韓國 컴퓨터情報學會 論文誌
第14卷 第1號, 2009. 1.

20098-14-1-4-6

유비쿼터스 메타서비스 온톨로지 자동 생성을 위한 번역기 개발

이 미 연*, 이 정 원**, 박 승 수***, 조 위 덕**

Development of a Translator for Automatic Generation of Ubiquitous Metaservice Ontology

Meeyeon Lee *, Jung-Won Lee **, Seung Soo Park ***, We Duke Cho **

요 약

유비쿼터스 컴퓨팅 환경에서 실시간의 상황을 고려한 동적인 서비스를 제공하기 위하여 이전 연구를 통해 메타서비스 개념과 기술 규격, 메타서비스 라이브러리 구축 방법을 제안한 바 있다. 하지만, 제안한 프로세스는 각 단계에서 UML, OWL, OWL-S 기반의 분리된 모델을 생성하게 되고 모델 간의 변환을 위한 일정한 체계를 제공하지 못하고 있다. 게다가, 다양한 온톨로지 언어와 온톨로지 편집 도구들, 제안한 메타서비스 규격에 대한 전문가의 개입을 전제로 한다. 본 연구에서는, 비전문가도 일관된 모델을 생성하고 메타서비스 라이브러리를 구축할 수 있도록 OWL 형식의 도메인 온톨로지에서 OWL-S 형식의 메타서비스 라이브러리로의 자동 변환 프로세스를 설계하고 이를 지원할 수 있는 시각 도구를 개발한다. 메타서비스 라이브러리 변환 프로세스는 일관성을 유지하면서 기존의 OWL 모델과 메타서비스 모델을 조합하여 메타서비스 라이브러리에 대한 OWL-S 코드를 자동으로 생성하는 것을 목표로 한다.

Abstract

To provide dynamic services for users in ubiquitous computing environments by considering context in real-time, in our previous work we proposed Metaservice concept, the description specification and the process for building a Metaservice library. However, our previous process generates separated models - UML, OWL, OWL-S based models - from each step, so it did not provide the established method for translation between models. Moreover, it premises aid of experts in various ontology languages, ontology editing tools and the proposed Metaservice specification. In this paper, we design the translation process from domain ontology in OWL to Metaservice Library in OWL-S and develop a visual tool in order to enable non-experts to generate consistent models and to construct a Metaservice library. The purpose of the Metaservice Library translation process is to maintain consistency in all models and to automatically generate OWL-S code for Metaservice library by integrating existing OWL model and Metaservice model.

▸ Keyword : Ubiquitous, Metaservice, Domain Ontology, Service Ontology, OWL(Web Ontology Language), OWL-S(OWL for Services), MDA(Model Driven Architecture), Eclipse

---

• 제1저자 : 이미연 교신저자 : 이정원
• 투고일 : 2008. 12. 18, 심사일 : 2008. 12. 19, 게재확정일 : 2009. 1. 23.
* 이화여자대학교 컴퓨터정보통신공학과 박사과정 ** 아주대학교 정보통신대학 전자공학부 교수
*** 이화여자대학교 공과대학 컴퓨터공학전공 교수
※ 본 연구는 지식경제 프론티어 기술개발사업의 일환으로 추진되고 있는 지식경제부의 유비쿼터스컴퓨팅및네트워크 원천기반기술개발사업의 09C1-T3-10M과제로 지원된 것임

Ⅰ. 서 론

유비쿼터스 컴퓨팅은 사용자의 개입을 최소화하면서 위치, 주변 기기 등에 상관없이 사용자의 의도에 맞는 효율적인 서비스를 제공할 수 있는 환경 구현을 목표로 한다[1, 2, 3, 4]. 유비쿼터스 컴퓨팅의 실현에 있어서 가장 핵심적인 기술은 목적 달성을 위해 실시간에 사용자의 주변 환경 상태를 파악하여 가장 적합한 서비스를 선택하고 각 서비스들을 합성하는 기법이라 할 수 있다. 따라서, 이전 연구[5, 6]에서 유비쿼터스 환경 내의 서비스를 추상적으로 모델링한 개념인 '메타서비스'를 제안하였고 서비스에 대한 정보를 효율적으로 표현하기 위한 메타서비스 기술 규격을 정의하였다. 또한, 실시간에 서비스 탐색이 가능하도록 메타서비스들을 서비스 온톨로지인 메타서비스 라이브러리 내에 의미적으로 계층화 및 구조화하기 위한 프로세스를 정립하였다.

그림 1의 상단의 프로세스와 같이, 최종적으로 메타서비스 라이브러리를 구축하는 과정은 UML의 다이어그램을 통해 특정 도메인을 모델링하여 서비스 기술에 필요한 객체(사용자, 기기)와 기기(device)의 기능(operation)의 이름을 추출하여 도메인 온톨로지를 생성하는 단계로부터 시작된다. 웹 서비스의 입력, 출력, 조합 구성과 같은 기능적 측면을 기술하는 서비스 온톨로지를 효율적으로 구축하기 위해서는 도메인 용어와 속성, 관계 정보와 같은 도메인 지식을 체계적으로 표현하고 있는 도메인 온톨로지가 필요하기 때문이다. 따라서, [5, 6]에서 사용한 메타서비스 라이브러리 구축 프로세스는 OWL (Web Ontology Language)[7] 언어를 사용하여 도메인 온톨로지를 구축한 후, 이를 기반으로 OWL-S (OWL for Services)[8] 언어를 사용하여 메타서비스를 규격에 맞게 기술하고 서비스 온톨로지를 구축하는 과정으로 설계되었다. 그러나 이 프로세스는 각 단계가 분리된 언어와 도구를 통해 분리된 모델을 갖게 됨으로써 각 모델 간의 의미적 일관성과 연결성은 개발자 한 사람에게 의존하게 된다. 또한, 메타서비스 기술 단계에서 특화된 도구의 부재로 인해 메타서비스의 정확하고 용이한 규격화에 한계가 있으며, 개발자는 모델링 언어인 UML, 온톨로지 언어인 OWL, OWL-S와 Rational Rose, Protégé와 같은 편집 도구의 사용에 능숙하여야 하고 메타서비스의 기술 규격도 정확하게 이해하고 있어야 한다.

본 논문에서는 UML, OWL, Metaservice, OWL-S 모델을 분석하고, 이 중에서 OWL 모델인 도메인 온톨로지로부터 OWL-S 모델인 메타서비스 라이브러리를 자동으로 생성할 수 있는 변환 규칙을 정의한다. 이를 기반으로 그림 1의 하단의 프로세스와 같이 개선하고 도구로 구현함으로써 OWL-to-메타서비스 라이브러리 간의 변환의 의미적 손실을 최소화하고 변환을 자동화할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 살펴보고, 3장에서는 이전 연구인 메타서비스와 메타서비스 라이브러리에 대해 소개한다. 4장에서는 도메인 온톨로지와 메타서비스 라이브러리간의 변환 프로세스와 방법을 제시하고, 5장에서 OWL-to-메타서비스 라이브러리 구축 도구의 구현을 다룬 후에 본 논문에서 제시한 방법과 기존 방법의 성능을 비교 평가한다. 마지막으로 6장에서 결론을 맺고 향후 연구방향을 제시한다.

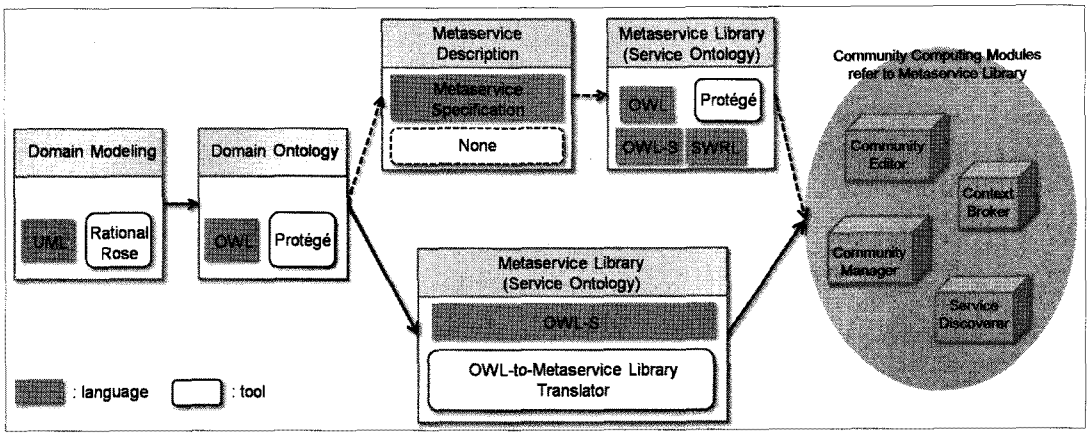

그림 1. 메타서비스 라이브러리 구축 프로세스
Fig. 1. Process for Building the Metaservice Library

II. 관련 연구

2.1 메타서비스 라이브러리

메타서비스 개념과 메타서비스 라이브러리로 온톨로지화 하기 위한 방법을 제안한 이전 연구[5, 6]의 내용 중에서 본 논문과 관련된 사항들을 3장에서 자세하게 설명한다.

2.2 온톨로지 자동 생성

최근, 도메인을 의미적으로 모델링하기 위한 방법으로 도메인 온톨로지와 웹서비스를 효과적으로 기술하기 위한 서비스 온톨로지에 대한 연구가 활발히 진행되면서, OWL이나 OWL-S와 같은 표준 온톨로지 언어가 개발되고 있다. 하지만, 이들은 규격과 문법이 복잡하여 일반 사용자가 직접 기술하기에 어려움이 따른다. 따라서, 다양한 도구들이 지원하고 있을 뿐만 아니라 모델링이 비교적 용이하고 모델링 결과도 비교적 쉽게 파악할 수 있다는 장점으로 인해 이미 널리 사용되고 있는 UML 모델로부터 도메인/서비스 온톨로지를 자동으로 생성하기 위한 연구가 진행되고 있다. 대표적으로 [9], [10]은 그림 2[9]의 변환 과정과 같이, UML 모델링 도구에서 UML 프로파일을 사용하여 다이어그램 모델을 정의하고 메타모델인 XMI (XML Metadata Interchange) 형태로 변환한 후에, 코드 변환 규칙을 정의한 XSLT를 적용하여 OWL 또는 OWL-S 코드를 생성해내는 방법론을 제안하고 있다. 하지만, 이들은 표현 대상이 웹 서비스이고, 웹 서비스의 속성 정보를 표현한 UML의 클래스 다이어그램과 조합 정보를 표현한 순차/협동 다이어그램으로부터 OWL-S의 Service Profile, Service Model, Service Grounding을 자동으로 기술하기 위한 연구들이 대부분이다. 게다가, OWL-S의 단일 프로세스만 기술할 수 있거나 복합 프로세스 중에서 순차적인 흐름만을 자동 기술할 수 있다는 한계점을 해결하지 못하고 있다.

또한, 구매, 지불과 같이 정적이며 순차적인 실행 순서를 갖는 웹 서비스와 달리 유비쿼터스 환경에서는 실행 프로세스를 미리 정의하기 어렵고, 사용자의 목적도 명확하지 않으며 실시간에 발생되는 상황 정보에 따라 서비스가 선택되는 동적 서비스 합성을 지원해야만 한다. 따라서, 이전 연구[5, 6]에서 사용자의 다양하고 모호한 목적 기술과 동적 서비스 합성을 가능케 하기 위한 '메타서비스 라이브러리'라는 온톨로지를 제안한 바 있고, 추가적으로 유비쿼터스 환경을 모델링한 UML 혹은 OWL 모델로부터 웹 서비스가 아닌 유비쿼터스 환경의 서비스 온톨로지를 자동 생성할 수 있는 방법에 대한 연구가 필요하다.

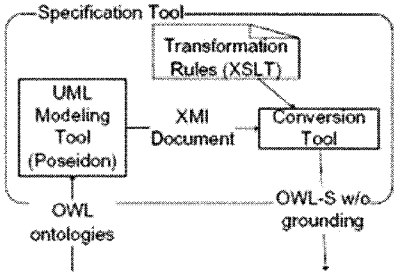

그림 2. UML to OWL-S 변환 과정
Fig. 2. Basic Architecture of the UML to OWL-S Conversion

2.3 MDA(Model Driven Architecture)

MDA는 모델 중심의 소프트웨어 개발 방법론으로, 플랫폼에 독립적인 모델을 정의하고 메타모델을 통한 모델간의 변환을 활용하여 어플리케이션 개발을 자동화하는 패러다임이다[11]. MDA 개발을 목표로 설계된 EMF (Eclipse Modeling Framework)는 XSD(XML Schema Definition)나 UML 과 같은 데이터 모델로부터 어플리케이션을 생성하기 위한 모델링 프레임워크이다[12]. GMF(Graphical Modeling Framework)는 EMF와 GEF 기반의 그래픽 편집기 개발용 프레임워크로서, 이클립스 플랫폼 상에서 단시간에 용이하게 에디터를 개발할 수 있도록 기본 컴포넌트와 런타임 인프라스트럭쳐를 제공한다[13]. GEF (Graphical Editing Framework)는 이클립스 상에서 그래픽 에디터를 생성하기 위한 프레임워크이다[14]. 즉, GMF는 에디터 생성 시 EMF를 기반으로 모델을 정의하고

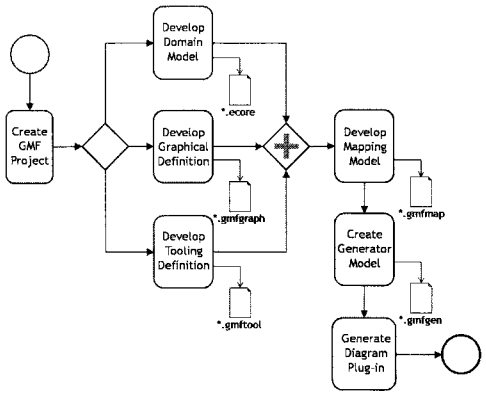

그림 3. GMF의 주요 컴포넌트 및 모델
Fig. 3. Main Components and Models in GMF

GEF를 기반으로 그래피컬한 다이어그램 편집 기능을 구현한다. 그림 3[13]은 GMF 기반으로 도구를 개발할 때 주로 사용되는 컴포넌트와 모델을 보여준다. 개발할 에디터 상의 모델링 요소들을 정의하는 메타모델인 Ecore 모델이 EMF에 의해 자동 생성되고, GMF의 핵심인 그래픽 모델 개념이 GEF의 기본적인 그래픽 요소와 추가적인 정의 모델을 포함하여 생성된다. GMF는 대부분의 코드가 자동 생성되므로 빠른 시간 내에 쉽게 에디터를 생성할 수 있게 할 뿐만 아니라, 재사용과 확장, 유지보수를 용이하게 한다.

III. 메타서비스 라이브러리

3.1 메타서비스

이전 연구를 통해 정의한 '메타서비스'는 커뮤니티 컴퓨팅이 목표로 하는 동적/자율적인 서비스의 지속적인 제공을 지원하기 위해 제안된 개념이다.

커뮤니티 컴퓨팅은 유비쿼터스 환경 내에 존재하는 다양한 기기들을 '커뮤니티'로 구성하여 사용자에게 서비스를 제공하고자 하는 컴퓨팅 모델이다. '커뮤니티'는 사용자의 목적 달성을 위해 필요한 기능을 수행할 수 있는 기기들의 그룹을 의미하며 CDL(Community Description Language) 기반의 커뮤니티 템플릿으로 표현되고, 커뮤니티 멤버들의 협력을 통해 최종 서비스가 제공된다[15]. 이전 연구에서는 실시간의 가변적인 상황(context)을 고려하여 필요한 서비스를 동적으로 검색하고 가장 적합한 서비스를 선택하여 바인딩하기 위한 기법으로 메타 서비스라는 개념을 제안하였다.

'메타 서비스'는 다양한 기기들의 기능(operation)을 추상화한 것으로서, 제공 가능한 기기, 위치, 실행 조건, 효과 등과 같은 기능에 대한 정보를 기술하고 계층적으로 구조화되는 단위이다. 메타서비스는 커뮤니티 템플릿 작성 시에 커뮤니티의 목표(goal) 달성에 필요한 역할(role)을 다양한 추상화 레벨로 기술하기 위해 사용된다. 즉, '보일러를 켜라'와 같이 필요한 서비스(역할)를 정확하게 알고 있을 경우에는 구체적인 서비스를 명시할 수 있다. 또한, 그렇지 않은 경우에 '실내 온도를 조절하라'와 같이 서비스를 추상적으로 명시하더라도 실행 시의 주변 상황과 각 메타서비스의 특성, 메타서비스 라이브러리의 구조에 따라 사용자에게 적합한 서비스가 동적으로 합성되어 제공될 수 있다. 서비스의 동적 합성(Dynamic Composition)을 가능케 할 수 있도록 메타서비스는 추상화 정도에 따라 3가지 타입으로 분류되고 표 1의 규격에 따라 기술된다.

표 1. 메타서비스 규격
Table 1. Metaservice Specification

메타서비스 규격		타입	설명
SID		*S.ScenarioNum.ServiceNum*	식별자
ServieceName		*String*	이름
SceneNo		*Integer*	추출된 상황 번호
AbsLevel		*{0, 1, 2}*	추상화 레벨
Child		*opt(SIDi, SIDj, ..., SIDn) // none*	자식 메타서비스들
Object		*{obj1, obj2, ..., objn}*	해당 메타서비스를 제공할 수 있는 플랫폼 혹은 기기
TerminatingService		*SIDk // none*	수행 중인 메타서비스를 종결할 수 있는 메타서비스를 명시
Precondition	Status	*objk.Status // none*	메타서비스를 실행할 기기의 상태
	Location	*?location*	메타서비스가 제공될 위치
	Priority	*?pvalue*	우선 순위
	Input	*{in1, in2, ..., inn}*	메타서비스 실행에 필요한 입력 값
Effect	Environmental/Functional	*{eff1, eff2, ..., effn}*	메타서비스 실행 후의 환경적 영향과 기능적 결과
	Device	*objk.Status // none*	메타서비스 실행 후의 기기의 상태
	Output	*?outputdata*	결과물

- Abstract 메타서비스 : 가장 추상화된 레벨의 서비스로서, 규격의 Environmental/Functional Effect에 의해 분류된다.
- Composite 메타서비스 : 중간 레벨의 서비스로서, 여러 층을 가질 수 있다.
- Atomic 메타서비스 : 가장 구체적인 서비스로서 실제 기기들의 기능이 이 레벨에 해당한다.

메타서비스의 타입 분류에 기반하여 그림 4와 같이 메타서비스를 계층화함으로써 다양한 목적 기술과 동적 합성을 가능케 할 수 있다. 또한, 유비쿼터스 환경의 이질성과 이동성으로 인해 복잡하고 변수가 많은 상황에서 목적을 달성하기 위해 서비스가 유연하게 조합될 수 있다.

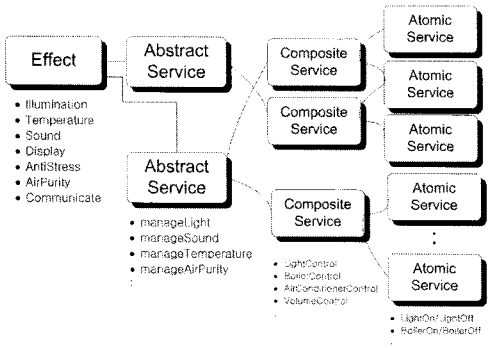

그림 4. 메타서비스 계층
Fig. 4. Metaservice Layer

3.2 메타서비스 라이브러리-서비스 온톨로지

메타서비스들을 저장 및 관리하는 구조는 메타서비스의 필수적인 정보인 규격을 기술할 수 있어야 하고 특히, 효율적인 탐색이 가능하도록 규격에 포함되어 있는 관계 정보, 즉 AbsLevel과 Child 정보를 표현할 수 있어야 한다. 따라서, 메타서비스들을 OWL-S 언어로 기술하여 메타서비스 라이브러리를 구성하였다. OWL-S는 웹 서비스를 의미적으로 기술하고 웹 서비스의 자동 발견/호출/합성/상호 운용 등을 지원하기 위해 제안된 W3C 표준 언어이다[8, 16]. OWL-S는 웹 서비스의 기능적 특성을 IOPE(Input / Output / Precondition / Effect)의 4가지 요소로 명세하고 기본 단위의 프로세스를 흐름에 맞게 조합하여 추상 프로세스를 표현할 수 있도록 지원한다. OWL-S를 통해 표 1의 규격 사항을 표현하고 메타서비스의 계층 관계를 명시할 수 있다.

OWL-S로 기술된 메타서비스들의 집합인 메타서비스 라이브러리는 커뮤니티 컴퓨팅의 주요 모듈인 커뮤니티 매니저(CM; Community Manager), 커뮤니티 에디터(CE; Community Editor), 컨텍스트 브로커(CB; Context Broker), 서비스 디스커버러(Service Discoverer)에 의해 참조됨으로써, 커뮤니티 실행 시에 서비스의 동적 합성과 대체 서비스 발견을 통해 자율적인 커뮤니티 구성을 가능케 할 뿐만 아니라 목적 달성도 보장할 수 있다. 따라서, 이러한 주요 모듈이 참조하게 되는 메타서비스 온톨로지를 그림 1의 하단의 프로세스와 같이 기존의 UML, OWL 모델로부터 체계적으로 용이하게 자동 생성해낼 수 있도록 방법과 도구를 제공하여야 한다.

IV. OWL-to-메타서비스 라이브러리 변환 프로세스의 설계

4.1 요구 사항 분석

커뮤니티 컴퓨팅의 실현을 위한 핵심 기반이라고 할 수 있는 메타서비스 라이브러리의 구축 과정을 자동화하기 위해서는 크게 모델 간의 번역과 사용자의 입력을 반영할 수 있는 다이어그램 기반의 모델링 방법이 필요하다.

메타서비스 라이브러리 구축 과정에서 사용되는 모델 간의 자동 번역 프로세스의 필요성은 크게 3가지로 정리할 수 있다. 첫째로, 기존의 메타서비스 라이브러리 구축 방법은 각 단계별로 서로 다른 모델을 사용하게 된다. 즉, 표 2의 유비쿼터스 도메인을 모델링하는 UML 요소, 표 3의 도메인 온톨로지의 OWL 규격, 표 4의 메타서비스 기술에 사용되는 OWL-S 규격 간의 변환은 개발자의 수동 작업에 의해 이루어지기 때문에, 모델 간의 일관성과 연결성을 보장하기 어렵다. 예를 들어, 유비쿼터스 환경 내에서 온도 조절을 위한 기능 중의 하나인 '보일러를 가동한다'는 UML 모델에서 표 2에 따라 '보일러'라는 'Object' 와 매니저 모듈 'Object'간의 'Relation'을 통한 'Message'로 표현되고, OWL 모델에서는 표 3에 따라 '보일러 Class'와 '보일러를 가동한다'라는 'Object Property'로 표현된다. 메타서비스 라이브러리 내에서는 기능을 서비스화하고 이름만 있었던 기술 부분을 확장하기 위하여 OWL-S 언어를 사용하여 14가지 규격 항목으로 상세히 기술하여야 한다. 따라서, 개발자의 수동 작업으로 수행될 경우에 발생할 수 있는 오류와 의미적 손실을 최소화하기 위해 모델 간의 자동 변환 방법이 필요하다.

둘째, 메타서비스 라이브러리 개발자는 메타서비스의 각 규격이 표현하는 사항을 파악하고 있어야 한다. 또한, 메타서비스 라이브러리는 웹 서비스에 특화되어 있는 OWL-S의 규격 항목 중에서 일부만을 사용하기 때문에, 기존의 온톨로지 에

표 2. 도메인 모델링에 사용되는 UML 협동 다이어그램의 표현 요소
Table 2. UML Collaboration Diagram Elements for Domain Modeling

대상	UML Element	UML Symbol
엔티티(사용자, 기기, 모듈 등)	Object	:ObjectName
엔티티들 간 상호 작용	Relation	
서비스 이름	Message	Number: ServiceName()
데이터 파라미터	Data	(DataName)

표 3. 도메인 온톨로지의 표현 요소
Table 3. Domain Ontology Elements

대상	OWL Element
엔티티	owl:Class
엔티티의 속성	owl:ObjectProperty
엔티티들 간의 관계	owl:DatatypeProperty
메타서비스 이름 리스트	owl:ObjectProperty

표 4. 메타서비스 기술에 사용되는 OWL-S 요소
Table 4. OWL-S Elements for Metaservice Description

대상 (메타서비스 규격)	Tag Element in OWL-S
AbsLevel	〈process:AtomicProcess〉 〈process:CompositeProcess〉
Child	〈process:Choice〉 〈process:Sequence〉 〈process:AnyOrder〉
Device	〈process:hasResult〉
Environmental / Functional	〈process:hasResult〉
Input	〈process:hasInput〉
Location	〈process:hasInput〉
Object	〈process:hasPrecondition〉
Output	〈process:hasOutput〉
Priority	〈process:hasInput〉
SceneNo	〈rdfs:comment〉
ServiceName	〈process:AtomicProcess〉 〈process:CompositeProcess〉
SID	〈rdfs:comment〉
Status	〈process:hasPrecondition〉
TerminatingService	〈rdfs:comment〉

디터를 사용하여 메타서비스를 기술하기 위해서는 표 4에 보인 OWL-S의 규격 규격 항목과 메타서비스 기술에 사용되는 항목의 매핑 관계를 미리 알고 있어야 한다.

마지막으로, 메타서비스 라이브러리를 구축하기 위한 과정은 모델링과 온톨로지에 대한 많은 전문 지식을 필요로 한다. 대표적인 모델링 언어인 UML과 도메인 온톨로지 언어인 OWL, 웹 서비스 온톨로지 언어인 OWL-S, 시맨틱 웹 규칙 언어인 SWRL의 규격과 문법을 알아야 하고 에디터인 Rational Rose와 Protégé의 사용에 능숙해야 한다.

또한, UML-OWL-Metaservice(OWL-S)간의 변환 프로세스를 자동화하기 위한 도구를 다이어그램 기반의 편집기로 구현해야 할 필요성은 다음과 같다. 첫째, 각 변환 단계는 전 단계의 모델을 서비스 차원에서 더 상세한 기술이 가능하도록 확장하기 때문에 필요한 추가 정보를 개발자가 쉽게 입력할 수 있도록 사용자 친화적인 인터페이스 환경을 제공해야 한다. 즉, UML-to-OWL 변환 과정에서 UML 모델에 명시적으로 표현되어 있지 않은 정보를 개발자로부터 입력받아 도메인 온톨로지에 반영할 수 있어야할 뿐만 아니라, OWL-to-메타서비스 라이브러리 변환 과정에서는 메타서비스 규격 항목에 대한 사용자 입력이 필수적이기 때문이다.

둘째, 메타서비스 라이브러리 내에서 메타서비스들은 기본적으로 계층 관계(hierarchy)를 이룬다. 하지만 기존의 온톨로지 에디터는 개발자에게 서비스의 조합 상태를 효율적으로 보여주지 못한다. 즉, Protégé와 같은 도구에서는 합성 서비스(composite service)의 하위 서비스들을 하나의 뷰에 보여주지 않는다. 예를 들어, 서비스 A-B-C의 연결 관계를 서비스 A에 대한 A-B 관계와 서비스 B에 대한 B-C의 관계로 분리하여 보여주기 때문에 편집하기에 번거로울 뿐만 아니라 전체 구조를 파악하기가 쉽지 않다. 따라서, 다양한 계층을 이루는 메타서비스 라이브러리 내의 구조를 생성하고 편집하기에 적합한 시각적인 환경이 필요하다.

4.2 OWL-to-메타서비스 라이브러리 변환 프로세스

OWL 기반의 도메인 온톨로지로부터 OWL-S 기반의 메타서비스 라이브러리로의 자동 변환 프로세스는 그림 5에서와 같이 크게 두 단계로 이루어진다.

4.2.1 1단계: 메타서비스 모델 및 변환 관계 정의

첫 번째 단계에서는 EMF와 GMF를 통해 모델 간 변환을 용이하게 하고 비주얼 환경을 구현하기 위해 메타서비스에 대한 XML 스키마 모델을 정의한다. 메타서비스 XSD 모델은 EMF를 통해 Ecore 모델로 변환되고, 메타서비스에 대한 GMF의 그래픽 요소의 사용자 입력 사항이 XSD 모델 기반의

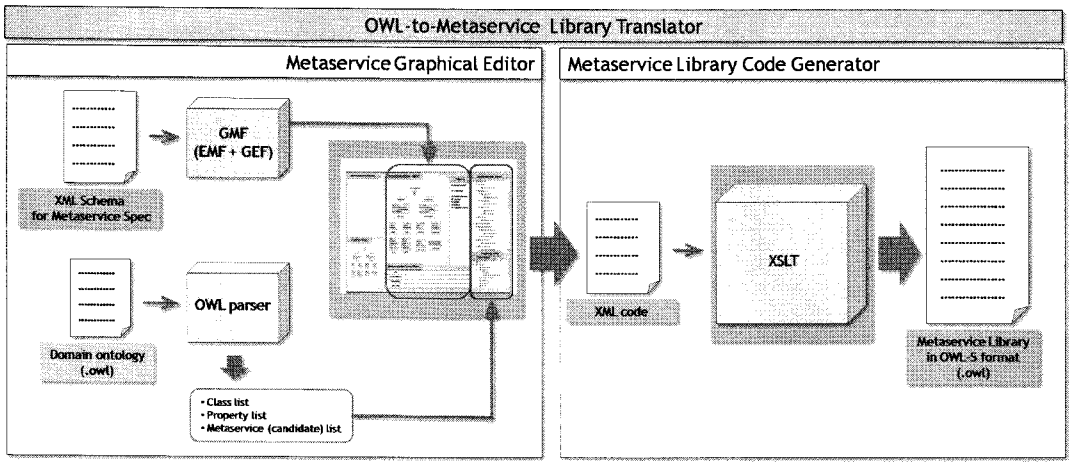

그림 5. OWL-to-메타서비스 라이브러리 변환 프로세스
Fig. 5. Translation Process of OWL-to-Metaservice Library

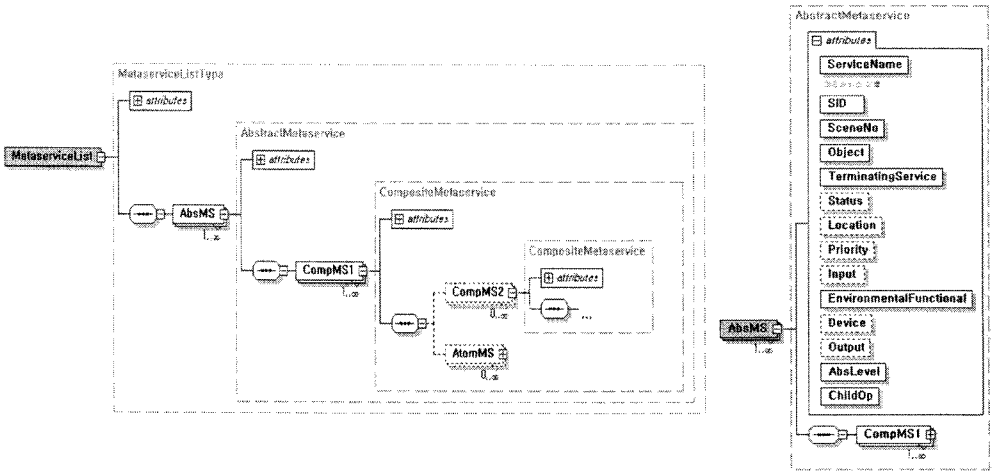

그림 6. 메타서비스의 XML 스키마 모델
Fig. 6. XML Schema Model for Metaservice

XML 코드로 변환된다. 메타서비스를 위한 XML 스키마 모델을 그림 6과 같이 정의하였다. 메타서비스의 3가지 타입에 대해, Abstract와 Composite 메타서비스는 14가지, Atomic 메타서비스는 자식 정보를 표현하는 Child 항목을 제외한 13가지 규격 항목에 따라 기술되어야 한다. 또한, 각 메타서비스는 메타서비스 라이브러리 내에서 다음과 같은 제약 조건에 따라 구조화되어야 하기 때문에 XML 스키마에 반영하여 설계하였다.

- 메타서비스 라이브러리 내의 모든 메타서비스는 Abstract 메타서비스를 루트로 가져야 한다.
- Abstract 메타서비스는 Composite 메타서비스만 자식으로 가질 수 있다.
- Composite 메타서비스는 다른 Composite 메타서비스와 Atomic 메타서비스를 자식으로 가질 수 있다.
- Atomic 메타서비스는 자식을 가질 수 없다.

이 XSD 모델로부터 EMF를 통해 그림 7의 Ecore 모델이 자동 생성되고, 각 요소는 GMF를 통해 그래픽적 요소로 변환되어 사용자에게 편집하기 쉽게 제공된다. XSD 모델과 Ecore 모델, 그래픽적 요소 간의 변환매핑 관계는 표 5에 보인다.

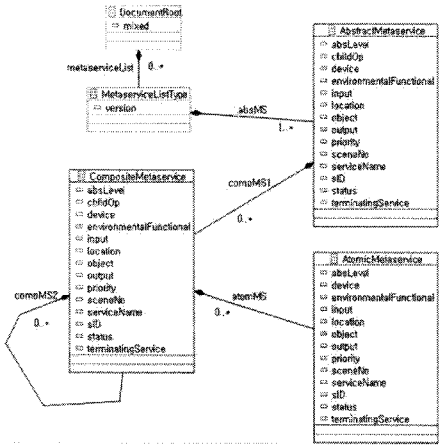

그림 7. 메타서비스의 Ecore 모델
Fig. 7. Ecore Model for Metaservice

4.2.2 2단계: 메타서비스 라이브러리를 위한 OWL-S 온톨로지 자동 생성

두 번째 단계에서는 사용자의 추가 정보로 완성된 메타서비스 라이브러리에 대한 XML 코드를 표 4의 매핑 관계에 기반하여 OWL-S 코드로 자동 변환하기 위한 변환 알고리즘을 그림 8과 같이 제안한다. 메타서비스 타입에 따라 Abstract와 Composite 메타서비스에 대해 Composite process를 생성하고 Child 항목을 추가하며(①), Atomic 메타서비스에 대해 Atomic process를 생성하고(②), 공통적인 11개의 규격 항목에 대해 변환을 수행한다(TranslatingCommon() 함수). 변환 규칙은 XSLT로 정의하는데, XSLT(eXtensible Stylesheet

```
procedure Generating_OWL-S_code (InD: XML doc)          // input : XML code
return OutD: OWL-S doc                                  // output : OWL-S code
begin
  while (!EndOfDoc(InD)) do
    begin
      for (each element) do
        begin
          switch (typeOfMS)
            {
        ①   case <AbsMS> or <CompMS> : // for Abstract/Composite Metaservice
            { write <CompositeProcess> element with ServiceName attribute in OutD;
              D = TranslatingCommon(element);
              write <process:Choice> or <process:Sequence> or <process:AnyOrder>
                element in OutD; }

        ②   case <AtomMS> :                             // for Atomic Metaservice
            { write <AtomicProcess> element with ServiceName attribute in OutD;
              D = TranslatingCommon(element); }
            }
        end
        write D in OutD;
    end
end

procedure TranslatingCommon (Element)          // input : each XML element
return D: OWL-S code                           // output : OWL-S code for 11 common specs
begin
  for (each attributes in Element) do
    begin
      switch (attribute)
      { case (EnvironmentalFunctional) :
          case (Device) :
            write <process:hasResult> element in D;
          case (Input) :
          case (Location) :
          case (Priority) :
            write <process:hasInput> element in D;
          case (Object) :
          case (Status) :
            write <process:hasPrecondition> element in D;
          case (Output) :
            write <process:hasOutput> element in D;
          case (SceneNo) :
          case (SID) :
          case (TerminatingService) :
            write <rdfs:comment> element in D;       }
    end
end
```

그림 8. OWL-S 온톨로지 생성 알고리즘
Fig. 8. Algorithm for Generating OWL-S Ontology

Language Transformation)는 특정 구조를 갖는 XML 문서를 다른 구조의 XML 문서로 변환하는 규칙을 정의하는 표준적인 방법이다[17]. 이 단계에서 정의한 XSLT 스크립트는 OWL-to-메타서비스 라이브러리 자동 변환의 핵심으로, 도메

표 5. 메타서비스 XSD 모델-Ecore 모델-그래픽 요소 간의 변환 관계
Table 5. Mapping Metaservice XSD Model - Ecore Model - Graphical Element

대상		XSD 모델	Ecore 모델	그래픽 요소
Abstract 메타서비스		〈AbsMS〉	EClass	Node
Composite 메타서비스		〈CompMS〉	EClass	Node
Atomic 메타서비스		〈AtomMS〉	EClass	Node
메타서비스 규격 (14가지 항목)	Child	Attribute	Aggregation & EAttribute	Connection & Node의 Property
	AbsLevel, Device, Environmental / Functional, Input, Location, Object, Output, Priority, SceneNo, ServiceName, SID, Status, TerminatingService		EAttribute	Node의 Property

인 온톨로지와 메타서비스 XML 코드를 조합하여 서비스 온톨로지인 OWL-S 코드를 자동 생성한다. 그림 9는 XSLT 스크립트의 일부로서, 기존의 온톨로지 편집 도구와도 연동할 수 있는 정확한 OWL-S 문서를 생성한다. ⓐ는 후보 메타서비스의 이름 리스트를 제외한 도메인 온톨로지의 내용을 가져오는 부분이고, 나머지 부분은 Atomic 프로세스의 13개 규격 항목에 대한 OWL-S 코드를 생성해내는 규칙이다.

```
<xsl:template match="owl:Class">
    <xsl:copy-of select='.'/>
</xsl:template>
<xsl:template match="owl:DatatypeProperty">
    <xsl:copy-of select='.'/>
</xsl:template>
<xsl:template match="owl:FunctionalProperty">
    <xsl:copy-of select='.'/>
</xsl:template>
<xsl:template match="owl:ObjectProperty">
    <xsl:if test="@rdf:ID='OtherObjectProperty'">
        <xsl:copy-of select='.'/>
    </xsl:if>
    <xsl:if test="rdfs:subPropertyOf/@rdf:resource='#OtherObjectProperty'">
        <xsl:copy-of select='.'/>
    </xsl:if>
</xsl:template>

<process:AtomicProcess rdf:ID="{@ServiceName}">
    <xsl:variable name="refResult" select="concat('#Result_', substring(@SID, 5, 3))"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
            SID=<xsl:value-of select="@SID"/>
            SceneNum=<xsl:value-of select="@SceneNo"/>
            TerminatingService=<xsl:value-of select="@TerminatingService"/>
</rdfs:comment>
    <process:hasInput>
        <process:Input rdf:ID="pvalue"/>
    </process:hasInput>
    <process:hasInput>
        <process:Input rdf:ID="location"/>
    </process:hasInput>
    <process:hasResult rdf:resource="{$refResult}"/>
    <xsl:if test="contains(@EnvironmentalFunctional,',')">
        <xsl:variable name="refResult2" select="concat($refResult, '0')"/>
        <process:hasResult rdf:resource="{$refResult2}"/>
        <xsl:variable name="eff22" select="substring-after(@EnvironmentalFunctional,',')"/>
        <xsl:if test="contains($eff22,',')">
            <xsl:variable name="eff23" select="substring-after($eff22,',')"/>
            <xsl:variable name="refResult3" select="concat($refResult2, '0')"/>
            <process:hasResult rdf:resource="{$refResult3}"/>
        </xsl:if>
    </xsl:if>
    <xsl:variable name="swrl" select="concat('#SWRL-Condition_', substring(@SID, 5, 3))"/>
    <process:hasPrecondition rdf:resource="{$swrl}"/>
</process:AtomicProcess>
```

ⓐ

그림 9. OWL-to-메타서비스 라이브러리 변환을 위한 XSLT 스크립트 (일부분)
Fig. 9. XSLT Script for OWL-to-Metaservice Library Translation

V. OWL-to-메타서비스 라이브러리 번역기 개발 및 적용 결과

4장의 설계를 바탕으로 OWL-to-메타서비스 라이브러리 변환 프로세스를 자동화하기 위한 도구를 구현한다. 그림 5에 보였듯이, 변환의 두 단계를 다이어그램 기반의 메타서비스 편집기와 메타서비스 라이브러리 코드 생성기를 통해 수행하게 된다.

5.1 다이어그램 기반의 메타서비스 편집기

OWL-to-메타서비스 라이브러리 생성 도구의 첫 번째 모듈인 그래픽 에디터는 메타서비스 라이브러리 개발자에게 도메인 온톨로지의 정보를 참조하기 쉽게 보여주고 메타서비스를 트리 구조로 비주얼하게 구조화하고 규격 정보를 입력할 수 있는 환경을 제공한다. 그림 10에 보인 메타서비스 편집기의 사용자 인터페이스는 다음의 7개 영역으로 구성된다.

① Package explorer view : 현재 작업 중인 메타서비스 라이브러리와 관련된 파일들을 보여준다. 도메인 온톨로지의 owl 파일을 임포트할 수 있다.
② Canvas : 메타서비스 그래픽 모델링 요소들을 그리고 편집할 수 있는 영역이다.
③ Palette : 그래픽 모델링 요소들인 노드와 링크를 보여준다.
④ Metaservice Candidate view : 도메인 온톨로지인 owl 파일로부터 OWL 파서를 사용하여 오브젝트 프로퍼티 중에서 후보 메타서비스의 이름 리스트를 추출하여 계층 구조를 유지한 채 보여준다.
⑤ Class/Object property/Datatype property view : 도메인 온톨로지인 owl 파일로부터 OWL 파서를 사용하여 클래스 정보와 후보 메타서비스를 제외한 나머지 오브젝트 프로퍼티, 데이터타입 프로퍼티를 추출하여 계층 구조를 유지한 채 보여준다.
⑥ Metaservice Specification view : 메타서비스의 규격 항목을 편집할 수 있는 영역이다.
⑦ Outline view : ②의 캔버스 상에 드로잉한 메타서비스 라이브러리 다이어그램의 전체 모습을 축소하여 보여준다.

5.2 메타서비스 라이브러리 코드 생성기

OWL-to-메타서비스 라이브러리 생성 도구의 두 번째 모듈인 코드 생성기는 다이어그램 기반의 메타서비스 편집기가 생성한 XML 코드의 내용을 OWL-S로 변환 및 확장하고 도메인 온톨로지 코드와 조합하여 서비스 온톨로지인 메타서비스 라이브러리를 구축하는 프로세스를 자동 수행한다. OWL-S 온톨로지 생성 알고리즘을 기반으로 3가지 타입의 메타서비스를 OWL-S의 process로 생성하고 14가지 메타서비스 규격 항목을 표 4의 매핑 관계에 따라 해당 태그 요소로 변환한다.

5.3 적용 및 평가

기존 방법과의 비교를 위해 [5, 6]에서 구축한 메타서비스 라이브러리의 일부를 본 연구에서 제안한 OWL-to-메타서비스

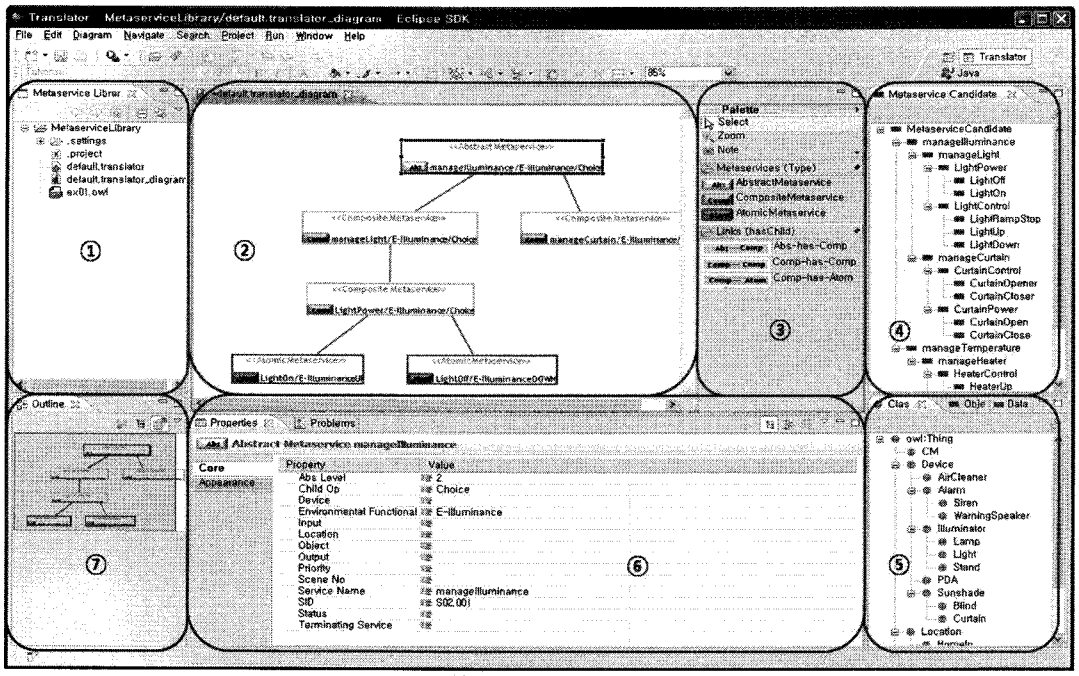

그림 10. 메타서비스 그래픽 에디터의 사용자 인터페이스
Fig. 10. User Interface of Metaservice Graphical Editor

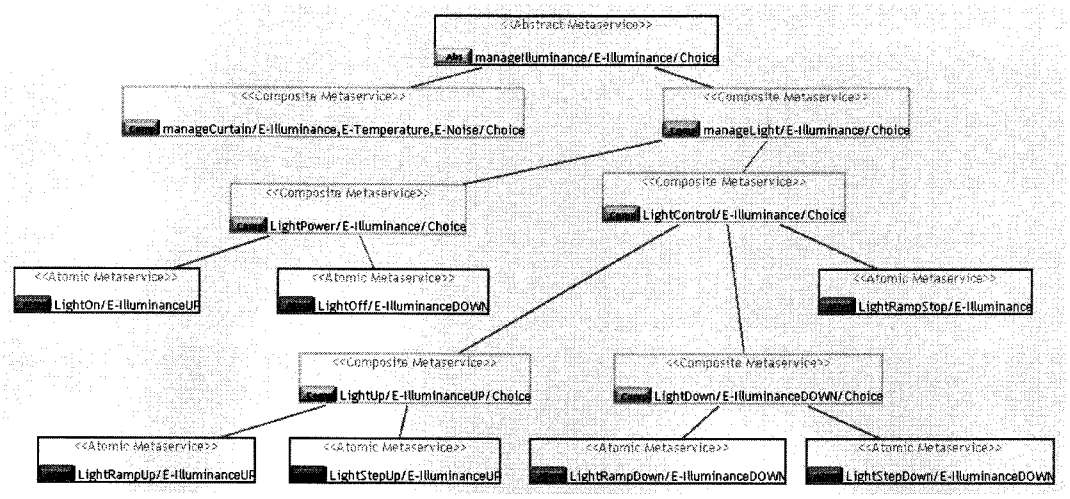

그림 11. 메타서비스 그래픽 요소로 편집한 메타서비스 샘플 (일부분)
Fig. 11. Metaservice Sample using Metaservice Graphical Element

라이브러리 변환 프로세스를 적용하여 생성하고 결과를 비교 검증해보았다. 그림 11은 변환 프로세스의 1단계에 해당하는 다이어그램 기반의 메타서비스 편집기 상에서 그래픽 요소를 사용하여 편집한 메타서비스 라이브러리 트리의 예이다. 3가지 타입의 메타서비스와 Child 관계를 명확하게 파악할 수 있고, 메타서비스 규격을 쉽게 입력할 수 있다. 그림 12는 그림 11의 편집 내용에 대해 자동 생성된 XML 코드로, 메타서비스 XSD 모델 구조를 따른다. 그림 13은 변환 프로세스의 2단계를 수행한 결과로서, 그림 12에 메타서비스 라이브러리 코드 생성기의 XSLT 변환 규칙을 적용하여 자동 생성한 OWL-S 코드이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<MetaserviceList>
  <AbsMS ChildOp="Choice" EnvironmentalFunctional="B-Illuminance"
Location="location" Priority="1" SceneNo="1" ServiceName="manageIlluminance"
SID="S02.001">
    <CompMS1 ChildOp="Choice" EnvironmentalFunctional="B-Illuminance"
Location="location" Object="Illuminator" Priority="1" SceneNo="1"
ServiceName="manageLight" SID="S02.002" Status="isLocatedIn(illuminator, location)"
TerminatingService="LightOff">
      <CompMS2 EnvironmentalFunctional="B-Illuminance" Location="location"
Object="Illuminator" Priority="1" SceneNo="1" ServiceName="LightPower"
SID="S02.003" Status="isLocatedIn(illuminator, location)"
TerminatingService="LightOff">
        <AtomMS Device="hasStatus(illuminator, &amp;quot;ON&amp;quot;)"
EnvironmentalFunctional="B-IlluminanceUP" Location="location" Object="Illuminator"
Priority="1" SceneNo="1" ServiceName="LightOn" SID="S02.004"
Status="isLocatedIn(illuminator, location)&amp;amp;hasStatus(illuminator,
&amp;quot;OFF&amp;quot;)" TerminatingService="LightOff"/>
        <AtomMS Device="hasStatus(illuminator, &amp;quot;OFF&amp;quot;)"
EnvironmentalFunctional="B-IlluminanceDOWN" Location="location"
Object="Illuminator" Priority="1" SceneNo="1" ServiceName="LightOff"
SID="S02.005" Status="isLocatedIn(illuminator, location)&amp;amp;hasStatus(illuminator,
&amp;quot;ON&amp;quot;)" TerminatingService="LightOff"/>
      </CompMS2>
      <CompMS2 ChildOp="Choice" EnvironmentalFunctional="B-Illuminance"
Location="location" Object="Illuminator" Priority="1" SceneNo="1"
ServiceName="LightControl" SID="S02.006" Status="isLocatedIn(illuminator,
location)&amp;amp;hasStatus(illuminator, &amp;quot;ON&amp;quot;)"
TerminatingService="LightOff">
        <CompMS2 EnvironmentalFunctional="B-IlluminanceUP" Location="location"
Object="Illuminator" Priority="1" SceneNo="1" ServiceName="LightUp" SID="S02.007"
Status="isLocatedIn(illuminator, location)&amp;amp;hasStatus(illuminator, &amp;quot;ON&amp;quot;)"
TerminatingService="LightOff">
.....
```

그림 12. XML 코드 샘플 (일부분) : 다이어그램 기반의 메타서비스 편집기의 최종 결과물
Fig. 12. XML Code Sample : the Result from Metaservice Graphical Editor

```
.....
<process:CompositeProcess rdf:ID="manageIlluminance">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
              SID=S02.001
              SceneNum=1</rdfs:comment>
  <process:hasInput>
    <process:Input rdf:ID="location"/>
  </process:hasInput>
  <process:hasResult rdf:resource="#Result_001"/>
  <process:composedOf rdf:resource="#Choice_001"/>
</process:CompositeProcess>
<process:Choice rdf:ID="Choice_001">
  <process:components>
    <process:ControlConstructBag rdf:ID="ControlConstructBag_001">
      <list:first>
        <process:Perform rdf:ID="Perform_001">
          <process:process rdf:resource="#manageLight"/>
        </process:Perform>
      </list:first>
      <list:rest>
        <process:ControlConstructBag rdf:ID="ControlConstructBag_0010">
          <list:first>
            <process:Perform rdf:ID="Perform_0010">
              <process:process rdf:resource="#manageCurtain"/>
            </process:Perform>
          </list:first>
          <list:rest rdf:resource="http://www.daml.org/services/owl-
s/1.2/generic/ObjectList.owl#nil"/>
        </process:ControlConstructBag>
      </list:rest>
    </process:ControlConstructBag>
  </process:components>
</process:Choice>
<process:Result rdf:ID="Result_001">
  <process:hasEffect>
    <expr:SWRL-Condition rdf:ID="B-Illuminance"/>
  </process:hasEffect>
</process:Result>
.....
```

그림 13. 생성된 OWL-S 코드 샘플 (일부분)
Fig. 13. Generated OWL-S Code Sample

기존의 메타서비스 라이브러리 구축 방법과의 성능 비교를 위해 정량적 평가 기준으로 생성된 OWL-S 코드의 길이를, 비정량적 기준으로 메타서비스 라이브러리 구축 프로세스의 일관성 유지 가능 여부, 메타서비스 라이브러리 구축을 위해 개발자가 미리 알고 있어야 하는 언어와 사용 도구, 구축의 용이성 및 방법을 선정하였고, 평가 결과는 다음과 같다.

- 자동 변환 프로세스를 통해 생성된 그림 13의 OWL-S 코드가 도구 개발 전 Protege에서 직접 작업한 OWL-S 코드보다 길이가 약 18% 감소됨을 확인하였다. 즉, 메타서비스 기술에 필요하지 않고 일관성을 떨어뜨릴 수 있는 〈profile:Profile〉, 〈service:supportedBy〉, 〈grounding:〉과 같은 불필요한 내용 없이 실제로 메타서비스 기술 규격이 필요로 하는 내용만을 포함하는 효율적인 코드 생성이 가능하다.
- 기존의 메타서비스 라이브러리 구축 방법에서는 각 단계별로 UML, OWL, 메타서비스(OWL-S)의 분리된 모델을 수동 작업으로 정의하는데 반해, 정형화된 OWL-to-메타서비스 라이브러리 변환 프로세스는 모델 간의 변환이 자동으로 이루어지며 개발자의 별도의 노력 없이 일관성 유지가 가능하다.
- Protégé와 같은 기존의 온톨로지 편집 도구를 사용할 경우와 달리, 번역기를 사용하면 메타서비스 규격, OWL, OWL-S, SWRL 언어에 능숙하지 않은 개발자도 라이브러리를 구축할 수 있다.
- 다이어그램 기반 편집기를 통해 하나의 화면 내에서 도메인 온톨로지의 내용을 참조할 수 있고 하나의 canvas 상에 모든 메타서비스들을 트리 구조로 구조화할 수 있기 때문에 작성이 용이하다.

결론적으로, 본 연구에서 제안한 OWL-to-메타서비스 라이브러리 간의 변환 프로세스는 정형화된 변환 체계를 제공함으로써 일관된 모델을 자동 생성할 수 있을 뿐만 아니라, 비전문가에게 비주얼한 구현 환경을 제공함으로써 추가 정보를 용이하게 편집할 수 있고 단기간에 검증된 메타서비스 라이브러리를 생성할 수 있다.

VI. 결 론

본 연구에서는 유비쿼터스 컴퓨팅 환경에서의 효과적인 서비스 검색과 동적 합성을 지원하기 위해 메타서비스 라이브러리의 사용을 제안했던 이전 연구의 확장으로, 도메인 온톨로지로부터 메타서비스 라이브러리를 쉽게 구축할 수 있는 자동 변환 방법을 제안하였다. 이를 위해, 메타서비스 라이브러리

구축에 필요한 OWL 모델과 메타서비스 모델, OWL-S 모델 간의 변환 관계와 자동 변환 방법을 정립하였다. 이 변환 프로세스를 통해, 모델 변환의 일관성을 고려하지 않고 전문가의 개입이 필수였던 기존의 메타서비스 라이브러리 구축 방법을 개선하였다. 메타서비스 라이브러리 개발자는 다이어그램 기반의 편집기 상에서 메타서비스를 정의하고 구조화할 수 있으며, 편집한 내용에 대한 OWL-S 코드를 자동으로 생성하여 서비스 온톨로지를 얻을 수 있다.

향후에는 분리되어 있는 두 모듈을 단일 도구로 통합하기 위한 연구를 진행할 것이다. 또한, 그림 1의 프로세스의 나머지 단계인 도메인 환경을 모델링하여 도메인 온톨로지를 구축하는 부분을 자동화하여 UML 모델로부터 메타서비스 라이브러리로의 변환을 위한 연구도 필요하다.

참고문헌

[1] M. Weiser, *"The Computer for the 21st Century,"* Scientific American, pp. 94-104, Sept. 1991.

[2] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," IEEE Personal Communications, pp. 10-17, Aug. 2001.

[3] 정헌만, 이정현, "유비쿼터스 컴퓨팅 환경에서의 상황 인식을 위한 확률 확장 온톨로지 모델," 한국컴퓨터정보학회논문지, 제11권, 제3호, 239-248쪽, 2006년 7월.

[4] 이기영, 김동오, "유비쿼터스 컴퓨팅 환경에서의 위치 데이터 관리 시스템의 설계," 한국컴퓨터정보학회논문지, 제12권, 제6호, 115-121쪽, 2007년 12월.

[5] M. Lee, S. Park and J. Lee, "Ontology-based Service Layering for Facilitating Alternative Service Discovery," In Proceedings of the Second International Conference on Ubiquitous Information Management and Communication (ICUIMC 2008), pp. 482-487, Jan. 2008.

[6] 이미연, 이정원, 박승수, 조위덕, "유비쿼터스 컴퓨팅을 위한 온톨로지 기반의 서비스 기술 및 오버로딩 기법," 정보처리학회논문지 B, 제15-B권 제5호, 465-476쪽, 2008년 10월.

[7] D. L. McGuinness, and F. van Harmelen, "OWL Web Ontology Language Overview," W3C Member Submission, Feb. 2004.

[8] D. Martin, et al., "OWL-S: Semantic Markup for Web Services," W3C Member Submission, Nov. 2004.

[9] J.T.E. Timm and G.C. Gannod, "A Model-Driven Approach for Specifying Semantic Web Services," In Proceedings of the 3rd IEEE International Conference on Web Services (ICWS2005), July 2005.

[10] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovic, "From UML to Ready-To-Use OWL Ontologies,", 2nd IEE International Conference on Intelligent Systems, June 2004.

[11] J. Miller and J. Mukerji, *"MDA Guide Version 1.0.1,"* OMG Specifications, June 2003.

[12] EMF (Eclipse Modeling Framework), *http://www.ecipse.org/modeling/emf/*

[13] GMF (Graphical Modeling Framework), *http://www.eclipse.org/modeling/gmf/*

[14] GEF (Graphical Editing Framework), *http://www.eclipse.org/gef/*

[15] K. Kang, "Community Computing White Paper," Ajou University, Center of Excellence for Ubiquitous System, Published as CUS Technical Report, Mar. 2006.

[16] 이용주, "시멘틱 e-워크플로우 프로세스를 이용한 동적 웹 서비스 조합," 한국컴퓨터정보학회논문지, 제10권, 제1호, 101-112쪽, 2005년 3월.

[17] M. Kay, *"XSL Transformation (XSLT) Version 2.0,"* W3C Recommendation, Jan. 2007.

저 자 소 개

이 미 연

2003: 이화여자대학교 컴퓨터학과 학사.
2005: 이화여자대학교 컴퓨터학과 석사.
현 재: 이화여자대학교 컴퓨터정보통신공학과 박사과정.
관심분야: 유비쿼터스 컴퓨팅, 온톨로지, SOA, 인공지능 등

이 정 원

2003: 이화여자대학교 컴퓨터공학 박사.
2003~2006: 이화여자대학교 컴퓨터학과 BK교수, 전임강사(대우).
현 재: 아주대학교 전자공학부 조교수.
관심분야: SOA, 유비쿼터스, 컴퓨팅, 임베디드 소프트웨어 등

박 승 수

1974: 서울대학교 수학과 학사.
1976: 한국과학기술원 전산학 석사.
1988: 미국 텍사스대학 전산학 박사.
1988~1991: 미국 캔사스대학 컴퓨터학과 조교수.
현 재: 이화여자대학교 컴퓨터공학과 교수.
관심분야: 시멘틱웹, 온톨로지, 인공지능 등

조 위 덕

1981: 서강대학교 전자공학과 학사.
1983: 한국과학기술원 전기 및 전자공학과 석사.
1987: 한국과학기술원 전기 및 전자공학과 박사.
1983~1990: 금성전기(현 LG전자) 기술연구소 DSP 연구실장.
1990~1991: 한국생산기술연구원 전자정보시스템연구부 팀장/조교수.
1991~2003: 전자부품연구원 시스템연구본부 본부장.
현 재: 유비쿼터스컴퓨팅사업단 단장, 아주대학교 전자공학부 교수.
관심분야: 유비쿼터스 컴퓨팅, 네트워크, 센서 네트워크, Post-PC(차세대 Smart PDA), Interactive DTV 방송 기술, 고품질 홈서버/게이트웨이 기술, 디지털 방송/이동통신 연계 융합플랫폼기술, 무선인터넷응용기술 등