차적 접근 방법에 의해 점차적으로 각 단계에서 하나씩 새로운 API가 조합에 첨가된다.

ABSTRACT

Mashups have become very popular over the last few years, and their use also varies for IT convergency services. In spite of their popularity, there are several challenging issues when combining Open APIs into mashups, First, since portal sites may have a large number of APIs available for mashups, manually searching and finding compatible APIs can be a tedious and time-consuming task. Second, none of the existing portal sites provides a way to leverage semantic techniques that have been developed to assist users in locating and integrating APIs like those seen in traditional SOAP-based web services. Third, although suitable APIs have been discovered, the integration of these APIs is required for in-depth programming knowledge. To solve these issues, we first show that existing techniques and algorithms used for finding and matching SOAP-based web services can be reused, with only minor changes. Next, we show how the characteristics of APIs can be syntactically defined and semantically described, and how to use the syntactic and semantic descriptions to aid the easy discovery and composition of Open APIs. Finally, we propose a goal-directed interactive approach for the dynamic composition of APIs, where the final mashup is gradually generated by a forward chaining of APIs. At each step, a new API is added to the composition.

☞ keyword : 매쉬업(mashup), 오픈 API(Open API), 시맨틱 온톨로지(semantic ontology), 대화형 매쉬업 조합 시스템(interactive mashup composition system), 유사성 탐색(similarity search), API 조합(API composition)

1. 서 론

Open API(Application Program Interface)는 웹 2.0의 근본 개념인 데이터의 개방 및 공유를 구현할 수 있는 핵심 기술로 자사의 API를 웹 서비스를 통해 외부로 공개한 것을 말한다. 매쉬업(mashup)은 이러한 공개된 Open API를 이용하여 두 가지 이상의 서로 다른 자원을 섞어서 완전히 새로운 가치의 서비스를 만드는 것이다. 매쉬업이란 단어를 누가 먼저 언급했는지는 아직도 잘 알려져 있지 않지만 HousingMaps[1]는 매쉬업이 대중에 널리 알려진 계기가 되었다. HousingMaps는 부동산 정보를 제공하는

* 정 회 원 : 경북대학교 과학기술대학 컴퓨터정보학부 교수
yongju@knu.ac.kr
☆ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(No. 2010-0008303)
[2012/02.08 투고 - 2012/02/14 심사 - 2012/06/04 심사완료]

CraigsList와 지도를 제공하는 구글 맵 Open API를 조합해 지도 위에 부동산 정보를 표현하는 매쉬업이다.

최근에 매쉬업은 미래 IT 융합 서비스의 효과적인 구현 방법으로써 그 관심도가 점점 높아지고 있으며, 그들의 활용도 엔터프라이즈 비즈니스 분야로부터 과학적 e-사이언스 분야에 이르기 까지 매우 다양하다. 그렇지만 이러한 높은 관심에도 불구하고 Open API들을 매쉬업 속으로 결합할 때 여러 가지 이슈들이 있을 수 있다. 특히, Open API들은 SOAP, REST, JavaScript, XML-RPC, Atom 등 다양한 프로토콜로 제공되므로 API들이 이런 다양한 프로토콜을 사용하고 있을 때 이는 더욱 심각해진다. 이러한 문제점들을 정리하면 다음과 같다.

첫 번째로, 대부분의 Open API 포털 사이트들은 매쉬업에서 사용 가능한 수많은 API들을 제공하고 있는데, 이들에 대한 조합 가능한 API들을 탐색하고 발견하는 것은 매우 힘들고 많은 시간이 소비되는 작업이다. 예를 들면, Open API와 매쉬업에 대한 대표적인 포털 사이트인 programmableWeb[2]은 2012년 1월 26일 현재 4913개의 Open API들을 제공하고 있는데, 이 사이트에서는 이용 가능한 API들을 발견하기 위해 단지 키워드 검색과 카테고리 검색만 지원하고 있다. 키워드 검색은 이미 여러 연구에서 밝혀진 바와 같이 나쁜 재현율과 나쁜 정확률 때문에 문제가 많으며, 카테고리별 검색 결과도 매쉬업 개발자에게는 별로 관심이 없는 알파벳 순서나 최근 개발된 날짜 순서로만 정렬되어 있어 원하는 결과를 찾는데 쉽지 않다. 통상 매쉬업 개발자들은 API들을 신속히 발견하기를 원하며 큰 프로그래밍 노력 없이 이들을 쉽게 결합할 수 있기를 바란다.

두 번째로, 현존하는 어떠한 매쉬업 포털 사이트들도 전통적인 SOAP 기반 웹 서비스 분야에서 보였던 것처럼 API들을 찾고 통합하는데 시맨틱 기법을 활용하는 사이트는 없다. SOAP 기반 웹 서비스들과는 달리 REST, JavaScript, XML-RPC, 그리고 Atom API들은 그들의 인터페이스를 명시하기 위한 WSDL(Web Service Description Language)을 사용하지 않는다. 더군다나 이런 API들을 정의하거나 분류하기 위한 어떠한 표준도 논의된 바 없고 단지 개발자들이 수동적으로 웹을 탐색하여 원하는 API들을 찾고 있다. 이는 API 발견 및 조합을 위한 쉽고 체계적인 접근 방법이 될 수 없다. 어떤 API들은 사용되는 프로토콜의 특수성으로 인해 매쉬업 애플리케이션에 적용하기에 부적합할 수 있으므로, 이런 경우 적합한 API들을 발견하기 위해 계속적으로 시행착오를 반복할 수 있다. 특히, API들이 다양한 프로토콜을 사용하고 있을 때 이런 현상은 더욱 문제가 된다.

세 번째로, 적합한 API들을 발견하였더라도 특별한 기술적 훈련 없이 값어치 있는 매쉬업을 생성하기란 현실적으로 어려운 일이다. 매쉬업을 만들기 위해서는 숙달된 프로그래밍 기술뿐만 아니라 매쉬업에 포함되는 모든 API들에 대해 깊이 파악하고 있어야 하기 때문이다. 매쉬업 개발자들은 최소한의 훈련으로 API들을 조합할 수 있어야 하고 이러한 작업을 위해 어려운 프로그래밍 기술이 요구되지 말아야 한다. 더욱 어려운 문제는 동적으로 API들을 조합할 때인데, 동적 API 조합은 자동적으로 API들의 능력(즉, 그들이 무엇을 할 수 있는가)을 이해 할 수 있어야 하고 그들 사이의 호환성을 동적으로 파악할 수 있어야 하기 때문이다.

본 논문에서는 위와 같은 이슈들을 해결하기 위해 먼저 기존의 SOAP 기반 웹 서비스 분야에서 사용된 시맨틱 기반 기술 및 알고리즘들을 최소의 수정만으로 재사용할 수 있음을 보인다. 본 논문에서 다루는 API들은 SOAP뿐만 아니라 REST, JavaScript, XML-RPC, 그리고 Atom API들을 포함하기 때문에 기존의 연구와는 상당한 차이가 있다. 다음으로, 조합 가능한 API들을 발견하기 위해 어떻게 API 특성들이 신택틱(syntactic)하게 정의되고 시맨틱(semantic)하게 묘사될 수 있는지 보인다. 그러고 이러한 신택틱/시맨틱 정보들이 어떻게 Open API들의 발견과 조합에 도움을 줄 수 있는지 보인다. 마지막으로, 동적 Open API 조합을 위한 대화형(interactive) 목표 지향 접근 방법을 제안한다. 여기서 최종 매쉬업은 API들의 순차적 접근 방법에 의해 점차적으로 각 단계에서 하나씩 새로운 API가 조합에 첨가된다.

본 논문의 구성은 다음과 같다. 2장에서 매쉬업에 관한 관련연구를 간단히 살펴보고 3장에서 Open API 특성들을 분석한다. 4장에서 시맨틱 온톨로지 구축 방법을 설명하고 5장에서 대화형 매쉬업 조합 시스템을 기술한다. 그리고 6장에서 실험 분석을 수행하고 7장에서 결론을 내린다.

2. 관련 연구

매쉬업에 대한 깊은 관심으로 지금까지 많은 연구들이 수행되어 왔으나, 이들 기술들과 개발 툴들은 주로 개인용과 기업용으로 크게 구별할 수 있다. 예를 들면, Pipes[3], Popfly[4], MashMaker[5]는 개별적인 개인사용을 목적으로 하고 있으며, Damia[6], MARIO[7], UQBE[8] 등은 주로 기업 인트라넷 환경에서 기업 비즈니스 생산성

향상을 목표로 하고 있다.

먼저 개인용 매쉬업 개발 툴들을 살펴보면, 야후의 Pipes[3]는 drag & drop 방식으로 인터넷 데이터 소스를 연결하고 흐름을 재구성하도록 함으로써 매쉬업 아이디어를 일반화한 비주얼 툴이다. Pipes는 몇 번의 클릭과 간단한 단어 입력만으로 사용자가 원하는 웹페이지를 프로그래밍 순서도 짜듯이 직관적으로 만들 수 있는 장점이 있다. 하지만 Pipes는 시각적 매쉬업 작성 툴을 처음으로 제시한 제품이지만 서비스 검색 기능이 한정적이고 개발자들이 미리 필요한 모듈들을 설계해 두어야 매쉬업이 가능하다. 또한 파이프의 흐름이 단순히 정적이고 순차적이다.

마이크로소프트의 Popfly[4]는 매쉬업을 손쉽게 할 수 있도록 실버라이트(Silverlight)를 이용하여 만들어진 웹 사이트이다. Popfly에서는 매쉬업을 위하여 Open API 검색과 매쉬업 가능한 서비스 검색을 제공하며, 상당히 많은 기능들이 이미 블록으로 만들어져 있어서 블록들을 선택하여 연결만 해주면 새로운 매쉬업을 만들 수 있다. 하지만 Popfly에서는 제한된 키워드 기반 매쉬업 검색만 제공하고, 아직 시맨틱 개념이 지원되지 않아 순위화된 유사 서비스 탐색 방법이 없다.

인텔의 MashMaker[5]는 사용자가 개별화된 매쉬업을 생성할 수 있도록 지원하는 브라우저의 부가기능이다. 사용자가 웹을 탐색하는 동안에 브라우저의 툴바에 매쉬업을 추천해 주고, 추천된 매쉬업을 선택하면 현재 보고 있는 웹 페이지에 관련된 정보가 추가된 매쉬업이 생성된다. 하지만 MashMaker는 매쉬업을 위한 편의는 도모하나 본 연구와 같은 시맨틱 기법을 활용한 매쉬업의 자동화에 관한 내용은 아니다. 또한 매쉬업 엔진이 브라우저 속에 삽입된 브라우저 확장판으로써 위젯 기반의 웹 애플리케이션 개발 저작도구는 아니다.

다음으로 기업용 매쉬업 개발 툴들을 살펴보면, IBM의 Damia[6]는 야후의 Pipes에서 취급하는 URL 기반 소스뿐만 아니라 Excel, Notes, 웹 서비스, 그리고 XML 문서와 같은 기업 형태의 데이터 소스까지 확장하였다. Damia에서는 3가지 중요한 오퍼레이션: 섭취(ingestion), 확장(augmentation), 그리고 출판(publication)을 제공한다. 섭취는 데이터 소스로부터 데이터를 시스템 안으로 가져오고, 확장은 새로운 매쉬업 오퍼레이터들의 생성을 허용하여 Pipes 오퍼레이터보다 더욱 강력한 확장성을 제공한다. 마지막으로 출판은 RSS, Atom, JSON과 같은 일반적인 출력 포맷으로 변환한다. Damia의 취약점으로는 사용할 수 있는 서비스의 종류가 한정되어 있으며, Open API의 탐색 및 선택은 고려하지 않는다.

MARIO[7]는 일반 사용자가 쉽게 데이터 매쉬업을 개발할 수 있는 태그(tag) 클라우드를 이용한 매쉬업 개발 도구이다. MARIO는 계층 구조를 갖는 태그로 매쉬업 컴포넌트들을 모델링하고, 태그 클라우드를 이용한 반복적 상호작용을 통해 사용자가 매쉬업 목적을 구체화할 수 있도록 한다. 이 과정에서 “wishful search”라 불리는 추상화를 통해 사용자 목적에 가장 적합한 매쉬업을 자동으로 생성하는 경량 플래너(planner)를 제공한다. UQBE[8]는 자동화된 스키마 매칭을 기반으로 최적의 매쉬업 결과를 리턴한다. 이러한 환경에서 분산된 데이터 소스들이 마치 릴레이션 테이블 조인처럼 공통적인 속성들에 의해 결합된다. 제안된 솔루션은 QBE(Query-By-Example) 원리에 의해 비-프로그래머들이 쉽게 접근할 수 있는 장점이 있다. 하지만 MARIO와 UQBE는 특정 도메인에 의존적이며 다른 상황의 애플리케이션에 재활용되기 어려운 단점이 있다.

3. Open API 특성 분석

현재 programmableWeb은 Open API 포털 사이트들 중 가장 유명한 사이트이기 때문에 이에 대한 분석은 최근의 Open API 현황을 대변할 수 있다. (표 1)은 programmableWeb에서 제공하고 있는 API들에 대한 프로토콜 타입의 분포를 보여주고 있다. 표에서 알 수 있듯이 제공되는 API의 대부분(72%)이 REST이고, 18%가 SOAP, 6%는 JavaScript, 2%가 XML-RPC, 그리고 Atom은 0%다. 따라서 본 논문에서 취급하는 Open API 프로토콜 타입은 REST, SOAP, JavaScript, 그리고 XML-RPC로 한정하며, 본 장에서 각 프로토콜별 API 특성들을 분석한다.

(표 1) Open API 프로토콜 타입 분포

API Type	%
REST	72
SOAP	18
JavaScript	6
XML-RPC	2
Atom	0

3.1 REST API

REST(Representational State Transfer)는 웹의 창시자 중 한 사람인 Roy Fielding의 박사학위 논문[9]에 의해 소개

되었다. 그는 현재의 웹 아키텍처가 웹이 지닌 본래의 설계 우수성을 충분히 활용하지 못하고 있다고 판단하고, 웹의 장점을 최대한 활용할 수 있는 네트워크 기반의 아키텍처를 제안했는데 그것이 바로 REST다. 이런 REST 아키텍처 스타일에 따라 정의되고 이용되는 Open API를 REST API라 한다. REST API의 가장 큰 특징 중의 하나는 모든 대상을 리소스(resource), 즉 자원으로 표현한다는 것이다. 이 리소스는 HTTP URI(Uniform Resource Identifier)에 의해 접근된다. REST 구조에서의 리소스는 HTTP의 기본 메소드인 POST, GET, PUT, DELETE만으로 접근할 수 있다. 리소스에 접근하기 위한 이러한 4개의 HTTP 메소드는 일반 CRUD(Create, Read, Update, Delete) 오퍼레이션에 각각 대응될 수 있다. 또한 HTTP의 기본 메소드로 전달되는 리소스는 다양한 방식으로 표현(representation)될 수 있는데, 이는 XML, JSON, HTML, 텍스트, 이미지 등이 가능하며 클라이언트에서 원하는 형식으로 표현하면 서버에서 이를 처리하게 된다. 마지막으로 REST API는 스테이트리스(stateless) 특성을 가진다. 스테이트리스란 웹 서비스 제공 서버 측에서 클라이언트의 상태(state) 정보를 저장, 관리하지 않는 것을 의미한다.

이상과 같이 REST API는 리소스의 URI만 알면 SOAP 기반 웹 서비스와 같은 부가적인 전송 레이어 없이 HTTP 프로토콜만으로 접근 가능한 아주 간단한 서비스라 할 수 있다. 이러한 단순 명료한 접근 방식 때문에 구글, 야후, 트위터 등에서 제공하는 대부분의 웹 2.0 Open API들이 REST API 방식으로 작성되어 있으며, 이는 위젯(widget)을 이용한 매쉬업을 활성화시킨 원동력이 되었다. SOAP 기반 웹 서비스에 비해 가볍고 구현하기 쉬운 REST API가 많은 주목을 받고 있음에 따라, REST API의 인터페이스를 기술하기 위한 다양한 방법들이 제안되고 있다. 기존의 WSDL에서도 2.0 버전에서는 REST API를 기술할 수 있는 HTTP 바인딩(binding) 확장 스펙이 제안되었지만 너무 복잡하다는 이유로 많이 쓰이지는 못하고 있다. 이에 썬마이크로시스템은 간략하면서도 범용성이 뛰어난 WADL(Web Application Description Language)을 발표하게 되었고, 간단하다는 장점 때문에 개발자들 사이에서 WADL의 사용은 점차 증가되고 있는 실정이다.

3.2 SOAP API

REST API와는 달리 SOAP API는 리소스를 엑세스하기 위해 직접 HTTP 메소드를 사용하지 않는다. SOAP API에서는 제공할 오퍼레이션들을 각각 정의하고 필요한 기능을 수행할 때 해당 오퍼레이션을 호출하는 방식으로 일반 프로그래밍 개념과 동일하다. SOAP 기반 웹 서비스는 느슨하게 연결된(loosely coupled) 컴포넌트들의 집합을 통하여 서비스들을 구성하기 때문에 서비스를 조합 생성할 때 뛰어난 재사용성과 확장성을 보여준다. 또한 W3C 표준에 의거한 SOAP, WSDL, 그리고 WS-* 와 같은 표준을 활용함으로써 보다 더 비즈니스 프로세스에 집중할 수 있도록 지원한다. 그러나 표준의 범위가 계속해서 확대됨에 따라 점차 복잡해지게 되었으며, 이에 따라 너무 많은 표준에 의해 제약 사항이 증가하였고 개발의 복잡성도 증가하는 문제점을 초래하였다.

REST API가 인터넷 서비스 업체들로부터 응용 개발자들에게 손쉬운 데이터 제공을 목적으로 출발한데 비해 SOAP API는 기업의 비즈니스 환경에서 응용 서비스 간 상호 운용을 목적으로 시작되었다. 따라서 SOAP API들은 서비스를 제공하고 이용하는 프로그램(기계)들이 잘 이해할 수 있도록 엄격한 문법에 따라 개발되었기 때문에 개발자들에게는 API 기본 스펙을 잘 알아야 하는 비교적 고난이도의 프로그래밍 능력이 요구되었다. 따라서 SOAP API를 활용한 응용 서비스의 개발 편의성을 도모하기 위해 다양한 개발 환경들이 지원되고 있다. 반면 REST API는 기계 보다는 사람이 이해하기 쉽도록 하기 위해 인터넷 기본(HTTP와 XML) 이외에는 별도의 개발 및 실행 환경을 필요로 하지 않는다. 한편 SOAP API는 서비스 기술문서인 WSDL에 의해 서비스 인터페이스와 입/출력 메시지 형식 등이 기술되어 있다.

3.3 JavaScript API

JavaScript API를 생성하기 위해서는 SOAP API를 생성할 때보다 몇 가지 추가적인 고려 사항이 요구된다. 일반적으로 JavaScript API를 구현할 때 첫 번째 단계는 GUI 기술을 결정하는 것이다(이는 보통 HTML 기반이 될 수 있다). 두 번째 단계는 GUI 내에 표현되는 데이터를 추출하는 것이다(이러한 정보는 관계형 데이터베이스의 레코드들로 구성된다). 세 번째 단계는 API 입/출력을 처리하기 위한 프로그래밍 코드를 작성하는 것이다. 여기까지는 JavaScript나 SOAP API를 생성하는데 큰 차이가 없다. 그러나 JavaScript API를 생성하기 위해 추가로 고려하여야 하는 첫 번째 단계는 공개되는 오퍼레이션과 입/출력을 결정하는 것이다. 두 번째 추가되는 단계는 이러한 오퍼레이션과 입/출력을 기술하는 WSDL 파일을 생성하는 것이다. 현재 널리 사용되고 있는 웹 서비스 개발 툴킷을 이

용하면 이러한 WSDL 파일의 자동 생성은 쉽게 수행될 수 있다[10].

3.4 XML-RPC API

RPC(Remote Procedure Call)는 분산 컴퓨터 환경에서 이기종 컴퓨터 자원을 사용하는 기술이다. RPC는 원격의 컴퓨터도 마치 자신의 컴퓨터에 존재하는 함수를 호출하는 것처럼 프로그램할 수 있게 만들어 준다. XML-RPC는 이 개념을 웹으로 옮겨온 것으로 프로그램에서 함수 호출을 하듯이 원격에 있는 사이트에 정보를 요청하고 받아올 수 있게 해 준다. 이때 주고받는 인자와 리턴 값은 XML로 인코딩되고, 실제로 데이터를 전송하는 수송 수단으로는 범용적인 HTTP(POST)를 사용한다.

XML-RPC는 SOAP보다는 사용하고 이해하기에 간단하다. 왜냐하면, XML-RPC는 단지 단방향 메소드 전송만 제공하는데 반해 SOAP은 다수의 다른 인코딩을 정의한다. 또한 XML-RPC는 SOAP보다 더욱 간단한 보안 모델을 가지고 있으며, XML-RPC는 WSDL의 생성을 요구하지 않는다. 하지만 WSDL에 의해 제공되는 기능은 WSDL의 서브셋인 XRDL(XML-RPC Description Language)에 의해 제공될 수 있다. 한편, SOAP API는 XML-RPC API보다 복잡하기는 하지만 XML-RPC의 원조로서 기능은 더욱 강력한 면이 있다.

4. 시맨틱 온톨로지 구축 방법

3장에서 분석된 바와 같이 REST, SOAP, JavaScript, 그리고 XML-RPC API들은 공통적으로 일반 프로그래밍과 같은 입력과 출력을 가지고 있다. 이러한 프로그램적인 입/출력은 Open API에 대한 사용하기 쉽고 직관적인 방법은 제공하지만 이들의 단점도 신중히 고려하여야 한다. 일반적으로 입/출력 매개변수(parameter)들은 잘 통제되고 관리된 어휘만 사용하는 것이 아니라 개발자의 자유롭고 임의적인 판단에 의해 용어들이 선택되어 지므로 이러한 모호성으로 인해 매칭되는 API들 간의 불일치가 야기될 수 있다. 또한 이러한 매개변수들은 의미적으로 계층구조를 형성하지 못하고 단지 일차원적으로 나열되어 있는 형태이므로 수많은 매개변수들이 존재할 때 이들 중 호환되는 API들을 찾는 것은 매우 어려운 문제이다.

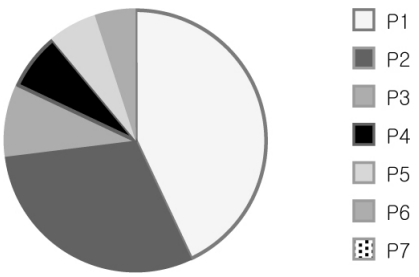
본 논문에서 사용되는 시맨틱 온톨로지 구축 방법은 이미 우리가 제안하였던 Open API 온톨로지 구축 방법[11]을 기반으로 수행된다. 이 방법의 핵심 내용은 Open API를 개발할 때 자동으로 생성되는 WSDL/WADL/XRDL 문서만 가지고 항목 간 숨어 있는 시맨틱 정보를 찾아내어 온톨로지를 자동 구축하는 것이다. 이를 위해 입/출력 항목들로부터 의미적으로 같은 개념들을 묶고(clustering), 각 항목들 간의 계층관계(hierarchical relationship)를 형성하여 자동적으로 시맨틱 온톨로지를 구축한다. 본 방법에서는 두 가지 기법(즉, 계층적 결합 클러스터링과 계층관계 형태소 분석 기법)이 소개되는데 이에 대한 간단한 설명은 다음과 같다.

4.1 계층적 결합 클러스터링

계층적 결합 클러스터링(hierarchical agglomerative clustering) 기법에서는 WSDL/WADL/XRDL 파일에 존재하는 신텍틱 정보를 이용하여 그들 사이에 숨겨져 있는 시맨틱 개념(concept)을 얻기 위해 마이닝(mining) 알고리즘을 적용한다. 주된 아이디어는 용어(term)들의 발생 빈도수를 측정하여 용어들을 개념들의 집합 속으로 클러스터(cluster)하는 것이다. 이러한 개념 클러스터는 Open API 사이의 유사성을 결정하는데 사용된다.

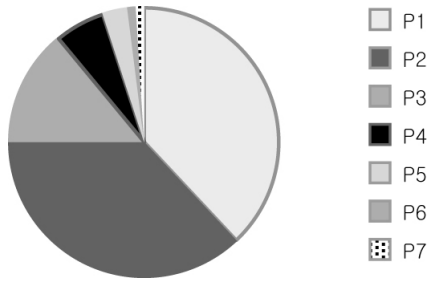
우선, 매개변수들을 토큰화하여 용어들로 분리한 후, 관련성이 많은 용어들에 대해 클러스터를 형성하면 이 클러스터는 각각의 단어가 아닌 하나의 의미 있는 개념을 나타낸다. 이러한 클러스터는 매개변수들이 동시에 자주 나타난다면, 그것들은 같은 개념을 나타내는 경향이 있다는 가정 하에 하나의 특별한 연관규칙(association rules)[12]에 따라 만들어 진다. 연관규칙은 용어 A가 일어나면 용어 B가 일어난다는 의미로 $A \Rightarrow B$로 표현될 수 있으며, 여기서 지지도(support)와 신뢰도(confidence)는 해당 규칙이 얼마나 유용한지를 나타내는 지표로서 사용된다. 지지도는 용어 A와 B를 동시에 포함하는 트랜잭션의 확률을 표현하며, 신뢰도는 A가 주어졌을 때 B가 동시에 나타날 트랜잭션의 확률을 나타낸다.

본 논문에서는 용어들의 집합을 시맨틱 개념들의 집합으로 전환하기 위해 계층적 결합 클러스터링 알고리즘을 사용한다. 이 알고리즘은 Apriori 알고리즘[13]의 연관규칙 탐사 결과에서 먼저 신뢰도를 내림차순으로 정렬한 다음 지지도를 내림차순으로 정렬한 후 각 단계에서 가장 최상위에 있는 규칙을 조사하여 만일 두 용어가 다른 클러스터에 속하면 이들을 결합한다. 결합하는 과정에서 가장 이상적인 클러스터를 형성하기 위하여 다음의 두 가지 클러스터 특성을 고려한다. 즉, 클러스터의 응집력(cohesion)-한 클러스터 내의 용어들과의 응집력-은 높게



No	Pattern	Occurrence
P1	Noun	3548 (43%)
P2	Noun1+Noun2	2435 (30%)
P3	Adjective+Noun	752 (9%)
P4	Verb+Noun	608 (7%)
P5	Noun1+Noun2+Noun3	472 (6%)
P6	Noun1+Preposition+Noun2	368 (5%)
P7	others	26 (0.3%)

(그림 1) REST API의 패턴 분석



No	Pattern	Occurrence
P1	Noun	485 (38%)
P2	Noun1+Noun2	470 (37%)
P3	Noun1+Noun2+Noun3	175 (14%)
P4	Verb+Noun	70 (6%)
P5	Noun1+Preposition+Noun2	40 (3%)
P6	Adjective+Noun	15 (1%)
P7	others	14 (1%)

(그림 2) SOAP API의 패턴 분석

하고, 클러스터 간의 연관성(correlation)-다른 클러스터 용어들 간의 상호관계-는 낮게 한다. 최종적으로 전체적인 클러스터 품질을 측정하기 위한 클러스터링 점수는 (응집력)/(연관성) 방식으로 계산되며, 이때 우리의 목표는 가장 높은 점수를 갖도록 클러스터를 형성하는 것이다. 이러한 과정은 유사한 클러스터를 병합한 후 새로 계산된 점수가 이전보다 더 높은 점수를 갖는지 조사하고, 더 이상 높은 점수를 얻을 수 없을 때까지 이 과정을 반복한다.

4.2 계층관계 형태소 분석

계층관계 형태소 분석 방법은 매개변수 내에 포함된 용어들 간의 상관관계를 취득하고, 만일 두 용어들이 서로 유사하며 상관관계가 조건에 일치한다면 그 매개변수를 매치하는 것이다. 이러한 접근방법은 사람들이 다수의 용어들을 가지고 매개변수를 만들 때 일반적으로 비슷한 패턴을 사용한다는 관찰로부터 유도되었다. 이러한 패턴들을 조사하기 위해 본 논문에서는 Open API의 대부분을 차지하는 REST, SOAP, JavaScript, XML-RPC API들에 대해 실험 데이터를 만들어 분석하였다. 하지만 JavaScript는 WSDL 파일을 생성하므로 SOAP API에 포함시킬 수 있고, XML-RPC는 (표 1)에서 알 수 있듯이 차지하는 비율이 낮아 분석에서 생략하였다.

이들 실험 데이터에 POS(part-of-speech) 형태소 분석기를 적용시킨 결과 (그림 1)과 (그림 2)와 같은 형태를 취하였다. 분석 결과 REST API에서는 단지 하나의 토큰으로 구성된 매개변수(예, City)가 전체의 43%로 가장 많았고, 명사+명사(30%), 형용사+명사(9%), 동사+명사(7%), 명사+명사+명사(6%), 명사+전치사+명사(5%), 그리고 기타(0.3%) 순으로 나타났다. SOAP API에서는 단지 하나의 토큰으로 구성된 매개변수는 38%를 차지하였으며, 명사+명사(37%), 명사+명사+명사(14%), 동사+명사(6%), 명사+전치사+명사(3%), 형용사+명사(1%), 그리고 기타(1%) 순으로 나타났다. 본 결과로부터 특이한 사항은 다른 프로토콜을 사용하거나, 다른 도메인을 선택하더라도 매개변수 패턴은 단지 출현 빈도의 순위에 다소간의 변동이 있을 뿐 도출된 5개의 패턴 종류는 동일한 사실을 알 수 있다.

따라서 (그림 1)과 (그림 2)의 관찰로부터 매개변수에 대한 온톨로지 변환 규칙은 다음과 같이 P1과 P7을 제외한 5개의 규칙으로 정할 수 있다. 일반적으로 온톨로지 개념은 속성(property)과 상-하위(subclass) 관계로 표현되므로 본 연구에서는 각 용어들 간의 속성 및 상-하위 관계만 중점적으로 취급하였다.

- 규칙1(Noun1+Noun2):
  Parameter **propertyOf** Noun1
- 규칙2(Adjective+Noun):
  Parameter **subClassOf** Noun
- 규칙3(Verb+Noun):
  Parameter **subClassOf** Noun

- 규칙4(Noun1+Noun2+Noun3):
  Parameter **propertyOf** Noun1
- 규칙5(Noun1+Preposition+Noun2):
  Parameter **propertyOf** Noun2

위와 같은 규칙을 사용하여 온톨로지가 구축되고 나면, 다음 단계는 질의문에 의해 두 개념 간 매칭을 시키는 것이다. 즉 두 개의 온톨로지 개념이 다음 조건을 만족하면 매치된다: (1) 어떤 개념이 다른 개념의 속성일 경우(예, CompanyID **propertyOf** Company), (2) 어떤 개념이 다른 개념의 자식관계인 경우(예, virtualAccount **subClassOf** Account).

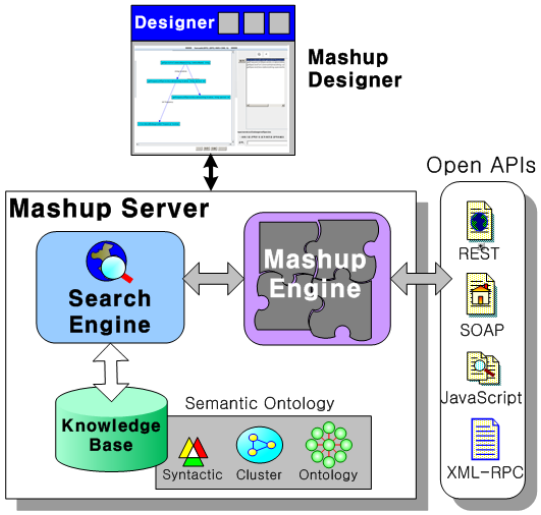
5. 대화형 매쉬업 조합 시스템

본 논문에서는 동적 Open API 조합 시스템을 구현하기 위해 대화형(interactive) 목표 지향 접근 방법을 제안한다. 즉, 워크플로우의 시작 단계에서부터 점차적으로 하나씩 새로운 API들이 추가된다. 각 단계에서 새로운 API를 워크플로우에 첨가시킬 때 사용자는 먼저 질의 Q를 생성한다. 일단 Q가 생성되면 이는 API 검색 엔진으로 보내지고, Q와 기존 API들 간의 유사도 측정에 따라 우선순위가 매겨진 API들이 반환된다. 이때 사용자는 그가 의도한 바를 가장 잘 이룰 수 있는 적합한 API를 하나 선택한다.

워크플로우를 표현하는 가장 자유스러운 방법은 사이클이 없는 방향성 그래프(DAG: Directed Acycle Graph) 방식이다. 여기서 정점(vertex)은 오퍼레이션을 표현하고 간선(edge)은 데이터 흐름을 나타내며 이 그래프는 *DAG=(V, E)*, 즉 정점들의 집합 V와 간선들의 집합 E로 구성된다. 구현된 디자이너는 drag & drop 방식으로 워크플로우를 작성함으로써 사용자에게 고수준의 인터페이스를 제공한다. 디자이너는 마법사처럼 만들어져 있으며, 여러 판넬에 의해 다음/이전 화면을 보여줄 수 있다. 사용자로부터 질의 Q가 주어지면 5.1절에서 설명되는 조합 가능한 API 발견 알고리즘을 적용하여 이용 가능한 API 리스트를 보여주고, 이들 중 가장 적합한 API를 선택한다. 이러한 방식으로 점차적으로 하나씩 새로운 API들이 첨가되고 나면 최종적으로 워크플로우가 완성된다. (그림 3)은 전체적인 대화형 매쉬업 조합 시스템의 구조를 설명하고 있다.

5.1 조합 가능한 API 발견 알고리즘

본 절에서는 시맨틱 온톨로지를 활용하여 조합 가능한



(그림 3) 대화형 매쉬업 조합 시스템

API들을 발견할 수 있는 알고리즘을 구현하였다. 이 알고리즘은 OWL-S Matchmaker[14]에서 구현된 것과 비슷하다. 조합 가능한 API 발견 알고리즘의 기본적인 원리는 질의의 입력항목을 사용하여 원하는 출력을 산출해 낼 수 있는 API들을 찾는 것이다. 이를 위해서 선택되는 API는 반드시 질의의 출력항목을 포함하고 있어야만 하고, 이 API의 입력항목들은 질의의 입력항목에 포함되어 있어야만 한다.

정의: 질의 Q의 모든 출력항목들이 API O의 출력항목들과 매치가 되고, 이 O의 모든 입력항목들이 Q의 입력항목들과 매치가 될 때, 이 O와 Q는 매치된다고 할 수 있다.

위의 정의는 매치되는 O가 Q의 모든 요구사항을 만족할 수 있고, 이 O를 올바로 작동시키기 위해 필요한 모든 입력항목들은 Q가 제공할 수 있다는 것을 보장한다. 이를 기반으로 한 조합 가능한 API 발견 과정은 (그림 4)와 같다. 본 알고리즘에서는 질의 Q를 API 저장소(API-Repository)에 있는 모든 오퍼레이션들과 비교한다. 만일 매치가 발견되면 기록되고 우선순위에 의해 정렬된다. Match( ) 함수는 질의 출력항목을 오퍼레이션 출력항목과 비교하여 유사도를 계산하고, 매치가 실패하지 않는다면 반대로 오퍼레이션 입력항목과 질의 입력항목을 비교한다.

```
Discovery(Q) {
    for all (O in API-Repository)
        if Match(Q, O) then result.append(O)
    return Sort(result)
}
Match(Q, O) {
    outputMatch(Q.outputs, O.outputs)
    inputMatch(O.inputs, Q.inputs)
}
```

(그림 4) 조합 가능한 API 발견 알고리즘

이 알고리즘에서 어떤 오퍼레이션들은 정확하게 매치되어 최종 결과로 선택되어 질 수도 있지만, 정확하지 않더라도 무조건 탈락되지 않고 상호 계층관계에 따라 유사도가 정해질 수도 있다. 따라서 본 논문에서는 유사성 측정기법(5.2절 참조)을 적용하여 우선순위별로 정렬하여 가장 높은 점수를 얻은 오퍼레이션을 최종 결과로 선택한다. outputMatch( ) 함수는 (그림 5)에 묘사되어 있으며, 만일 질의 출력항목 중 하나라도 오퍼레이션 출력항목과 매치되지 않는다면 이 매치는 실패한다. inputMatch( )도 outputMatch( )와 똑 같은 과정으로 처리되지만 단지 질의와 오퍼레이션 순서가 바뀌어 수행된다.

```
outputMatch(Q.outputs, O.outputs) {
    degreeMatch = 0
    for all Q.out in Q.outputs
        for all O.out in O.outputs
            degree = Similarity(Q.out, O.out)
            if degree > degreeMatch
                then degreeMatch = degree
    if degreeMatch = 0 then return fail
        else return degreeMatch
}
```

(그림 5) 출력항목 매치 알고리즘

5.2 API 매개변수 유사성 탐색 방법

본 절에서는 Open API 매개변수 유사도가 어떻게 측정되는지 기술한다. 본 연구에서는 매칭되는 API들을 효율적으로 발견하기 위해 신택틱과 시맨틱 정보를 혼합 사용하는 방법을 탐구한다. 먼저, WSDL/WADL/XRDL 파일에 작성된 신택틱 정보를 파싱하여 다음과 같이 토큰화(tokenization), POS(Part-Of-Speech), 불용어(stop-word) 필터링, 약어 확장, 그리고 동의어 탐색을 수행한다.

- 복합단어 토큰화: API 기술 문서를 파싱하여 모든 매개변수들을 뽑아낸 후에 복합단어는 여러 개의 용어로 분리한다. 예를 들면 ClientName은 Client와 Name으로 나눈다. 단어를 토큰화하기 위해 프로그래머들에 의해 사용되는 일반적인 명명 규칙들을 사용한다. 본 연구에서는 대소문자, 빈칸, 하이픈(-), 언더스코어 문자(_), 문자 내 숫자 등과 같은 구분 문자를 사용하여 복합단어를 분리한다.
- POS와 불용어 필터링: POS는 접두사 또는 어미 등 어근에 붙어있는 부분을 제거하는 알고리즘으로서 단어를 어근으로 분리하므로 같은 단어이지만 접미사나 어미의 변화에 의해 다른 단어로 인식되는 것을 막을 수 있다. 또한, 미리 만들어진 불용어 리스트에 의해 불용어들이 필터링된다. 본 연구에서 사용되는 불용어는 상용 검색 엔진에서 사용되는 것과 비슷한 and, or, the, is 등이 된다.
- 약어 확장: 약어(abbreviation)는 완전한 단어로 확장된다. 예를 들면 CustomerInfo는 CustomerInformation으로 확장이 수행된다. 여러 개의 확장 단어 후보가 존재할 경우 복수 개의 단어 확장도 가능하다. 따라서 CustPurch는 CustomerPurchase와 CustomaryPurchase 등으로 확장될 것이다.
- 동의어 탐색: 용어들에 대한 동의어 리스트를 발견하기 위해 WordNet과 같은 시소러스(thesaurus)를 사용한다. 시소러스란 같은 의미를 갖고 있는 단어이지만 단어의 철자가 다른 경우 이를 해결하기 위해서 제안된 동의어 사전이다. 예를 들면, town과 city는 같은 의미를 갖고 있으나 서로 철자가 틀리므로 다른 단어로 인식된다.

본 논문 4.1에서 오퍼레이션 유사성을 측정하기 위한 토대로 개념들을 클러스터링 하였다. 클러스터링을 고려한 입력은 형식적으로 벡터 $Is = < v_i, C_i >$로 묘사된다(출력도 이와 비슷하게 $Os = < v_o, C_o >$로 표현될 수 있다). 여기서 v_i는 입력 매개변수들의 집합이고, C_i는 v_i와 연관된 클러스터링 개념이다. 따라서 입력 유사도는 위의 전처리 과정을 수행하여 이로부터 나온 다음의 각 용어들에 대해 관련된 개념들을 찾아 교환하고 이에 대한 유사도를 계산한다(출력도 비슷한 방법으로 처리된다).

하나의 질의와 저장소로부터 매치되는 임의의 후보 매개변수 쌍을 (X, Y)라 하고, X와 Y에는 각각 m과 n개의

토큰들이 있다고 가정하면, X=($x_1, x_2, \cdots x_m$)와 Y=($y_1, y_2, \cdots y_n$)로 표현될 수 있으며 X와 Y 간의 클러스터링 기반 매개변수 유사도(Similarity)는

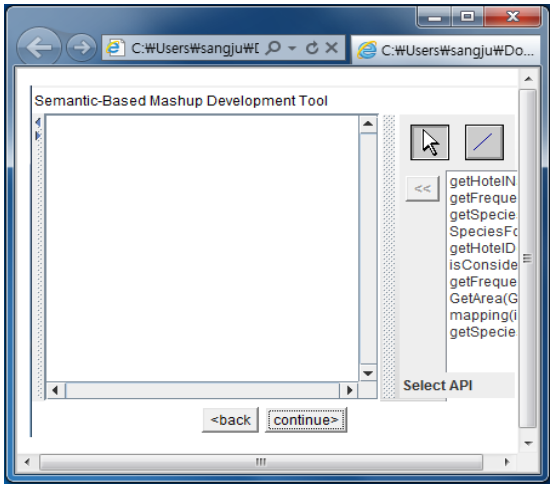
$$Similarity(X,\ Y) = 2 * Match(X, Y)/(m+n)$$

여기서, $Match(X, Y)$는 X, Y 간 매칭되는 토큰의 수이다.

한편, 위의 유사도 측정 방법에서는 4.2절에서 설명한 계층관계 형태소 분석 방법은 고려하지 않았다. 이를 통한 온톨로지를 활용함으로써 검색 결과의 정확률을 향상시킬 수 있다. 클러스터링만을 사용한 매개변수 유사성 탐색에서는 연관성 높은 단어들을 한 클러스터에 묶어서 단지 동일한 개념처럼 취급할 뿐 용어들 사이의 계층관계를 무시함으로써 사용자의 의도와는 관계없는 매칭(즉, false positives)이 발생할 수 있다. 따라서 매개변수 유사도를 계산할 때 계층관계 온톨로지 개념에 위배되는 매개변수 쌍들을 배제하여 사용자가 만족할 수 있는 품질 좋은 것만 선택할 수 있는 정제과정이 필요하다. 기본적인 아이디어는 매치되는 매개변수 쌍들 중에서 4.2절에서 설명한 패턴 상관관계를 조사하여 부모 클래스가 일치하지 않을 경우 이 쌍들을 검색 결과에서 배제한다.

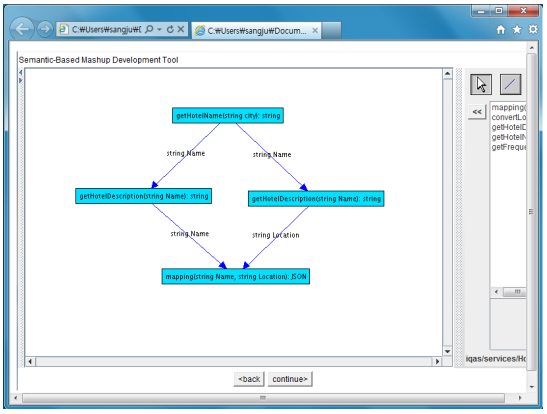
5.3 대화형 매쉬업 툴 활용 예

(그림 6)은 최초 매쉬업 개발 툴 작업 과정을 보여준다. 오른쪽 판넬에 매쉬업에 활용 가능한 오퍼레이션들의 리스트를 보여주고 있다. 왼쪽 판넬은 시맨틱 기반 매쉬업 개발 디자이너 캔버스이다. 매쉬업을 수행하기 위해 먼저 사용자는 쿼리문을 사용하여 요구사항을 질의한다. 만일 여러 개의 오퍼레이션들이 탐색된다면 이들 중 유사도가 가장 높은 오퍼레이션을 선정한다. 이러한 순차적 접근 과정을 거쳐 목표 지향적 매쉬업 워크플로우를 완성한다.



(그림 6) 최초 매쉬업 개발 작업 단계

(그림 7)에서는 4개의 오퍼레이션들(즉, *HotelSearch, HotelFinder, GetLocation, MappingService*)이 매쉬업을 구성하고 있다. *HotelSerch*에서는 출력 매개변수로 hotel name이 산출되고 이들은 *HotelFinder*와 *GetLocation*에 의해 hotel description과 location이 산출된다. 이들을 입력으로 받아 *MappingService*에서는 hotel description과 map이 혼합된 최종 매쉬업 결과물을 출력한다.



(그림 7) 최종 매쉬업 워크플로우

6. 실험 분석

본 장에서는 제안된 시맨틱 온톨로지 구축 방법과 대화형 매쉬업 조합 시스템의 성능을 평가하기 위한 실험과 그 분석 결과를 설명한다. 실험 분석을 위한 자료는 Open API 포털 사이트인 programmableWeb 사이트로부터 수집할 수 있다. 하지만 이 사이트에서는 현재 WSDL/WADL과 같은 기술 문서는 제공하고 있지 않다. 따라서 우선 간편한 데이터 구축을 위해 SOAP API 기술 문서인 WSDL 파일을 제공하고 있는 xmethods.net으로부터 50개의 WSDL 파일을 다운로드 받아, 이를 파싱한 후 전처리 과정을 거쳐 우리의 알고리즘들이 적절히 수행될 수 있는 데이터 파일을 준비하였다. 하지만 REST API 기술 문서인 WADL 파일을 제공하는 사이트는 현재 우리가 아

는 한 거의 없으므로 programmableWeb에서 REST API를 제공하고 있는 공급자들로부터 실험 분석에 필요한 자료를 직접 수작업으로 추출하였다. 본 실험에서는 168개의 REST API 공급자들로부터 264개의 입/출력과 8209개의 매개변수를 추출하였으며, 이는 501개의 용어들로 구성되어 있다.

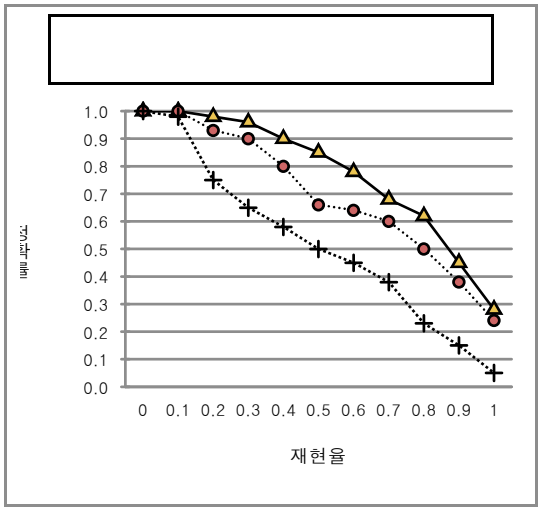
(표 2) 실험 분석을 위한 데이터 요약

API 종류	REST	SOAP
기술 문서	WADL	WSDL
데이터 소스	programmableWeb	xmethods.net
API 개수	168	50
입/출력 수	264	85
매개변수 수	8209	1245
용어 수	501	104
용어 평균수	11.2/tran	4.1/tran
도메인	mapping/travel/ weather	zip/weather/ address
최저 지지도	3%	10%
최저 신뢰도	80%	80%

실험 분석을 위한 데이터의 특성을 요약한 내용은 (표 2)와 같다. 여기서 각 트랜잭션 당 용어의 평균 개수는 REST는 11.2, SOAP은 4.1로 REST가 SOAP보다 2.7배 많다. 이는 SOAP API는 일반 프로그래밍 개념과 비슷하게 오퍼레이션 능력을 제공하고자 하는 기술이지만, REST API는 URI를 기반으로 리소스를 개방하고자 하는 기술이기 때문에 이는 주로 매쉬업 데이터 디스플레이에 사용되기 때문이다. programmableWeb에서는 총 54개의 도메인별로 Open API들이 정리되어 있으나, xmehods.net에서는 도메인별 분류 없이 전체 서비스 리스트만 제공하고 있다. 따라서 본 실험에서는 programmableWeb에서는 mapping, travel, weather 도메인만, xmethods.net에서는 zip, weather, address 도메인만 적용하였다. 이는 너무 관련 없는 API들 간의 클러스터링은 결과가 거의 없거나 클러스터링이 별 의미가 없을 수 있기 때문이다. 본 연구에서는 실험 파일에 대해 최저 지지도와 최저 신뢰도를 변경시켜 가면서 클러스터 결과를 도출 분석하였는데, 그 결과 최저 지지도는 REST 3%, SOAP 10%, 최저 신뢰도는 REST 80%, SOAP 80%에서 의미 있는 클러스터를 구성할 수 있는 하나의 적절한 임계값이 될 수 있음을 알 수 있었다.

위에서 설명된 실험 데이터로부터 계층적 결합 클러스터링 기법(4.1 참조)을 실행시킨 결과 REST API에서는 최종 클러스터링 결과로 {identification, latitude, longitude}, {area, city, code, country}, {origin, destination}, {end, start}, {latitude, longitude, time, type}, {postal, code, city, name}, {code, state, city}, 그리고 {result, latitude, longitude}의 8개 최적 클러스터가 생성되었고, 반면에 SOAP API에서는 {zip, city, area, state}, {zip, code}, 그리고 {country, city} 3개의 클러스터가 생성되었다. 이와 관련된 계층관계 형태소 분석(4.2 참조) 결과는 REST API에서는 CountryID **propertyOf** Country, FullName **subClassOf** Name, 그리고 ArriveTime **subClassOf** Time과 같은 패턴이 자주 적용되었고, SOAP API에서는 ZipCode **propertyOf** Zip, TelephoneAreaCode **propertyOf** Telephone, 그리고 GetCity **subClassOf** City와 같은 패턴이 비교적 많이 사용되었다.

다음으로 본 논문 5장에서 제안하는 대화형 매쉬업 조합 시스템의 우수성을 분석하기 위해 기존의 IR 분야에서 가장 보편적으로 활용되고 있는 재현율(recall)과 정확률(precision)을 사용하여 평가하였다. 본 실험에서는 (그림 8)과 같은 R-P 커브(curve)를 작성하였다. R-P 커브는 IR 분야에서 검색 엔진의 효율성을 가장 잘 표현하는 그래프로써 알려져 있다. 이 그래프에서 X 축은 재현율을 표시하고 Y 축은 정확률을 나타내며, 가장 높이 있는 커브가 가장 좋은 성능을 나타낸다. 실험은 기존의 전통적인 키워드 검색 방법과 클러스터링 기법을 적용한 Woogle [15] 방법을 우리의 API 매개변수 유사성 탐색 방법과 비교하여 제안된 방법의 우수성을 보여주고 있다.



(그림 8) R-P 커브

(그림 8)에서 알 수 있듯이 기존의 키워드 검색(Keyword) 방법은 가장 낮은 성능을 보이고 있다. Woogle 방법은 키워드 검색보다는 상당한 성능 향상을 보이는데, 이는 연관성이 높은 용어들을 한 클러스터에 묶어 동일한 개념처럼 취급함으로써 검색의 재현율을 향상시킬 수 있기 때문이다. 그러나 재현율이 향상된 만큼 비례적으로 사용자의 의도와는 관계없는 false positive 매칭이 발생됨에 따라 정확률의 증가는 크게 기대할 수 없다. 우리의 API 매개변수 유사성 탐색(Semantic) 방법은 계층적 결합 클러스터링 기법에 계층관계 형태소 분석 기법을 추가하여 검색 결과들 중 부적합한 API들을 정제함으로써 재현율과 정확률을 함께 증가시킬 수 있다. 결론적으로 Woogle과 Semantic 방법은 기존의 Keyword 방법보다는 더 나은 성능을 보여주고 있으며, 본 논문에서 제안하는 Semantic 방법은 이들 중 가장 좋은 성능을 보여주고 있다.

7. 결론 및 향후 연구 방향

본 논문에서는 REST, SOAP, JavaScript, 그리고 XML-RPC와 같은 다양한 Open API 타입들을 지원하는 시맨틱 기반 매쉬업 개발 툴을 구현하였다. 이를 위해 기존의 SOAP 기반 웹 서비스 분야에서 사용된 시맨틱 기술 및 관련 알고리즘들을 최소한의 수정만으로 재사용하였다. 먼저 시맨틱 온톨로지를 구축하기 위해 WSDL/WADL/XRDL 파일에 존재하는 신텍틱 정보를 활용하고 그들 사이에 숨겨져 있는 시맨틱 개념을 얻기 위해 마이닝 알고리즘을 적용하였다. 계층관계 형태소 분석 방법은 API 매개변수 내에 포함된 용어들 간의 상관관계를 취득하고, 만일 두 용어들이 서로 유사하고 상관관계가 조건에 일치한다면 그 매개변수를 매치한다. 대화형 매쉬업 조합 시스템을 구현하기 위한 조합 가능한 API 발견 알고리즘은 질의에 매치될 가능성이 없는 API들을 사전에 배제하여 검색 엔진의 효율을 향상시키고, API 매개변수 유사성 탐색 방법은 자동으로 구축된 시맨틱 온톨로지를 활용하여 검색 재현율과 정확률을 향상시킨다. 본 연구에서 제안된 기법들은 관련 포털 사이트로부터 실제 사용되고 있는 자료를 다운로드 받아 실험 분석을 수행하였다. 실험 결과 기존의 키워드 검색 방법과 Woogle 방법보다 재현율/정확률 측면에서 상당한 성능 향상을 보였다.

본 논문에서는 동적 Open API 조합 시스템을 구현하기 위해 대화형 목표 지향 접근 방법을 제안하고 있으나 전체 시스템 중 시맨틱 온톨로지 구축 부분과 조합 가능한 API 발견 알고리즘, 그리고 워크플로우 디자이너만 구축된 상태이고 나머지 기능들은 현재 개발 중에 있다. 또한 본 논문에서는 API 조합 자동화에 관한 연구는 수행되지 못한 상태이다. 인공 지능 분야의 계획 관리자(planner)에 관한 연구 결과물들을 본 논문에 접목함으로써 매쉬업 조합의 전 자동화가 가능할 수 있을 것이다.

참 고 문 헌

[1] http://www.housingmaps.com

[2] http://www.programmableweb.com

[3] http://pipes.yahoo.com/pipes

[4] T. Loton, Introduction to Microsoft Popfly, No Programming Required, Lotontech Limited, 2008.

[5] R. Ennals and M. Garofalakis, “MashMaker: Mashups for the Masses,” In Proceedings of the 2007 ACM SIGMOD International Conference Management of Data, pp. 1116-1118, 2007.

[6] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh, “Dimia: Data Mashups for Intranet Applications,” In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1171-1182, 2008.

[7] A. V. Riabov, E. Bouillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, “Wishful Search: Interactive Composition of Data Mashups,” In Proceedings of the ACM International Conference on World Wide Web, pp. 775-784, 2008.

[8] J. Tatemura, S. Chen, F. Liao, O. Po, K. S. Candan, and D. Agrawal, “UQBE: Uncertain Query By Example for Web Service Mashup,” In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1275-1280, 2008.

[9] R. Fielding, Architectural Styles and The Design of Network-based Software Architectures, PhD thesis, University of California, 2000.

[10] A. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H. Y. Paik, “Semantic-Based Mashup of Composite Applications,” IEEE Transactions on Services Computing, Vol. 3, No. 1, pp. 2-15, 2010.

[11] 이용주, “스마트 매쉬업을 위한 시맨틱 기반 Open API 온톨로지 구축 기법,” 디지털산업정보학회 논문지, 제7권, 제3호, pp. 11-23, 2011.

[12] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules between Sets of Items in Large Databases,” In Proceedings of the ACM-SIGMOD International Conference Management of Data, pp. 207-216, 1993.

[13] R. Agrawal and R. Srikant, “Fast Algorithm for Mining Associations Rules,” In Proceedings of the 20th VLDB Conference, Santiage, Chile, pp. 487-499, 1994.

[14] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, “Semantic Matching of Web Services Capabilites,” In Proceedings of the International Semantic Web Conference, pp. 333-347, 2002.

[15] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, “Similarity Search for Web Services,” In Proceedings of VLDB, pp. 372-383, 2004.

◐ 저 자 소 개 ◑

이 용 주

1983년 울산대학교 산업공학과(학사)
1985년 한국과학기술원 산업공학과 정보검색전공(석사)
1997년 한국과학기술원 정보및통신공학과 컴퓨터공학전공(박사)
1985년~1989년 KIST 시스템공학연구소 연구원
1989년~1994년 삼보컴퓨터 근무
1998년~2007년 상주대학교 컴퓨터공학과 부교수
2008년~현재 경북대학교 과학기술대학 컴퓨터정보학부 교수
관심분야 : 웹 데이터베이스, 정보검색, 공간 데이터베이스
E-mail : yongju@knu.ac.kr