제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

서비스 온톨로지와 규칙표현의 통합에 관한 근간

양진혁, 정인정
고려대학교 전산학과
e-mail:{grjinh, chung}@korea.ac.kr

Foundation on Integration of Service Ontology and Rule Representation

Jin Hyuk Yang, In Jeong Chung
Dept of Computer Science, Korea University

요 약

본 논문에서는 지능형 e-비즈니스의 효과적인 수행을 지원하기 위하여 서비스 온톨로지 표현 언어 OWL-S와 규칙표현 언어 RuleML 사이의 관계를 살펴봄으로써 서비스 온토롤지 및 규칙 표현 모두가 함께 사용될 수 있는 통합모델의 이론적인 근간을 제공한다. 이를 위하여 OWL-S의 정형 시맨틱스를 기술하고 로직 프로그램과의 관계를 분석한 후 두 마크업 표현의 상호매핑을 보인다.

1. 서론

시맨틱 웹 서비스[1] 기술은 웹 서비스의 의미를 자동으로 추론하고 처리할 수 있기 때문에 효과적인 지능형 e-비즈니스의 수행을 지원하기 위한 핵심기술이다[2]. OWL-S[3]는 시맨틱 웹 서비스 마크업을 위한 사실상의 표준으로서 표현력이 풍부하고 온톨로지 표준 마크업인 OWL[4][1]로 표현되는 서비스 온톨로지이다. 한편 규칙 마크업인 RuleML[5]은 전통적으로 e-비즈니스의 다양한 분야에서 사용되어온 표현이다. 지능형 e-비즈니스를 더욱더 효과적으로 수행하기 위해서는 서비스 온톨로지 정보와 규칙표현이 함께 사용될 수 있는 통합모델이 요구된다. 이에 본 논문에서는 서비스 온톨로지 마크업인 OWL-S와 규칙표현 RuleML 사이의 관계를 분석함으로써 통합모델의 이론적인 근간을 제공한다. 이를 위하여 먼저 OWL-S의 정형 시맨틱스를 기술하고 이와 규칙표현과의 상호관계를 분석함으로써 통합모델의 실현성을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 OWL-S와 RuleML 통합모델의 중요성들에 대해 언급하고 3장에서는 OWL-S와 RuleML 사이의 상호매핑을 정의 및 분석함으로써 통합모델의 이론적인 근간을 제공한다. 마지막으로 4장에서는 결론과 향후과제를 언급한다.

2. 관련연구 및 통합모델의 중요성

통합모델의 이론적인 근간을 통한 본 논문의 주요 동기들 및 목적들은 다음과 같다.

- OWL-S의 정형 시맨틱스의 제공: OWL-S는 기술로직(Description Logic: DL) 기반의 OWL로 작성된 표현력이 풍부하고 추론이 가능한 시맨틱 웹 서비스를 기술하기 위한 언어이다. 그러나 [2]에서 지적되고 있듯이 알려진 많은 한계점들이 존재한다. 특히 OWL-S의 가장 큰 문제점은 정형 시맨틱스의 부족이다. OWL-S의 정형 시맨틱스에 관한 연구는 Situation Calculus(SC)와 페트리 넷을 이용하여 operational 시맨틱스를 제안한 [6]과 subtype개념을 이용하여 제안한 [7]이 존재한다.

본 논문에서 중요하게 고려하는 접근법은 OWL-S의 axiomatic 시맨틱스인데 그 이유는 공리들의 집합으로 표현된 OWL-S의 프로세스 정보는 로직 프로그램(LP: Logic Program)으로의 매핑정의를 통하여(3장에서 언급됨) 추론엔진을 통한 추론이 가능해지기 때문이다. 따라서 우리는 3장에서 OWL-S의 axiomatic 시맨틱스를 SC로 기술한 [6]의 정형 시맨틱스를 복합(composite) 프로세스를 포함하도록 확장하여 제안한다.

- LP 기반의 추론엔진 활용: OWL-S는 DL기반의 OWL로 작성된 서비스 온톨로지이므로 이론적으로 OWL-S로 작성된 온톨로지 정보에 대해 추론하기 위해서는 DL기반의 추론엔진을 사용해야만 한다. 그러나 DL기반의 추론엔진에 비해 LP기반의 엔진들은 훨씬 더 좋은 성능으로 더 많이 알려져 있다[8].
- OWL-S의 한계점들 극복: 우리의 선 연구 [2]와 같은 관련연구들은 온톨로지와 규칙의 통합모델이 OWL-S의 표현력 및 한계점들을 향상시킬 수 있음을 보이고 있다.

이밖에 [8]에서 언급되듯이 온톨로지와 규칙 언어들 사이의 매핑은 시맨틱 웹 언어의 레이어링, 표현력이 강한 질의, 데이터 통합 및 시맨틱 웹 서비스를 위해 중요하다.

3. 서비스 온톨로지 마크업과 규칙표현 사이의 매핑

3장에서 정의할 서비스 온톨로지와 규칙 사이의 매핑은 신택스와 시맨틱스 두 부분으로 나누어 설명된다. 왜냐하면 OWL-S가 OWL로 작성된 서비스 온톨로지이나 그것이 서비스의 행동을 모델링하는 수단이므로 OWL의 신택스에서 제공되는 프리미티브들로 OWL-S의 의도된 시맨틱스를 이끌어 낼 수 있을 만큼 OWL이 충분히 강력하지 않기 때문이다[7]. 따라서 우리는 3.1절에서 먼저 OWL과 RuleML사이의 관계를 조사하고, 3.2절에서 OWL-S의 동적인 행동들의 시맨틱스와 RuleML사이의 관계를 기술한다. 그리고 3.3절에서 매핑에 관한 분석을 제공한다.

3.1 OWL과 RuleML사이의 매핑

OWL과 RuleML사이의 매핑은 기본적으로 [8]의 연구결과에 의존한다. [8]에서는 FOL(First Order Logic)의 부분집합들인 DL과 LP사이의 교집합을 DHL(Description Horn Logic)로 정의하였고 이에 대응되는 LP를 DLP(Description Logic Program)으로 정의하였다. DLP는 def-Horn의 표현력을 가지는데, 여기서 def-Horn은 definite equality-free Datalog LP를 말한다. 즉, 로직 프로그램을 구성하는 클로즈들의 바디에서 NAF(Negation As Failure)가 출현하지 않고 equality를 명시하는 predicate가 존재하지 않으며 0보다 큰 arity를 가지는 function 기호가 존재하지 않는 로직 프로그램을 말한다.

다시 말해 [8]에서 제안된 온톨로지와 규칙사이의 매핑정의 이

1) OWL은 OWL Lite, OWL DL 및 OWL Full로 구성되는데, OWL-S는 OWL DL로 표현된다.

면의 아이디어는 FOL의 부분집합들 DL과 LP의 교집합을 DLP로 정의하였고, DLP를 DL 및 LP사이의 상호매핑 기준으로 고려하였다. 이 연구결과는 DL기반의 언어들인 RDFS와 DAML+OIL의 클래스들 및 속성들에 대한 DLP로의 매핑을 가능하게 하고, 그럼으로써 온토롤지 정보가 규칙정보로 표현될 수 있게 한다. 이는 DL에 비해 상대적으로 많은 추론엔진들과 효율적인 알고리즘을 가지는 LP관련 기술들을 DL추론을 위해 사용할 수 있는 이론적인 근간을 제공한다는 측면에서 매우 의미가 크다.

RDF(S) 및 DAML+OIL의 프리미티브들과 클래스 생성을 위한 구조물들에 대한 DL 및 FOL로의 표현과 이 표현의 LP로의 매핑에 대한 자세한 정의는 [8]을 참조한다.

3.2 OWL-S의 동적인 행동들에 대한 매핑

OWL-S는 OWL로 작성된 서비스 온톨로지이므로, 즉 OWL-S가 OWL의 신택스에 의존하여 작성되므로 OWL-S와 규칙과의 매핑에 대한 부가적인 매핑정의는 불필요하게 보일 수도 있다. 그러나, OWL-S는 동적인 행동을 가지는 서비스, 즉 프로세스를 모델링하기 위한 수단이므로 DAML+OIL의 표현력을 이용하여 OWL-S가 가지는 의도된 해석들을 모두 이끌어낼 수 없다[6,7]. 따라서, OWL-S와 규칙사이의 매핑 정의를 완전하게 구축하기 위해서는 서비스의 동적인 행동을 모델링하고 있는 OWL-S의 프로세스 모델 온톨로지와 규칙표현사이의 매핑에 관한 연구 및 분석이 필요하다.

이를 위하여 본 논문에서는 OWL-S의 정형 시맨틱스에 관한 연구결과들 중 SC로 OWL-S의 프로세스 모델 시맨틱스를 기술한 연구결과 [6]을 부분적으로 활용하고(atomic 프로세스 정의) 보완하여(complex 프로세스들의 정의 및 한계점 지적) OWL-S의 완전한 정형 시맨틱스를 제공한다. 그리고 난 후 SC로 기술된 OWL-S의 프로세스 모델 시맨틱스를 가지고 로직 프로그램과의 매핑 정의에 관한 연구결과를 제공함으로써 FOL의 두 부분집합들인 DL기반의 OWL-S와 LP기반의 RuleML사이의 매핑을 기술한다.

3.2.1 OWL-S의 정형 시맨틱스

OWL-S를 SC로 완전히 기술하기 위해서는 작용 이론(action theory) D로 불리는 공리들의 집합들 $D=\Sigma\cup D_{ss}\cup D_{\phi}\cup D_{una}\cup D_{S_0}$를 명시하는 것이 필요하다[6]. Σ는 situation들을 위한 foundational 공리들이고, D_{ss}는 functional 및 relational fluent들을 위한 successor state 공리들이다. 여기에서 fluent는 situation 인수를 그 fluent의 마지막 인수로 취하는 SC에서의 관계(relation)를 말한다. D_{ϕ}는 작용 precondition 공리들이고, D_{una}는 작용들을 위한 고유 명칭들(unique names) 공리들이며, D_{S_0}는 S_0에서 uniform[9]한 first-order 문장들이다.

이제 기본적인 작용 이론 D의 측면에서 OWL-S의 프로세스 모델에 대한 정형 시맨틱스를 정의한다. 먼저 OWL-S의 atomic 프로세스 측면[6]에서 먼저 기술하고 난 후 composite 프로세스를 구성하기 위한 구조물들에 대한 시맨틱스를 기술한다. OWL-S 안에서 IOPE(Input, Output, Precondition, Effect)를 가지는 (atomic) 프로세스들은 SC안에서 입력 매개변수들을 가지는 작용으로서 모델링될 수 있다.

다음은 [6]에서 정의한 DAML-S의 atomic 프로세스에 대한 SC 시맨틱스이다.

• Atomic 프로세스의 effects들은 SC안에서 다음 형태의 positive 및 negative effect 공리들로서 표현된다.

$Poss(a,s)\wedge\gamma^{+}_{F}(x,a,s)\rightarrow F(x,do(a,s))$

$Poss(a,s)\wedge\gamma^{-}_{F}(x,a,s)\rightarrow\neg F(x,do(a,s))$

여기에서 $\gamma^{+/-}_{F}(x,a,s)$는 fluent F를 작용 a의 실행 후에 각각 참/거짓으로 만들 작용들과 조건들의 모든 서로 다른 조합들을 포함한다.

• 출력들은 에이전트가 알게 되는 정보이므로, 의미적으로 지식 effects들로서 취급될 수 있다. 이런 effects들은 KRef, KWhether 또는 Knows 표현이다[6,9][2].

• Atomic 프로세스를 위한 OWL-S preconditions들은 SC안에서 well-formed 포뮬러[9]로서 표현된다. Atomic 프로세스의 각 precondition은 SC안에서 작용들을 위한 필요조건으로서 표현된다. 즉, $Poss(a,s)\rightarrow\pi_i$ 여기에서, π_i는 s로 상대화된(relativized) 포뮬러이다. 여럿의 preconditions들에 대해서 이것은 $Poss(a,s)\rightarrow\pi_1\wedge\pi_2\wedge...\wedge\pi_n$으로 일반화된다.

• 출력들이 지식 effects들로서 취급되는 것과 똑같이 입력들 또한 의미적으로 지식 precondition들로서 취급된다. 에이전트는 그것이 그 서비스를 실행시킬 수 있기 전에 그 서비스로의 입력들의 값을 알아야만 한다. 그러므로 OWL-S의 atomic 프로세스 a의 각 입력 ϕ_i에 대해서 입력들은 $Poss(a,s)\rightarrow KRef(\phi_1,s)\wedge...\wedge KRef(\phi_n,s)$로 표현된다.

• Preconditions들이 atomic 프로세스를 위해 모든 의도된 preconditions들을 인코드한다는 완전성 가정 하에서, 작용을 위한 이런 필요조건들은 $Poss(a,s)\equiv\pi_1\wedge\pi_2\wedge...\wedge\pi_n\wedge KRef(\phi_1,s)\wedge...\wedge KRef(\phi_n,s)$형태의 작용 precondition 공리들로 컴파일된다.

지금껏 [6]에서 기술된 OWL-S의 atomic 프로세스에 관한 시맨틱스를 살펴보았다. 다음은 OWL-S에서 제공되는 10개의 구조물들인 Sequence, Split, Split+Join, Choice, Unordered, Condition, If-Then-Else, Iterate, Repeat-While 및 Repeat-Until을 이용하여 조합될 수 있는 composite 프로세스의 시맨틱스를 SC로 정의한다. 이를 위하여 먼저 $Do(\delta,s,s')$의 귀납적 정의가 필요하다. $Do(\delta,s,s')$[9]는 δ에 의해 명시된 원론의 작용들을 실행함으로써 그것이 situation s'에서부터 s에 다다르는 것이 가능하다는 것을 말한다.

• 프리미티브 작용들: $Do(a,s,s')\equiv Poss(a[s],s)\wedge s'=do(a[s],s)$. 표기 a[s]는 situation 인수(argument) s를 작용 텀(term) a에 의해서 언급되는 모든 functional fluents들로 복구함으로써 야기되는 결과를 의미한다.

• 테스트 작용들: $Do(\phi,s,s')\equiv\phi[s]\wedge s=s'$. 여기에서 테스트 표현 ϕ는 모든 situation 인수들이 압축(suppressed)된, SC의 언어 안의 포뮬러를 구성하는 표현(SC 포뮬러가 아님)이다. $\phi[s]$는 situation 변수 s를 ϕ에서 언급된 모든 fluent 명칭들(relational 및 functional)로 복구함으로써 ϕ로부터 획득되는 SC 포뮬러를 나타낸다.

• 순서(sequence):
$Do(\delta_1;\delta_2,s,s')\equiv(\exists s'').Do(\delta_1,s,s'')\wedge Do(\delta_2,s'',s')$.

• 2개의 작용들 중 비결정적인 선택:
$Do(\delta_1|\delta_2,s,s')\equiv Do(\delta_1,s,s')\vee Do(\delta_2,s,s')$

• 작용 인수들 중 비결정적인 선택:
$Do((\pi x)\delta,s,s')\equiv(\exists x)Do(\delta x),s,s')$.

• 비결정적인 반복: δ를 0번 또는 그 이상 실행하라. $Do(\delta^*,s,s')\equiv(\forall P).\{(\forall s1)P(s1,s1)\wedge(\forall s1,s2,s3)[Do(\delta s1,s2)\wedge P(s2,s3)\supset P(s1,s3)]\}\supset P(s,s')$. 다시 말해, δ를 0번 또는 그 이상 수행하는 것은 s로부터 s'으로 이동한다.

Conditional들과 while-loops들은 다음과 같이 이전의 구조물들 측면에서 정의될 수 있다.

• **if** ϕ **then** δ_1 **else** δ_2 **endIf** $\equiv[\phi?;\delta_1]|[\neg\phi?;\delta_2]$,

• **while** ϕ **do** δ **endWhile** $\equiv[\phi?;\delta^*;\neg\phi?]$.

[9]에서 정의된 Do에 대한 상기의 정의들을 가지고, 이제 OWL-S composite 프로세스를 위한 구조물들의 시맨틱스를 다음과 같이 정의한다. [6]에서 예상하였고 또한 아래에서 볼 수 있듯이, 대부분의 OWL-S 제어 구조물들의 시맨틱스는 상기 Do 매크로-확장[9] 정의를 활용하여 쉽게 기술될 수 있다. 그러나, Split 및 Split+Join 구조물들의 완전한 시맨틱스를 기술하는 데에는 부가적인 작용 이론의 수정이 요구된다.

□Sequence: OWL-S의 sequence는 순서대로 행해질 프로세스들의 목록을 나타낸다. 이는 SC안에서 $Do(\delta_1;\delta_2,s,s')\equiv(\exists s'').Do(\delta_1,s,s'')\wedge Do(\delta_2,s'',s')$으로 표현될 수 있다.

□Split: OWL-S의 Split는 동시에 수행될 프로세스 컴포넌트들의 백(bag)으로서 표현된다. OWL-S에서는 대기(waiting)나 동기화에 대해서 언급이 없다. 그러나 이 부분은 동시성과 시간성을 위한 SC의 foundational 공리들[9][3]을 이용하여 기술될 수 있다.

예를 들어 만약 어떤 composite 프로세스가 waking, singing, chewingGum으로 나뉘어 진다고 가정하면, 이러한 모델은 아래 (그림 1)로 충분히 기술가능하다. 아래 그림은 3개의 중첩되는 프로세스들, walking(x,y,s), singing(s), 그리고 chewingGum(s)의 인스턴스들을 위한 가능한 시간 라인을 보인다. 즉, OWL-S에서

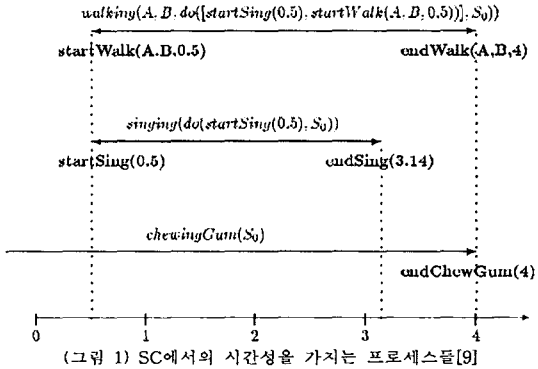
2) KRef, KWheter 및 Knows 표현과 관련된 사항은 [9]를 참조한다.

3) 동시성과 시간성을 위해서는 기본 작용 이론 D를 확장해야만 한다. 이에 대한 자세한 확장방법 및 필요한 용어들은 지면관계상 생략한다. 자세한 내용은 [9]를 참조한다.

명백하게 제공되지 못하는 동시성의 개념을 SC temporal 덤 t를 통해서 시뮬레이트하는 것이 가능함을 알 수 있다.

□Split+Join: 프로세스는 장벽(barrier) 동기화를 가지는 한 다발의 프로세스 컴포넌트들의 동시 실행으로 구성된다. SPLIT와 SPLIT+JOIN을 가지고 우리는 부분 동기화를 가지는 프로세스들을 정의할 수 있다.

현재 OWL-S에서는 대기(waiting) 및 동기화를 위한 어떠한 프리미티브들도 제공하지 않으므로 Split후 Join을 위해서는 대기 및 동기화에 대한 수단이 요구된다. 이를 위하여 새로운 relational fluent waiting과 syncJoin을 도입한다. 예로, Split에서 사용한 보기에서 waitingWalk를 참으로 만드는 작용은 endWalk이고 다시 syncJoin을 참으로 만드는 작용은 waitingWalk이다. 이와 같은 방법은 Split의 시맨틱스를 기술하기 위하여 Split절에서 SC의 기본 작용 이론을 확장하면서 도입된 walking과 같은 순간적인(instantaneous) 작용의 사용과 같은 접근법이다.



(그림 1) SC에서의 시간성을 가지는 프로세스들[9]

□Choice: CHOICE는 부가적인 속성들 'chosen'과 'chooseFrom'을 가지는 제어 구조물이다. 이러한 속성들은 프로세스를 위해서 그리고 실행 제어를 위해서 또는 양쪽 모두를 위해서 사용될 수 있다. 뿐만 아니라 "m개로부터 적어도 n을 선택하라", "m개로부터 정확히 n개를 선택하라", "m개로부터 최대 n개를 선택하라" 등등과 같이 새로운 subclasses들을 건설하기 위해서 사용될 수 있다.

상기의 시맨틱스를 포착하기 위한 방법은 선택에 관한 공리 $Do(\delta_1|\delta_2,s,s') \equiv Do(\delta_1,s,s') \vee Do(\delta_2,s,s')$를 기본 작용 a 대신 복합 작용 c를 포함하도록 확장하는 것이다. 그리고 cardinality에 대한 명시는 작용 precondition 공리들의 집합에 추가적으로 명시한다.

□Unordered: Bag으로서 명시된 프로세스 컴포넌트들이 어떤 명시되지 않은 순서로 또는 동시에 실행되는 것을 허용하는 구조물이다. 이 구조물 안의 모든 컴포넌트들은 실행되어야만 하고, Split+Join에서와 같이, 모든 컴포넌트들이 완료되어야만 한다. Unordered 구조물 자체는 실행의 순서에 대해 그 어떠한 제약사항들도 부과하지는 않지만, 몇몇 경우들에 있어서 반드시 지켜져야만 하는 하부 컴포넌트들과 연관된 제약사항들이 존재할 수 있다.

OWL-S의 Unordered 구조물에 대한 SC 시맨틱스는 비결정적인 공리의 선택을 나타내는 공리 $Do(\delta_1|\delta_2,s,s') \equiv Do(\delta_1,s,s') \vee Do(\delta_2,s,s')$과 순서를 나타내는 공리 $Do(\delta_1;\delta_2,s,s') \equiv (\exists s'').Do(\delta_1,s,s'') \wedge Do(\delta_2,s'',s')$의 조합으로 기술될 수 있다. 즉, $Unordered(\delta_1,\delta_2) \equiv Do(\delta_1,s,s') \wedge Do(\delta_2,s,s'') \vee Do(\delta_2,s,s') \wedge Do(\delta_1,s,s'')$.

□Condition: OWL-S의 현재 스펙에서 명시되어 있지 않다. 추후 논리 표현들의 클래스들로 정의될 것으로 명시되어 있다. 조건과 관련된 클래스는 OWL-S라기 보다는 규칙표현이 자연스럽고 그래서 조건 클래스를 명시하는 것은 RuleML과의 통합모델에 의존한다. 즉 LP의 표현력을 이용한다.

□If-Then-Else: If-Then-Else 시맨틱스의 SC 기술은 if ϕ then δ_1 else δ_2 endIf $\equiv [\phi?;\delta_1] | [\neg\phi?;\delta_2]$으로 기술될 수 있다.

□Iterate: OWL-S의 Iterate 구조물은 이것의 'nextProcessComponent' 속성이 현재 프로세스 컴포넌트와 같은 값을 가지는 제어 구조물이다. Repeat는 Iterate 클래스의 동의어로서 정의된다. Repeat/Iterate 프로세스는 얼마나 많은 반복들이 만들어지는지 또는 언제 시작하고, 종료되는지, 또는 재개되는지에 대한 어떠한 가정도 하지 않는다. 시작, 종료, 또는 유지 조건은 'whileCondition' 또는 'untilCondition'과 함께 명시될 수 있다.

상기 Iterate 구조물의 시맨틱스는 Do의 매크로-확장정의를 이용하여 아래와 같이 쉽게 표현될 수 있다.

$Do(\delta^*,s,s') \equiv (\forall P).\{(\forall s1)P(s1,s1) \wedge (\forall s1,s2,s3)[Do(\delta,s1,s2) \wedge P(s2,s3) \supset P(s1,s3)]\} \supset P(s,s')$

□Repeat-While/Repeat-Until: Repeat-Until 클래스는 이것이 'ifCondition'이 'untilCondition'과 똑같은 If-Then-Else를 구체화한다는 의미에서 Repeat-While과 유사하고, else 속성이 되풀이되는 프로세스라는 의미에서 Repeat-Whiler 클래스와 다르다. 그래서 프로세스는 'untilCondition'이 참이 될 때까지 되풀이된다.

따라서 Repeat-While 및 Repeat-Until은 SC 시맨틱스 while ϕ do δ endWhile $\equiv [\phi?;\delta^*;\neg\phi?]$으로 표현할 수 있다.

이상으로 OWL-S의 프로세스 지향적인 동적 행동을 모델링하는 프로세스 모델 온톨로지의 IOPE와 atomic 프로세스 및 composite 프로세스의 정형 시맨틱스를 SC로 기술하였다.

3.2.2 OWL-S 프로세스 모델과 규칙과의 매핑

본 섹션에서는 OWL-S의 프로세스 모델의 SC 표현과 규칙표현 사이의 관계를 분석함으로써 상호운용할 수 있는 매핑을 기술한다. 이를 위하여 [10]의 연구결과를 활용한다. [10]의 기본적인 아이디어는 로직 프로그램을 구성하는 클로즈들은 규칙들로 취급될 수 있고, 그런 규칙의 적용은 SC에서 작용을 수행하는 것과 같다는 것이다. 클로즈를 실행하는 것은 그 클로즈의 헤드를 그 클로즈의 바디가 현재 situation에서 참이면 새로운 situation안에서 참으로 만든다. 그리고 나면 프로그램 클로즈들은 그것들이 작용들의 axiomatic 이론들 안에서 이해되므로 SC effect 공리들로 식별된다. 그래서, $F(\vec{x}):-G^{4)}$ 형태의 클로즈는 SC안에서 작용의 effects들의 명세와 같다. 이러한 생각을 염두에 두고서, 프로그램 P에 대한 작용 이론 D가 클락의 정리[10]를 이끌어 낸다는 것을 보임으로써 SC로 기술된 작용 이론 D와 로직 프로그램 P와의 매핑을 정의한다. 뿐만 아니라 이 결과는 SC로 기술된 작용 이론들의 구현을 위한 이론적인 근간을 제공한다.

가정하기를 SC 언어 안에서 어떤 클로즈를 작용 $A(\vec{x})$로 명명한다고 하자. 그러면 A의 effect를 기술하는 effect 공리는 $Poss(A(\vec{x}),s) \supset (G[s] \supset F(\vec{x}do(A(\vec{x}),s))$이다. SC 안에서 작용들은 항상 가능하다고 가정될 수 있으므로[9,10], $G[s] \supset F(\vec{x}do(A(\vec{x}),s))$과 동치이다. 이는 다시 $(\exists \vec{y})G[s] \wedge a=A(\vec{x}) \supset F(\vec{x}do(a,s))$로 재작성될 수 있다. 예로 gf(x,y)가 클로즈 grandfather(x,y):-parent(x,z) & parent(z,y) & not female(x)를 명명(naming)하는 작용이라고 가정하면, effect 공리으로서 $(\exists z)[parent(x,z,s) \wedge parent(z,y,s)] \wedge \neg(\exists s')female(x,s') \wedge a=gf(x,y) \supset grandfather(x,y,do(a,s))$를 얻는다.

Fluent F를 위한 프레임 문제[9]를 해결하면, F에 대한 successor state 공리 $F(\vec{x}do(a,s)) \equiv \{(\exists \vec{y_1}) G_1[s] \wedge a=A_1(\vec{x}) \vee ... \vee (\exists \vec{y_n}) G_n[s] \wedge a=A_n(\vec{x}) \vee F(\vec{x}s)\}$를 얻는다.

이제 로직 프로그램들의 의미(meaning)를 SC안에서 정의할 수 있다. 각 클로즈에 대해서 그 클로즈를 명명하는 고유 작용 기호가 존재하고, 그 클로즈의 헤드안의 predicate의 그것과 똑같은 개수의 인수들을 가진다고 가정한다.

■ Lin과 Reiter의 정리[10]. P를 프로그램, D를 P의 작용 이론, 그리고 F를 fluent라고 하자. D안의 F를 위한 successor state 공리는 $F(\vec{x}do(a,s)) \equiv \{(\exists \vec{y_1}) G_1[s] \wedge a=A_1(\vec{x}) \vee ... \vee (\exists \vec{y_n}) G_n[s] \wedge a=A_n(\vec{x}) \vee F(\vec{x}s)\}$이고, G_i는 $l_{i1} \& ... \& l_{ik}(1 \le i \le n)$라고 하자. 그러면 D는 F를 위한 다음의 클락 정리를 이끌어낸다.

$(\exists s)F(\vec{x}s) \equiv \{(\exists \vec{y_1})((\exists s)\, l_{11}[s] \wedge ... \wedge (\exists s)\, l_{1k_1}[s]\} \vee ... \vee (\exists \vec{y_n})\{(\exists s)\, l_{n1}[s] \wedge ... \wedge (\exists s)\, l_{nk_n}[s]\}\}$.

예로써 다음 클로즈들을 가지는 definite 프로그램 P_1을 고려하자.

ancestor(x,y):-parent(x,y)
ancestor(x,y):-ancestor(x,z)&ancestor(z,y)
parent(x,y):-x=John&y=Joe

4) 여기에서 F는 fluent 기호, $\vec{x}$는 n(n≥0)의 서로 다른 변수들의 튜플이고 G는 $l_1 \& ... \& l_n$형태의 표현되는 goal이다. 여기에서 l_1 ... l_n(n≥0)들은 리터럴들이다.

parent(x,y):-x=Joe&y=Bill
parent(x,y):-x=Joe&y=Susan

A_1(x,y), A_2(x,y), B_1(x,y), B_2(x,y), B_3(x,y)가 각각 이 5개의 클로즈들을 명명하는 작용들이라고 하자. fluent ancestor에 대해서, 우리는 다음 2개의 effect 공리들을 가진다:

parent(x,y,s)∧a= A_1(x,y)⊃ancestor(x,y,do(a,s)),

(∃z)[ancestor(x,z,s)∧ancestor(z,y,s)]∧a= A_2(x,y)⊃ancestor(x,y,do(a,s)).

그래서 우리는 ancestor를 위한 다음의 successor state 공리을 가진다.

ancestor(x,y,do(a,s))≡(a= A_1(x,y)∧parent(x,y,s)∨
a= A_2(x,y)∧(∃z).ancestor(x,z,s)∧ancestor(z,y,s)∨
ancestor(x,y,s))

유사하게, 우리는 parent를 위한 다음의 successor state 공리을 가진다.

parent(x,y,do(a,s))≡(x=John∧y=Joe∧a= B_1(x,y)∨
x=Joe∧y=Bill∧a= B_2(x,y)∨
x=Joe∧y=Susan∧a= B_3(x,y)∨
parent(x,y,s))

D_1을 P_1을 위한 작용 이론이라고 하자. Lin과 Reiter의 정리에 의해서, 우리는 다음을 알 수 있다.

D_1⊨(∀x,y){(∃s)parent(x,y,s)≡[(x=John∧y=Joe)∨
(x=Joe∧y=Bill)∨(x=Joe∧y=Susan)]}

그리고, ancestor에 대해서는 다음을 생성한다.

D_1⊨(∀x,y){(∃s)ancestor(x,y,s)≡
[(∃s)parent(x,y,s)∨
(∃z).(∃s)ancestor(x,z,s)∧
(∃s)ancestor(z,y,s)]}

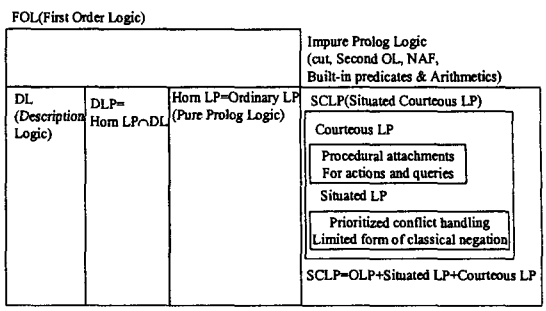
이상으로 로직 프로그램 안의 클로즈의 유도(derivation)가 SC 안의 작용의 실행과 같다는 아이디어에서 출발한 매핑정의는 successor state 공리에 의하여 상호 변환될 수 있고, 이 시맨틱스는 클락의 정리에 의해 보장된다는 것을 살펴보았다.

3.3 OWL-S와 규칙표현과의 상호유용성에 관한 분석

우리는 이상으로 3.1절에서 SC로 기술된 OWL-S의 시맨틱스와 규칙표현과의 매핑 가능성을 기술하였다. 그러나 3.2절에서 기술된 Lin과 Reiter의 정리를 살펴보면 클락의 정리를 이용하고 있는데 이 클락의 정리는 SC의 프롤로그 구현을 위해 매우 중요한 정리이지만 한편으로는 SC의 표현력을 매우 제한시킨다. 그 이유는 SC로 기술된 기본 작용이론이 클락의 정리를 결과로 초래하기 위해서는 소위 definitional 이론[9]이어야 하기 때문이다. 이로 인해서 발생하는 표현상의 한계점들은 다음과 같다. 먼저 클락의 정리가 적용되기 위해서는 이론들이 적당한 고유 명칭들의 공리들을 반드시 포함해야만 하는데 이것은 덤들의 표기법에 있어 매우 심각한 제한이다. 왜냐하면 서로 다른 덤들은 서로 다른 것들을 나타내기 때문에 father(Sue)=Bill과 같은 것을 말할 수 없기 때문이다. 두 번째로 모든 predicate들이 definitional 이론에서 정의하는 definition이어야만 한다. 그래서 예로 (∃x)loves(Sue,x)와 같은 존재적 사실들 또는 loves(John, Mary)∨loves(John, Sue)와 같은 disjunction들과 같은 응용 도메인에 대한 부정확한 정보를 표현할 수 없다[9].

이러한 사실들은 definitional 이론들이 부정확한 정보를 표현하기 위해 매우 제한된 메커니즘들을 제공한다는 것을 말하고 있고 그런 공리화(axiomatization)는 단혀진 세상(closed world) 표현들로 불린다. 그러나 존재적 한정, disjunction 및 negation을 수반하는 FOL의 모든 표현력 능력을 포기하는 대신에 우리는 이러한 이론들에 대해 프롤로그로의 클락 번역을 굉장히 단순화시킬 수 있고 프로그램들의 결과계산을 효율적으로 할 수 있는 장점을 가질 수 있다[9].

이상의 분석결과로서 우리는 서비스 온톨로지 OWL-S와 로직프로그램의 표현 RuleML이 정확하게 다른 하나로 매핑 또는 번역되는 것은 불가능하다고 유추할 수 있다. 그러나 이것은 모든 경우에 대해서 그렇다는 것은 아니라는 것에 주의할 필요가 있다. 실제로 어떤 서비스 온톨로지 정보가 definitional 이론에서 요구되는 형태로 표현될 경우는 정확하게 로직 프로그램으로 매핑될 수 있기 때문이다. 다음 (그림 2)는 FOL, DL 및 LP와의 표현력 관계를 말하고 있다. 이 그림에서 알 수 있듯이 만약 주어진 서비스 온톨로지의 표현력을 만약 우리가 DLP의 범주로 한정시킨다면 우리는 그 서비스 온톨로지 정보의 추론에 있어 LP엔진을 사용할 수 있다.



(그림 2) FOL, DL 및 LP들의 상관도[11]

4. 결론 및 향후과제

본 논문에서 우리는 지능형 e-비즈니스의 효과적인 수행을 지원하기 위한 모델링 언어로서 서비스 온톨로지 마크업 OWL-S와 규칙표현 RuleML 사이의 관계를 분석함으로써 상호매핑을 기술하였다. OWL-S의 제어 구조물들에 대한 의도된 모든 시맨틱스가 항상 규칙표현으로 정확하게 매핑될 수는 없지만 우리의 이러한 접근법은 시맨틱 웹 영역의 언어 레이어링 측면(버너스리에 의해 주창된 시맨틱 웹 아키텍처의 구현)과 OWL-S의 정형 시맨틱스의 제공 등에 있어 중요한 의미를 가진다. OWL-S와 LP의 부분적 매핑에 대한 우리의 이론적인 근간 및 분석 결과는 서비스 온톨로지와 규칙표현이 각각의 고유한 표현력을 가지고 (완벽한 상호유용 및 매핑이 불가능함) 있기 때문에 두 마크업이 지능형 e-비즈니스에서 상호보완적으로 사용되는 것이 바람직하다는 사실을 말한다.

향후 우리는 실제 응용에서 LP엔진의 효율성을 활용하기 위하여 OLP의 범위를 벗어난 표현력을 가지는 SCLP RuleML과 같은 버전을 고려하고 있다. 특히 우리의 선 연구 [11]은 SCLP RuleML이 제공하고 있는 NAF와 절차적인 프리미티브들을 첨가하는 기능을 포함한 형태에서 이를 확장한 CLP RuleML을 제안하였는데 추후 이에 대한 이론적인 분석과 구현을 통하여 타당성을 검증할 것이다.

참고문헌

[1] T.C. Son S. McIlraith and H. Zeng, Semantic Web Services. IEEE Intelligent Systems, Special Issue on the Semantic Web, 16(2):46-53, 2001.

[2] 양진혁, 민재홍, 이유수, 정인정, 지능형 e-비즈니스를 위한 플랫폼에 관한 연구: DAML-S의 규칙기반 프레임워크로의 확장 및 통합방안, KIPS'04 춘계학술발표 논문집 11권 1호, pp. 373-376, 2004.5.

[3] http://www.daml.org/services/owl-s/1.1B/

[4] http://www.w3.org/TR/owl-ref/

[5] http://www.ruleml.org/0.87/

[6] Srinin Narayanan and Sheila A. McIlraith, Semantic web services: Simulation, verification and automated composition of web services, In Proceedings of the Eleventh International World Wide Web Conference, pages 77-88, ACM Press, 2003.

[7] Anupriya Ankolekar, Frank Huch, and Katia Sycara, Concurrent Execution Semantics for DAML-S with Subtypes, In The First International Semantic Web Conference, 2002.

[8] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker, Description Logic Programs: Combining Logic Programs with Description Logic, In Proc. of 12th Intl. Conf. on the World Wide Web (WWW-2003), Budapest, Hungary, May 20-23, 2003.

[9] Raymond Reiter, Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, MIT Press, 2001.

[10] F. Lin and R. Reiter, Rule as Actions: A Situation Calculus Semantics for Logic Programs, Journal of Logic Programming, Special issue on Reasoning about Action and Change, 31:299-330, 1997.

[11] 양진혁, 정인정, 지능형 e-비즈니스를 위한 플랫폼에 관한 연구: OWL-S와 CLP와의 통합에 관한 연구, KIPS'04 춘계학술발표 논문집 11권1호, pp. 377-380, 2004.5.