시맨틱 웹 서비스의 기술을 위한 모델지향접근방식

김일웅[0] 김연석 이경호
연세대학교 컴퓨터과학과
{iwkim[0], yskim}@icl.yonsei.ac.kr, khlee@cs.yonsei.ac.kr

A Model Driven Approach for Describing Semantic Web Service

Il-Woong Kim[0], Yeon-Seok Kim, and Kyong-Ho Lee
Dept. of Computer Science, Yonsei University

요 약

본 논문에서는 UML 다이어그램으로부터 시맨틱 웹 서비스 기술 표준인 OWL-S문서를 생성하는 방법을 제안한다. 특히 온톨로지의 기술을 위해 클래스 다이어그램을 사용하고, 프로세스의 흐름을 기술하기 위해 시퀀스 다이어그램 및 액티비티 다이어그램을 사용한다. 제안된 방법은 UML 모델의 재사용 및 기존 연구들이 처리하지 못하는 복합프로세스의 생성이 가능하다는 장점을 가진다.

1. 서론

시맨틱 웹 서비스는 웹 서비스에 대한 온톨로지를 구축하고 기계가 처리할 수 있는 형태로 시맨틱 마크업(markup)을 함으로써 지능적 웹 서비스의 발견, 실행, 조합, 모니터링이 가능하도록 한 기술이다. 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 위해서는 반드시 요구되는 기술이라고 할 수 있다. OWL-S [1] 는 Ontology Web Language (OWL)을 기반으로 시맨틱 웹 서비스를 기술하기 위해 작성된 상위 온톨로지이다. Unified Modeling Language (UML)은 여러 가지 다이어그램들을 제시함으로써 소프트웨어 개발과정의 산출물들을 비주얼하게 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 신뢰성있는 시스템으로 평가받을 수 있다. 이미 산업계의 표준으로 자리 잡은 UML을 사용해 개발된 많은 시스템이 존재한다. 웹 서비스가 시스템 통합플랫폼으로 활성화될 것이라는 기대를 받고 있고, 기존의 시스템들의 내부자원 통합 및 연계에 웹 서비스가 활용할 수 있기에 이런 시스템들은 웹 서비스로의 변환이 필요하게 되었다. 그러나 현재 OWL-S 온톨로지를 구축할 수 있는 저작 도구가 매우 빈약한 상태이고, OWL-S의 개념을 이해하여 온톨로지를 활용할 수 있는 전문 기술을 배우는 것은 많은 시간을 필요로 한다. 이렇듯 OWL-S를 직접 저작하는 일은 쉽지 않다. 기존의 UML 다이어그램을 재사용해 자동 또는 반자동으로 OWL-S를 생성할 수 있다면 위의 문제를 쉽게 해결 할 수 있다.

기존 UML을 OWL-S로 변환하는 연구들이 존재한다. Timm과 Gannod [2] 은 클래스 다이어그램을 사용하여 단일 프로세스(Atomic Process)를 생성한다. 그러나 OWL-S 1.0을 지원하고, Semantic Web Rule Language (SWRL)이나 Knowledge Interchange Format (KIF)로 조건을 기술하지 못하고, 복합 프로세스(Composite Process)는 생성하지 못하는 단점을 가진다. Grønmo 등 [3]은 UML 프로파일을 통한 확장으로 OWL-S 1.1의 단일프로세스를 생성하고, 클래스 다이어그램을 통해 온톨로지를 표현하지만 복합 프로세스는 생성하지 못한다. Skogan 등 [4]은 액티비티 다이어그램과 클래스 다이어그램을 사용해 Business Process Execution Language for Web Services (BPEL4WS)를 생성한다.

※ 이 연구는 정보통신부(정보통신연구진흥원)에서 지원하는 2005년도 IT기초기술연구지원사업의 연구결과임.

본 논문에서는 OWL-S의 생성을 위해 기존의 UML 모델을 적절하게 수정하고, 그 결과로 생성된 XML Metadata Interchange (XMI) 문서를 Extensible Stylesheet Language Transformations (XSLT) 스크립트를 사용해 OWL-S로 변환하는 방법을 제안한다.

2. 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 시퀀스 다이어그램 또는 액티비티 다이어그램으로 표현한다. 변환 단계는 이전 단계를 통해 만들어진 UML 모델로부터 생성된 XMI 문서를 XSLT 스크립트를 이용해 OWL-S로 변환한다. 각 단계에 대한 자세한 설명은 다음과 같다.

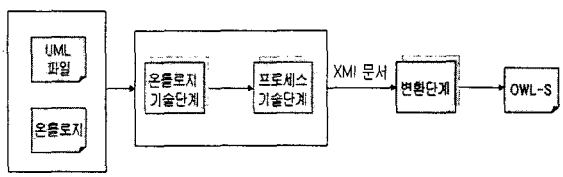

그림 1. 제안된 변환 방법.

2.1 온톨로지 기술

먼저 시맨틱 웹 서비스를 기술하는데 필요한 온톨로지를 기술한다. UML로 표현된 각각의 프로세스는 입력과 출력에 매핑되는 온톨로지의 개념들에 대한 정보를 필요로 한다. 이 온톨로지를 클래스 다이어그램을 사용해 기술한다. 입력파일로 주어진 온톨로지의 개념간의 계층관계와 각각의 개념이 갖는 속성을 클래스 다이어그램으로 기술한다(그림 2참조). 온톨로지를 표현한 클래스 다어그램은 패키지로 묶이며, 스테레오타입 <<ontology>>를 사용해 온톨로지임을 명시한다. 온톨로지를 나타내는 클래스는 스테레오타입 <<OntClass>>를 사용해 명시한다. OWL과의 매핑관계는 표 1과 같다. 클래스 이름이 온톨로지 개념의 이름이 되고, 속성은 스테레오타입을 사용하여 데이터타입속성과 오브젝트속성으로 구분된다. 일반화 관계

(Generalization)를 이용하여 개념간의 계층을 표현할 수 있다.

표 1. 클래스 다이어그램과 OWL의 매핑테이블.

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

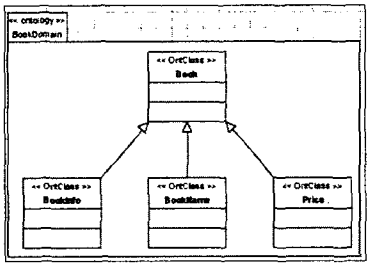

그림 2. 온톨로지를 표현한 클래스 다이어그램의 예.

2.2 프로세스 기술

UML에서 행위를 기술하는 다이어그램으로는 대표적으로 시퀀스 다이어그램과 액티비티 다이어그램이 있다. 기존에 존재하는 이 다이어그램들을 OWL-S로 직접 변환하는 것은 불가능하다. 따라서 OWL-S로의 변환에 필요한 정보를 삽입하거나 기존의 다이어그램을 적절하게 수정해야한다. 이 단계에서는 스테레오타입 및 제약(constrain)의 UML 프로파일을 통해서 다이어그램을 수정하는 방법을 제안한다.

시퀀스 다이어그램은 각각의 객체에 대한 정보를 클래스 다이어그램을 통해 얻으므로 클래스 다이어그램이 필요하며 함께 수정해준다.

2.2.1 시퀀스 다이어그램

시퀀스 다이어그램은 행위에 대한 정보만 가지고 있어 각각의 객체에 대한 정보를 담고 있는 클래스 다이어그램이 필요하다(그림 4참조). 클래스 다이어그램에서 오퍼레이션의 인자 값은 각각의 단일 프로세스의 입력이다. 오퍼레이션의 반환형이 불린(boolean)인 경우에 제약을 사용해 If-then-else를 기술한다. 출력과 입력의 바인딩은 노트를 사용한다(그림 3참조). 시퀀스 다이어그램과 OWL-S의 매핑관계는 표 2에서 확인할 수 있다.

표 2. 시퀀스 다이어그램과 OWL-S의 매핑테이블.

표준 UML 2.0 상태 모델 요소	OWL-S
Operation	AtomicProcess
Synchronous Asynchronous	Sequence
constraints	If-then-else

2.2.2 액티비티 다이어그램

액티비티 다이어그램은 UML에서 제공하는 여러 모델 중에 OWL-S로의 변환에 가장 적합한 다이어그램이다. 액티비티 다이어그램과 OWL-S의 매핑관계는 표 3에서 확인할 수 있다. 각각의 액티비티의 입·출력에 대해 Pin을 사용해 표현한다.

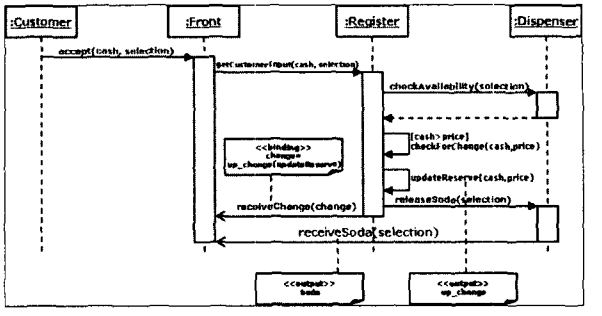

그림 3. 시퀀스 다이어그램의 예.

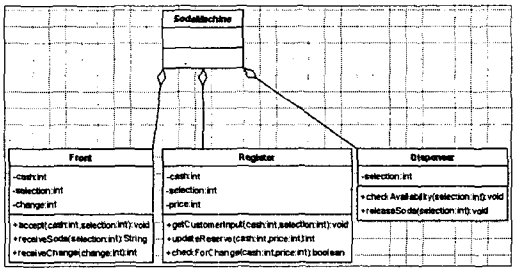

그림 4. 시퀀스 다이어그램을 위한 클래스 다이어그램의 예.

입·출력의 구분은 스테레오타입을 사용해서 표현하고, 각각의 프로세스간의 입·출력을 연결해 준다. Condition은 노트의 제약을 사용해서 표현한다. Choice, If-Then-Else, Repeat-Until, Repeat-While, Iterate는 DecisionNode를 사용해서 표현하고, 각각의 컴포넌트를 구분하기위해 스테레오타입을 사용한다(그림 5참조).

새로운 OWL-S를 작성할 경우에도 본 논문에서 제안한 방법으로 액티비티 다이어그램을 이용해 쉽게 OWL-S를 기술할 수 있다.

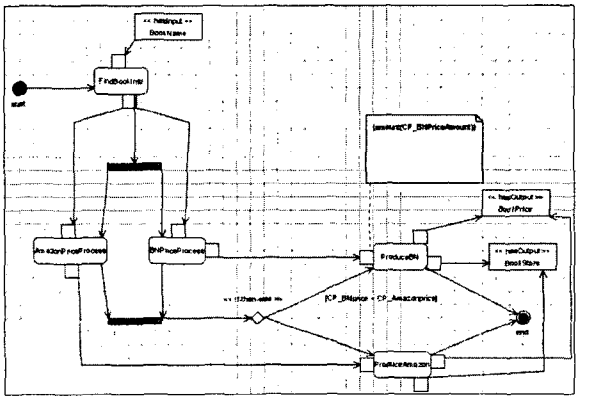

그림 5. 액티비티 다이어그램의 예.

2.3 변환

UML 툴을 통해 수정된 UML 모델은 하나의 xmi 파일로 추출된다. xmi 파일은 xslt 스크립트(그림 6참조)를 통하여

OWL-S 문서로 변환된다. 변환되어져 생성되는 파일은 프로파일과 프로세스모델이다.

표 3. 액티비티 다이어그램과 OWL-S의 매핑테이블.

표준 UML 2.0 상태 모델 요소	OWL-S
Sequence	Sequence
Fork	Split
Fork + Join	Split+Join
....	Any-Order
DecisionNode	Choice
	If-then-else
	Repeat-Until
	Repeat-While
	Iterate
Pin	Input
	Output
note	Condition

```
</xsl:template>

<xsl:template match="UML2:Activity.node">
    ........
    <xsl:apply-templates select=".../DecisionNode"/>
    ........
</xsl:template>
</xsl:template>
........
<xsl:comment>DecisonNode</xsl:comment>
<xsl:template match="UML:Namespace.ownedElement">
    <xsl:variable name="stereo">
        <xsl:value-of select="UML2:DecisionNdoe/
        UML:ModelElement.stereotype/UML:Stereotype/@xmi.idref"/>
    </xsl:variable>
    <xsl:variable name="stereo_name">
    <xsl:value-of select="UML:Stereotype[@xmi.id=$stereo]/
    @name"/>
    </xsl:variable>
    <xsl:choose>
        <xsl:when test="$stereo_name='If-then-else'">
            <process:If-then-else>
                <xsl:variable name="constraint">
                    <xsl:value-of select="UML2:DecisionNdoe/
                    UML:ModelElement.constraint/UML:Constraint/
                    @xmi.idref"/>
                <xsl:variable name="constraint_body">
                    <xsl:value-of select="UML:Constraint[@xmi.id=$constraint]/
                    UML:Constraint.body/UML:BooleanExpression/@body"/>
                ........
                </xsl:variable>
                ........
                <process:ifCondition>
                    <expr:SWRL-Condition>
                        <expr:expressionBody rdf:parseType="Literal">
                            <swrl:AtomList>
                                <rdf:First>
                                    ........
                            </swrl:AtomList>
                        </expr:expressionBody
                    </expr:SWRL-Condition>
                </process:ifCondition>
            </process:If-then-else>
        </xsl:when>
        <xsl:when test="$stereo_name='Choice'">
        ........
        </xsl:when>
        <xsl:when test="$stereo_name='Repeat-Until'">
        ........
        </xsl:when>
        ........
    </xsl:choose>
</xsl:template>
.....
<xsl:template match="text()|@*">
</xsl:template>
```

그림 6. xslt 스크립트의 예.

3. 성능분석

제안된 방법의 변환 과정을 거치면 그림 7의 OWL-S 문서가 생성된다.

본 논문에서 제안한 변환 방법은 복합 웹서비스에 대한 풍부한 기술이 가능하다(표 3참조). 시퀀스 다이어그램의 경우에는 Sequence와 If-then-else의 기술로 제한되는데, 이것은 시퀀스 다이어그램의 한계로 인한 것이다. 컨디션에 대한 명세는 조건문(If-Then-Else, Iterate), 입·출력에 대한 컨디션, Effect

```
<process:If-Then-Else>
  <process:ifCondition>
    <expr:SWRL-Condition>
      <expr:expressionBody rdf:parseType="Literal">
        <swrl:AtomList>
          <rdf:first>
            <swrl:BuiltinAtom>
              <swrl:builtin rdf:resource="&swrlb;#lessThan"/>
              <swrl:arguments>
                <rdf:List>
                  <rdf:first rdf:resource="#FindBNPriceBookPrice"/>
                  <rdf:rest>
                    <rdf:List>
                      <rdf:first rdf:resource="#FindAmazonPriceBookPrice"/>
                      <rdf:rest rdf:resource="&rdf;#nil"/>
                    </rdf:List>
                  </rdf:rest>
                </rdf:List>
              </swrl:arguments>
            </swrl:BuiltinAtom>
          </rdf:first>
          <rdf:rest rdf:resource="&rdf;#nil"/>
        </swrl:AtomList>
      </expr:expressionBody>
    </expr:SWRL-Condition>
  </process:ifCondition>
  <process:then>
    <process:Sequence>
      <process:components>
        <process:ControlConstructList>
          <list:first>
            <process:Perform rdf:ID="ProduceBNPerform">
              <process:process rdf:resource="#ProduceBN"/>
              <process:hasDataFrom>
                <process:InputBinding>
                  <process:toParam rdf:resource="#ProduceBNPrice"/>
                  <process:valueSource>
                    <process:ValueOf>
                      <process:fromProcess rdf:resource="#FindBNPrice"/>
                      <process:theVar rdf:resource="#FindBNPriceBookPrice"/>
                    </process:ValueOf>
                  </process:valueSource>
                </process:InputBinding>
              </process:hasDataFrom>
            </process:Perform>
          </list:first>
          <list:rest rdf:resource="&list;#nil"/>
        </process:ControlConstructList>
      </process:components>
    </process:Sequence>
  </process:then>
```

(a)

(b)

그림 7. 변환된 OWL-S 문서의 예.

에서 사용된다. 따라서 컨디션에 대해 풍부하게 기술할 수 있어야한다. 본 논문에서 제안하는 방법은 컨디션에 대한 풍부한 기술이 가능하다는 장점을 가진다. 그림 7의 (a)는 UML 프로파일인 제약을 통해 기술되는 컨디션이 SWRL을 통해 어떻게 기술되는지를 보여준다. Precondition과 Effect에 관한 기술은 노트를 사용한다. 각각의 프로세스간의 입·출력이 갖는 관계에 대한 기술은 복합 웹 서비스 기술 시 반드시 필요하다. (b)는 각각의 프로세스의 입력이 어떻게 바인딩되는지를 보여준다. 액티비티 다이어그램은 그림 5에서 확인할 수 있듯이 각각의 액티비티 간의 입·출력 관계를 연결해줄 수 있고, 그 정보를 이용해 입·출력 바인딩을 기술할 수 있다. 본 논문에서 제안하는 UML을 통한 OWL-S 변환 및 생성은 시각적인 작업이기 때문에 직접 OWL-S를 기술하는 것보다 더 효과적인 방법이다.

참고문헌

[1] OWL-S: Semantic Markup for Web Services [Online] Available http://www.daml.org/services/owl-s/1.1/overview/, Dec. 2004.

[2] John T. E. Timm and Gerald C. Gannod, "A Model-Driven Approach for Specifying Semantic Web Services," Proc. the 3rd IEEE Int'l Conf. Web Services, pp. 313-320, 2005

[3] Roy Grønmo, Michael C. Jaeger, and Hjørdis Hoff, "Transformations Between UML and OWL-S," Proc. Foundations and Applications: First European Conf, pp. 269-283, 2005

[4] David Skogan, Roy Grønmo, and Ida Solheim, "Web Service Composition in UML ," Proc. The Eighth IEEE International Enterprise Distributed Object Computing Conf, pp. 47-57, 2004.