OWL-S와 문맥 정보를 가진 에이전트를 사용한 웹 서비스 발견 기법

이창호°, 김진한, 박동민, 이병정
서울시립대학교 컴퓨터과학부
{leechangho°, kimjinhan, parkdongmin, bjlee}@venus.uos.ac.kr

A Web Service Discovery Using OWL-S and Agent with Context

Changho Lee°, Jinhan Kim, Dongmin Park, Byungjeong Lee
School of Computer Science, University of Seoul

요 약

로봇이 적응성을 위해 웹 서비스를 통해 문제를 해결하려고 할 때, 웹 서비스는 서비스의 의미로 서비스를 찾을 수 있는 방법을 제공하지 않아서 원하는 기능의 서비스를 찾기가 어렵다. 하지만 시멘틱 웹에서는 온톨로지를 이용하여 서비스의 의미를 표현하기 때문에 위의 문제를 해결할 수 있다. OWL-S를 이용하면 웹 서비스의 표현을 용이하게 해준다. 하지만 문맥(Context) 정보를 고려한 서비스의 검색이나 검색을 통해 가져온 서비스를 검증하는 수단을 제공하지 않고 있다. 본 논문에서는 OWL-S과 에이전트를 기반으로 하여 적합한 서비스를 찾는 방법을 제안한다.

1. 서 론

적응형 소프트웨어란 " 외부의 환경 변화에 대하여 자신의 행동을 스스로 고칠 수 있는 소프트웨어" 를 말한다 [1]. 요즘 활발히 연구가 이루어지고 있는 로봇 분야에서도 적응성은 필수의 요소로 여겨지고 있다. 하지만 로봇이나 임베디드 분야에서는 한정된 내부 저장소로 인해 적은 일정한 양의 데이터나 컴포넌트를 저장할 수 밖에 없다. 그래서 외부 저장소를 이용하여 문제를 해결하고자 하는 연구들[1, 2]이 이루어지고 있다. [2]와 같은 연구에서는 온톨로지를 사용하여 좀 더 적합한 자원을 찾으려는 시도가 있었지만 웹 서비스를 찾는데 있어서 OWL-S와 에이전트를 이용하여 적합한 서비스를 찾아 적응성을 해결하려는 시도는 없었다.

한편 웹 서비스는 표준 기술들을 사용하여 웹을 통해 다른 이종의 프로그램들과의 통신을 가능하게 해준다. 하지만 WSDL (Web Service Description Language)를 이용해서는 서비스의 인터페이스만을 알 수 있을 뿐 서비스가 가지는 기능과 의미를 표현할 수가 없다. 그래서 이런 웹 서비스를 기반으로 자원들의 의미적인 표현을 가능하게 해주는 시멘틱 웹의 중요성은 점점 커지고 있다. 시멘틱 웹에서는 웹 상의 자원들을 온톨로지를 사용하여 그 자원을 기술하고 있다. 따라서 시멘틱 웹을 사용하면 사용자가 원하는 기능을 가진, 단순히 서비스 이름이 아닌 그 서비스가 가진 의미로써 서비스의 등록, 검색, 바인딩이 이루어지게 된다 [3]. 하지만 온톨로지만을 가지고서는 전후의 문맥적인 상황이나 조합된 서비스들의 검증을 위한 수단을 제공하지 못하고 있다.

본 논문은 로봇 내부에서 문제를 해결하지 못하는 상황에서 웹 서비스를 통해 문제를 해결하고자 하는 방법을 설명한다. 본 논문의 구성은 다음과 같다. 2장에서는 에이전트와 OWL-S에 대한 내용과 문제점을 설명한다. 3장에서는 그러한 기술을 바탕으로 원하는 서비스를 효율적으로 검색하기 위한 방법과 과정에 대해 설명한다. 그리고 검색된 서비스들의 조합을 검증하는 방법도 소개한다. 그리고 4장에서는 사용사례를 보이고 5장의 결론과 향후 연구로 마무리를 짓는다.

2. 관련 연구

2.1. Agent

에이전트는 사용자들을 위해서 독립적으로 작업을 수행하기 위해 행동하는 소프트웨어의 부분을 말한다 [4]. 많

본 연구는 한국과학재단 특정기초연구(R01-2006-000-11150-0) 지원으로 수행되었음.

은 에이전트들은 직접 명령을 내리는 것 보다는 높은 수준의 목적들을 명세 하는 기법을 기반으로 하고 있다. 일반적으로 에이전트의 특징은 다른 에이전트와 구별할 수 있는 고유한 식별성 (Identiy)을 가지고 주변 환경을 인지하며 그것에 관하여 추론을 하며 추론을 바탕으로 자신의 행동을 수정할 수 있다 [3]. 그리고 에이전트는 다른 에이전트들과 통신할 수 있는 능력을 가지고 있다. 그림 1은 이런 특징들을 가지면서 간단히 환경의 변화를 감지해서 그에 맞는 반응을 보여주는 에이전트의 구조를 보여준다.

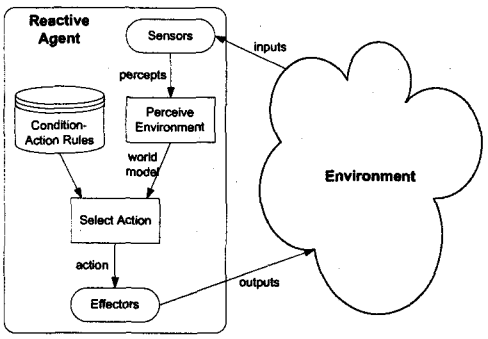

그림1. 환경의 변화에 반응을 나타내는 에이전트

에이전트를 웹 서비스에 응용하는 것은 많은 장점을 가진다. 서비스들이 제공되는 데에 있어서 많은 제약 사항들을 잡아낼 수 있다. 그럼으로써 서비스 조합을 위한 요구사항들을 좀 더 풍부하게 표현해준다. 그리고 에이전트는 신뢰할 수 있는 서비스들을 발견하는 것을 도와준다. 그리고 서비스 제공자들과의 사이에서 협상 (Negotiation)을 도와준다. 협상 후 서비스 제공자들이 특별한 서비스 조합에 관한 협약을 따르는지를 판단할 수 있게 해준다. 이러한 특징들은 자연스럽게 웹 서비스에 응용이 될 수 있지만 아직까진 많은 연구들이 이루어지지 않고 있다.

2.2. OWL-S

OWL-S는 서비스를 위한 웹 온톨로지 언어이다. 서비스를 기술하기 위해 OWL-S에는 세 가지 구성 요소들이 있다. 서비스프로필(service profile)은 사용자들로부터 요구하는 것과 사용자들에게 제공할 수 있는 것을 기술한다. 서비스모델(service model)은 어떻게 서비스가 작동하는지를 명세 한다. 그리고 마지막으로 서비스그라운딩(service grounding)은 어떻게 서비스를 사용하는지에 대한 정보를 가지고 있다. 이 중에 서비스모델의 서브클래스 중 프로세스모델(process model)은 IOPE(Input, Output, Precondition, Effect)와 서브 프로세스로 서비스를 기술한다. 프로세스 모델에서는 복합의 프로세스들과 그들의 연관 관계, 그리고 상호작용을 기술할 수 있다. 그리고 OWL-S의 프로세스에는 세가지 타입이 있다. 그 세가지는 서브 프로세스를 가지지 않는 원자 프로세스와 마찬가지로 서브 프로세스를 가지지 않지만 호출될 수 있는 단일 프로세스, 그리고 서브 프로세스들을 가지는 복합 프로세스이다. 그림 2는 위에서 언급된 내용을 나타낸다 [3, 5].

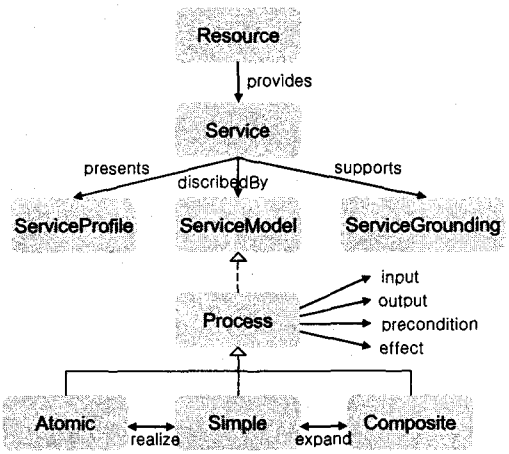

그림2. 서비스의 구성을 보여주는 모델의 부분

OWL-S의 목적은 웹 서비스에 관해서 추론과 서비스 조합을 위한 플랜, 그리고 에이전트들에 의한 서비스들의 사용을 자동화를 가능하게 해주는 것이다. 그리고 OWL-S는 웹 서비스들을 기술하기 위한 OWL 온톨로지를 제공한다. OWL-S를 사용하여 서비스 제공자들은 잠재적인 사용자들에게 자신들의 서비스의 기능을 알릴 수 있다. 그러면 소비자들은 중개 에이전트를 통해서 요구하는 기능을 가진 서비스를 찾을 수 있다. 서비스 검색에 있어서도 OWL-S는 단순히 단어의 매치뿐만 아니라 의미를 가지고 서비스를 찾는 것을 가능하게 해준다. 또한 비기능적인 속성들을 표현할 수 가 있기 때문에 의미와 이런 속성 정보들을 이용해 기존의 UDDI (Universal Description, Discovery, and Integration)를 이용한 웹 서비스를 검색하는 것 보다 더 적합한 검색 결과를 보여준다. 하지만 등록된 웹 서비스의 빠르고 효과적인 검색을 위해 개선을 하기 위한 시도가 있었지만 [6], 문맥 정보 (Context)를 추가한다거나 검색을 위한 다른 방법을 제공하지 못하고 있다.

3. 에이전트를 이용한 웹 서비스 발견 아키텍처

시멘틱 웹에서 서비스를 발견해서 이용하기 위한 과정은 크게 세 단계로 나눌 수 있다. 첫 번째 단계는 사용자의 목적을 달성하기 위해서 후보 서비스들을 발견한다. 그런 다음 두 번째 단계에서는 후보 서비스들 중 협상을 통해 적절한 서비스를 선택하게 된다. 그러면 마지막 단계로 서비스의 바인딩이 일어나게 된다 [7]. 그림 3은 로봇이 사용자 에이전트를 통해 서비스의 요청이 이루어지면 그림

에 나와있는 에이전트들을 통해 위의 과정들을 수행하게 된다. 본 논문에서는 이러한 과정들이 되도록 빠르고 효율적으로 일어나도록 레지스트리 아래에 하위 레지스트리를 두어 검색에 있어서 최소한의 검색이 일어나도록 하였다. 그리고 검색으로 가져온 결과가 로봇에 넘겨지기 전에 사용자 에이전트에서 서비스의 조합인 프로세스의 정보를 변환시켜 검증하는 과정을 추가하였다. 본 논문에서는 온톨로지에 대해서 많은 사람들이 동의하는 표준 온톨로지가 존재하고, 그에 따라 도메인이 잘 분류가 되어 있고, OWL-S로 표현되어 있다는 가정을 하고 있다.

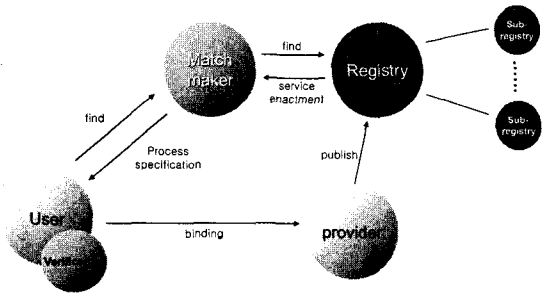

그림3. 에이전트를 이용한 웹 서비스 발견 아키텍처

3.1 레지스트리의 분류

일반적으로 시멘틱 웹에서의 서비스의 검색은 브로커 역할을 하는 그림 3에서 Match maker를 통해 위에서 설명한 세 단계의 과정이 일어나게 된다. 보통은 레지스트리에 등록된 수많은 서비스들 중에서 사용자의 목적에 맞는 후보 서비스들을 발견해서 그 서비스들과의 협상을 통해 적합한 서비스를 찾게 된다. 하나의 서비스가 목적을 만족하지 못하게 되면 여러 서비스들의 조합을 플랜을 통해 찾게 되는데 이러한 과정은 직관적으로도 많은 계산을 필요로 한다. 따라서 로봇에 빠른 응답을 해주기 위해서는 시스템적으로 서비스 발견에 대한 오버헤드를 줄여줄 필요가 있다. 본 논문에서는 이러한 문제를 해결하기 위해 하나의 레지스트리와 여러 개의 하위 레지스트리로 기존의 레지스트리를 분류한다.

레지스트리에는 온톨로지에서 도메인에 대한 정보를 가지고 있고 하위 레지스트리에서는 가장 작은 도메인 단위에 속하는 서비스들에 대한 정보를 가지고 있다. 즉, 온톨로지를 트리 구조로 생각을 하고 리프를 웹 서비스로 생각을 하면 루트에서부터 리프 전까지의 정보와 최하위 도메인에 속한 웹 서비스들의 정보를 가진 하위 레지스트리의 위치에 관한 정보를 가지게 된다. 따라서 서비스를 요청하는 정보에 사용자가 (이 논문에서는 로봇) 원하는 도메인에 대한 정보를 추가하는 것만으로 검색의 범위를 좁혀주기 때문에 검색되는 후보 서비스들의 수도 상당히 작아지게 된다. 이에 따라 검색에 대한 시간을 대폭 줄일 수 있다.

3.2 사용자 에이전트 기술 (Description)

이 장에서는 그림 3에 나오는 에이전트들 중, 사용자 에이전트에 관해서 설명을 한다. 기본적으로 에이전트의 특징을 살펴보면 주변의 환경을 인지하고 추론을 할 수 있다는 능력은 에이전트가 적응성을 가지고 있다는 것을 말해준다. 이런 기본적인 에이전트를 간단한 의사 코드로 나타내면 표 1과 같다.

표1. 일반적인 에이전트의 간단한 의사 코드

```
Environment e;
RuleSet r;
While (true) {
    state = senseEnvironment(e);
    a = chooseAction(state, r);
    e.applyAction(a)
}
```

표 1의 내용을 기본으로 사용자 에이전트의 기술을 확장한다. 사용자 에이전트의 역할은 로봇으로부터 문제 해결을 위한 목적에 대한 정보와 도메인 정보, 문맥 정보를 받아서 Match maker에 이 정보들을 넘겨주고 결과로 Match maker로부터 프로세스 명세(Process Specification)을 받게 된다. 프로세스 명세에는 서비스 조합이 어떤 식으로 구성되어 있는지에 대한 정보를 담고 있다. 표 2는 프로세스 명세에 대한 예를 보여준다. 여기에서는 세 개의 서비스의 조합을 OWL-S로 표현한다. 이해를 쉽게 하기 위해 필요 없는 부분을 생략한다. 이 예에서 프로세스는 세 개의 서비스로 구성이 되어있다. 이 프로세스의 시작은 서비스 A로부터 시작한다. 그리고 서비스 A가 끝나면 서비스 B와 C가 병렬적으로 실행되는 것을 보여준다.

표2. 프로세스 명세의 예

```
<daml:Class rdf:ID=" Example" >
  <daml:subClassOf rdf:resource=" Process.CompositeProcess" />
  <daml:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource=" Process#composedOf" />
      <daml:toClass>
        <daml:Class>
          <daml:intersectionOf rdf:parseType=" daml:collection" >
            <daml:Class rdf:about=" process:Sequence" >
              <daml:Restriction>
                <daml:onProperty rdf:resource ≈
" Process#components" />
                  <daml:toClass>
                    <daml:Class>
                      <process:listOfInstancesOf rdf:parseType ≈
" daml:collection" >
                        <daml:Class rdf:about=" #serviceA" />
                        <daml:Class rdf:about=" process:Split" >
                          <daml:Restriction>
```

```
                    <daml:onProperty        rdf:resource        =
" Process#components" />
                    <daml:toClass>
                      <daml:Class>
                        <process:listOfInstancesOf  rdf:parseType
=" daml:collection" >
                          <daml:Class rdf:about=" #serviceB" />
                          <daml:Class rdf:about=" #serviceC" />
                        </process:listOfInstancesOf>
                      </daml:Class>
...
```

그리고 이 프로세스 명세를 바로 로봇에 건네주기 전에, 먼저 검증을 한 후 이상이 없다고 판단을 내린 경우 로봇에 결과를 건네주게 된다. 따라서 사용자 에이전트는 추가로 도메인 정보와 문맥 정보를 위한 저장 공간이 필요하다. 그리고 OWL-S 문서에 대해서 검증하기 위한 메커니즘이 필요하게 된다. 사용자 에이전트를 위한 의사 코드는 표 3과 같다.

표3. 사용자 에이전트의 의사 코드

```
Environment e;
RuleSet r;
Domain d;
Context c;
Process p;
While (true) {
    state = acceptRequest(e);
    p = sendRequest(state, r, d, c);
    p.translationToPetrinet();
    p.verify();
    sendResult(p);
}
```

3.3 프로세스 검증

로봇에서 프로세스 명세 정보에 의해서 실질적으로 서비스의 호출이 일어나기 전에 일련의 과정을 통해 구성된 서비스들의 특징들(예를 들어, security, dependability)의 정확성을 검증할 필요가 있다. 하지만 OWL-S는 서비스와 그 속성들을 표현하고 프로세스를 명세 하는 방법은 제공하지만 위의 특징들을 검증하는 방법은 제공하지 못한다. 그래서 이 점을 해결하기 위해 OWL-S에서 기술하는 내용을 대수적으로 표현 가능한 언어나 유한 상태 머신 같은 모델로 변환을 해서 검증하는 방법들이 제시되었다 [3, 5, 8]. 여기에서는 프로세스 명세의 정보를 Petri-net으로 변환하여 정확성을 검증한다. Petri-net은 잘 설계된 프로세스를 모델링 하는 방법을 제시해주며 도달성 그래프 (Reachability graph)로 표현이 가능하다. Petri-net을 통해 서비스를 모델링하여 데드락 (deadlock), 무한 루프 (infinite loop), 그리고 중복성 (redundancy)을 포함하는 서비스들의 행동적인 속성들을 관찰하거나 오류를 검출할 수 있다. 이 방법에서는 검증을 위해 정교한 대수적인 방법을 사용하기 때문에 서비스 조합이 일정한 단계 안에서 종료가 될 수 있는지를 증명할 수 있다.

프로세스 명세를 검증하기 위해서는 두 가지 과정을 거쳐야 한다. 먼저 이 문서에 대해서 문법적으로 오류가 없는지를 OWL-S를 위한 파서를 통해 검사를 한다. 그런 다음 이 명세를 Petri-net으로의 변환과정이 일어난다. 그리고 Petri-net과 여기서부터 유도된 도달성 그래프의 비교를 통해 검증을 하게 된다. 웹 서비스를 위한 Petri-net으로의 변환은 [8]을 참조한다. 이 연구에서는 WSPN (Web Service Petri-net)을 다음과 같이 정의한다.

표4. WSPN의 정의

$WSPN = (P, T, F, M_0, P_{initial})$

$P = (P_{initial}, P_1, P_2, \cdots, P_m)$는 장소들의 유한한 집합을 말한다. 각각의 장소는 서비스에서의 하나의 프로세스를 나타낸다.

$T = (T_1, T_2, \cdots, T_n)$는 트랜지션들의 유한한 집합을 말한다. 각각의 트랜지션은 서비스에서 제어 구문을 나타낸다.

$F = (P \times T) \cup (T \times P)$는 호들의 집합이다. 각각의 호는 서비스에서 프로세스들 사이의 데이터 흐름을 나타낸다.

$M_0 = (1, 0, \cdots, 0)$는 처음을 나타내는 마킹이다. 토큰은 요청을 나타낸다. 간단하게 토큰은 하나만 사용한다.

$P_{initial} \in P$는 서비스의 시작 지점을 나타낸다.

4. 사용 사례

본 논문에서 제시하는 아키텍처에서의 서비스의 발견은 다음과 같은 순서를 따른다. 처음 로봇과 같은 사용자가 문맥 정보를 가지고 서비스를 사용자 에이전트에 요청을 한다. 그러면 사용자 에이전트는 match maker를 통해 레지스트리를 뒤져서 하나의 서비스나 여러 서비스로 이루어진 프로세스의 명세를 받게 된다. 이 명세를 로봇으로 보내기 전에 사용자 에이전트에서는 프로세스에 대한 검증을 한다. 그래서 아무런 이상이 발견되지 않으면 프로세스 명세를 로봇에게 보내고 로봇에서는 실질적으로 서비스를 이용하게 된다.

사용자 에이전트에 도달할 프로세스 명세는 그림 3과 같은 형태이다. 그림 3에서는 간단한 프로세스의 예를 들었기 때문에 이 장에서는 좀 더 복잡한 프로세스 명세를 받았다는 가정한다. 그리고 추가적으로 그 명세를 바탕으로 Petri-net으로의 변환이 이루어진다는 가정을 한다. 그림 4는 프로세스 명세에서 Petri-net으로의 변환을 보여준다. 그리고 그림 5는 Petri-net에서 유도된 도달성 그래프

를 보여준다.

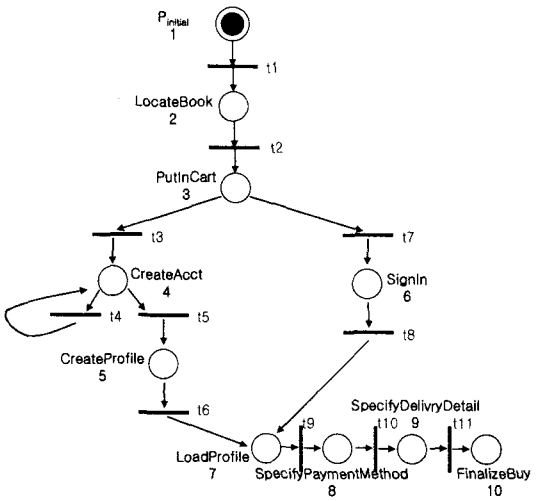

그림 4. WSPN으로의 변환

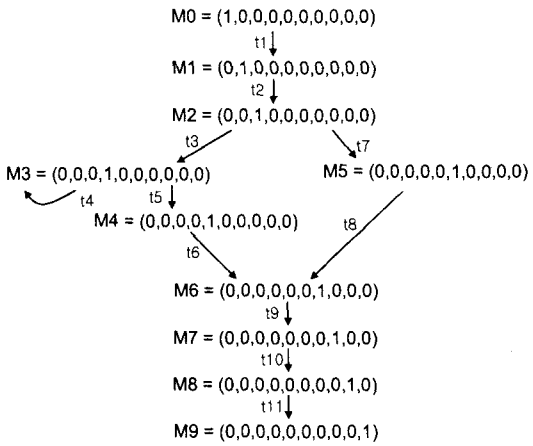

그림 5. 유도된 도달성 그래프

그림 4와 그림 5의 두 그래프를 가지고 프로세스에 대한 검증이 일어나게 된다. 그림 5를 보면 데드락이 없다는 것을 알 수 있다. 왜냐하면 각각의 마킹으로부터 나가는 호가 M9를 제외하고는 하나씩 있기 때문이다. 그리고 그림 4의 Petri-net에서의 전이(Transition)과 그림 5의 호가 일치하기 때문에 중복적이지 않다. 그렇지만 그림 6에서 M3에서 M3로의 루프가 존재한다. 따라서 검증 과정은 실패로 끝나게 된다. 그래서 이 명세는 로봇에게 전달되지 않을 것이다. 이런 경우 사용자 에이전트는 다시 로봇의 요청에 맞는 서비스를 찾기 위한 시도를 하게 된다.

5. 결론 및 향후 연구

본 연구에서는 OWL-S와 에이전트를 사용하여 적응형 소프트웨어에 대한 웹 서비스 발견 방법을 제시하였다. OWL-S를 통해 서비스를 표현하고, 웹 서비스에 관해 추론과 서비스 조합을 위한 플랜을 가능하게 하였다. 그리고 검색에 있어서 효율성을 높이기 위해 레지스트리를 두 종류로 나누고 최하위 도메인 마다 하위 레지스트리를 두었다. 그래서 로봇이 적합한 서비스를 찾을 때에 도메인 정보를 삽입하여 빠른 검색이 일어나도록 하였다. 그리고 검색은 사용자 에이전트를 통해서 필요한 정보가 Match maker에 넘어가서 목적에 맞는 서비스의 조합을 기술하는 프로세스 명세를 가져오도록 하였다. 최종적으로 로봇에게 결과가 넘어가기 전에 Petri-net으로의 변환을 통해 OWL-S에서 서비스들의 특징에 대한 정확성을 검증하였다.

향후 연구로는 로봇에서 적응성이 일어나는 데에 있어 전체적인 방향과 일관성을 제시할 수 있는 룰(Rule)이나 정책 (Policy)을 제시하고 기술하는 연구가 이루어져야 할 것이다. 그리고 로봇의 최소한의 모듈 단위를 서비스로 하여 OWL-S로 체계적으로 기술을 하여 내부 플랜에서 사용하는 정보를 가지고 그대로 외부 레지스트리에서 일어나는 플랜에 사용할 수 있도록 연구가 이루어져야 할 것이다. 그럼으로써 로봇에 더 적합하고, 효율적인 서비스를 찾도록 가능하게 해야 할 것이다.

참고 문헌

[1] 김진한, 이창호, 이광우, 이병정, 김희천, " 적응형 소프트웨어에 대한 웹 서비스 발견 기반 기법," *한국컴퓨터종합학술대회 논문집*, 제 33권, 제 1(C)호, pp. 175-177, July 2006.

[2] 구형민, 박유식, 김기현, 고인영, 최호진, "아키텍처 기반의 자가 성장 로봇 소프트웨어를 위한 저장소 구조," *Proc. of Korea Conference on Software Engineering*, pp. 219 - 227, 2006.

[3] M. P. Singh and M. N. Huhns, *Service-Oriented Computing*, John Wiley & Sons, Ltd., 2005.

[4] Z. Maamar, S.K. Mostefaoui and H. Yahyaoui, " Toward an Agent-Based and Context-Oriented Approach for Web Services Composition," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, Iss. 5, pp. 686 - 697, 2005.

[5] N. Milanovic and M. Malek, " Current solutions for Web service composition," *IEEE Internet Computing*, Vol. 8, Iss. 6, pp. 51 - 59, 2004.

[6] W. Xinqi and Y. Xueli, " A OWL-Based Semantic Web Service Discovery Framework," *Proc. of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*, 2006.

[7] M. Burstein, C. Bussler, T. Finin, M. N. Huhns, M. Paolucci, A. P. Sheth, S. Williams and M. Zaremba, " A semantic Web services architecture," *IEEE Internet Computing*, Vol. 9, Iss. 5, pp. 72 - 81, 2005.

[8] S.J.H. Yang, B.C.W. Lan and J.-Y. Chung, " A New Approach for Context Aware SOA," *Proc. of IEEE International Conference on e-Technology, e-Commerce and e-Service*, pp. 438 - 443, 2005.