시맨틱과 워크플로우 혼합기법에 의한 자동화된 웹 서비스 조합시스템

이 용 주†

요 약

본 논문에서는 BPEL 기법에 OWL-S 기법을 도입하는 혼합기법을 사용하여 자동화된 웹 서비스 조합시스템을 구현한다. BPEL 기법은 에러 처리나 트랜잭션 관리와 같은 비즈니스 환경에서 요구되는 실질적인 전체 기능을 지원하고 있으나, 주된 단점은 정적 조합 기법으로써 서비스 선택 및 워크플로우 관리가 사전에 수동으로 이루어져야만 한다. 반면에, OWL-S 기법은 자동적인 웹 서비스 발견 및 통합을 실현하기 위해 기계 가독형으로 웹 서비스 기능을 묘사할 수 있는 메카니즘, 즉 온톨로지(ontology)를 사용한다. 이에 따라 호환 가능한 웹 서비스들 간의 동적 통합이 가능하고, 웹 서비스 조합 실행 시에 웹 서비스 발견이 가능하다. 그러나 이러한 기법은 아직 연구 중에 있으며, 실제 적용을 위해서는 BPEL의 상용기법이 요구된다. 본 연구에서는 BPEL4WS와 OWL-S 혼합 시스템인 SemanticBPEL의 구조를 설계하고 웹 서비스 탐색 및 통합 알고리즘을 제안한다. 특히, SemanticBPEL 시스템은 오픈 소스 툴들을 기반으로 구현되었으며, 기존에 개발되어 있는 다양한 BPEL 시스템들과 기능을 비교하여 그 우수성을 보여주고 있다.

키워드 : 웹 서비스 조합, 혼합기법, BPEL4WS, OWL-S, 탐색 및 통합 알고리즘, 구현기법

Automated Composition System of Web Services by Semantic and Workflow based Hybrid Techniques

Yong-Ju Lee†

ABSTRACT

In this paper, we implement an automated composition system of web services using hybrid techniques that merge the benefit of BPEL techniques, with the advantage of OWL-S. BPEL techniques have practical capabilities that fulfil the needs of the business environment, such as fault handling and transaction management. However, the main shortcoming of these techniques is the static composition approach, where the service selection and flow management are done a priori and manually. In contrast, OWL-S techniques use ontologies to provide a mechanism to describe the web services functionality in machine-understandable form, making it possible to discover, and integrate web services automatically. This allows for the dynamic integration of compatible web services, possibly discovered at run-time, into the composition schema. However, the development of these approaches is still in its infancy and has been largely detached from the BPEL composition effort. In this work, we describe the design of the SemanticBPEL architecture that is a hybrid system of BPEL4WS and OWL-S, and propose algorithms for web service search and integration. In particular, the SemanticBPEL has been implemented based on the open source tools. The proposed system is compared with existing BPEL systems by functional analysis. These comparisions show that our system outperforms existing systems.

Key Words : Web Services Composition, Hybrid Techniques, BPEL4WS, OWL-S, Search and Integration Algorithms, Implementation Techniques

1. 서 론

최근 웹 서비스 기술(Web Services Technologies)의 출현으로 분산 애플리케이션 통합 문제에 대한 차세대 솔루션인 서비스 지향 아키텍처(SOA: Service Oriented Architectures)의 구현이 가능하게 되었다. SOAP[1], WSDL[2], UDDI[3], 그리고 BPEL4WS[4]와 같은 표준 웹 서비스 기술들은 SOA[5]에서 요구되는 이기종 분산 플랫폼 상의 복잡한 애플리케이션 통합 문제를 유연하게 해결할 수 있는 다양한 인터페이스 기술들을 제공하고 있다.

그렇지만, 웹 서비스 기술의 활발한 도입과 관심에도 불구하고 아직까지는 기업의 내부 통합 프로젝트에서만 이들

※"이 논문은 2006년도 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임" (KRF-2006-521-D00411).
† 정 회 원 : 상주대학교 컴퓨터공학과 교수
논문접수 : 2006년 12월 11일, 심사완료 : 2007년 2월 1일

이 사용되고, 외부 서비스들이 '온 디맨드(on demand)' 방식으로 연결되는 엔터프라이즈급 IT 플랫폼에서는 아직 활용되지 못하고 있는 실정이다. 이러한 주된 이유는 현재의 웹 서비스 기술들이 동적인 웹 서비스 발견 및 통합, 즉 자동화된 웹 서비스 조합에 대한 적절한 기법을 제공하지 못하기 때문이다. 현재의 기술들은 자동화를 위한 시맨틱 개념이 부족하여 프로그래머에 의한 수동으로 모든 작업을 수행하고 있다.

웹 서비스 조합에 관한 연구는 크게 두 가지 방향으로 추진되고 있다. 첫째는 IBM, 마이크로소프트, BEA 등 컴퓨터 대형 업체들이 주도적으로 추진하고 있는 비즈니스 워크플로우 관리 이론을 기반으로 한 웹 서비스 조합기법이다. 비즈니스 프로세스 실행 언어(BPEL: Business Process Execution Language)를 사용하는 이러한 기법들은 에러 처리나 트랜잭션 관리와 같은 비즈니스 환경에서 요구되는 실질적인 전체 기능을 지원하고 있다. 그렇지만 이러한 기법의 주된 단점은 정적 조합 기법으로써 서비스 선택 및 워크플로우 관리가 사전에 수동으로 이루어져야만 한다[6].

두 번째 접근 방법은 주로 학계에서 추진되고 있는 OWL-S와 같은 시맨틱 웹을 기반으로 한 웹 서비스 조합기법이다. OWL-S[7]는 자동적인 웹 서비스 발견 및 통합을 실현하기 위해 기계 가독형으로 웹 서비스 기능을 묘사할 수 있는 메카니즘, 즉 온톨로지(ontology)를 사용한다. 이에 따라 호환 가능한 웹 서비스들 간의 동적 통합이 가능하고, 웹 서비스 조합 실행 시에 웹 서비스 발견이 가능하다. 그러나 이러한 기법은 아직 연구 중에 있으며, 실제 적용을 위해서는 BPEL4WS 스타일의 워크플로우 실행계획 및 엔진이 필요하다[8].

본 연구에서는 자동화된 웹 서비스 조합을 구현하기 위해 BPEL 기법에 OWL-S 기법을 도입하는 혼합(hybrid) 조합기법을 제안한다. BPEL 조합기법(예, BPEL4WS)은 실제 비즈니스에 널리 사용되고 있으나, 정적 기법이어서 수동으로 조합을 수행해야 하므로 작업하기 어렵고, 시간이 많이 소비되며, 에러가 발생되기 쉽다. 이를 보완하기 위해 OWL-S 조합기법을 적용하면 동적으로 웹 서비스 발견 및 통합이 가능할 수 있다.

본 연구와 유사한 혼합 조합기법들을 살펴보면, [6]에서 동적 웹 서비스 발견을 위해 BPEL4WS를 중심으로 OWL-S 개념을 도입한 보텀-업(bottom-up) 방식을 제안하였다. 이 논문에서는 동적 웹 서비스 발견에 대한 방법론은 제시하고 있으나, 본 연구에서 수행하고자 하는 웹 서비스 조합 자동화에 대한 언급은 없다. [9]에서는 자동화된 웹 서비스 조합을 구현하기 위해 AI 분야의 계획 기법을 제안하고 있으나 이 기법에서 웹 서비스 발견 및 선택은 수동으로 이루어지고 있으며, 실제 비즈니스 적용보다는 이론에 더 많이 치중된 연구이다. [10]에서는 BPEL 기법에 OWL-S 기법을 도입함으로써 발생되는 두 기법 사이의 간격에 대한 중계자 역할을 기술하였다. 그렇지만 이 논문은 우리 연구의 중요 부분인 웹 서비스 매칭 알고리즘에 대해서는 차후 연구과제로 언급하였으며, 웹 서비스 조합 자동화를 위한 전반적인 워크플로우 처리 과정은 기술하고 있지 않다.

본 연구에서는 기존의 연구들과는 달리 웹 서비스 조합 자동화를 위해 워크플로우 구성, 웹 서비스 탐색 및 통합, 그리고 조합 실행 등 비즈니스 워크플로우 전체 과정을 다루고 있다. 특히, 비즈니스 분야에서 성공적으로 활용되고 있는 워크플로우 관리시스템(WfMS: Workflow Management System) 기법[11]들을 적용하여 웹 서비스 조합 전 과정의 자동화를 지원한다. 또한 여러 사이트에 분산 저장되어 있는 웹 서비스들을 효율적으로 탐색하고, 선택된 서비스들의 적절한 통합을 지원하는 계획 에이전트의 구현이 새롭게 소개된다. 웹 서비스 매칭 알고리즘은 기존 연구를 확대 발전시켰으나, BPEL4WS 워크플로우 실행계획을 만든 후 이를 상용 실행엔진과 연결시키는 작업은 기존 연구들과 다른 새로운 기법이다.

본 논문의 구성은 다음과 같다. 2장에서는 자동화된 웹 서비스 조합에 관한 관련 연구를 설명하고, 3장에서 본 시스템의 전체 구조 및 웹 서비스 탐색 및 통합 알고리즘을 제안한다. 4장에서 시스템 구현 및 분석에 관해 서술하고, 5장에서 결론을 내렸다.

2. 관련 연구

웹 서비스 조합기법에 관한 연구는 두 가지 방향, 즉 워크플로우 기반과 시맨틱 기반 웹 서비스 조합 언어로 양분되어 진행되고 있다. 이 장에서는 이에 대한 기술 동향을 먼저 살펴보고, 본 연구와 유사한 관련 시스템들을 살펴본다.

2.1 워크플로우 기반 웹 서비스 조합 언어

IBM의 WSFL[12]과 마이크로소프트사의 XLang[13]는 웹 서비스 조합을 위해 가장 초기에 제안된 언어들이다. 둘 다 WSDL을 확장한 것으로써 웹 서비스의 구문적(syntactic) 관점을 묘사하기 위해 사용된 표준 언어이다. 특히, WSFL과 XLang은 비즈니스 워크플로우 정의 언어를 이용하여 특정 비즈니스 플로우를 정적으로 정의하고 이때 요구되어지는 웹 서비스의 조합을 수행하고 있다. BPEL4WS는 보다 최근에 제안된 스펙으로써 WSFL과 XLang의 장점을 취합한 새로운 비즈니스 프로세스 정의 언어이다. BPEL4WS는 그래프 중심으로 프로세스를 표현하는 WSFL과 구조적 중심으로 프로세스를 표현하는 XLang을 통합한 새로운 표준 언어이지만 시맨틱 개념은 지원하고 있지 않다.

BPEL4WS는 기본적으로 비즈니스 프로세스를 구현하는 언어이다. 비즈니스 프로세스는 비즈니스 목표를 실현하기 위해 액티비티(activity)들의 그룹으로 표현될 수 있으며 BPEL4WS의 액티비티로는 <receive>, <invoke>, <reply> 등이 있다. 이러한 액티비티들은 <sequence>, <switch>, <while>과 같은 제어 구성자(control structure)를 사용하여 좀 더 복잡한 프로세스로 조합될 수 있다. BPEL4WS가 비즈니스 커뮤니티에서 성공할 수 있었던 주요 요인으로는 첫째, 비즈니스 분야에서 성공적으로 활용되고 있는 기존의 워크플로우 관리 기법을 바탕으로 개발되었기 때문에 비즈니스 프로세스 상호작용을 모델링하기가 매우 적합하였다. 둘째로는, 이젠

확고한 표준으로 자리를 잡았고 비즈니스 프로세스 처리를 위한 전반적인 기능들을 충분히 제공하고 있기 때문이다. 예를 들면, BPEL4WS의 트랜잭션 관리, 상태관리, 오류 핸들링 기능 등은 길고 중단 없는 프로세스 처리 능력을 보장한다. 이 표준 언어는 웹 서비스 조합기법으로써 IBM, 마이크로소프트, BEA 등 대형 IT 업체에서 적극적으로 연구되고 있으나 정적 기법이어서 서비스의 선택 및 워크플로우 관리가 사전에 수동으로 수행되어야 하는 단점이 있다.

2.2 시맨틱 기반 웹 서비스 조합 언어

시맨틱 웹(Semantic Web)의 근본적인 전제는 정보를 획득하는 과정에서 현재처럼 사람이 직접 수행해야 하는 인간 중심적인 인터페이스를 컴퓨터 프로그램이 이해하고 처리할 수 있는 형태로 확장시키는 것이다[14]. 따라서 시맨틱 웹 개념을 서비스 조합에 적용시키면 자동화된 서비스 발견 및 실행이 가능해 진다. 자동화는 웹 서비스를 시맨틱하게 표현함으로써 구현 가능하며, 이로 인해 에이전트는 서비스 기능들을 인지하고 서비스 조합과 관련된 모든 결정들을 사용자 입장에서 지능적으로 수행할 수 있다.

OWL-S는 가장 최근에 발표된 시맨틱 기반 웹 서비스 언어이다. OWL-S 온톨로지는 기계 가독형으로 웹 서비스 기능을 묘사할 수 있는 메카니즘을 제공한다. 이 메카니즘은 자동화된 웹 서비스 발견 및 통합을 가능하게 만든다. 그러나 이러한 시맨틱 기반 웹 서비스 조합 언어는 학계에서의 열렬한 연구에도 불구하고 아직 연구 중에 있으며 명확한 개념 설정이나 표준화 작업이 완료되어 있지 않은 상황이다. <표 1>은 워크플로우 및 시맨틱 기반 웹 서비스 조합 언어들을 정리 요약한 것이다.

2.3 자동화된 웹 서비스 조합에 관한 연구

웹 서비스 조합 자동화에 관한 연구는 매릴랜드 대학(University of Maryland)의 Mindswap과 조지아 대학(University of Georgia)의 LSDIS 연구실에서 활발히 진행되고 있다. 매릴랜드 대학에서 개발된 WS-Composer[15]는 OWL-S를 사용한 동적 웹 서비스 조합 지원 시스템으로써, 조합을 수행하는 동안에 자체 탐색 알고리즘을 적용하여 목표 지향적이고 대화식 조합 기법을 제안하고 있다. 그러나 이 시스템은 비즈니스 프로세스 실행언어는 고려하지 않았기 때문에 사람이 개입된 반자동 시스템으로만 작동되는 단점이 있다. 조지아 대학에서 개발된 METEOR-S[16]는 워크플로우 기반 시맨틱 웹 서비스 조합 기법을 제안하고 있다. METEOR-S는 본 연구와 비슷하게 시맨틱 프로세스 템플릿을 사용하여 이를 워크플로우 실행계획으로 변환한 후 실행 엔진에서 수행하는 일련의 과정을 기술하고 있다. 그러나 이 시스템은 정적 웹 서비스 조합 접근방법이며 비즈니스 전반적인 워크플로우 처리 능력을 제공하지 못하는 실험실 수준인 단점이 있다.

시맨틱과 워크플로우 혼합기법에 관한 연구는 [6, 9, 10]에서 수행되었다. [6]에서는 BPEL4WS 구현 엔진인 BPWS4J[17]가 확장 가능하지 못하기 때문에 웹 서비스 탐색을 위한 중개자로써 SDS(Semantic Discovery Service)를 구현하였다. 비록 이 논문에서는 동적 웹 서비스 발견에 대해서는 기술하고 있으나, 웹 서비스 조합 자동화에 대한 내용은 취급되어지지 않았다. [9]에서는 자동화된 웹 서비스 조합을 구현하기 위해 서비스 조합 단계에서 OWL-S를 사용하고, 이를 실행시키기 위해 BPEL4WS로 변환하는 AI 분야의 계획 기법을 제안하였다. 이 논문의 목표는 비록 웹 서비스 조합의 자동화이지만, 실제적인 적용에 중점을 둔 것보다는 이론에 치우친 논문이다. [10]에서는 동적 웹 서비스 발견을 위해 WSDL 파일과 OWL 파일을 도메인 내에 함께 저장하고, 동적 웹 서비스 조합을 구현하기 위해 기존의 BPEL 템플릿 파일에 새로운 웹 서비스를 첨가할 수 있는 프로그래밍 기반 프레임워크를 제공하고 있다. 그러나 이 논문은 웹 서비스 조합 자동화를 위한 전체적인 과정을 기술한 것이 아니라 단지 혼합기법에 대한 기본 틀만 제공하고 있다.

3. 자동화된 웹 서비스 조합 시스템

3.1 시스템 구조

BPEL4WS와 OWL-S 혼합기법을 적용한 SemanticBPEL 시스템의 전체적인 구조는 (그림 1)과 같으며, 서비스 공급자(provider), 서비스 사용자(requester), 그리고 서비스 중개

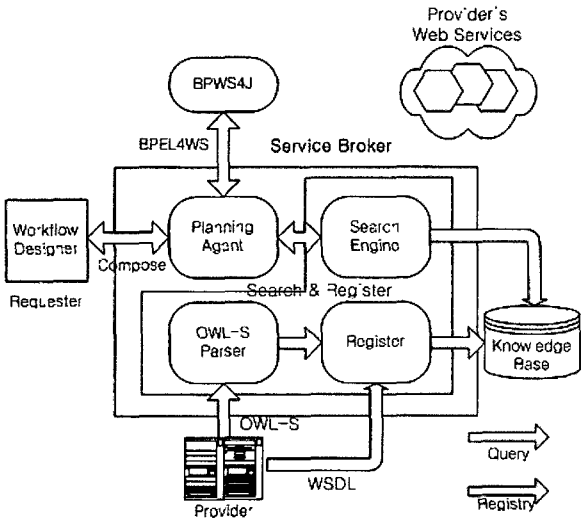
〈표 1〉 웹 서비스 조합 언어

	워크플로우 기반			시맨틱 기반
품명	WSFL	XLANG	BPEL4WS	OWL-S
기반기술	WSDL	WSDL	WSFL과 XLANG의 통합	DAML+OIL
주요기술	그래프 중심으로 프로세스 표현	구조적 중심으로 프로세스 표현	액티비티와 제어 구성자 사용으로 프로세스 조합	온톨로지 사용으로 자동화된 메카니즘 제공
시맨틱 개념	지원안됨	지원안됨	지원안됨	시맨틱 지원
조합방법	정적	정적	정적	동적
단점	수동처리	수동처리	수동처리	상용화 미비
개발업체	IBM	Microsoft	IBM, Microsoft, BEA 등	DARPA

자(service broker)로 구성되어 있다. 서비스 중개자는 계획 에이전트(planning agent)와 웹 서비스 탐색(search) 및 등록(register) 모듈로 이루어져 있으며, 중개자는 SemanticBPEL 시스템의 미들웨어(middleware) 역할을 수행한다. Semantic BPEL 시스템은 본 연구실에서 이미 개발되어 있는 동적 웹 서비스 조합을 위한 워크플로우 기반 서비스 중개자 시스템인 DWSC(Dynamic Web Services Composition)[18]를 기반으로 확장 발전시켰다. 구현 시스템은 기존의 DWSC 시스템과 개념 및 구조적인 면에서는 큰 차이가 없으나 자체 워크플로우 엔진의 기능적 한계 때문에 BPWS4J 엔진으로 대치되고, GUI 기반 워크플로우 디자이너가 새로 개발됨에 따라 많은 모듈들이 새로 개발되었다. (그림 1)에 대한 자세한 설명은 다음과 같다.

- 서비스 공급자: 웹 서비스 메타 정보를 등록한다. OWL-S와 WSDL로 작성된 웹 서비스 객체를 등록하기 위해 등록 모듈을 엑세스 한다.
- 서비스 사용자: 워크플로우 디자이너를 사용하여 워크플로우를 작성하고 이를 실행한다.
- 계획 에이전트: 탐색엔진을 이용하여 웹 서비스를 검색하고, BPEL4WS 스팩을 생성하여 BPWS4J 엔진에 의해 이들 워크플로우 문서를 등록(deploy)한다.
- 웹 서비스 탐색 및 등록 모듈: OWL-S 파서, 웹 서비스 등록기, 탐색엔진으로 구성되어 있으며, 웹 서비스 등록 및 탐색을 수행한다.
- BPWS4J 엔진: IBM alpahWorks로부터 이용 가능한 BPEL4WS 구현 엔진이다.

SemanticBPEL 시스템에서 공급자는 메타 데이터를 사용하여 웹 서비스를 등록할 수 있어야 하고 사용자는 계획 에이전트를 이용하여 웹 서비스 조합 및 실행을 수행할 수 있어야 한다. 공급자는 자신의 웹 서비스를 등록하기 위해 등록 모듈을 접속한다. OWL-S로 작성된 웹 서비스 객체는



(그림 1) SemanticBPEL 시스템

OWL-S 파서에 의해 파싱된 후 웹 서비스 등록기에 전달되고, WSDL은 바로 웹 서비스 등록기에 전달된 후 지식베이스(Knowledge Base)에 저장된다. 한편, 사용자는 워크플로우 디자이너를 사용하여 워크플로우를 작성하고 계획 에이전트로부터 이를 BPEL4WS 스팩으로 변환한 후 BPWS4J 엔진에 워크플로우를 등록하고, 최종적으로 웹 서비스 조합을 실행한다.

사용자 처리과정을 좀 더 자세히 살펴보면, 사용자는 먼저 워크플로우 디자이너를 사용하여 비즈니스 프로세스를 작성하는데, 사전 작업으로 시맨틱 웹 서비스 탐색 과정을 수행한다. 즉 워크플로우의 각 태스크는 탐색 엔진의 웹 서비스 탐색 알고리즘에 의해 관련 웹 서비스들이 검색되고, 선택된 웹 서비스는 프로세스 전후 관계에 따라 입출력 인터페이스가 연결되며 최종적으로 BPEL4WS로 변환된다. 이 실행 계획은 BPWS4J 엔진에 의해 웹 서비스 프로세스가 실행된다.

이러한 처리과정에서 SemanticBPEL 시스템에서는 기존 워크플로우 시스템에서는 볼 수 없었던 새로운 웹 서비스 탐색 및 통합 문제의 해결을 요구한다.

1) 웹 서비스 탐색 문제: SemanticBPEL 시스템에서는 전통적인 워크플로우 시스템과는 달리 인터넷상의 수 없이 많은 웹 서비스들 중에서 가장 적당한 것을 찾아야 하는데 이는 너무 시간이 많이 소비되고 지루한 일이 된다. 따라서 수동으로 이러한 작업을 수행하는 것은 거의 불가능하게 보이며, 이를 효율적으로 지원할 수 있는 새로운 웹 서비스 탐색 메카니즘이 제공되어야만 한다.
2) 웹 서비스 통합 문제: 일단 웹 서비스가 발견되었더라도 완벽하게 워크플로우에 일치되고 상호 운용 가능하리라고는 보장할 수 없다. 사용자가 발견한 웹 서비스의 출력부분은 앞으로 탐색될 웹 서비스의 입력부분에 자연스럽게 연결되어야 한다. 그러나 발견된 웹 서비스가 입출력 인터페이스 부분에서 구조적 또는 의미적으로 상이할 수 있으므로 이 작업을 효율적으로 지원할 수 있는 통합 메카니즘이 필요하다.

3.2 웹 서비스 탐색 및 통합 알고리즘

웹 서비스 탐색 알고리즘의 기본적인 원리는 질의 입력항목을 사용하여 원하는 출력을 산출해 낼 수 있는 웹 서비스들을 찾는 것이다. 이를 위해서 선택되는 웹 서비스는 반드시 질의문의 출력항목을 포함하고 있어야만 하고, 이 서비스의 입력항목들은 질의문의 입력항목에 포함되어 있어야만 한다. 이러한 원리를 기반으로 웹 서비스 탐색 알고리즘을 작성하면 (그림 2)와 같다. (그림 2)에서 search( ) 함수는 질의문 Q를 지식베이스에 있는 모든 웹 서비스 S들과 비교한다. 만일 매치가 발견되면 기록되고 우선순위에 의해 정렬된다. match( ) 함수는 먼저 질의문 출력항목을 웹 서비스 출력항목과 비교하여 유사도를 계산하고, 매치가 실패하지 않는다면 반대로 웹 서비스 입력항목과 질의문 입력항목을 비교한다.

```
search(Q) {
 Record = empty;
 for each S ∈ KnowledgeBase {
  if match(Q, S) then Record.append(S) }
 return sort(Record)
}

match(Q, S) {
 outputMatch(Q{Os}, S{Os})
 inputMatch(S{Is}, Q{Is})
}
```

(그림 2) 웹 서비스 탐색 알고리즘

```
ClientApplet {
 enter query statement;
 Client = new HttpClient( );
 Method = new PostMethod(URL);
 Client.executeMethod(Method); //서블릿 호출
 GraphEditor( ); //워크플로우 작성
}

ServerServlet {
 perform matching algorithm;
 for each S ∈ Record {
  wsdlReader.readWSDL(URL);
  readOperations( ); //portType, operation, input, output 추출
 }
 return Services;
}
```

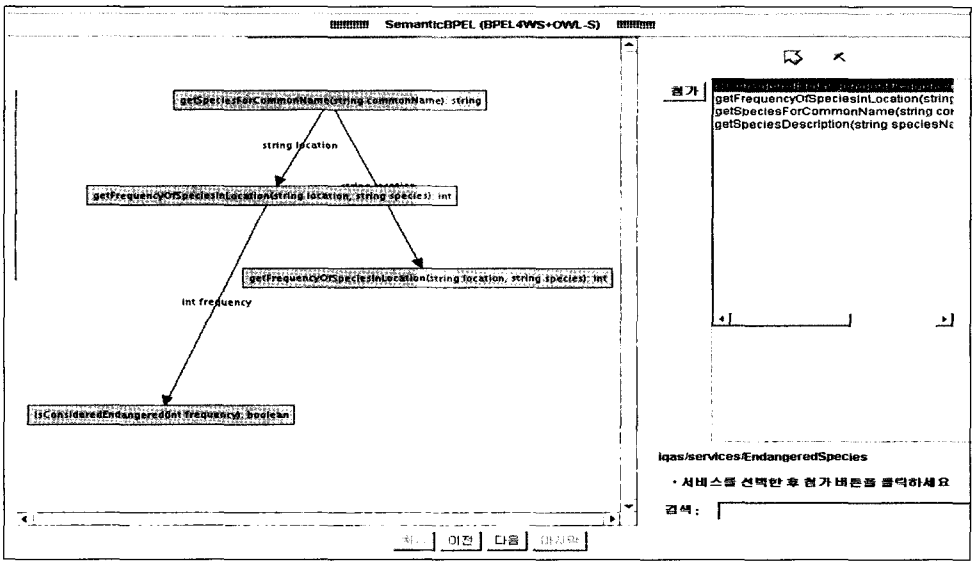
(그림 3) 워크플로우 작성 알고리즘

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 [18]에서 제안한 온톨로지 기반 유사성 측정기법을 적용하여 우선순위별로 정렬하여 가장 높은 점수를 얻은 웹 서비스를 최종 결과로 선택하도록 한다.[1)]

SemanticBPEL 시스템은 클라이언트-서버 프로그래밍 구조로 되어 있다. 클라이언트 측 워크플로우 디자이너는 "드래그 & 드롭" 인터페이스 방식으로 워크플로우를 작성한다. 이를 구현하기 위한 클라이언트 프로그래밍 기술은 자바 애플릿을 사용한다. 자바 애플릿은 사용자가 웹 브라우저로 웹 사이트를 접속할 때 사용자의 PC로 프로그램이 자동으로 다운로드된 후 실행되기 때문에 서버의 로드를 줄여줄 수 있다. 먼저 검색 필드에 질의문이 주어지면 (그림 2)의 웹 서비스 탐색 알고리즘을 적용하여 이용 가능한 웹 서비스 리스트를 보여주고, 이들 중 가장 적합한 웹 서비스를 선택한다. 이러한 방식으로 점차적으로 하나씩 새로운 웹 서비스가 첨가되고 나면 최종적으로 웹 서비스 워크플로우가 완성된다. 이때 Vertex는 태스크를 표현하고 Edge는 데이터 흐름을 나타낸다. 다만 Edge는 두 개의 태스크 사이에 출력 및 입력 XML 스키마가 매치되는 경우에만 연결될 수 있다. 워크플로우 디자이너에서 워크플로우 작성 알고리즘은 (그림 3)과 같으며, 이 알고리즘에서 클라이언트-서버 간 통신을 위해 HttpClient를, WSDL 파싱을 위해 wsdlReader 컴포넌트를 사용한다. (그림 4)는 워크플로우 디자이너 실행 결과를 보여주고 있다.

일단 워크플로우가 완성되고 나면 이를 서버 쪽에 전송하여 BPEL4WS 스팩으로 변환하고 BPWS4J 엔진에 등록시켜야 한다. 이를 위해 먼저 클라이언트 프로그램에서는 완성된 워크플로우를 binary 형태로 서블릿에 전달한다. 서블릿은 플랫폼 독립적인 서버 측 자바 프로그램이다. 서블릿은 binary 형태로 된 그래프를 받아서 GraphParser를 호출한다. GraphParser는 워크플로우를 파싱하여 관련된 WSDL과 BPEL4WS 파일을 만들고 등록함수 Deploy( )를 부른다. 이러한 BPEL4WS 변환 알고리즘은 (그림 5)와 같으며, (그림 4)의 워크플로우로부터 도출된 WSDL 및 BPEL4WS 파일은 (그림 6), (그림 7)과 같다.

WSDL과 BPEL4WS 문서가 생성되고 나면 BPWS4J 엔진에 의해 이들 문서가 등록될 수 있다. 먼저 BPEL4WS 파일의 partnerLink에서 partnerLinkType을 찾은 다음 WSDL



(그림 4) 워크플로우 디자이너 실행 결과

1) 온톨로지 기반 유사성 측정기법은 지면상 생략하고 자세한 내용은 참고문헌 [18]을 참조하기 바람.

```
ClientApplet {
 GraphEditor.getGraph( );
 Client = new HttpClient( );
 Method = new PostMethod(URL);
 In = new ByteArrayInputStream(bytes);
 Method.setRequestBody(In); //binary 형태로 세팅
 Client.executeMethod(Method);
}

ServerServlet {
 G = readObject.getGraph( );
 Parser = new GraphParser(G);
 Parser.parse( );
 wsdlWriter.writeWSDL(wsdlFile);
 bpelWriter.writeBPEL(bpelFile);
 Deploy( );
}
```

(그림 5) BPEL4WS 변환 알고리즘

!!!!! WSDL !!!!!

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="bpel.processes" xmlns:tns="bpel.proc
  <wsdl:import namespace="iqas/services/EndangeredSpecies" location="h
  <wsdl:import namespace="iqas/services/CommonNames" location="http:/
  <wsdl:import namespace="iqas/services/SpeciesInLocation" location="htt
  <wsdl:message name="outputMessage">
    <wsdl:part name="result0" type="xsd:boolean"/>
    <wsdl:part name="result1" type="xsd:int"/>
  </wsdl:message>
  <wsdl:message name="inputMessage">
    <wsdl:part name="commonName" type="xsd:string"/>
    <wsdl:part name="species0" type="xsd:string"/>
    <wsdl:part name="species" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="semanticBpelPT">
    <wsdl:operation name="semanticBpel">
      <wsdl:input message="tns:inputMessage"/>
      <wsdl:output message="tns:outputMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:service name="semanticBpelService">
  </wsdl:service>
<plnk:partnerLinkType name="semanticBpelLT">
<plnk:role name="semanticBpelGenerator">
<plnk:portType name="tns:semanticBpelPT"/>
</plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="getSpeciesForCommonNameLT">
<plnk:role name="getSpeciesForCommonNameGenerator">
<plnk:portType name="ns2:CommonNames"/>
</plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="getFrequencyOfSpeciesInLocationLT">
<plnk:role name="getFrequencyOfSpeciesInLocationGenerator">
<plnk:portType name="ns1:SpeciesInLocation"/>
</plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="isConsideredEndangeredLT">
<plnk:role name="isConsideredEndangeredGenerator">
<plnk:portType name="ns0:EndangeredSpecies"/>
</plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="getFrequencyOfSpeciesInLocationLT">
<plnk:role name="getFrequencyOfSpeciesInLocationGenerator">
<plnk:portType name="ns1:SpeciesInLocation"/>
```

(그림 6) WSDL 화면

파일에서 동일한 이름을 가진 partnerLinkType들을 찾는다. 이로부터 role, portType, 그리고 실제 웹 서비스 URL을 얻은 후 서블릿을 호출한다. 서블릿은 IBM BPWS4J 엔진을 대신해서 기동된다. 이는 현재의 BPWS4J 엔진이 단지 웹 기반 관리자 툴을 통해서만 등록을 허용하기 때문에 엔진 그 자체로는 자동적인 BPEL 등록이 불가능하다. 따라서 프로그래밍 인터페이스가 지원되는 엔진 프록시를 만들어야만 한다. 처리과정은 (그림 8)과 같이 IBM WSGateway 컴포넌트를 초기화한 후 WSDL과 BPEL 파일을 불러온다. 그리고 SOAP RPC(Remote Procedure Call) 채널을 통해 BPEL4WS 문서를 BPWS4J 엔진에 등록한다. 지금까지 설명한 SemanticBPEL 클라이언트-서버 프로그램의 전체 처리과정을 정리 요약하면 (그림 9)와 같다.

!!!!! BPEL !!!!!

```
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-proc
      name="semanticBpelProcess"
      targetNamespace="bpel.processesProcess"
      xmlns:tns="bpel.processesProcess"
      abstractProcess="no">
 <partnerLinks>
  <partnerLink name="semanticBpel" xmlns:ns1="bpel.processes" partnerL
  <partnerLink name="getSpeciesForCommonName" xmlns:ns2="bpel.proc
  <partnerLink name="getFrequencyOfSpeciesInLocation" xmlns:ns3="bpel.
  <partnerLink name="isConsideredEndangered" xmlns:ns4="bpel.processes
  <partnerLink name="getFrequencyOfSpeciesInLocation" xmlns:ns5="bpel.
 </partnerLinks>
 <variables>
  <variable name="var0" xmlns:ns6="iqas/services/CommonNames" mess
  <variable name="var1" xmlns:ns7="iqas/services/CommonNames" mess
  <variable name="var2" xmlns:ns8="iqas/services/SpeciesInLocation" me
  <variable name="var3" xmlns:ns9="iqas/services/SpeciesInLocation" me
  <variable name="var4" xmlns:ns10="iqas/services/EndangeredSpecies"
  <variable name="var5" xmlns:ns11="iqas/services/EndangeredSpecies"
  <variable name="var6" xmlns:ns12="iqas/services/SpeciesInLocation" m
  <variable name="var7" xmlns:ns13="iqas/services/SpeciesInLocation" m
  <variable name="output" xmlns:ns14="def" messageType="ns14:outputMe
  <variable name="input" xmlns:ns15="def" messageType="ns15:inputMess
 </variables>

 <sequence>
  <receive partnerLink="semanticBpel" xmlns:ns16="bpel.processes" portT
       variable="input" createInstance="yes">
  </receive>
  <assign >
   <copy>
    <from variable="input" part="commonName"/>
    <to variable="var1" part="commonName"/>
   </copy>
   <copy>
    <from variable="input" part="species"/>
    <to variable="var3" part="species"/>
   </copy>
   <copy>
    <from variable="input" part="species0"/>
    <to variable="var7" part="species"/>
   </copy>
  </assign>
  <invoke name="getSpeciesForCommonName"
       partnerLink="getSpeciesForCommonName" xmlns:ns17="iqas/servi
```

(그림 7) BPEL 화면

```
ClientApplet {
 Client = new HttpClient( );
 Method = new PostMethod(URL);
 PartnerLinks = BPEL.getPartnerLinkType( );
 while (PartnerLinks = wsdlPartnerLinkTypes) {
  WSDL.getRoles( );
  WSDL.getPortTypes( );
  WSDL.getURLs( );
 }
 Client.executeMethod(Method);
}

ServerServlet {
 InitializeWSGateway( );
 openWsdlFile( );
 openBpelFile( );
 ChannelID = Gateway.findChannel( );
 DeployChannels.add(ChannelID); //SOAP RPC 채널
 Gateway.addService(DeployChannels); //등록
}
```

(그림 8) BPEL 등록 알고리즘

```
ClientApplet {
 Display main menu;
 Enter query statement;
 Retrieve available web services (by server);
 Construct a workflow;
 Convert from a workflow into BPEL4WS (by server);
 Return result;
}

ServerServlet {
 case ("retrieving") {
  Perform matching algorithm;
 }
 case ("converting") {
  Transform a workflow into BPEL4WS;
  Deployed by BPWS4J;
 }
}
```

(그림 9) 클라이언트-서버 프로그래밍 전체 처리 과정

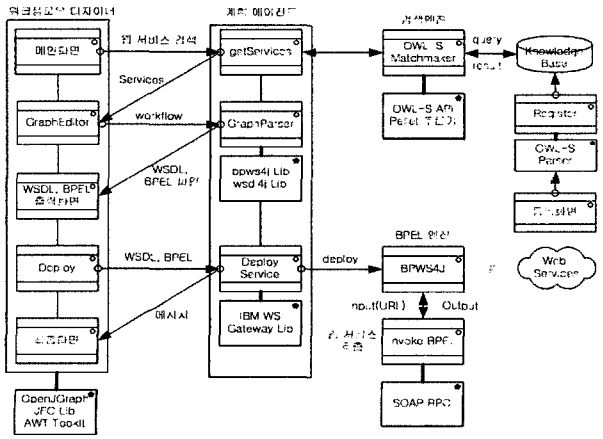
4. 시스템 구현 및 분석

SemanticBPEL 시스템은 자바 언어(JDK1.5)로 구현되었으며, 지원 툴들은 모두 오픈 소스를 선택하였다. 서버 쪽 웹 컨테이너로 아파치 톰캣 5.5를 이용하였고, 웹 서비스 컨테이너로는 아파치 Axis 1.3을 사용하였다. 또한 BPEL4WS 실행을 위해서 IBM의 BPWS4J 엔진이 사용되었으며, 워크플로우 작성 시 요구되는 그래프 구현을 처리하기 위해 opengraph graphing 팩키지[19]가 사용되었다. 마지막으로 OWL-S 파서와 온톨로지 등록 및 탐색을 처리하기 위해 매릴랜드 대학의 OWL-S API 모듈[15]을 이용하고 있다. 이런 툴들을 기반으로 시스템 구현 작업은 IBM의 Eclipse 개발 환경에서 수행되었다.

시스템 전체 프로시저의 자료 흐름도는 (그림 10)과 같다. 워크플로우 디자이너 모듈에는 메인화면, GraphEditor, WSDL/BPEL 출력화면, deploy, 그리고 최종화면으로 구성되어 있으며 OpenJGraph, JFC(Java Foundation Class), AWT(Abstract Window Tookit) 라이브러리를 이용한다. 계획 에이전트 모듈은 getServices, GraphParser, Deploy Service로 구성되어 있으며 bpws4j, wsdl4j, IBM WSGateway 라이브러리를 사용한다. getServices 프로시저는 검색엔진 OWL-S Matchmaker에 의해 웹 서비스들이 검색되며 이 Matchmaker는 OWL-S API 및 Pellet 추론기를 이용한다. 생성된 BPEL 파일은 BPWS4J 엔진에 의해 등록되는데 등록된 웹 서비스 호출은 SOAP RPC 환경 하에서 Invoke BPEL 프로시저를 통해 이루어진다. 마지막으로 웹 서비스 등록업무는 OWL-S Parser를 이용하여 Register 프로시저에서 수행된다.

SemanticBPEL 시스템의 성능을 분석하기 위해 기존에 개발되어 있는 다양한 BPEL 시스템들을 설치·분석하였다. 컴퓨터 환경은 3.20GHz Intel CPU와 MS Window Server 2003 버전에 IBM의 BPWS4J[17], Active Endpoints의 ActiveBPEL[20], 그리고 Stanford 대학의 SDS[6] 시스템을 설치하여 실험하였다.

웹 서비스 조합 자동화 문제는 현재 연구가 활발히 진행되고 있는 분야이므로 아직까지 명확한 기능 설정이나 표준 스펙이 완료되지 않은 상황이다. 따라서 규모나 개발 범위가 상당히 다른 이들 시스템 간 성능 분석은 큰 의미가 없으므로 본 연구에서는 <표 2>와 같이 SemanticBPEL 시스템을 BPWS4J, ActiveBPEL, 그리고 SDS 시스템과 기능 및 구현 기술을 비교 분석하였다.



(그림 10) 프로시저 자료 흐름도

BPWS4J는 IBM alphaWorks로부터 이용가능하며 최초로 개발된 BPEL 엔진으로써 대중적으로 가장 널리 알려져 있다. 워크플로우 디자이너는 텍스트 형태의 다중 윈도우 뷰를 제공하고 있으며, 프로세스 탭에서 트리 방식의 워크플로우를 작성하면 소스 탭에 BPEL 코드가 자동 생성된다. 단점으로는 에디터로 BPEL 파일은 자동으로 만들 수 있으나 WSDL 파일은 수동으로 작성해야 한다. 웹 서비스 등록은 엔진의 관리자 툴을 통해서만 지원되기 때문에 전 과정의 자동화가 불가능하다. 또한 소스 오픈은 제공되지 않고 있으며 현재 개발이 중단된 상태이다.

ActiveBPEL은 상용제품 수준의 오픈 소스 BPEL 엔진이다. 수준 높은 워크플로우 디자이너를 무료로 제공하고 있으며 사용자 매뉴얼 및 튜토리얼을 지원하고 있다. 그러나 BPWS4J와 마찬가지로 디자이너로 BPEL 파일은 자동으로 만들 수 있으나 WSDL 파일은 수동으로 작성해야 하며, 웹 서비스 등록은 BPEL 프로세스에 대한 자세한 정보(예, 파트너 링크 위치정보)를 가지고 있는 프로세스 서술자(descriptor)를 작성한 후 jar 명령을 이용하여 bpr 파일을 만드는 복잡한 과정을 거친다.

SDS는 동적 웹 서비스 발견을 위해 BPWS4J 엔진에 시맨틱 웹 기법을 도입한 확장된 BPEL 시스템이다. 수동으로 웹 서비스를 탐색하는 기존의 방식과는 달리 SDS는 BPWS4J 전방에서 자동 탐색을 위해 중개자 역할을 수행한다. 그러나 SDS는 자동화된 웹 서비스 조합 기능은 지원하지 않는다. 따라서 워크플로우 디자이너나 WSDL 자동 생성 기능도 없다. 웹 서비스 등록도 BPWS4J 방식을 따르므로 관리자 툴을 통해서만 가능하다. 현재 Stanford KSL 홈페이지에서 데모 버전만 제공하고 있으며 오픈 소스에 대한 계획도 아직은 없다.

SemanticBPEL은 BPWS4J와 ActiveBPEL과 비교하여 가

〈표 2〉 유사 시스템 간 기능 및 구현기술 비교

기능	SemanticBPEL	BPWS4J	ActiveBPEL	SDS
시맨틱 사용	Yes	No	No	Yes
디자이너 (사용환경) (라이센스)	GUI Applet 연구용	Text(윈도우 뷰) Eclipse PlugIn 무료	GUI Eclipse PlugIn 상용	GUI(탐색용) Applet 연구용
자동화	전체과정	BPEL 생성	BPEL 생성	BPEL 생성
웹서비스 탐색	자동	수동	수동	자동
워크플로우 관리	동적	정적	정적	없음
등록	자동	수동(관리자 툴)	수동(jar 명령)	수동(관리자 툴)
완전성	부족	보통	우수	미흡
플랫폼	J2EE	J2EE	J2EE	J2EE
오픈 소스	Yes	No	Yes	No
벤더		IBM	Active Endpoints	Stanford 대학
추가 기능	개발중	개발중	중단	개발중

장 큰 차이점은 시맨틱 기법을 도입한 것이다. 다른 시스템들에 비해 유일하게 자동화된 웹 서비스 탐색 및 동적 워크플로우 관리가 가능하고 웹 서비스 등록의 자동화가 가능하다. 또한 웹 서비스 조합 전체 과정의 자동화를 지원하는 것도 특이하다. 한편 SDS는 시맨틱 기법은 도입하고 있으나 웹 서비스 조합 기능이 없기 때문에 워크플로우 디자이너나 워크플로우 관리 부분이 SemanticBPEL에 비해 그 기능이 떨어진다.

5. 결 론

본 논문에서는 자동화된 웹 서비스 조합을 지원하기 위해 BPEL4WS에 OWL-S 기법을 도입하는 혼합 조합기법을 제안하였다. BPEL4WS와 OWL-S 혼합 시스템인 SemanticBPEL의 전체적인 구조를 설계하고, 웹 서비스 탐색 및 통합 알고리즘을 제안하였다. 특히, SemanticBPEL 시스템은 오픈 소스 툴들을 기반으로 구현되었으며 자동화된 웹 서비스 탐색 및 동적 워크플로우 관리가 가능하고 웹 서비스 등록의 자동화가 지원된다. 이 시스템은 기존에 이미 개발되어 있는 다양한 BPEL 시스템들과 기능을 비교하였고 그 우수성을 보여주었다.

하지만 SemanticBPEL은 상용화되어 있는 타 시스템들에 비해 완전성이 부족하여 이미 나와 있는 BPEL 1.1 표준 스펙을 모두 만족시키기 위해서는 많은 테스트 및 추가 모듈의 개발이 필요하다. SemanticBPEL 프로젝트는 현재 진행중에 있으며 향후 실제 상황에서 작동될 때 야기될 수 있는 여러 가지 문제점들을 해결할 수 있는 다양한 기술들을 개발할 예정이다.

참 고 문 헌

[1] W3C, SOAP Version 1.2 Part0: Primer(Second Edition), http://www.w3.org/TR/2006/PER-soap12-part0-20061219/, 2006.

[2] W3C, Web Services Description Language(WSDL) Version 2.0 Part1: Core Language, "http://www.w3c.org/TR/2006/CR-wsdl20-20060327/, 2006.

[3] UDDI, The Evolution of UDDI, UDDI.org White Paper, http://www.uddi.org/, 2002.

[4] Andrews T. et al., Business Process Execution Language for Web Services, Version 1.1, http://www-128.ibm.com/developerworks/ library/specification/ws-bpel/, 2003.

[5] Portier B., SOA terminology overview, Part 1: Service, architecture, governance, and business terms, http://www-128.ibm.com/developerworks/webservices/library/ws-soa-term1/, 2006.

[6] Mandell D. and McIlraith S., Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, in Proceeding of the 2nd International Semantic Web Conference(ISWC2003), Sanibel Island, Florida, 2003.

[7] OWL Services Coalition, OWL-S: Semantic Markup for Web Services, OWL-S White Paper, http://www.daml.org/services/owl-s/1.2/overview/, 2006.

[8] Thakker D., Osman T., and Al-Dabass D., Web Services Composition: A Pragmatic View of the Present and the Future, in Proceedings of 19th European Conference on Modelling and Simulation, Yuri Merkuryev, 2005.

[9] Pistore M., Marconi A., Bertoli P., Traverso P., Automated Composition of Web Services by Planning at the Knowledge Level, International Joint Conference on Artificial Intelligence (IJCAI), 2005.

[10] Osman T., Thakker D., and Al-Dabass D., Bridging the Gap between Workflow andSemantic-based Web Services Composition, WWW Service Composition with Semantic Web Services, 2005.

[11] WfMC, WfMC(Workflow Management Coalition) Strand Documents, Technical Report, http://www.wfmc.org/, 1998.

[12] Leymann P., Web Service Flow Language (WSFL) 1.0, http://www-4.ibm/software/solutions/webservices/pdf/WSFL.pdf, 2002.

[13] Thatte S., XLang: Web Services for Business Process Design, http://www.gotdotnetcom/team/xml_wsspecs/xlang-c/default.htm, 2002.

[14] Antoniou G. and Harmelen F. V., A Semantic Web Primer, MIT Press, 2005.

[15] 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.

[16] Sivashanmugam K., Miller J., Sheth A., and Verma K., Framework for Semantic Web Process Composition, International Journal of Electronic Commerce, Vol.9(2), pp.71-106, 2005.

[17] IBM, BPWS4J, http://www.alphaWorks.ibm.com/tech/bpws4j.

[18] 이용주, 동적 웹 서비스 조합을 위한 시맨틱 웹 서비스 발견 및 실행 기법, 정보처리학회논문지D, 제12-D권 제6호, pp. 889-898, 2005.

[19] Openjgraph, http://sourceforge.net/projects/openjgraph/

[20] Active Endpoints, http://www.active-endpoints.com/

이 용 주

e-mail: yongju@sangju.ac.kr

1985년 한국과학기술원 산업공학과 정보검색전공(석사)

1997년 한국과학기술원 정보및통신공학과 컴퓨터공학전공(박사)

1985년 ~ 1989년 시스템공학연구소 연구원

1987년 ~ 1988년 일본 IBM TRL 연구소 연구원

1989년 ~ 1994년 삼보컴퓨터 근무

1998년 ~ 현재 상주대학교 컴퓨터공학과 부교수

2004년 ~ 현재 국립상주대학교 전산정보원 원장

관심분야: 웹 데이터베이스, 정보검색, 공간 데이터베이스