# UML 모델을 OWL-S 온톨로지로 변환하기 위한 모델지향접근방식

## (A Model-Driven Approach for Converting UML Model to OWL-S Ontology)

김 일 웅 † 이 경 호 ††
(Il-Woong Kim) (Kyong-Ho Lee)

**요 약** 온톨로지에 기반한 시맨틱 웹 서비스는 웹 서비스의 자동화된 탐색, 선택, 조합을 지원한다. 특히 시맨틱 웹 서비스 기술 표준인 OWL-S는 서비스의 의미적 기술을 제공하기 위해 만들어진 온톨로지 언어이다. 한편 OWL-S는 문법이 복잡하여 일반 사용자가 OWL-S 문서를 직접 생성하는 것은 어렵다. 본 논문에서는 현재 소프트웨어 설계 및 개발을 위해서 널리 사용 중인 UML 다이어그램으로부터 OWL-S 문서를 편리하게 생성할 수 있는 방법을 제안한다. 제안한 방법은 프로세스의 흐름을 기술한 순차 다이어그램 및 활동 다이어그램으로부터 OWL-S 온톨로지를 생성하기 위해서 UML 프로파일을 기반으로 하고 있다. UML 다이어그램은 XMI 파일로 추출되고 XSLT 스크립트를 통해 OWL-S 온톨로지로 변환된다. 제안한 방법은 복합 프로세스의 다양한 제어구조를 지원할 수 있는데, 이 논문에서는 다양한 종류의 UML 다이어그램을 대상으로 실험함으로써 이를 보였다.

**키워드** : 시맨틱 웹 서비스, UML, 모델지향접근방식, OWL-S

***Abstract*** Based on ontologies, semantic Web services enable the discovery, selection, and composition. OWL-S is a de facto standard ontology for describing semantics of Web services. Due to the difficulty of the OWL-S grammar, the learning curve for constructing OWL-S description manually can be steep. This paper presents an efficient method for generating OWL-S descriptions from UML diagrams, which are widely used for software design and development. The proposed method is based on UML profiles to generate an OWL-S ontology from sequence or activity diagrams, which represent the behavior of a business process. Specifically, an XMI file extracted from UML diagrams is transformed into an OWL-S description via an XSLT script. Experimental results with a large volume of UML diagrams show that the proposed method deals with the control flow of complex processes and is superior to previous methods.

**Key words** : Semantic Web Services, UML, MDA, OWL-S

## 1. 서 론

웹 서비스(Web Service)는 플랫폼에 상관없이 어플리케이션의 상호 운용을 가능하게 하는 소프트웨어 기술이다. 웹 서비스는 SOAP, WSDL, UDDI와 같은 표준 인터페이스를 통해 서비스의 재사용 및 서비스 간의 상호 작용을 지원한다. 웹 서비스는 서비스 지향적 분산 컴퓨팅 기술의 일종이다. 기존 분산 컴퓨팅 기술과 비교했을 때 느슨한 연결, XML 유니코드 부호화 사용, 객체 지향적이기 보다는 메시지 지향적인 차이점을 갖는다. 시맨틱 웹 서비스는 웹 서비스에 대한 온톨로지를 구축하고 기계가 처리할 수 있는 형태로 시맨틱 마크업(markup)을 한다. 그 결과 웹에 있는 정보를 컴퓨터가 더 쉽게 이해할 수 있도록 도와주는 표준과 기술을 개발하여 시맨틱 검색, 데이타 통합, 내비게이션, 태스크의 자동화 등을 지원한다. 시맨틱 웹 서비스 기술은 시맨틱 웹 기술을 웹 서비스에 적용시킴으로써 웹 서비스가 갖고 있는 기술적 한계들을 해결하고자 하는 시도라고 할 수 있고, 향후 복잡한 웹 서비스의 조합과 합성을 위해서 반드시 요구되는 기술이다.

시맨틱 웹 서비스를 기술하기 위해 풍부한 의미적 기

· 이 논문은 서울시 산학연 협력사업(10705)의 지원에 의해 연구되었음
† 학생회원 : 연세대학교 컴퓨터과학과
iwkim@icl.yonsei.ac.kr
†† 종신회원 : 연세대학교 컴퓨터과학과 교수
khlee@cs.yonsei.ac.kr
논문접수 : 2007년 1월 24일
심사완료 : 2007년 6월 13일

술이 가능한 OWL-S [1] 와 같은 언어가 개발되었다. OWL-S는 OWL 기반의 웹 서비스 온톨로지로서 기존 OWL의 형식을 그대로 사용하며 모호하지 않고 컴퓨터가 이해할 수 있는 형태의 마크업 언어 구조를 웹 서비스를 기술하기 위해 제공한다. OWL-S 사용 목적은 자동화된 웹 서비스 탐색, 호출, 조합, 상호운용과 자동화된 웹 서비스 실행의 모니터링이다. OWL-S는 또한 시맨틱 웹상에서 서비스 간의 상호작용을 허락한다.

Model Driven Architecture(MDA)[2]는 프로그램 코드 보다 모델 생성에 중심을 둔 소프트웨어 개발 방법론이다. MDA의 목적은 소프트웨어의 구현과 명세의 관계를 구조적으로 분리함으로써 휴대성, 상호운용성, 재사용성을 가능케 하는 것이다. 본 논문에서는 MDA 기반의 방법론 중에서 Unified Modeling Language(UML) 모델의 개발을 통한 소프트웨어 생성에 초점을 맞춘다. UML은 요구 분석, 시스템 설계, 시스템 구현 등의 과정에서 사용되는 모델링 언어다. UML은 오늘날의 객체지향 시스템 개발 분야에서 가장 각광받는 도구 중 하나로서 시스템 개발자가 자신의 비전을 구축하고 반영하는데 있어서 표준적이고 이해하기 쉬운 방법으로 할 수 있도록 도와주며, 자신의 설계 결과물을 다른 사람과 효과적으로 주고받으며 공유할 수 있는 메커니즘을 제공한다. UML은 시스템을 모델링 할 수 있는 다양한 도구들을 제공하기 때문에, 도메인을 모델링하기가 훨씬 용이할 뿐만 아니라 모델링한 결과를 쉽게 파악할 수 있게 된다. 또한 산업계 표준으로 채택되었기 때문에 UML을 적용한 시스템은 신뢰성 있는 시스템으로 평가받는다.

한편 OWL-S는 스펙이 복잡하고 문법이 장황해서 일반 사용자가 OWL-S 문서를 직접 기술하는 것은 어렵다. 그러나 소프트웨어 설계 및 개발을 위해서 널리 사용 중인 UML 다이어그램을 재사용하면 OWL-S를 편리하게 기술할 수 있다.

기존 연구에서는 UML 모델을 사용해 OWL-S와 BPEL4WS같은 웹 서비스 조합 언어를 기술한다. BPEL4WS의 경우 웹 서비스 간의 행위에 대한 기술을 지원하지만 의미적 정보는 표현할 수 없다. OWL-S에 관한 기존 연구의 경우 단일 프로세스(atomic process)의 기술은 가능하지만 여러 웹 서비스를 조합한 복합 프로세스(composite process)의 다양한 제어구조를 기술하지는 못한다. 대부분의 시스템은 복합 프로세스의 다양한 제어구조를 사용해 기술하기 때문에 기존의 제안된 방법으로는 이런 문제를 해결 할 수 없다. 또한 이미 존재하는 UML 파일의 재사용에 관해서는 고려하지 않는다.

기존 연구의 문제점을 해결하기 위해 본 논문에서는 프로세스의 흐름을 기술한 순차 다이어그램 및 활동 다이어그램으로부터 OWL-S 온톨로지를 생성하기 위해서 UML 프로파일을 기반으로 하고 있다. 본 논문에서는 OWL-S의 생성을 위해 기존의 UML 모델을 알맞게 수정하고, 그 결과로 생성된 XML Metadata Interchange (XMI) 문서를 Extensible Stylesheet Language Transformations(XSLT)[3] 스크립트를 사용해 OWL-S로 변환하는 방법을 제안한다. 총 22개의 UML 모델을 대상으로 전문가가 직접 작성한 OWL-S와 제안된 방법으로 생성된 OWL-S를 비교한 실험결과는 의미적으로 완전하게 일치했다. 제안한 방법을 이용하면 UML 모델의 재사용이 가능하고 기존 연구들이 처리하지 못하는 복합 프로세스의 다양한 제어구조를 기술하는 데에도 쉽게 적용될 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 MDA 기반방법으로 웹 서비스 조합 언어를 생성하는 기존의 연구에 대해 살펴본다. 3장에서는 UML 다이어그램으로부터 시맨틱 웹 서비스 기술 표준인 OWL-S문서를 생성하는 방법을 제안한다. 4장에서는 제안한 방법을 평가하기 위한 실험 결과와 분석을 기술한다. 끝으로 5장에서는 결론 및 향후 연구 방향에 대해 기술한다.

## 2. 관련 연구

본 장에서는 기존의 MDA를 기반으로 웹 서비스를 기술하는 연구의 특징에 대해 설명한다. MDA 기반의 웹 서비스 기술에 대한 연구는 시맨틱 웹 서비스의 기술의 편의성과 복합 웹 서비스의 조합, 검색의 효율성을 높이기 위한 목적에서 시작되었다. 특히 UML 관점의 접근에 초점을 맞추는 이유는 웹 서비스의 상호 협동하며 동작하는 방식을 직관적으로 표현할 수 있고, 새로운 툴을 개발할 필요 없이 이미 산업계 표준으로 자리 잡은 UML을 사용할 수 있기 때문이다. 표 1은 기존의 MDA 기반의 웹 서비스 기술에 관한 연구목록이다.

기존 연구는 크게 BPEL4WS와 OWL-S를 생성하는 방법으로 나눌 수 있다. BPEL4WS를 생성하는 기존 연구는 웹 서비스 호출, 데이타 조작, 오류 보고, 프로세스 종료 같은 다른 작동들을 하나로 만들어서 복합적인 프로세스를 만든다. BPEL4WS는 프로세스의 행위는 기술할 수 있지만 의미적 정보를 포함하지는 않는다. 그러나 OWL-S를 생성하는 기존 연구는 프로세스의 행위뿐만 아니라 의미적 정보의 기술 또한 가능하다. 다만 기존 연구는 OWL-S의 단일 프로세스에 대한 기술과 복합 프로세스 중 순차적 흐름만을 고려하고 있을 뿐 복합 프로세스의 다양한 제어구조에 대한 기술은 고려하지 않는다. 본 논문은 기존 연구에서 간과된 OWL-S의 복합 프로세스의 다양한 행위에 대한 기술이 가능하고,

표 1 MDA 기반의 웹 서비스 기술에 관한 연구

| 저자 | 연도 | 특징 | 결과물 |
|---|---|---|---|
| Grønmo 등 [4] | 2005 | 시맨틱 웹 서비스의 탐색, 호출, 조합을 위해 UML을 이용해 OWL-S문서를 기술하고 사용자는 자신이 호출하기 원하는 웹 서비스를 UML로 모델링 | OWL-S |
| Alberto 등 [5] | 2005 | 웹 서비스 조합을 쉽고, 정교하게 하기 위한 저작도구인 ZenFlow는 BPEL4WS의 구조적이고 의미적인 인식이 가능한 다중 뷰 (Multi-view) 그래픽 에디터를 제공 | BPEL4WS |
| Grønmo 와 Oldevik [6] | 2005 | UML 툴을 통해 생성된 XMI 파일을 선단계로서 Light XMI 파일로 변환하고 생성된 Light XMI 파일은 UMT를 통해 원하는 다른 타입(text, BPEL4WS , WSDL, XML)으로 변환되지만 OWL-S를 지원하지는 않음 | WSDL, BPEL4WS, text문서 등등 |
| Konrad 등 [7] | 2005 | 웹 서비스 조합을 위한 방법론 중 모델 지향 방법론의 특징과 해결해야할 문제를 제기하고 추상적 모델을 바탕으로 하는 하향식 설계(Top-Down)방식과 존재하는 서비스들의 연결을 바탕으로 하는 상향식 설계(Bottom-Up)의 특징에 대해 설명 | - |
| Timm 과 Gannod [8] | 2005 | UML의 클래스 다이어그램을 통해 OWL-S의 단일 프로세스를 기술. 클래스간의 관계를 통해 Service, Profile, Model, Grounding을 기술 | OWL-S |
| Michael 등 [9] | 2005 | WSDL을 사용해 추상적인 템플릿을 설계하는 단계, 서비스 요소의 의미적 기술을 위한 적절한 온톨로지를 분류하는 단계 와 전 단계의 서비스 요소와 온톨로지를 적용해 웹 서비스 조합을 완성 | OWL-S |
| Dragan 등 [10] | 2004 | 온톨로지 개발을 위해 UML을 확장하는 개념인 UML profile을 사용하고 Ontology UML Profile(OUP) 온톨로지는 XSLT 스크립트를 사용해 OWL로 변환. | OWL |
| Skogan 등 [11] | 2004 | 기존의 웹 서비스들을 조합해 새로운 웹 서비스를 생성하기 위한 UML의 소개와 웹 서비스를 위한 특정 언어에 의존되지 않는 독립적 방법론 제시하나 OWL-S는 고려하지 않음 | - |
| Gannod 와 timm [12] | 2004 | 시맨틱 웹의 성능도 향상시키고, 다양한 사용자 계층이 이용 가능한 기술 방법으로 MDA 방법론을 제안하고 특별히 하향식 설계(Top-Down) 방식을 제안 | - |
| Koch 등 [13] | 2004 | 동적 모델 처리 방법으로 PIM(Platform Independet Model) 레벨에서 Web Service Choreography를 순차 다이어그램으로 모델링 하고, 이 모델을 BPEL4WS로 변환 | BPEL4WS |
| Sebastian 등 [14] | 2002 | 프로세스지향접근방법론에 따라 이용 가능한 웹 서비스와 컴포넌트 사이의 데이타의 흐름으로서 웹 서비스 조합을 기술하고 조합언어로는 BPEL4WS를 대상으로 하고 시각적 언어로 UML을 사용 | BPEL4WS |

OWL-S의 특징에 대한 자세한 표현이 가능하다.

## 3. 제안된 방법

제안된 방법은 그림 1과 같이 온톨로지 기술, 프로세스 기술, 변환의 세 단계를 사용하여 온톨로지 및 UML 파일로부터 OWL-S를 생성한다. 온톨로지 기술단계는 온톨로지를 입력으로 받아 클래스 다이어그램으로 표현한다. 프로세스 기술단계는 프로세스의 흐름을 순차 다이어그램 또는 활동 다이어그램으로 표현한다. 위의 두 단계는 UML 도구를 통해 이루어진다. 변환 단계는 이전 단계를 통해 만들어진 UML 모델로부터 생성된 XMI 문서를 XSLT 스크립트를 이용해 OWL-S로 변환한다. 각 단계에 대한 자세한 설명은 다음과 같다.

### 3.1 온톨로지 기술

UML로 표현된 각각의 프로세스는 입력과 출력에 매핑 되는 온톨로지의 개념들에 대한 정보가 필요하다. 온

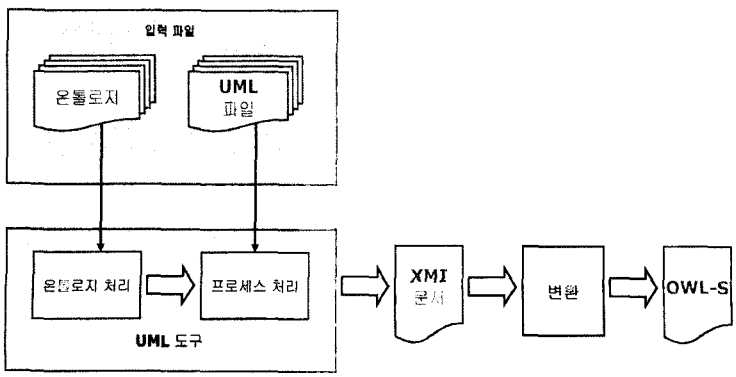



그림 1 UML 모델을 OWL-S로 변환하는 과정

톨로지는 단어와 관계들로 구성된 일종의 사전으로서 생각할 수 있으며, 그 속에는 특정 도메인에 관련된 단어들이 계층적으로 표현되어 있고, 추가적으로 이를 확장할 수 있는 추론 규칙이 포함되어 있어, 웹 기반의 지식 처리나 응용 프로그램 사이의 지식 공유, 재사용 등이 가능하도록 되어 있다. 온톨로지는 시맨틱 웹 응용의 가장 중심적 개념이다. 이를 표현하기 위해 스키마와 구문 구조 등을 정의한 언어를 온톨로지 언어라 하며, 대표적으로 OWL이 있다.

본 논문에서 사용되는 온톨로지는 도메인 온톨로지를 기본으로 한다. 이 온톨로지는 자동으로 UML 모델로 변환되고 웹 서비스 설계 시 사용된다. 개발하고자 하는 서비스에 적합한 도메인 온톨로지가 존재하지 않을 경우 본 논문에서 제안한 방법을 이용하여 온톨로지를 설계할 수 있다.

온톨로지는 클래스 다이어그램을 사용해 기술한다. 입력파일로 주어진 온톨로지 개념간의 계층관계와 각각의 개념이 갖는 속성을 클래스 다이어그램으로 기술한다(그림 2). 온톨로지의 각각의 개념을 표현한 클래스 다이어그램들은 패키지로 묶이며, 스테레오타입 "<<ontology>>"를 사용해 온톨로지임을 명시한다. 온톨로지를 나타내는 각각의 클래스는 스테레오타입 "<<OntClass>>"를 사용해 명시한다. 온톨로지의 개념들은 속성을 가질 수 있다. 본 논문에서 제안한 방법을 이용하면 데이타타입속성과 오브젝트속성에 대한 표현을 쉽게 할 수 있다. 그림 2와 같이 클래스 다이어그램에 속성을 명시하고, 그 종류에 따라 "<<datatype>>" 혹은 "<<object>>"를 사용해서 속성의 종류를 구별한다. 이것은 출판물에 대한 온톨로지로서 "Book"과 "Pamphlet"과 "Magazine"이 "Publication"과 상속관계에 있음을 보여준다. 또한 "Book"은 String 타입을 가지는 데이타타입속성 "hasName"과 오브젝트 타입 Price를 가지는 오브젝트속성 "hasPrice"를 속성으로 갖는다.

OWL과의 매핑관계는 표 2와 같다. 클래스 이름이 온톨로지 개념이 되고, 클래스 다이어그램의 속성을 사용해 온톨로지 개념의 속성을 정의하고, 스테레오타입을 사용해 데이타타입속성과 오브젝트속성을 구분한다. 일반화 관계(Generalization)를 이용하여 개념간의 계층을 표현한다.

표 2 OWL과 클래스 다이어그램의 매핑테이블

| OWL | 표준 UML 2.0 상태 모델 요소 |
|---|---|
| owl:Class | Class Name |
| owl:DatatypeProperty | Attribute |
| owl:ObjectProperty | |
| rdfs:subClassOf | Generalization |

### 3.2 프로세스 기술

UML은 시스템을 여러 가지 시점에서 점검하고 관찰할 수 있는 도구이다. 다양한 관점에서 시스템을 바라보게 되는데 프로세스를 기술하기 위해서는 행위의 기술이 가능해야 한다. 클래스 다이어그램과 객체 다이어그램은 정적인 정보를 나타낸다. 하지만 특정한 행동을 수행하는 시스템에서는 여러 개의 객체들이 서로 메시지를 주고받으며 작업을 진행하는 것이 일반적이기 때문에 행위를 표현할 수 있는 순차 다이어그램 또는 활동 다이어그램을 사용해야한다.

UML 모델은 OWL-S를 생성하기 위한 모델이 아니라 시스템을 설계하기 위해 개발된 모델이기 때문에 기존 UML 모델을 OWL-S로 완전하게 변환하는 것은 어

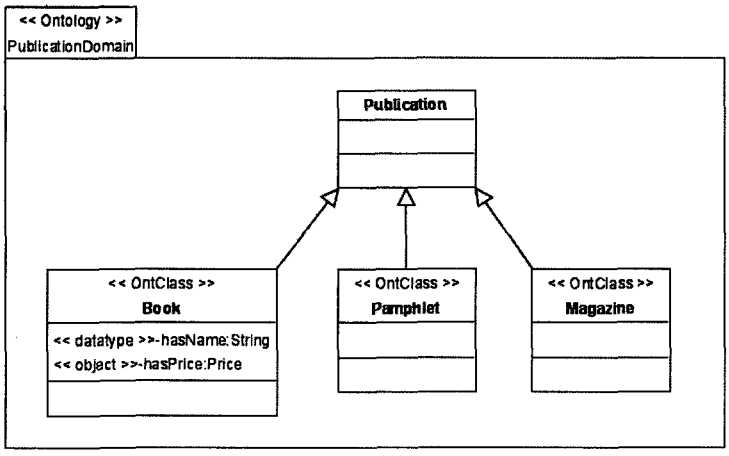



그림 2 온톨로지를 표현한 클래스 다이어그램의 예

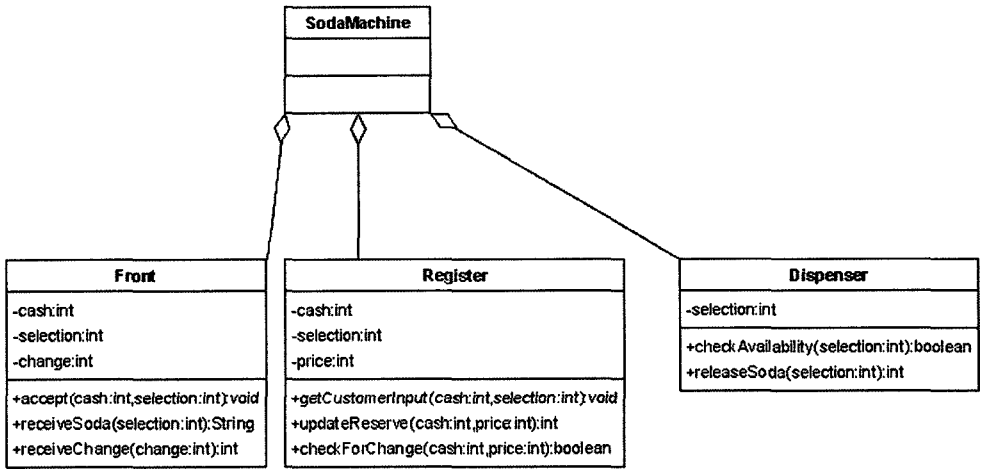



그림 3 순차 다이어그램을 위한 클래스 다이어그램의 예

렵다. 따라서 OWL-S로 변환하기 위해 필요한 정보를 추가하거나 각각의 다이어그램들을 적절하게 수정할 필요가 있다. 본 장에서는 UML 프로파일인 스테레오타입(stereotype), 제약(constraint), 태그 값(tagged value)을 사용해서 OWL-S 변환에 알맞은 다이어그램을 작성하는 방법을 제안한다.

3.2.1 OWL-S의 제어구조를 표현하는 순차 다이어그램

UML의 동적 모델들 중에 객체와 객체 사이의 동적 상호관계의 정의를 위한 모델에는 순차 다이어그램이 있다. 순차 다이어그램은 해결해야 할 문제가 주어진 상황에서 그 문제를 해결하기 위해 필요한 객체를 정의하고, 객체간의 동적인 상호관계를 시간 순서에 따라 정의함으로써 주어진 문제를 해결하는 모델이다.

시퀀스 다이어그램을 OWL-S로 변환하기 위해서는 각각의 객체에 대한 정보를 담고 있는 클래스 다이어그램이 필요하다. 왜냐하면 순차 다이어그램은 행위에 대한 정보만을 가지고 있기 때문이다. 그림 3은 "SodaMachine"에 대한 클래스 다이어그램이다. 이 다이어그램은 "Front", "Register", 와 "Dispenser"의 오퍼레이션과 속성에 대한 정보를 포함한다. 오퍼레이션은 인자와 반환형을 표현한다. 이 클래스 다이어그램에서 각각의 객체가 가지는 오퍼레이션은 하나의 단일 프로세스가 되고 속성은 프로세스의 입력이 되고 반환 값은 출력이 된다.

클래스 다이어그램의 속성은 타입을 가질 수 있다. 온톨로지를 표현한 클래스와 속성을 연결함으로써 속성의 타입을 정의한다. 기본 데이타형의 경우는 기본적으로 UML에서 제공한다.

오퍼레이션 기술 시 반드시 각각의 오퍼레이션의 인자 값과 반환형을 명시해주어야 한다. 그 이유는 오퍼레이션의 인자 값은 각각의 단일 프로세스의 입력이 되고, 반환 값은 단일 프로세스의 출력이 되기 때문이다. 각각의 오퍼레이션의 실행을 위한 조건은 제약을 사용해 기술한다. 이 조건은 OWL-S 프로세스의 If-Then-Else 혹은 반복문에 사용된다. 순차 다이어그램과 OWL-S의 매핑관계는 표 3[1]과 같다.

표 3 OWL-S와 순차 다이어그램의 매핑테이블

| 구분 | OWL-S | 표준 UML 2.0 상태 모델 요소 |
|---|---|---|
| Process | AtomicProcess | Operation |
| Control Construct | Sequence | Synchronous<br>Asynchronous |
| | If-then-else | Interaction Fragment(alt) |
| | Choice | |
| | Repeat-While | *Interaction Fragment(loop)* |
| | Split + Join | Interaction Fragment(par) |
| Parameter | Input | Arguments of operation |
| | Output | Return of operation |
| | parameterType | Type of Attributes |
| Category | categoryName | tagged value |
| | taxonomy | |
| Profile | serviceName | |
| | textDescription | |
| Condition | Precondition | constraints |
| | Result | |
| | Effect | Note |
| Parameter Binding | InputBinding | Note |
| | OutputBinding | |

1) 순차 다이어그램은 OWL-S의 제어구조 중 Any-order, Split, Repeat-Until을 표현할 수 없다. 그 이유는 위에서 언급한 각각의 제어구조를 표현하기에 알맞은 기호를 제공하지 않기 때문이다.

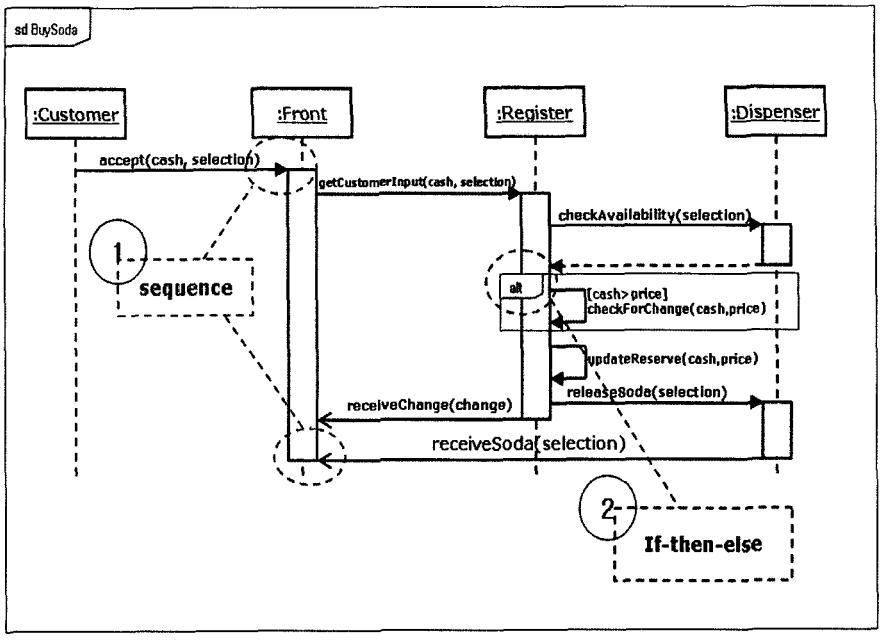


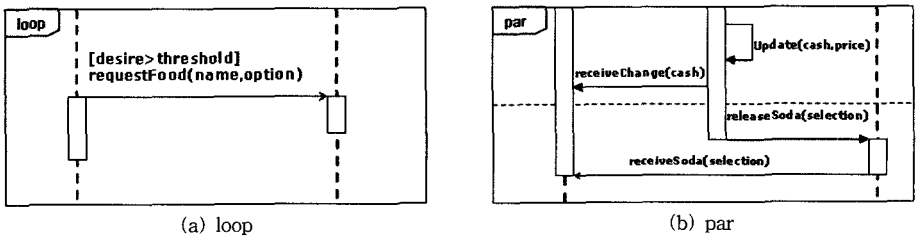

그림 4 순차 다이어그램의 예

그림 4는 음료수 자동판매기의 작동 방법을 순차 다이어그램으로 표현한 것이다. 이를 위해 3개의 클래스 프론트(Front), 금전 등록기(Register), 디스펜서(Dispenser)가 존재한다. 프론트는 음료수 선택을 받아들이고 돈을 받는 일(accept), 고객에게 돈을 돌려주는 일(receiveChange)과 디스펜서로부터 소다 캔을 받아 고객에게 주는 일(receiveSoda)을 담당하는 오퍼레이션을 갖는다. 금전 등록기는 프론트로부터 고객의 입력을 전달받는 일(getCustomerInput), 현금 잔고를 업데이트하는 일(updateReserve)과 잔돈을 체크하는 일(checkForChange)을 담당하는 오퍼레이션을 갖는다. 디스펜서는 선택된 음료가 이용 가능한지를 체크(checkAvailability)하고 음료수를 방출하는 일(releaseSoda)을 담당하는 오퍼레이션을 갖는다.

순차 다이어그램은 메시지의 종류에 상관없이 시간축의 순서대로 프로세스가 처리되기 때문에 동기메시지와 비동기메시지는 모두 OWL-S의 제어구조 중 Sequence에 대응된다. 예를 들어 그림 4의 ①을 살펴보면 Front 객체의 오퍼레이션 accept()는 동기메시지이고, receiveSoda()는 비동기메시지이다. 이 두 메시지는 모두 순차적으로 수행된다.

제약은 If-Then-Else 혹은 Repeat-While의 조건을 기술하는데 사용한다. Register객체의 checkForChange()은 잔돈을 체크하는 일을 수행하는 오퍼레이션이다. 이 오퍼레이션을 수행하기 위해서는 "[cash>price]"과 같은 조건이 필요하고 이 조건은 제약으로 표현되며 그림 4의 ②와 같다.

UML 2.0부터 순차 다이어그램에 교류조각(Interaction Fragment)이 도입되었다. 이 교류조각은 순차 다이어그램의 한 부분을 일컫는 UML 2.0의 일반적인 이름이다. 이 교류조각을 사용해 순차 다이어그램에서 OWL-S의 다양한 제어구조를 표현할 수 있다. 각각의 제어구조의 예제는 그림 5에서 확인할 수 있다. 그림 5의 (a)는 loop로서 제어구조의 Repeat-While에 대한 표




(a) loop (b) par

그림 5 교류조각의 예

현이 가능하다. (b)는 par로서 여러 작업을 동시에 처리하고자 할 때 사용하는 오퍼레이션이다. 병행 처리는 여러 상호 작용 그룹을 동시적으로 처리함을 의미한다.

이밖에도 교류조각은 병행처리에 관한 critical, 메시지 전달 순서를 결정하는 seq와 strict, 테스트에 관련된 assert, 메시지 선택에 관한 ignore와 consider 그리고 발생해서는 안 되는 상황 작성에 관한 neg가 있다. 이런 연산자들을 표현하기 위한 적절한 문법이 OWL-S에서 제공되지 않기 때문에 이 연산자들은 OWL-S로 변환이 불가능하다.

3.2.2 OWL-S의 제어구조를 표현하는 활동 다이어그램

활동 다이어그램은 업무영역이나 시스템 영역에서 다양하게 존재하는 각종 처리로직이나 조건에 따른 처리 흐름을 순서에 입각하여 정의한 모델이다. 활동 다이어그램은 하나의 액티비티에서 다음 액티비티로 순서가 바뀌면서 처리되는 과정을 표현하기 때문에 순서와 분기와 처리절차의 표현을 필요로 하는 대상에 대해 제한없이 적용가능하다. 따라서 OWL-S의 복합 프로세스 기술에 용이하다.

활동 다이어그램은 UML에서 제공하는 여러 모델 중에 OWL-S로의 변환에 가장 적합한 다이어그램이다. 왜냐하면 활동 다이어그램은 복합 프로세스의 다양한 제어구조를 기술할 수 있고, OWL-S로의 변환에 적합한 기호들을 충분하게 제공해주기 때문이다. 활동 다이어그램과 OWL-S의 제어구조 매핑관계는 표 4[2]와 같다.

활동 다이어그램의 각각의 액티비티는 OWL-S의 단일 프로세스이다. 프로세스의 순차적 흐름은 OWL-S의 sequence에 대응된다. Fork 노드를 사용해서 Split을 표현하고, Fork와 Join 노드를 함께 사용해서 Split+Join을 표현한다. Fork 노드는 프로세스의 병행처리를 표현한다. Decision 노드는 조건문과 반복문을 표현한다. 조건문은 Choice와 If-Then-Else로 나뉘고, 반복문은 Repeat-Until과 Repeat-While로 나뉜다. Choice는 프로그래밍 언어의 switch문과 같고, If-Then-Else는 If문과 같다. Iterate는 Repeat-Until과 Repeat-While의 슈퍼클래스로서 추상클래스이고, Repeat-Until(그림 6)은 조건이 참이면 프로세스가 종료되고 거짓이면 프로세스가 순환되어 실행된다. 반대로 Repeat-While(그림 7)은 조건이 거짓이면 프로세스가 종료되고 참이면 프로세스가 순환되어 실행된다. 각각은 스테레오타입을 통해 구분한다.

UML 2.0이 되면서 활동 다이어그램에 pin기호가 추가 되었다. 이로써 각각의 액티비티에 직접 입·출력을

표 4 활동 다이어그램과 OWL-S의 매핑테이블

| 구분 | OWL-S | 표준 UML 2.0 상태 모델 요소 |
|---|---|---|
| Control Construct | Sequence | Sequence |
| | Split | Fork |
| | Split+Join | Fork + Join |
| | Choice | Decision |
| | If-Then-Else | |
| | Repeat-Until | |
| | Repeat-While | |
| Parameter | Input | Pin |
| | Output | |
| | parameterType | Pin의 속성 |
| Category | categoryName | Tagged Value |
| | taxonomy | |
| Profile | serviceName | |
| | textDescription | |
| Condition | Precondition | constraints |
| | Result | |
| | Effect | Note |
| Parameter Binding | InputBinding | Pin과 Pin의 연결 |
| | OutputBinding | |

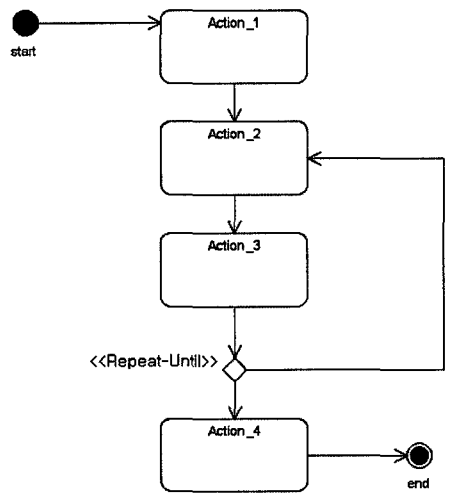



그림 6 Repeat-Until

표현할 수 있게 되었다. 또한 이 pin들 간의 연결이 가능하기 때문에 한 프로세스의 출력이 다른 프로세스의 입력에 바인딩 되는 것을 직관적으로 표현할 수 있다. 입력과 출력의 구분은 스테레오타입을 통해 나타낸다.

활동 다이어그램을 이용한 복합 프로세스 생성은 다음과 같다. 복합 프로세스에 필요한 단일 프로세스들을 각각의 액티비티로 표현하고 액티비티의 입력과 출력을 표현하기 위한 pin을 붙인다. 이 pin과 온톨로지를 표현한 클래스 다이어그램과 연결시킴으로써 입력과 출력의

2) 활동 다이어그램은 OWL-S의 제어구조 중 Any-order를 표현할 수 없다. 그 이유는 Any-order를 표현하기에 알맞은 기호를 제공하지 않기 때문이다.

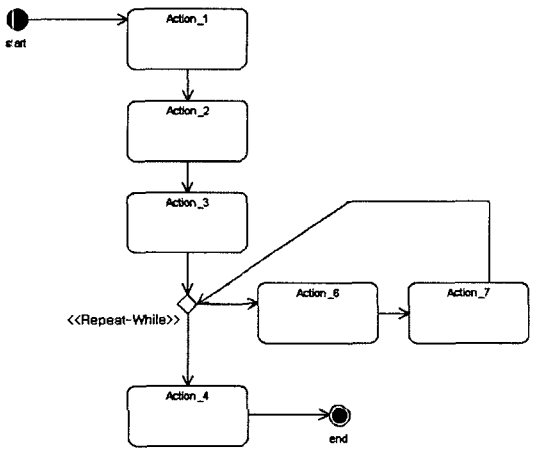



그림 7 Repeat-While

타입을 결정한다. 순차 다이어그램과 마찬가지로 기본 데이타 형은 기본 UML이 제공하기 때문에 바인딩 없이 직접 기술이 가능하다. 의미적 정보를 기술할 수 있는 preCondition과 Result는 제약을 사용해서 표현한다.

이렇게 만들어진 복합 프로세스는 하나의 커다란 시맨틱 웹 서비스다. 이 하나의 서비스의 입력과 출력은 복합 프로세스를 구성하는 프로세스의 입력과 출력에 연결돼서 표현된다.

그림 8은 사용자가 책을 싼 가격에 구입할 수 있는 사이트를 찾는 서비스를 활동 다이어그램으로 표현한 것이다. 이 서비스는 책의 이름을 입력으로 받아서 가장 저렴한 가격으로 책을 구입할 수 있는 온라인 서점과 그 가격을 출력으로 반환한다. 이 다이어그램은 총 5개의 프로세스로 구성된다. 그림 8의 ①은 책의 정보를 찾는 프로세스다. 한 개의 입력과 출력을 갖는다. FindBookInfo의 입력은 전체를 나타내는 프로세스인 FindCheaperBook의 입력인 BookName과 연결되고 출력은 AmazonPriceProcess와 BNPriceProcess의 입력과 각각 연결된다. 이 두 프로세스는 책의 가격을 반환하는 프로세스다. 두 프로세스는 ②와 같이 제어구조 중 Split + join으로 병렬 처리된다. ③은 FindBookInfo의 출력 값인 bookInfo는 AmazonPriceProcess의 입력 값인 bookInfo와 연결되는 것을 보여준다. 이것은 InputBinding의 예이다. ④는 책의 가격을 비교해 어떤 프로세스를 수행할지를 결정하는 Decision 노드다. ⑤는 전체를 나타내는 프로세스인 FindCheaperBook의 출력 BookStore가 ProduceBN의 출력과 연결되는 것을 보여준다. 이것은 OutputBinding의 예를 보여준다.

또한 제안한 방법을 이용하면 기존의 UML 모델을

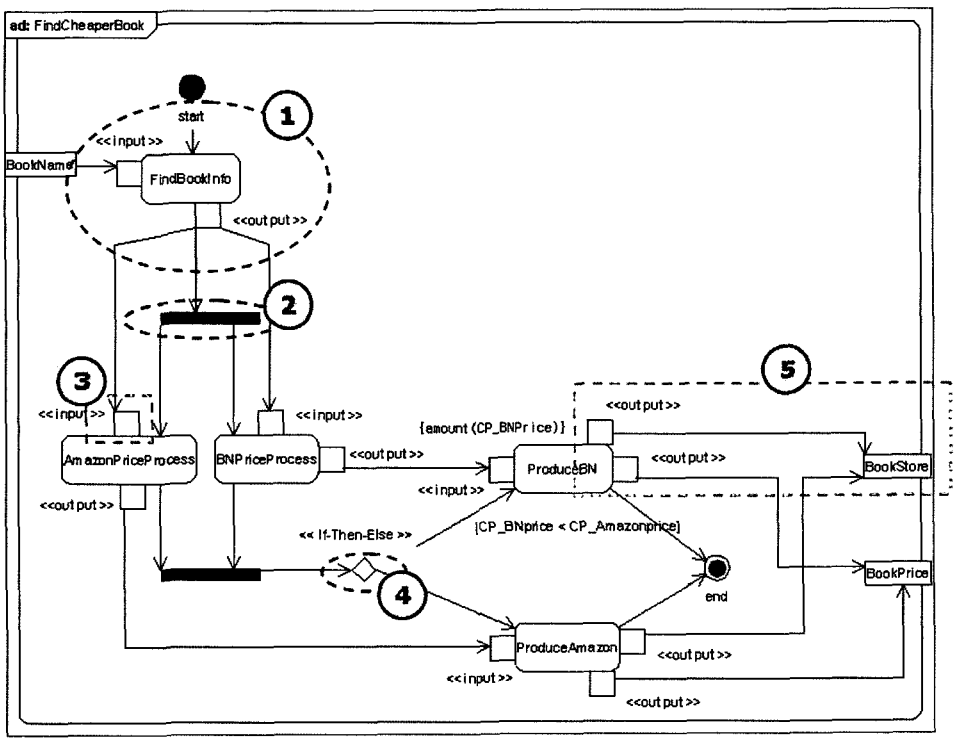



그림 8 활동 다이어그램을 활용한 기술의 예

재사용해 OWL-S를 생성하는 것뿐만 아니라 새로운 OWL-S 개발하는 데에도 쉽게 적용될 수 있다.

### 3.3 변환

이 장에서는 UML 모델을 OWL-S로 변환하는 방법에 대해 설명한다. 본 논문에서 제안한 방법을 이용해 작성된 UML 모델(온톨로지 모델과 프로세스 모델)은 하나의 XMI 파일로 추출되고 XMI 파일은 XSLT 스크립트를 통해 OWL-S 문서로 변환된다.

XMI 파일은 개발자와 다른 사용자들 간의 메타데이타에 관한 정보를 교환하기 위한 표준 방식이다. UML 모델링을 통해 생성된 XMI 파일은 다양한 다이어그램의 기호에 관한 정보와 그 기호의 그래픽적인 특징에 대한 정보로 구성된다. 변환 작업을 처리하기 위해서는 기호의 그래픽적인 특징에 대한 정보는 필요 없다. 그 이유는 XSLT 스크립트를 통해 변환되는 정보는 기호에 관한 정보이기 때문이다.

표 5는 UML과 OWL-S의 변환 매핑표이다. UML 모델과 OWL-S의 매핑을 위해 XMI 파일을 사용한다. 활동 다이어그램의 action은 OWL-S의 프로세스로 매핑 된다. pin은 입력과 출력으로 매핑되고 pin to pin은 바인딩을 나타낸다. 태그 값은 스테레오타입을 사용해서 각각 categoryName, taxonomy, serviceName과 textDescription로 분류되어 매핑 된다. 제약도 스테레오타입을 사용해서 Precondition과 Result로 매핑 된다. Note는 Effect와 매핑 된다. Decision은 스테레오타입을 사용해 각각 Choice, If-Then-Else, Repeat-Until, Repeat-While로 분류되어 매핑 된다.

변환된 OWL-S문서에서 조건 절은 SWRL[15]로 기술된다. SWRL 외에도 KIF와 같은 언어도 존재하지만 OWL-S의 차기 버전에서 SWRL을 기본으로 채택하였기 때문에 본 논문에서는 SWRL을 사용해 조건 절을 기술한다.

## 4. 실험 결과

본 장에서는 제안된 방법의 성능을 평가하기 위해 수행한 실험 결과를 기술한다. 실험은 총 22개의 UML 모델을 대상으로 전문가가 직접 작성한 OWL-S와 제안한 방법을 이용해 생성된 OWL-S를 비교분석한 것이다.

표 5 변환 매핑표

| UML | XMI | OWL-S |
|---|---|---|
| Action | <UML2:Activity> | Process |
| pin | <UML2:InputPin> | Input |
| pin | <UML2:OutputPin> | Output |
| pin to pin | <UML2:ActivityEdge> | {In\|Out}put Binding |
| Tagged Value | <UML:TaggedValue><br><UML:Stereotype name='categoryName'> | categoryName |
| Tagged Value | <UML:TaggedValue><br><UML:Stereotype name='taxonomy'> | taxonomy |
| Tagged Value | <UML:TaggedValue><br><UML:Stereotype name='serviceName'> | serviceName |
| Tagged Value | <UML:TaggedValue><br><UML:Stereotype name='textDescription'> | textDescription |
| constraints | <UML:Constraint><br><UML:Stereotype name='preCondition'> | Precondition |
| constraints | <UML:Constraint><br><UML:Stereotype name='Result'> | Result |
| Note | <UML:Comment> | Effect |
| Sequence | <UML2:ActivityEdge> | Sequence |
| Fork | <UML2:ForkNode> | Split |
| Fork+Join | <UML2:ForkNode><br><UML2:JoinNode> | Split+Join |
| Decision | <UML2:DecisionNode><br><UML:Stereotype name='Choice'> | Choice |
| Decision | <UML2:DecisionNode><br><UML:Stereotype name='If-Then-Else'> | If-Then-Else |
| Decision | <UML2:DecisionNode><br><UML:Stereotype name='Repeat-Until'> | Repeat-Until |
| Decision | <UML2:DecisionNode><br><UML:Stereotype name='Repeat-While'> | Repeat-While |

실험 데이타 및 성능 분석에 대한 기술은 다음과 같다.

### 4.1 실험 데이타

제안된 OWL-S로의 변환 방법의 성능을 평가하기 위해 표 6과 같은 UML 파일을 수집하여 실험하였다. 실험 데이타로 사용된 다이어그램은 순차 다이어그램 8개와 활동 다이어그램 14개다.

표 6 실험 데이타

| 다이어그램 | UML 파일 | 사용된 제어구조 |
|---|---|---|
| 순차 다이어그램 | Banking.zuml | Sequence<br>If-Then-Else |
| | BuySoda.zuml | Sequence<br>If-Then-Else |
| | GuaranteeReservation.zuml | Sequence<br>If-Then-Else<br>Repeat-While |
| | MemberManager.zuml | Sequence |
| | Order.zuml | Sequence |
| | RefundMembershipfee.zuml | Sequence<br>If-Then-Else |
| | BuySoda2.zuml | Sequence<br>Split + Join |
| | Conditional.zuml | Sequence<br>Choice |
| 활동 다이어그램 | BusinessModeling.zuml | Sequence<br>Choice<br>Split + Join |
| | Engagement.zuml | Sequence<br>Choice<br>If-Then-Else |
| | MatchJourney.zuml | Sequence<br>If-Then-Else<br>Repeat-While |
| | Membership.zuml | Sequence<br>Repeat-Until |
| | PlanTrip.zuml | Sequence<br>Choice<br>If-Then-Else<br>Repeat-While |
| | ProcessSale.zuml | Sequence<br>If-Then-Else |
| | Product.zuml | Sequence<br>Choice |
| | Serving.zuml | Sequence<br>Split + Join |
| | Word.zuml | Sequence<br>If-Then-Else |
| | Congobuy.zuml | Sequence<br>Choice<br>If-Then-Else |
| | Bravoair.zuml | Sequence<br>If-Then-Else |
| | BookPrice.zuml | Sequence |
| | FrenchDictionary.zuml | Sequence |
| | FindCheaperBook.zuml | Sequence<br>If-Then-Else<br>Split + Join |

※ 실험데이타 : http://icl.yonsei.ac.kr/mda/testdata

### 4.2 성능평가

본 실험에서는 제안된 방법을 통해 생성된 OWL-S와 전문가가 작성한 OWL-S를 비교함으로써 제안된 방법의 성능을 평가한다.

그림 9는 실험에 사용된 예제 파일 중 하나인 GuaranteeReservation.zuml이다. 이 다이어그램은 8개의 오퍼레이션으로 구성되어 있고 제어구조로 Sequence, 교류조각의 alt와 loop 노드를 가진다. 이 UML 모델은 예약을 위한 정보를 입력하고 결과로 예약 번호를 받는 프로세스이다. 그림 10의 왼쪽 열은 그림 9의 UML 모델로부터 생성된 OWL-S의 일부이고, 오른쪽 열은 UML 모델로부터 OWL-S를 생성하는 XSLT 스크립트의 일부이다. XSLT 스크립트는 각각의 오퍼레이션을 순차 다이어그램과 연관되는 클래스 다이어그램으로부터 입력과 출력에 대한 정보를 얻어 단일 프로세스로 변환하고 교류조각을 사용하는 제어구조는 교류조각의 이름과 스테레오타입을 바탕으로 각각에 해당하는 템플릿을 통해 OWL-S의 여러 제어구조로 변환한다.

UML 모델을 보고 전문가가 기술한 OWL-S와 제안한 방법을 이용해 생성된 OWL-S를 비교하기 위해 각각의 실험데이타에 대해 표 7과 같은 분류를 사용해 비교분석한다. 전문가가 직접 기술한 OWL-S 요소의 개수와 제안된 방법을 이용해 생성된 OWL-S 요소의 개수가 같을 경우 본 실험에서는 동일한 요소가 생성되었다. 예를 들어 두개의 OWL-S가 3개의 "hasInput"을 가지고 있다고 하면 두개의 OWL-S는 각각 A, B, C와 A, B, D의 서로 다른 3개의 "hasInput" 요소가 아닌 A, B, C의 동일한 3개의 "hasInput"을 가짐을 의미한다.

그림 11은 순차 다이어그램을 전문가가 직접 기술한 것과 제안한 방법을 이용해 생성된 OWL-S의 요소 개수를 비교한 그래프이다. 그림 12는 활동 다이어그램을 통해 생성된 OWL-S를 비교한 것이다. 요소의 개수가 같다는 것은 위에서 언급했듯이 전문가가 기술한 요소와 제안한 방법을 이용해 생성된 요소가 동일하다는 것을 의미한다.

그림 11과 12의 그래프를 보면 몇몇 실험 데이타에서 전문가의 방법보다 제안한 방법을 이용해 생성된 OWL-S 요소의 개수가 더 많음을 확인할 수 있다. 그 이유는 제어구조 중 Sequence를 제외한 다른 제어구조 다음에 단지 하나의 단일 프로세스가 올 경우에도 Sequence 요소가 추가되기 때문이다. 하나의 프로세스만이 수행되기 때문에 Sequence를 통해 그 단일 프로세스를 묶을 필요가 없으나 제안한 방법을 이용하면 이

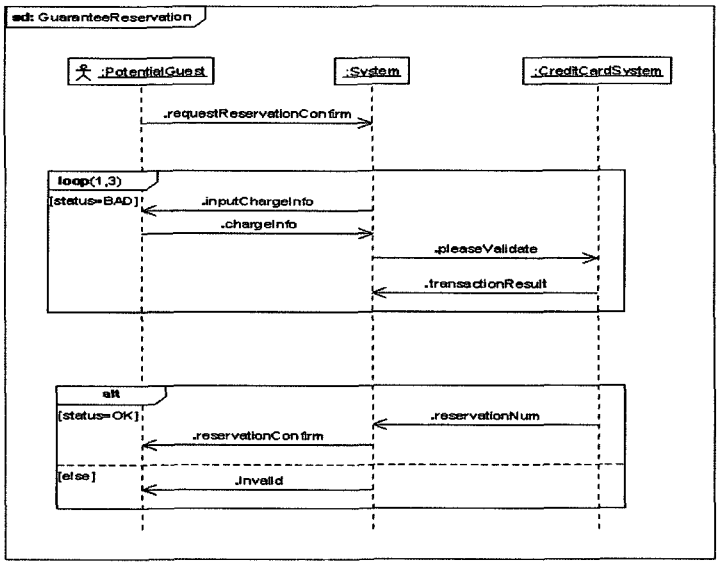



그림 9 순차 다이어그램(GuaranteeReservation.zuml)의 예

```
<process:CompositeProcess rdf:ID="GuaranteeReservationProcess">
 <process:composedOf>
  <process:Sequence>
   <process:components>
    <process:ControlConstructList>
     <list:first>
      <process:Perform rdf:ID="RequestReservationPerform">
       <process:process rdf:resource="requestReservationConfirm"/>
      </process:Perform>
     </list:first>
     <list:rest>
      <process:ControlConstructList>
       <list:first>
        <process:Repeat-While>
         <process:whileCondition>
          <expr:Condition rdf:ID="RW_Condition">
           <expr:expressionLanguage rdf:resource="#SWRL"/>
          </expr:Condition>
         </process:whileCondition>
         <process:whileProcess>
          <process:Sequence>
           <process:components>
            <process:ControlConstructList>
             <list:first>
              <process:Perform rdf:ID="inputChargeInfoPerform">
               <process:process rdf:resource="inputChargeInfo"/>
              </process:Perform>
             </list:first>
             <list:rest>
              <process:ControlConstructList>
               <list:first>
                <process:Perform rdf:ID="chargeInfoPerform">
                 <process:process rdf:resource="chargeInfo"/>
                </process:Perform>
               </list:first>
               .....
              </process:ControlConstructList>
             </list:rest>
            </process:ControlConstructList>
           </process:components>
          </process:Sequence>
         </process:whileProcess>
        </process:Repeat-While>
       </list:first>
       <list:rest>
        <process:ControlConstructList>
         <list:first rdf:resource="#If-Then-Else_35"/>
         <list:rest rdf:resource="#nil"/>
        </process:ControlConstructList>
       </list:rest>
      </process:ControlConstructList>
     </list:rest>
    </process:ControlConstructList>
   </process:components>
  </process:Sequence>
 </process:composedOf>
</process:CompositeProcess>
```

(a) OWL-S 온톨로지

```
<xsl:template match="UML2:Interaction.fragment">
  .....
 <xsl:apply-templates select=".../Interaction.fragment"/>
  .....
</xsl:template>
  .....
<xsl:comment>Interactionfragment</xsl:comment>
<xsl:template match="UML2:Interaction">
 <xsl:variable name="control">
  <xsl:value-of select="UML2:Interaction.fragment
   /UML2:CombinedFragment/@interactionOperator"/>
 </xsl:variable>
 <xsl:choose>
  <xsl:when test="$control='loop'">
   <process:Repeat-While>
    <xsl:variable name="constraint">
     <xsl:value-of select="UML2:CombinedFragment/
      UML:ModelElement.constraint/
      UML:Constraint/@xmi.idref"/>
    </xsl:variable>
    <xsl:variable name="constraint_body">
     <xsl:value-of select="UML:Constraint[@xmi.id=
      $constraint]/UML:Constraint.body/
      UML:BooleanExpression/@body"/>
      .....
    </xsl:variable>
    .....
    <process:whileCondition>
     <expr:SWRL-Condition>
      <expr:expressionBody rdf:parseType="Literal">
       <swrl:AtomList>
        <ref:first>
         .....
       </swrl:AtomList>
      </expr:expressionBody>
     </expr:SWRL-Condition>
    </process:whileCondition>
  </xsl:when>
  <xsl:when test="$control='par'">
   .....
  </xsl:when>
  <xsl:when test="$control='alt'">
   .....
  </xsl:when>
 </xsl:choose>
</xsl:template>
  .....
<xsl:template match="text()|@*">
</xsl:template>
```

(b) XSLT 스크립트

그림 10 그림 9로부터 생성된 OWL-S 온톨로지와 이를 위한 XSLT 스크립트

런 경우에도 Sequence 요소가 추가된다. 그림 13은 Split+Join을 사용한 OWL-S의 구조에서 전문가에 의해 직접 기술된 OWL-S와 제안한 방법을 이용해 생성된 OWL-S의 구조적 차이를 보여준다. 전문가의 방법에서는 병행처리 되는 Process1과 Process2를 Sequence를 사용해 묶지 않는다. 그러나 제안된 방법에서는 각각의 프로세스를 Sequence를 사용해 묶고 있는 것을 확인 할 수 있다. 이것은 전문가가 기술한 OWL-S와 비교할 때 구조적 최적화에서 차이를 가질 뿐 두 OWL-S 문서 사이의 의미적 차이를 갖는 것을 의미하지는 않는다.

표 7 전문가와 제안한 방법을 이용해 생성된 OWL-S의 요소 비교의 예

| 요소 | 전문가 | 제안된 방법 |
|---|---|---|
| compositeProcess | 2 | 2 |
| atomicProcess | 7 | 7 |
| hasInput | 10 | 10 |
| hasOutput | 13 | 13 |
| hasPrecondion | 2 | 2 |
| inputBinding | 7 | 7 |
| outputBinding | 4 | 4 |
| hasResult | 1 | 1 |
| Sequence | 3 | 6 |
| If-Then-Else | 1 | 1 |
| Split + Join | 1 | 1 |

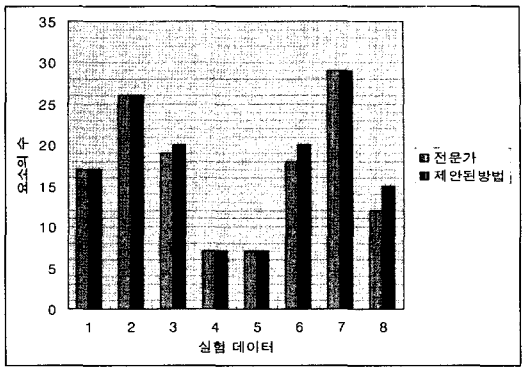



그림 11 순차 다이어그램을 통해 생성된 OWL-S의 비교 그래프

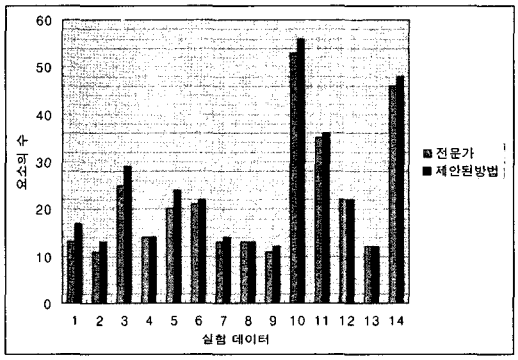



그림 12 활동 다이어그램을 통해 생성된 OWL-S의 비교 그래프

**전문가에 의해 기술된 OWL-S의 구조**

```
<process:Split-Join>
 <list:first>
  <process:Perform rdf:ID="Process_1Perform>
   <process:process rdf:resource="#Process_1">
   </process:process>
  </process:Perform>
 </list:first>
 <list:rest>
  <list:first>
   <process:Perform rdf:ID="Process_2Perform>
    <process:process rdf:resource="#Process_2">
    </process:process>
   </process:Perform>
  </list:first>
 <list:rest rdf:resource="#nil"/>
 <list:rest rdf:resource="#nil"/>
</process:Split-Join>
```

**제안된 방법으로 생성된 OWL-S의 구조**

```
<process:Split-Join>
 <list:first>
  <process:Sequence>
   <list:first>
    <process:Perform rdf:ID="Process_1Perform>
     <process:process rdf:resource="#Process_1">
     </process:process>
    </process:Perform>
   </list:first>
   <list:rest rdf:resource="#nil"/>
  </process:Sequence>
 </list:first>
 <list:rest>
  <list:first>
   <process:Sequence>
    <list:first>
     <process:Perform rdf:ID="Process_2Perform>
      <process:process rdf:resource="#Process_2">
      </process:process>
     </process:Perform>
    </list:first>
    <list:rest rdf:resource="#nil"/>
   </process:Sequence>
  </list:first>
  <list:rest rdf:resource="#nil"/>
 </list:rest>
</process:Split-Join>
```

그림 13 전문가의 방법과 제안한 방법을 이용해 생성된 OWL-S의 구조적 차이에 대한 예

### 4.3 기존 연구와의 비교

Timm과 Gannod는 UML의 클래스 다이어그램을 이용해 OWL-S를 기술한다. 1.0 버전의 OWL-S를 생성하고, 클래스 다이어그램을 사용하기 때문에 복합 프로세스의 다양한 제어구조를 표현하지 못하고 순차와 활동 다이어그램에 대해서는 고려하지 않고 있다.

Grønmo 등은 시맨틱 웹 서비스의 탐색, 호출, 조합을 위해 UML을 이용해 OWL-S문서를 기술한다. 1.1 버전의 OWL-S를 기술하지만 복합 프로세스의 다양한 제어구조에 대한 기술은 불가능하고 순차 다이어그램에 대해서는 고려하지 않고 있다.

MDA 기반의 OWL-S 생성에 과한 기존 연구는 단일 프로세스의 처리, 입력과 출력, preCondition, Effect 등의 기술은 가능하다. 그러나 웹 서비스는 하나의 오퍼레이션에 의해 처리되는 경우는 극히 드물고 이런 단일 프로세스들의 결합을 통해 생성되는 복합 프로세스가 될 확률이 높다. 따라서 복합 프로세스의 다양한 제어구조를 기술하는 방법이 필요하다. 기존 연구에서는 이 방법을 향후 연구로 언급하고 있다. 그러나 본 논문에서는 복합 프로세스의 다양한 제어구조를 기술하는 방법을 제안함으로써 이 문제를 해결했다. 표 8은 제안된 방법과 기존 연구의 비교분석표이다.

표 8의 결과에서 알 수 있듯이 기존 연구는 복합 프로세스의 다양한 제어구조를 제대로 표현하지 못한다. 활동 다이어그램만을 고려하고 있다. 그러나 제안된 방법을 이용하면 다양한 제어구조의 표현이 가능하고 순차 다이어그램도 표현할 수 있다. 제안된 방법과 기존 연구에서 Any-order를 표현하지 못하는 것은 UML 다이어그램으로 이 제어구조를 표현하지 못하기 때문이다. 본 논문에서 제안한 방법을 이용하면 가장 큰 특징으로 시스템 설계 시 사용된 기존의 UML 파일을 재사용할 수 있다. 이것은 문법이 복잡하여 일반 사용자가 직접 작성하기 어려운 OWL-S를 편리하게 생성하는 것을 가능하게 한다.

표 8 제안된 방법과 기존 연구의 실험결과 비교

| | Timm과 Gannod | Grønmo 등 | 제안된 방법 |
|---|---|---|---|
| atomicProcess | O | O | O |
| compositeProcess | X | △ | O |
| sequence | X | O | O |
| if-then-else | X | O | O |
| choice | X | O | O |
| any-order | X | X | X |
| split | X | O | O |
| split+join | X | O | O |
| repeat-until | X | X | O |
| repeat-while | X | X | O |
| input | O | O | O |
| output | O | O | O |
| preCondition | O | O | O |
| effect | O | O | O |
| (in\|out)put_binding | X | O | O |
| parameterType | O | O | O |
| UML 파일의 재사용 | X | X | O |
| 순차다이어그램 | X | X | O |
| 활동다이어그램 | X | O | O |

## 5. 결론 및 향후 연구 방향

시맨틱 웹 서비스는 웹 서비스에 대한 온톨로지를 구축하고 기계가 처리할 수 있는 형태로 시맨틱 마크업을 한다. 그 결과 웹에 있는 정보를 컴퓨터가 좀 더 이해할 수 있도록 도와주는 표준과 기술을 개발하여 시맨틱 검색, 데이타 통합, 네비게이션, 태스크의 자동화 등을 지원한다. 시맨틱 웹 서비스 기술 표준인 OWL-S는 서비스의 의미적 기술을 제공하기 위해 만들어진 온톨로지 언어이다.

일반 사용자가 OWL-S를 직접 기술하는 것은 OWL-S의 스펙이 복잡하고 문법이 장황하기 때문에 어렵지만 UML 모델을 재사용하면 시맨틱 웹 서비스 기술 언어인 OWL-S를 편리하게 기술할 수 있다. 제안된 방법은 온톨로지 기술을 위해 클래스 다이어그램을 사용하고, 프로세스의 흐름을 기술하기 위해 순차 또는 활동 다이어그램을 사용한다. 또한 OWL-S의 특징을 자세히 기술하기 위해 UML 프로파일을 사용한다. UML로 작성된 다이어그램들은 하나의 XMI 파일로 추출되고 XSLT 스크립트를 통해 OWL-S로 변환된다.

기존 연구들은 UML 모델의 재사용을 고려하지 않고 OWL-S 복합 프로세스의 다양한 제어구조를 기술 할 수 없다는 단점을 갖는다. 그러나 제안된 방법은 UML 모델의 재사용을 고려하고 있고, 순차 다이어그램과 활동 다이어그램을 사용해 다양한 제어구조를 기술한다. 제안한 방법을 이용하면 OWL-S로 기술할 수 없는 UML 기호들이 존재하기 때문에 UML 모델의 모든 기호들을 OWL-S로 변환하지 못한다는 제약점을 갖게 된다. 또한 제안된 방법은 그라운딩 정보가 없는 OWL-S를 생성한다. 따라서 향후연구로는 그라운딩 정보를 갖는 OWL-S의 생성을 가능하게 할 것이다.

## 참 고 문 헌

[1] OWL Service Coalition, Owl-s: Semantic markup for web services. White paper - http://www.daml.org/ services, July 2004.

[2] Miller, J and Mukerji, J., et al, MDA Guide Version 1.0.1. Technical Report omg/2003-06-01,

Object Management Group, June 2003.
[3] World Wide Web Consortium, XSL Transformations (XSLT) v1.0, W3C Recommendation http://www.w3c.org/TR/1999/REC-xslt-19991116, 1999.
[4] Grønmo, R., Jaeger, M.C., and Hoff, H., "Transformations Between UML and OWL-S," *Proc. Conf. Foundations and Applications, First European*, pp. 269-283, 2005.
[5] Martinez, A., Marta, P., Ricardo J., and Francisco P., "ZenFlow: A Visual Web Service Composition Tool for BPEL4WS," *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 181-188, 2005.
[6] Grønmo, R. and Oldevik, J., "An Empirical Study of the UML Model Transformation Tool (UMT)," *Proc. First Int'l Conf. Interoperability of Enterprise Software and Applications*, 2005.
[7] Pfadenhauer, K., Kittl, B., and Dustdar, S., "Challenges and Solutions for Model Driven Web Service Composition," *Proc. 14th IEEE Int'l Workshops on Enabling Technologies*, pp 126-134, 2005.
[8] Timm, J. T. E. and Gannod, G. C., "A Model-Driven Approach for Specifying Semantic Web Services," *Proc. 3rd IEEE Int'l Conf. Web Services*, pp. 313-320, 2005.
[9] Jaeger, M. C., Engel, L., and Geihs, K., "A Methodology for Developing OWL-S Descriptions," *Proc. First Int'l Conf. Interoperability of Enterprise Software and Applications*, pp 153-166, 2005.
[10] Djuric, D., Gasevic, D., Devedzic, V., and Damjanovic, V., "A UML Profile for OWL Ontologies," *Proc. Model Driven Architecture: Foundations and Applications*, pp. 138-152, 2004.
[11] Skogan, D., Grønmo, R., and Solheim, I., "Web Service Composition in UML," *Proc. 8th IEEE Int'l Conf. Enterprise Distributed Object Computing*, pp. 47-57, 2004.
[12] Gannod, G. C. and Timm, J. T. E., "An MDA-based Approach for Facilitating Adoption of Semantic Web Service Technology," *Proc. 8th IEEE Enterprise Distributed Object Computing Conference Workshop on Model-Driven Semantic Web*, 2004.
[13] Koch, N., Fraternali, P., and Wirsing, M., "MDA Applied: From Sequence Diagrams to Web Service Choreography," *Proc. 4th Int'l Conf. Web Engineering*, pp. 132-136, 2004.
[14] Thöne, S., Depke, R., and Engels, G., "Process-Oriented, Flexible Composition of Web Services with UML," *Proc. 3rd Int'l Joint Workshop on Conceptual Modeling Approaches for E-business: A Web Service Perspective*, pp 390-401, 2002.
[15] Schmuller, J., Teach Yourself UML in 24 Hours, 3rd., p422, Sams, 1999.
[16] Ian Horrocks et al, SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical Report, http://www.w3.org/Submission/2004/SUBM-SWRL- 20040521/, May 2004.

김 일 웅
2005년 2월 가톨릭대학교 컴퓨터공학과 졸업(학사). 2005년 9월~현재 연세대학교 컴퓨터과학과 석사과정. 관심분야는 Model Driven Architecture for Semantic Web Services

이 경 호
1995년 2월 연세대학교 전산과학과 졸업(학사). 1997년 2월 연세대학교 컴퓨터과학과 졸업(석사). 2001년 2월 연세대학교 컴퓨터과학과 졸업(박사). 2001년 3월 National Institute of Standard and Technology(NIST) 객원연구원. 2002년~현재 연세대학교 컴퓨터과학과 부교수. 관심분야는 Internet Computing, Service-Oriented Computing, Multimedia Document Engineering