반응형 계획에 기초한 자동화된 시맨틱 웹서비스의 조합

진 훈[†] · 김 인 철[††]

요 약

최근 들어 인공지능 계획기법을 이용하여 자동화된 시맨틱 웹서비스들 간의 조합을 구현하려는 연구들이 활발하게 이루어지고 있다. 하지만 일반적으로 전통적인 인공지능 계획기법들은 복잡한 제어구조를 포함하는 웹서비스 프로세스를 하나의 단위 행동이나 계획으로 표현하기 어렵고, 온톨로지에 포함된 의미 정보들을 계획생성에 충분히 반영할 수 없으며, 웹서비스들 사이의 입출력 데이터 흐름을 직접 모델링할 수 없고, 계획단계와 실행단계가 분리되어 있어 웹서비스 실행단계의 불확실성과 가변성을 계획단계에서 충분히 고려할 수 없다는 등의 한계점을 가지고 있다. 본 연구에서는 이러한 문제점들을 해결하기 위한 접근 방법으로서 반응형 계획을 이용한 시맨틱 웹서비스 조합을 제안하고, 프로토타입 시스템인 SWEEP을 구현하였다. e-Commerce 분야의 예제 웹서비스들을 대상으로 실험을 통해, 우리는 반응형 계획이 자동화된 시맨틱 웹서비스의 조합과 실행을 구현하기 위한 효과적인 기술임을 확인할 수 있었다.

키워드 : 시맨틱 웹서비스, 시맨틱웹, 서비스 조합, 반응형 계획기, OWL-S, 프로세스 모델, 계획 문제

Automated Composition of Semantic Web Services Based on Reactive Planning

Jin Hoon[†] · Kim Incheol[††]

ABSTRACT

Recently, there have been a lot of works trying to realize automated composition of semantic web services through application of AI planning techniques. The traditional AI planning techniques, however, have some limitations: it is not easy to represent a web service process with complex control constructs as an action or a plan; it is hardly possible to consider enough the rich information contained in domain ontologies during the planning process; it is impossible to model directly the data flow from the outputs of a web service to the inputs of another web service; it is difficult to predict and deal with uncertainty and dynamics of the environment because the plan generation phase is supposed to be separated from the plan execution phase. In order to overcome some of these limitations, this paper suggests a reactive planning approach to automated composition of semantic web services. Through some experiments using several e-commerce web services, we found that the reactive planning is an effective way to realize automated composition of semantic web services.

Key Words : Semantic Web Service, Semantic Web, Service Composition, Reactive Planner, OWL-S, Process Model, Planning Problems

1. 서 론

웹서비스란 네트워크를 통해 컴퓨터들 간의 상호작용을 지원하도록 개발된 소프트웨어 시스템으로서, WSDL(Web Services Description Language), SOAP(Simple Object Access Protocol), UDDI(Universal Description, Discovery and Integration)를 이용하여 정의될 수 있다[4]. 이러한 웹서비스들은 사용자의 복잡한 요구가 있거나, 이미 존재하는 웹서비스들의 수행만으로는 원하는 결과를 직접 얻을 수 없을 때, 이들을 조합(composition)함으로써 복합 서비스를 제공할 수 있다. 그러나 웹서비스는 의미가 결여된 기능적 정보들만을 이용하여 작성되기 때문에, 자동화된 서비스 조합을 구현하기에는 적절하지 않다. 그래서 기존의 웹서비스에 의미적인 기술(annotations)를 추가하여 컴퓨터가 웹서비스의 기능을 이해할 수 있도록 확장시킴으로써, 컴퓨터들 간의 자동화된 처리를 위해 시맨틱 웹서비스가 개발되었다. 시맨틱 웹서비스와 관련된 이슈들로는 자동화된 서비스 발견, 호출, 조합, 감시 등이 있으나, 이중에서도 특히 조합에 관한 연구들이 활발하게 연구되고 있다.

시맨틱 웹서비스 조합은 시맨틱 웹서비스들을 대상으로

† 준 회 원 : 경기대학교 정보과학부 외래교수
†† 종신회원 : 경기대학교 전자계산학과 교수
논문접수 : 2006년 9월 25일, 심사완료 : 2007년 3월 21일

주어진 목표에 대해, 서비스를 구성하는 여러 개념들과 그들 간의 관계를 의미적으로 고려하여, 자동으로 복합 서비스를 구성하는 것을 목표로 삼는다. 시맨틱 웹서비스 조합에 관한 연구들은 시맨틱 웹서비스들을 대상으로 하여 자동화된 조합을 구현하기 위해, 주로 인공지능 연구자들을 중심으로 연구되어 왔다. 이들은 전통적인 인공지능 분야의 계획기법을 이용함으로써 조합 문제를 해결하고자 하였다.

그러나 시맨틱 웹서비스 조합을 위해 전통적인 인공지능 분야의 계획수립 기법을 이용하는데 있어, 해결해야 할 몇 가지 중요한 문제점들이 존재한다. 전통적인 계획수립 기법들은 첫째로, OWL-S와 같은 서비스 온톨로지에서 정의하고 있는 다양하고 복잡한 제어구조를 하나의 단위 행동이나 계획으로 표현하지 못한다는 것이다. 둘째로 온톨로지에 포함된 여러 개념들 간의 의미와 관계를 계획생성에 충분히 반영할 수 없다는 것이다. 셋째로 사전에 웹서비스 실행 시에 발생할 수 있는 데이터의 타입 정보 또는 값을 정의할 수 없음으로 인해, 웹서비스들 사이의 입출력 데이터 흐름을 직접 모델링 할 수 없다는 것이다. 넷째로 계획단계와 실행단계가 분리되어 있기 때문에, 서비스 실행단계에서의 불확실성과 가변성을 계획단계에서 충분히 고려할 수 없다는 것이다. 이와 같은 문제들은 웹서비스 조합을 구현하기 위한 계획 기술의 적용을 어렵게 만드는 요인들이 된다.

이에 본 논문에서는 시맨틱 웹서비스 조합을 위한 방법으로서, 반응형 계획 기술의 사용을 제안한다. 반응형 계획 기술은 계획수립과정과 실행과정을 병행하면서, 불확실하고 가변적인 환경에서 환경의 변화에 대해, 동적으로 계획을 재수립하고 실행한다. 또한 주어진 문제를 해결하기 위해, 목표 지향적이면서 계층적인 추론과정을 수행할 수 있다. 이와 같은 반응형 계획 기술을 적용하기 위해서는, 서비스가 제공하는 명세를 계획으로 변환시키는 과정과, 변환된 계획들을 효과적으로 계획수립 과정에 참여시키는 과정이 필요하다. 우리는 4장에서 이에 관하여 설명한다. 또한 우리는 제안한 반응형 계획기를 이용하는 조합 도구인 SWEEP(Semantic Web service Execution Environment based on Plans)을 구현하였다. 그리고 e-Commerce 분야의 예제 웹서비스들을 대상으로 실험하고, 결과를 통해 제안된 접근 방식의 적용 가능성과 효과를 알아보고자 한다, 이에 관해서는 5장에서 다룬다. 마지막으로 6장에서는 결론과 연구 의의, 그리고 향후 연구 방향을 소개한다.

2. 관련 연구

2.1 웹서비스 조합 연구

웹서비스 조합 연구는 주로 기업들을 중심으로 진행되었고, WSDL로 표현된 개별 웹서비스들을 대상으로 BPEL4WS(Business Process Execution Language for Web Services)와 같은 언어들을 이용하여 이들 간의 서비스 흐름 관계를 표현하고 실행하는 과정을 다룬다. 이때 WSDL은 추상화된 웹서비스의 기능에 대한 XML형식의 명세로서, 구체적인 웹서비스로의 연결을 나타내기 위한 프레임웍을 제공한다. WSDL 명세는 서비스 소비자가 서비스 제공자와 통신하도록 허용하는, 추상화된 웹서비스 인터페이스를 나타낸다.

BPEL4WS는 프로세스에 참여하는 파트너들의 역할과 이들로부터 발생되는 메시지 교환에 관한 논리적인 흐름을 표현하는데 이용되며, 웹서비스들을 비즈니스 프로세스 구성을 위한 추상적이고, 표준화된 인터페이스로 인식한다. 그러므로 서비스들 간의 흐름 관계는 의미적 연결이라기보다는 문법적인 연결 관계를 나타낸다. 또한 BPEL4WS와 같은 언어들에서 강조되는 내용이 에러 처리나 메시지 의존 관계 등임을 감안할 때, 실행 과정 동안 예측된 환경 속에서 견고하고, 효과적으로 동작한다는 장점을 갖는다. 하지만, 자동성이라는 측면에서, 전문가의 적극적인 개입이 요구되며, 가변적 환경에서의 동적인 변화를 수용할 수 없다는 한계성을 갖는다[10,19].

이에 비해, 시맨틱 웹서비스 조합 연구는 주로 인공지능 연구자들을 중심으로 진행되었고, OWL-S로 작성된 서비스들을 대상으로 계획 기술을 이용하여 자동화된 서비스 조합을 구현하는 과정을 다룬다. OWL-S는 하나의 서비스 온톨로지를 가지며, 서비스 온톨로지는 3가지 속성들인 presents, describedBy, supports 을 갖는다. 그리고 이러한 속성들은 속성에 대한 범위(range)로서 각각 서비스 프로파일(Service Profile), 서비스 모델(Service Model), 서비스 그라운딩(Service Grounding)을 갖는다. 서비스 프로파일은 서비스 수행에 영향을 미치는 기능적 정보와 비기능적 정보(주석 정보)들로 구성된다. 그리고 서비스 모델은 프로세스 모델이라고도 불리며, 실제 서비스들 간의 동작과정을 나타내기 위한 프로세스 수준의 처리과정을 기술한다. 이때 프로세스 모델을 구성하는 프로세스들은 기본(Atomic) 프로세스와 단순(Simple) 프로세스, 그리고 복합(Composite) 프로세스로 나눌 수 있다. 서비스 그라운딩은 실제 실행을 위한 프로토콜, 바인딩 정보, 메시지 타입 등을 정의하는 곳이며 주로 WSDL과의 연결을 위한 형식 정보가 기술되어 있다[3].

계획 기술을 이용한 연구들로는 Golog, Rule 기반의 연구들 및 Shop2를 이용한 연구와 OWLS-Xplan에 관한 연구 등이 있다. Shop2는 도메인에 독립적인 HTN(Hierarchical Task Network)기반의 계획수립 시스템이다. Shop2는 계획수립을 위해 기본 프로세스, 단순 프로세스, 복합 프로세스로 정의된 OWL-S 서비스들을 계획으로 변환하여 이용할 수 있다. 또한 계획수립 과정 중 정보획득 작업을 통해 실제의 웹서비스들을 대상으로 하여, 계획수립에 필요한 정보들을 현재의 상태정보로 이용할 수 있다. 그러나 정보획득 작업 중에 모든 웹서비스들을 대상으로 정보획득 작업을 수행해야 하고, 실행 시에 기 실행된 서비스들을 재실행해야 한다는 오버헤드를 갖는다. 결국 Shop2를 이용한 조합방식 역시, 계획과정과 실행과정의 분리로 인해 환경의 변화에 효과적으로 반응할 수 없다는 한계를 갖는다[17]. OWLS-Xplan은 시맨틱웹 기반의 OWL-S 서비스 조합을 위한 계

〈표 1〉 계획기법들의 기능 비교

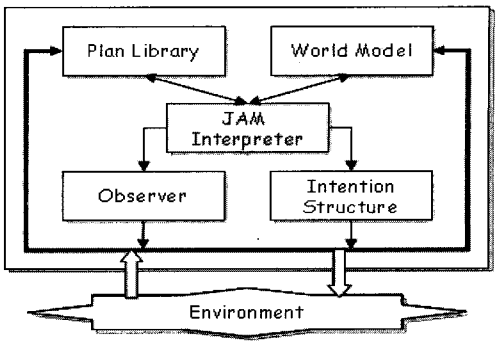
	BPEL4WS	Golog	Rule-based	SHOP2	OWL-S2Xplan	Reactive Planning
자동성	△	○	○	○	○	○
복합 제어구조 표현	○	△	?	△	△	○
풍부한 데이터 타입 사용	X	△	△	△	△	△
동적 객체처리	△	X	X	△	△	○
실시간 계획수립과 실행	△	X	X	X	X	○

획기로서, OWL-S 서비스들을 대상으로 하여 빠르고 유연하게 조합을 구현한다. OWLS-Xplan은 효과적인 그래프-계획 기반의 FF(FastForward) 계획기에 HTN 계획요소를 갖도록 확장시킨 하이브리드(hybrid) 계획기인 Xplan를 이용한다. 그러므로 OWLS-Xplan은 목표 달성을 위해, OWL-S 1.1 명세들을 대상으로 PDDL(Planning Domain Definition Language) 2.1기반의 영역 지식 파일과 문제 파일을 생성하여 Xplan 계획수립을 위한 입력 요소로서 사용한다. Xplan은 실행 중에 사전 계획과정에서 생성된 그래프 기반의 경로를 재탐색하는 기능을 제공한다. 그러나 OWLS-Xplan은 사용자에 의해 주어진 완전한 정보만을 이용함으로써 실제의 서비스 환경에 적용하기에는 어려움이 존재하며, 역시 실행과정에서 발생 가능한 환경의 변화에 대해 효과적으로 반응할 수 없다[10].

2.2 반응형 계획 기술

가변성이 높고 불확실성이 존재하는 환경들에서는, 모든 가능한 상황들을 고려하는 반응 행위를 생성하고 저장하는 것이 불가능하다. 이를 위해 몇몇 연구들은 실행 중인 소프트웨어에게 계획수립 능력을 제공하는 것을 제안하였다. 이와 같은 소프트웨어는 환경의 변화를 감지하여 반응을 위한 행위를 계산해 낼 수 있다. 이러한 기능을 제공하는 계획기를 반응형 계획기라고 한다[13]. 반응형 계획기는 일반화된 계획들을 미리 내장한 상태에서 외부의 환경 변화를 인식하고 이에 능동적으로 대처하는 재계획 수립 과정을 지원한다. 또한 복수 개의 목표들을 동적으로 달성하도록 지원하고, 계획과정과 실행과정의 병행, 환경의 불확실성과 변화에 대한 높은 적응성을 특징으로 갖는다. 이와 같은 반응형 계획 기능을 제공하는 시스템들로는 RAP, TCA, PRS, JAM 등이 있다. RAP은 주로 로봇 행위를 다루기 위해 개발된 시스템이며 Lisp 언어로 개발되었다. TCA 역시 모바일 로봇을 위한 태스크 수준의 제어 시스템으로서 Lisp과 C 언어로 개발되었고, 수직적인 태스크 세분화 방식을 취한다. PRS의 경우 동적인 환경에서 복잡한 태스크를 수행할 수 있는 실시간 추론 시스템 개발을 위한 프레임웍이며, C++ 언어로 개발되었다. 그리고 JAM은 PRS의 Java 언어 버전이라 할 수 있는데, 이를 구체적으로 설명하면 다음과 같다.

JAM은 (그림 1)과 같이 5가지의 구성 요소를 갖는다. 월드모델(World Model)은 믿음을 나타내는 일종의 데이터베이스이다. 계획집합(Plan Library)은 목표를 달성하기 위해 이용할 수 있는 계획들의 집합이다. 인터프리터(Interpreter)는 두뇌역할을 담당하는데, 무엇을, 언제, 어떻게 해야 할지를 추론한다. 의도구조(Intention Structure)는 에이전트가 성취하고자 하는 현재의 목표들에 대한 내부 모델이고, 목표를 성취해 나가는 과정을 책임지고 기록한다. 감시기(Observer)는 사용자가 직접 작성하는 함수로서, JAM이 수행하는 보통의 목표의 범위 또는 계획 기반의 추론 범위를 넘어서는 기능을 수행하기 위해, 계획수립 과정들 중에서 교차 수행하는 가벼운 수준의 선언적 절차이다[9].



(그림 1) JAM 구조

JAM은 계획라이브러리, 즉 재료가 되는 계획들과 달성하고자 하는 전체 목표(top level goal)를 이용하여 계획을 수립한다. 이를 항목별로 보면, GOAL부에는 계획실행을 통해 달성할 수 있는 목표를 기술하고, BODY부에는 목표 달성을 위한 세부적인 절차들을 기술한다. 또한 BODY부에는 하부목표에 대한 기술도 포함한다. PRECONDITION부에는 계획수립과정에 적용가능성을 판단하기 위해 계획 선택 이전에 한번 수행하는 조건들을 기술하게 된다. CONTEXT부 역시 PRECODNTION부와 유사한 내용을 기술하나, 계획수립과

```
Plan: {
    GOAL: [goal specification] or CONCLUDE: [world model relation]
    NAME: [string]
    BODY: [procedure]
    <DOCUMENTATION: [string]>
    <PRECONDITION: [expression]>
    <CONTEXT: [expression]>
    <UTILITY: [numeric expression]>
    <FAILURE: [non-subgoaling procedure]>
    <EFFECTS: [non-subgoaling procedure]>
    <ATTRIBUTES: [string]>
}
```

(그림 2) JAM 계획 명세

정 동안 주기적으로 수행될 수 있다. 이때 PRECODITION부와 CONTEXT부에 기술되는 조건들은 계획 수행을 위한 입력 요소로 사용될 수 없고, 하부목표 수행을 통해 달성될 수 없다. EFFECTS부는 계획이 성공적으로 수행되었을 때 월드모델 상의 변화를 나타내고, FAILURE부는 계획이 실패했을 때 수행할 동작들을 기술한다. 또한 UTILITY부는 계획의 유용성을 나타낸다. PRECONDITION부, CONTEXT부, EFFECTS부, FAILURE부에는 BODY부에 기술될 수 있는 여러 행위들로 구성되는 절차들을 기술할 수 있다. 이와 같은 JAM 계획 명세를 고려할 때, 결국 JAM 계획을 수행하기 위한 입력정보들은 BODY부 내에 기술되어야 하고, 출력정보들은 대체로 GOAL부와 EFFECTS부에 작성될 필요가 있다.

3. 웹서비스의 조합 문제

2장에서 소개한 바와 같이 기업들을 중심으로 한 BPEL4W와 같은 연구들은 실행과정에서의 자동화는 구현하고 있으나, 조합과정은 사람에 의해 수동적으로 작성해야 하는 한계성을 갖는다. 이에 비해 계획 기술을 이용한 연구들은 조합과정에서의 자동화를 이루기 위한 시도로써, 인공지능 기술의 발전과 더불어 오랜 세월 동안 연구되었던 계획 기법들을 이용한다. 그러므로 본 연구에서는 웹서비스 조합을 자동화시키기 위한 방법으로써, 계획 기술을 이용하고자 한다.

3.1 웹서비스의 예

본 논문에서는 시맨틱 웹서비스 온톨로지로서 OWL-S를 이용하기로 한다. 다음은 MindSwap에서 제공되는 예제 웹서비스들[19]이다.

(그림 3)은 아마존 도서검색 검색서비스의 OWL-S 명세이고, 서비스 모델로서 기본 프로세스인 Amazon PriceProcess를 가진다. AmazonPriceProcess는 입력인수로서 BookInfo를 필요로 하며, 출력인수로서 BookPrice를 정의하고 있다. 정의된 온톨로지에 의해, BookInfo의 인수 타입은 Book임을 알 수 있다.

(그림 4)는 복합 서비스로 주어진 최저 도서가격 검색 서비스로서, 복합 프로세스의 정의와 이를 구성하는 요소 프로세스들, 그리고 이들 간의 관계를 나타내는 제어구조가

```
... <!-- Process description -->
<process:AtomicProcess rdf:ID= "AmazonPriceProcess" > ...
    <process:Input rdf:ID= "BookInfo" > ...
     <process:parameterType
          rdf:datatype= "&xsd;anyURI" >&bibtex;Book</process:parameterType> ...
    <process:Output rdf:ID= "BookPrice" > ...
<grounding:WsdlGrounding rdf:ID= "AmazonPriceGrounding" > ...
     <grounding:wsdlDocument rdf:datatype= "&xsd;anyURI" >
          http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl ...
```

(그림 3) AmazonBookPrice.owl

```
... <!-- Process description -->
<process:CompositeProcess rdf:ID= "FindCheaperBookProcess" > ...
     <process:Sequence> ...
          <process:composedOf> ...
               <process:Split-Join> ...
     <process:Perform rdf:ID= "FindBookInfo" > ...
     <process:Perform rdf:ID= "ComparePrices" > ...
<process:CompositeProcess rdf:ID= "ComparePricesProcess" > ...
     <process:composedOf> ...
          <process:If-Then-Else> ...
     <process:Produce rdf:ID= "ProduceBNPrice" > ...
     <process:Produce rdf:ID= "ProduceAmazonName" > ...
```

(그림 4) FindCheaperBook.owl

표현되어 있다. FindCheaperBookProcess는 현재 서비스의 메인 프로세스를 나타내고, 복합 프로세스를 위한 제어구조로서 Sequence, Split-Join, If-Then-Else가 사용된 것을 알 수 있다. FindBookInfo는 메인 프로세스를 구성하는 개별 프로세스를 나타내며, 실제의 외부 서비스를 호출하기 위한 대리적 성격을 갖는다. 마지막으로 프로세스 Produce BNPrice는 복합 프로세스의 출력을 담당하는 프로세스이다.

3.2 문제 및 가정

(그림 3, 4)와 같은 시맨틱 웹서비스들을 대상으로 조합을 구현하기 위해서는 먼저 웹서비스 조합이 갖는 의미와 문제점들을 살펴볼 필요가 있다. 웹서비스 조합은 원시 웹서비스들을 사용하여 복합 프로세스를 갖는 웹서비스를 구성하는 것이라 정의할 수 있다. 복합 프로세스를 구성하기 위해서는 원시 웹서비스들로부터 IOPE(Inputs, Outputs, Preconditions, Effects)와 같은 인수들로 표현되는 프로세스 모델정보가 이용된다. 이때 원시 웹서비스란 OWL-S 에서 정의하고 있는 기본, 단순, 복합 프로세스를 갖는 서비스로서, 보다 복잡한 웹서비스를 구현하기 위해 이용된다. 이와 같은 웹서비스 조합을 효과적으로 구현하기 위해서는 웹서비스 위치화(location) 문제와 웹서비스 기술명세 변환 문제를 고려해야 한다.

먼저 조합을 위한 웹서비스 위치화와 관련하여, 보통 하나의 서비스는 사전조건 또는 입력을 인수로 하여 현재 서비스의 실행과정을 통해 출력 또는 사후영향을 발생시킨다. 이때 현재 서비스를 수행하기 위해서는 반드시 사전조건을

만족시키거나 입력값을 제공해야만 한다. 그러나 임의의 웹서비스들을 대상으로 조합을 구현하고자 할 경우, 연결되어야 할 서비스들이 어디에 있고, 어떻게 해당 서비스들을 찾아야 할지 알 수 없다. 또한 설령 적절한 서비스를 찾았다 할 지라도 각각의 서비스들의 입출력 정보들은 연계성을 고려할 때에 서로 판이하게 다를 수 있다. 웹서비스 기술명세(descriptions) 변환은 BPEL4WS와 같은 기업을 중심으로 한 연구 기법 또는 인공지능 분야의 계획 기술을 이용하여 조합을 구현하기 위해서 고려해야 할 또 하나의 중요한 문제이다. 이는 웹서비스 조합이 해당 기법을 이용하여 작성되는 워크플로우 또는 수립된 계획으로 표현되기 때문에, 조합을 위해서는 먼저 웹서비스를 기술하고 있는 명세의 변환작업이 수행되어야 한다. 이때 각각의 기법들이 웹서비스의 형식적, 의미적인 측면에서의 기술명세들을 얼마나 잘 표현하고 지원하는가가 조합기법 선택에 있어서 중요한 잣대가 될 수 있다.

이와 같은 문제점들에 대해 본 논문에서는 몇 가지의 가정을 통해 어느 정도 제한된 수준의 웹서비스들로 한정시키고, 이들을 대상으로 자동화된 웹서비스 조합 방법을 제안하고자 한다.

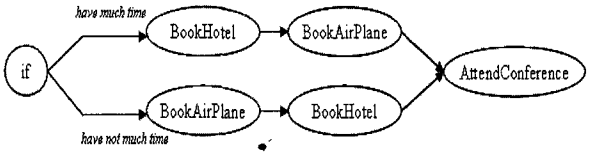
- -첫째, 조합을 구현하기 위한 원시 서비스들은 사전에 정의된다; 전술한 바와 같이 웹서비스 조합을 구성될 웹서비스들은 조합과정에 앞서 미리 어떠한 형태로든 미리 정의된 서비스들만을 이용할 필요가 있다. 일반적으로 UDDI와 같은 웹서비스를 위한 온라인 레지스트리에 등록된 서비스들만을 대상으로 이용하거나 로컬 컴퓨터 상에 미리 해당 서비스 기술명세들을 저장시키고 이용할 수 있다. 그리고 서비스들 간의 연결을 위해서 사람의 입력을 필요로 하는 서비스를 제외한 모든 서비스들에 대해, 반드시 다른 서비스들로부터의 결과를 현재 서비스 수행을 위한 사전조건, 또는 입력으로 이용될 수 있는 서비스들로만 한정지을 필요가 있다.
- -둘째, 조합을 위해서는 원시 서비스들의 프로세스 모델 정보만을 대상으로 한다; OWL-S 서비스 온톨로지의 정의에 따라 작성된 서비스들은 서비스 프로파일, 프로세스 모델, 서비스 그라운딩 정보로 구성된다. 이때 OWL-S의 정의에 따라 작성된 서비스들은 프로파일부의 정보들과 프로세스 모델부의 정보가 상이할 수 있다. 그러나 서비스 요청자에 의해서 수행되어야 하는 서비스의 모든 동작과정은 프로세스 모델부에 기술된다. 그러므로 웹서비스 조합을 위해 사용되는 모든 웹서비스들은 반드시 프로세스 모델 정보를 가져야 하며, 이들만을 대상으로 조합을 구현한다.
- -셋째, 웹서비스 인수들에 대한 온톨로지 정보는 변환의 대상에서 제외한다; SHOP2에 관한 연구와 OWL-S2Xplan을 이용한 연구 등에서는 온톨로지로 표현된 의미적 정보들을 계획수립을 위한 지식으로 변환하여 이용한다. 하지만 이러한 방법들 역시 열린 세계의 지식을 대상으로 하는 웹서비스 정보들을 닫힌 세계의 지식을 대상으로 하는 계획 기술에서 이용하는 데에는 한계가 존재한다. 이를 해결하기 위해서는 다양한 수준의 논리적 기호들과 규칙으로 표현된 대량의 온톨로지 정보를 실시간적으로 변환하고 이용하기 위한 별도의 기술 개발과정이 요구된다. 그러므로 서비스 수행을 위한 인수 정보들에 대해 복잡한 의미해석 과정을 필요로 하는 웹서비스들은 이용의 대상에서 제외한다.
- -넷째, 복합 프로세스들 간의 데이터 흐름 처리는 기술명세 변환 과정에서 예외로 처리한다; 현재까지 알려져 있는 OWL-S 복합 서비스들은 대부분 인위적으로 작성된 것들이며, 설령 소프트웨어 도구 등을 이용하여 자동으로 작성된 것일지라도 사람의 개입과정을 요구한 경우가 많다. 이때 제어 흐름의 경우 자동화 처리를 위한 기법들이 많이 도입되어 사용되고 있는 추세이나 이들을 기반으로 한 데이터들 간의 흐름 처리는, 해당 프로세스들만의 고유한 기능과 방법들을 사용함에 따라 현실적으로 자동화시키기 어려운 형편이다.

3.3 고려 사항

일반적으로 계획기법들은 계획 문제를 $P=\{S, S_0, G, A, T\}$로 정의하고, 원시(primitive) 계획은 하나의 단일 행위 또는 오퍼레이터로 정의하고 이용된다(S:월드 모델에 대한 모든 가능한 상태 정보, S_0:초기 상태, G:목표 상태 집합, A:행위 상태 집합, T:각 행위들의 의미를 나타내는 전이 관계, $T \subseteq SxAxS$). 목표를 달성하기 위해 수립된 계획은 부분적으로 순서화된 행위들의 연속(sequence)으로 표현한다. 이때 이러한 행위들은 기본(atomic) 행위들로서 개별적으로 적용될 수 있고, 적용 결과가 예측된 조건들에 포함된 유효한 행위 기술들로부터 계획수립 과정을 시작한다. 하지만 웹서비스의 거대한 탐색 공간과 실세계의 문제들을 다루는 정보들을 대상으로 하여 제한된 지식 세계만을 완전한 것으로 고려하고 이를 풀기 위해 계획 기술을 이용한 접근 방법을 취하는 것은 대단히 위험할 수 있다. 그러므로 웹서비스가 다루는 실세계 정보를 완전하고도 정확하게 표현할 수 없다는 것은 전통적인 계획기법을 적용하는데 있어서 중요한 문제로 대두된다. 이와 같은 문제점들을 <표 1>에서 나타낸 기능적 분류에 따라 간략하게 기술하면 다음과 같다.

3.3.1 복합 제어 구조 표현

전통적인 계획기법들은 (그림 5)에서와 같이 다양한 복합 제어 구조로 연결된 서비스들을 일련의 연속된 행위들로밖에 표현할 수 없는 한계로 인해, 제대로 표현해 낼 수가 없다.



(그림 5) 복합 예약 서비스 흐름 구조

3.3.2 풍부한 데이터 타입 사용

- 의미적 개념 - 예. Ticket(영화표?, 비행기표?)
- 개념들 간의 관계 - 예. 자동차-세단, SUV

위에서와 같이 웹서비스는 데이터 타입 정보들을 이용하여 사용된 용어들 간의 의미적인 개념 및 그들 간의 관계정보를 파악해낼 수 있다. 하지만 계획기법들은 계획 작성을 위해 주로 단순한 선언 방식만을 사용하여 타입 정보를 이용한다는 한계를 갖는다.

예) ?room1 - room (PDDL), robot_status "OK"(JAM)

3.3.3 동적 객체 처리

<process:Output rdf:ID="BookPrice"> ←(BookPrice=$10? Or $5?)

전통적인 계획기법들은 위와 같은 경우에 "BookPrice"라는 값이 얼마인지를 정의하기가 어렵다. 이는 웹서비스의 수행이 데이터 처리라는 관점에서는 데이터의 입출력에 관한 처리이며, 상태 변경이라는 관점에서 이전 상태를 이후 상태로 변경시키는 경우를 모두 포함할 수 있지만, 계획에서의 동작(행위)은 단지 이전 상태에서 이후 상태로의 상태의 변경만을 다루는 한계에서 비롯된다.

3.3.4 실시간 계획수립과 실행

조합된 서비스를 실행할 때, 여러 파트너들과의 상호 작용 속에서 수시로 조정 작업이 발생할 수 있다. 이러한 작업은 필요에 따라 이미 생성된 서비스를 교체하거나, 파트너를 교체하는 과정까지도 포함한다. 하지만 전통적인 인공지능 계획 기법들은 계획수립 단계와 실행 단계가 분리되어 있으므로, 이 같은 작업이 불가능하다. 실제로 웹서비스들은 인터넷을 기반으로 동작함으로써, 인터넷 환경이 지닌 여러 가지 오류나 에러 상황 등에 직면할 수가 있다. 그러므로 이러한 문제들을 내포하고 있는 웹서비스들을 대상으로 계획을 수립하고, 이렇게 미리 정해진 상태에 기반하여 수립된 계획을 별도의 단계에서 실행하는 것은 대단히 불안정하며, 신뢰할 수 없다.

이와 같은 문제점들을 고려할 때, 반응형 계획기법은 환경 변화에 대한 반응성과 계획과 실행의 병행적 수행, 그리고 계획의 계층적 전개 능력을 지원할 뿐만 아니라, 목표 또는 데이터 지향적인 추론 방식, 절차적 행위 기술, 절차적인 구조를 위한 풍부한 표현 양식을 제공함으로써 대안으로 사용될 수 있다. 그러므로 본 연구에서는 반응형 계획기법 중의 하나인 JAM을 이용하기로 하였다. JAM은 Java 언어로 개발된 시스템으로서, 반응형 계획 기술의 특징인 외부 환경에 반응하여 재계획을 수립하는 기능과, 주어진 태스크를 계층적으로 세분화시켜 나가면서 문제를 해결해 나가는 기능, 또한 다양한 제어 구조들과 변수 기능을 제공한다. 또한 JAM은 계획수립 과정과 실행과정을 동시에 병행 수행함으로써 동적으로 변화하는 환경에서 계획을 효과적으로 수립하고 실행하는 기능을 제공한다.

4. 조합을 위한 기술명세 변환

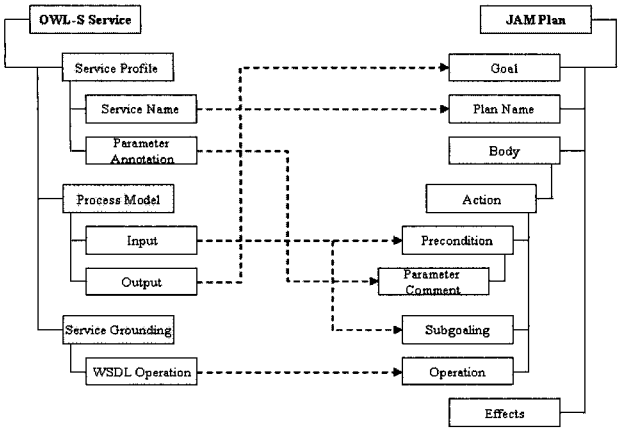
OWL-S로 정의된 시맨틱 웹서비스들을 대상으로 JAM과 같은 반응형 계획기를 이용한 조합을 구현하기 위해서는 먼저 웹서비스 기술 명세를 계획(또는 계획 기술명세)로 변환시켜야 한다. OWL-S 서비스는 기본, 단순, 복합 프로세스들로 구성되며, 이들 각각을 위한 변환과정을 설명하면 다음과 같다.

4.1 기본 서비스 변환

계획기법에서는 OWL-S의 3가지 구성 요소 중 프로세스 모델을 대상으로 하여 계획수립 과정에 이용한다. 프로세스 모델은 해당 서비스를 통해 수행하려는 프로세스들과 이들을 위한 인수 정보들을 포함한다. 또한 복수의 프로세스들을 이용하는 경우, 복합 제어 구조를 통해 연결된다. 이에 비해 JAM 명세는 목표는 GOAL부에, 계획수립을 위해 요구되는 지식 또는 상태 정보들과 동작을 기술하기 위한 행위들을 주로 BODY부에 기술한다. 그리고 계획실행의 결과를 반영하기 위한 행위는 EFFECTS부에 기술한다.

OWL-S 서비스는 서비스를 구성하는 메인 프로세스가 기본 프로세스, 단순 프로세스, 복합 프로세스인지에 따라 기본 서비스, 단순 서비스, 복합 서비스로 나뉠 수 있다. 하나의 OWL-S 서비스는 입력을 위한 인수정보(parameters)로 입력(input)과 사전조건(preconditions)을 모두 가질 수 있고, 출력으로 출력(output)과 사후결과(effects)를 모두 가질 수 있다. 대개의 경우, 각각은 입력 또는 사전조건 만을, 출력 또는 사후결과만을 갖는 편이지만 때로는 사전조건에 입력이 포함될 수 있고, 사후결과에 출력이 포함되는 경우도 있다. 그러므로 이와 같은 경우들을 모두 감안하는 JAM 계획으로의 변환절차가 필요하다. 또한 복수 개의 입출력 인수들을 갖는 서비스들에 대하여 조합과정을 이루기 위해서는 서비스들 간의 해당 입출력 인수에 대한 온톨로지를 이용할 필요가 있다.

OWL-S 서비스는 프로세스 모델부에 프로세스들과, 프로세스들을 수행한 결과로서 사후결과 또는 출력을 갖는다. 이에 비해 계획은 GOAL부에 행위의 목적이 되는 목표를 상태정보로 표시하고, 정해진 목표를 달성하기 위한 행위들을 BODY부에 기술한다. 즉, OWL-S 서비스는 서비스 동작의 결과로서 출력인수들을 출력하지만, JAM은 주어진 목표를 달성하기 위한 행위들을 기술한다. 그러므로 이를 의미적 관점에서 생각해보면, 서비스의 수행 결과에 대해 계획의 목표로 정의되는 것이 가능하다. 그리고 OWL-S는 입력인수인 입력, 사전조건을 갖는데 비해, JAM 계획은 사전조건을 갖는다. 하지만 OWL-S의 사전조건이 호출된 서비스의 실행을 위한 전제조건이며, 필요에 따라 하부목표 달성과정을 통해서 달성이 가능한 내용을 가지는데 비해 JAM의 PRECONDITION부는 이와는 다르다. 그러므로, OWL-S의



(그림 6) 기본 서비스 명세와 JAM 계획 명세 비교

사전조건은 JAM의 PRECONDITION부에 기술될 수 없고 BODY부에 기술되어야 한다.

또한 OWL-S 서비스는 프로세스 모델로서 복합 프로세스를 가질 수 있다. 복합 프로세스는 기본, 단순, 복합 프로세스들을 가질 수 있고, 다양한 종류의 제어 구조로 표현될 수 있다. 이에 비해 JAM 계획은 편의상 OWL-S에 대한 대응 개념으로서, 단일 행위가 기술된 기본 계획(atomic plan)과, 아무런 동작(operation)을 수행하지 않고 단순히 하부 목표만을 추구하는 구문으로만 기술된 추상 계획(abstract plan), 그리고 복합 제어 구조와 여러 행위들로 구성된 복합 계획(complex plan)으로 구분지어 나타낼 수 있다. 또한 조합 및 계획수립을 위해 사용되는 서비스 또는 계획을 각각 원시 서비스 및 원시 계획이라고 한다. 이외에도 OWL-S 서비스와 같은 웹서비스들은 실행 중 발생할 수 있는 기존 데이터 값의 변화 또는 새로운 데이터 타입의 생성에 관하여, 변수처리를 통해 쉽게 정의하고 이용할 수 있다. JAM은 이에 대해, 닫힌 세계의 지식만을 강하게 고수하는 전통적인 계획수립 기법들과는 달리, 이와 같은 동적 객체정보들을 처리할 수 있는 변수 기능을 제공한다.

기본 서비스의 프로세스 모델에서 정의하고 있는 기본 프로세스에서 제공하는 입력인수들은 계획에서의 PRECONDITION부 또는 하부목표에 나타낼 수 있다, 그리고 출력인수들은 계획의 목표 또는 EFFECTS부에 나타낼 수 있다. 구성형태가 유사하다는 것은 용이하게 변환 과정을 수행할 수 있음을 의미한다.

(그림 3)의 명세와는 다르게, 일반적으로 OWL-S 서비스들은 복수 개의 입력과 복수 개의 출력을 제공할 수 있다. 그러므로 이런 측면을 함께 고려하여 OWL-S 기본 서비스를 JAM 기본 계획으로 변환시키기 위한 알고리즘을 기술하면 다음과 같다. <표 5>에서 Q는 임의의 OWL-S 서비스인 W의 기본 프로세스이며, I는 입력인수 집합, C는 사전조건 집합, O는 출력인수 집합 O, E는 사후결과 집합, P는 변환과정을 통해 생성된 JAM의 기본 계획 집합, G는 P집합의 원소 계획들이 갖는 목표정보 집합, SG는 하부목표정보 집합을 의미한다.

〈표 2〉 기본 서비스 명세 변환

■ **입력**: I와 O, C와 E를 인수로 갖는 Q에 대한 OWL-S 정의
■ **출력**: P
■ **절차**:
(1) O와 E에 대하여,
  A. O와 E가 모두 존재하며, 서로 독립인 경우, 각각 서로 다른 PO와 PE의 GO와 GP로 정의한다.
  B. O와 E가 모두 존재하며, 서로 독립이 아닌 경우, PO의 GO를 정의한다 (GE정의에 관하여는 사용자 처리 요구).
  C. O 또는 E만 존재하는 경우, GO를 가진 PO 또는 GE를 가진 PE만 정의한다.
(2) I와 C에 대하여,
  A. I와 C가 모두 존재하며, 서로 독립인 경우, 각각 P의 SGI, SGC로 정의한다.
  B. I와 C가 모두 존재하며, 서로 독립이 아닌 경우, P의 SGI를 정의한다 (SGC정의에 관하여는 사용자 처리 요구)
  C. I 또는 C만 존재하는 경우, SGI 또는 SGC 만 정의한다.
(3) Q의 이름은 P의 이름으로 정의한다.
(4) Q의 온톨로지 정보는 제한적으로 FACTS부에 정의한다.
(5) I의 주석 정보는 BODY부에, 사용자 입력 수용을 위한 사용자 정의함수에 정의한다.
(6) W의 그라운딩 정보는 실행을 위한 사용자 정의함수의 동작 대상으로 정의한다.

<표 2>의 기본 서비스 명세 변환 알고리즘을 이용하여 (그림 3)을 JAM 계획으로 변환시킨 결과는 (그림 7)과 같다.

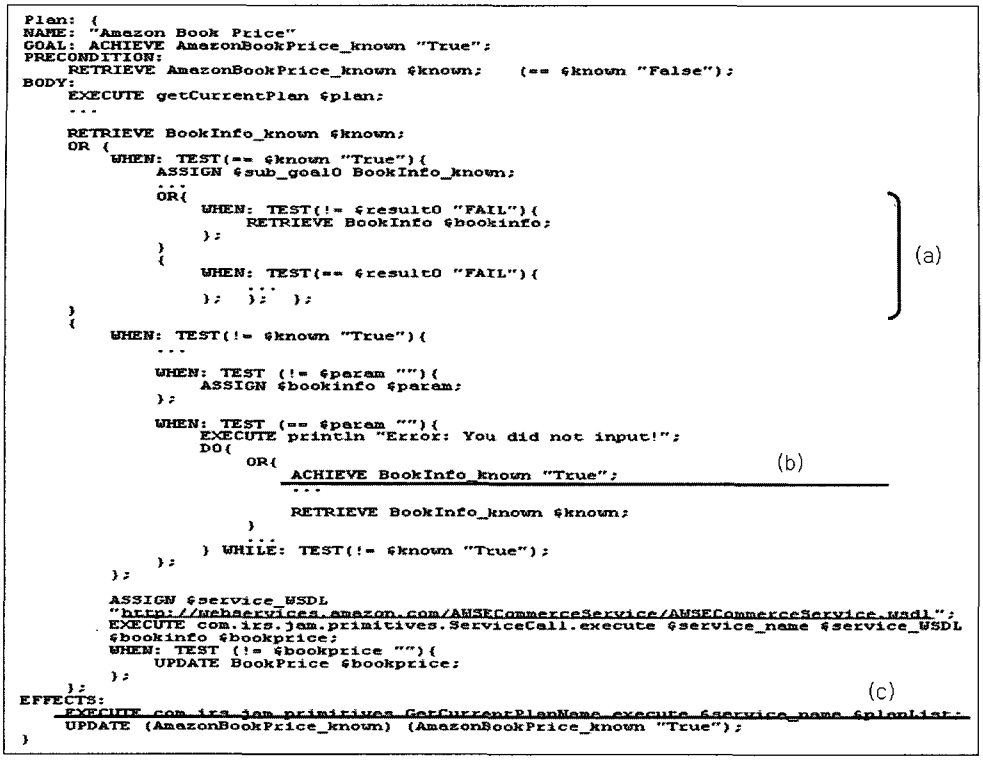
(그림 7)에서 (a)는 서비스 실행을 위한 입력요소인 BookInfo가 현재 서비스 수행 이전에 다른 서비스에 의해 구해진 경우에, 인수 타입들 간의 매칭 관계에 따라 서비스 연결을 허용할 지의 여부를 결정하는 구문이다. (b)는 BookInfo 값을 알지 못하는 경우에, 값을 얻기 위해 하부목표 달성을 수행하기 위한 구문이고, (c)는 아마존 도서가격 검색 서비스가 정상적으로 수행됨으로 인해, 최종 수립된 계획을 나타내는 계획 리스트에 추가시키는 구문이다.

4.2 단순 서비스 변환

단순 프로세스는 주로 조합서비스의 설계 시에, 먼저 이에 대응하는 구체적인 프로세스가 알려질 때 이용된다. 만약 그렇지 않을 경우, 설계 시에 정의된 단순 프로세스는 실행 시에 실제로 이를 구체화시키는 프로세스로 자동 대체될 필요가 있다. (그림 8)은 단순 프로세스의 정의와 제공된 예를 기반으로 하여 작성된 가상의 단순 프로세스이다.

(그림 8)에서 FindCheaperBookPrice는 입력인수로 BookName을 갖고 출력인수로 BookPrice를 내는 기본 프로세스 또는 복합 프로세스로 대체될 수 있다. 현재까지 알려진 정의에 의하면, 단순 프로세스는 프로세스명과 입력인수 및 출력인수 정보들로만 구성된다.

먼저 입력인수의 경우를 살펴보면, 단순 프로세스 역시 수행을 위한 입력값들을 알지 못할 때, 이들을 구하기 위해



(그림 7) AmazonBookPriceProcess.jam

```
<process:SimpleProcess rdf:ID="FindCheaperBookPrice">
  <process:hasInput>
    <process:Input rdf:ID="BookName"/>
    <process:parameterType rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
  … <process:hasOuput>
    <process:Output rdf:ID="BookPrice"/> …
</process:SimpleProcess>
```

(그림 8) FindCheaperBookPrice.owl

JAM 계획 내에서 하부목표 달성을 위한 구문을 기술한다는 면에서 기본 프로세스의 경우와 동일하다. 하지만 출력인수의 경우는 다른데, 단순 프로세스에서의 출력값은 자체적인 실행을 통해 도출될 수 없기 때문에, 기본 서비스의 변환과정에서와 같이, 변환된 JAM 계획에서 목표로 설정될 수 없다. 그러므로 단순 프로세스에서의 출력값들은 단순 프로세스를 대체할 수 있는 구체화된 프로세스들로부터 가져올 수 있어야 한다. 다른 프로세스의 수행으로부터 도출된 결과를 가져오는 과정은 JAM 계획에서 하부목표 달성과정을 통해서 이루어질 수 있다. 결국 입출력 인수들을 갖는 단순 프로세스를 JAM 계획으로 변환하는 과정은, 먼저 입력인수 값들을 구하기 위한 하부목표 달성구문을 기술하고, 이들을 이용하여 출력인수 값들에 대한 하부목표 달성구문을 추가로 기술하는 것이다.

자동화된 서비스 조합을 이루기 위해서 변환될 JAM 계획은 단순 프로세스를 구성하는 인수들로부터 유도되어야 한다. 단순 프로세스를 구성하는 인수들 중에서 입출력 인수 정보를 제외하면, 프로세스의 이름정보가 남는다. 또한 단순 프로세스가 대체로 조합 설계 시에 임의적으로 작성되고 이용되는 점을 감안할 때, 이를 변환시킨 JAM 계획이 자동화된 추론과정을 통해서 발견되고, 호출되도록 하기는 대단히 어렵다. 즉 단순 프로세스는 추론과정을 통해 자동으로 호출될 것을 고려하지 않고, 주어진 명세에 대해 구체화된 프로세스들로 대체시킬 목적으로 작성된 프로세스이다. 이를 종합해 보면, 단순 프로세스는 서비스 조합과정에 사용되는 서비스라기보다는, 사용자의 요구사항이 표현된 OWL-S 명세로부터 유도되어, 조합을 구현하기 위한 첫 단계로 이용될 수 있는 프로세스라고 말할 수 있다. 그러므로 단순 프로세스에 명기된 프로세스의 이름정보는 JAM 계획의 목표로 설정될 수 있다. 프로세스의 이름을 변환될 JAM 계획의 목표로 설정하는 방법은, 입출력 인수들을 갖지 않고 대체될 프로세스만이 기술된 경우에도 적용될 수 있다. 이때 JAM 계획 내부에는 대체될 프로세스만을 하부목표로 달성하는 구문을 기술하게 된다.

JAM은 이와 같은 단순 프로세스에 대응할 수 있는 계획 형태로서 추상 계획을 제공하며, 이를 이용한 단순 서비스

〈표 3〉 단순 서비스 명세 변환

■ **조건1**: Q가 I,O,P,E를 가질 때
- **입력**: Q에 대한 OWL-S 정의
- **출력**: P
- **절차**:
  (1) Q의 이름을 목표 정보 G로 정의한다.
  (2) I와 C에 대하여
    A. I와 C가 모두 존재하며, 서로 독립인 경우, 각각 SG_I, SG_C로 정의한다.
    B. I와 C가 모두 존재하며, 서로 독립이 아닌 경우, SG_I를 정의한다 (SG_C정의에 관하여는 사용자 처리 요구).
    C. I 또는 C만 존재하는 경우, SG_I 또는 SG_C 만 정의한다.
  (3) (2)를 A1에 기술한다.
  (4) O와 E에 대하여
    A. O와 E가 모두 존재하며, 서로 독립인 경우, 각각 G_O와 G_E로 정의한다.
    B. O와 E가 모두 존재하며, 서로 독립이 아닌 경우, G_O를 정의한다(G_E정의에 관하여는 사용자 처리 요구).
    C. O또는 E만 존재하는 경우, G_O또는 G_E만 정의한다.
  (5) (4)를 A2에 기술한다.
  (6) M = WHEN: TEST(A1의 결과가 참인 조건){A2;};
  (7) M을 P부에 추가한다.

■ **조건2**: Q가 I,O,P,E를 갖지 않을 때
- **입력**: Q에 대한 OWL-S정의
- **출력**: P
- **절차**:
  (1) Q의 이름을 목표 정보 G로 정의한다.
  (2) Q 내에서 Q를 구체화시키는 프로세스 Q'에 대하여,
  Q'를 달성하기 위한 하부목표를 SG'로 정의한다.
  (3) SG'를 달성하기 위한 구문을 A3에 기술한다.
  (4) A3를 P에 추가한다.

```
plan:{
NAME: "FindCheaperBookPrice"
GOAL: FindCheaperBookPrice_known "True";(d)
PRECONDITION:
        RETRIEVE FindCheaperBookPrice_known $known;
        (== $known "False");
BODY:
        DO{
                OR{
                        ACHIEVE BookName_known "True";
                        RETRIEVE BookName_known $known;
                }
                {
                        EXECUTE println "Subgoaling Error!";
                };
        } WHILE: TEST(!= $known "True");
    RETRIEVE BookName $bookname;
    WHEN: TEST(!= $bookname "Unknown")     {
        DO{
            OR{
                ACHIEVE FindCheaperBookProcess_known "True";
                RETRIEVE FindCheaperBookProcess_known $known;
            }
            {
                EXECUTE println "Subgoaling Error!";
            };
        } WHILE: TEST(!= $known "True");
    };
EFFECTS:
        EXECUTE com.irs.jam.primitives.GetCurrentPlanName.execute $service_name $planList;
        UPDATE (FindCheaperBookPrice_known) (FindCheaperBookPrice_known "True");
}
```

(e)

(f)

(그림 9) FindCheaperBookPrice.jam

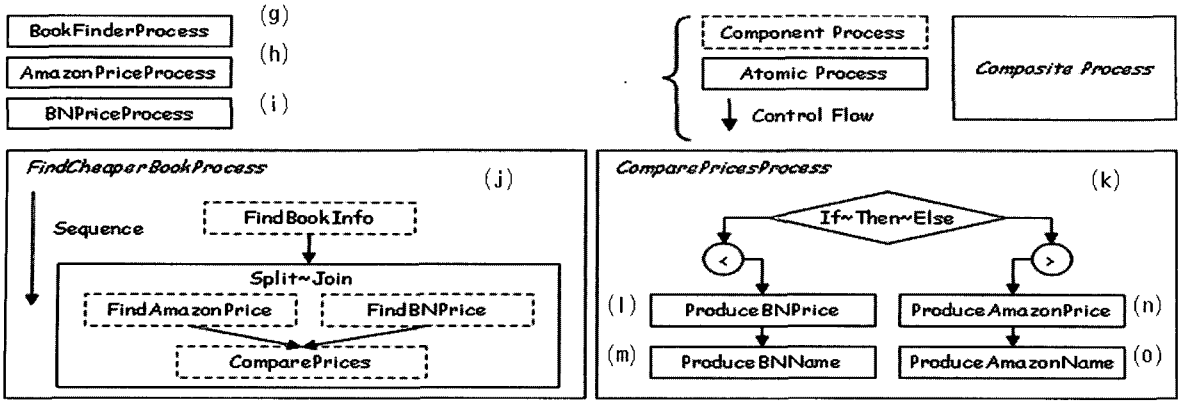
의 변환 알고리즘을 기술하면 다음과 같다. 우리는 <표 3>에서 단순 프로세스를 구체화시키는 프로세스가 단일한 기본 프로세스 또는 복합 프로세스인 경우만을 가정한다. 이때 Q는 임의의 OWL-S 서비스인 W의 단순 프로세스이며, P는 추상 계획, G는 목표정보, SG는 하부목표정보를 의미한다. 그리고 A1,A2,A3는 임의의 JAM 행위이며, M은 JAM의 복합 행위 구조이다.

<표 3>의 단순 서비스 명세 변환 알고리즘을 이용하여 (그림 8)을 JAM 계획으로 변환시킨 결과는 다음과 같다.

(그림 9)에서 (d)는 FindCheaperBookPrice 서비스의 이름으로부터 유도된 추상 계획의 목표를 나타낸다. (e)와 (f)는 FindCheaperBookPrice 서비스가 갖는 입력요소와 출력요소를 각각 하부목표 달성과정을 통해서 구하는 구문들이다.

4.3 복합 서비스 변환

복합 서비스를 효과적으로 JAM 계획으로 변환시키기 위



(그림 10) FindCheaperBookProcess 제어 구조

```
Plan: {
NAME: "FindCheaperBookProcess"
GOAL: ACHIEVE FindCheaperBookProcess_known "True";
PRECONDITION:
    RETRIEVE FindCheaperBookProcess_known $known;
    (== $known "False");
BODY:
    EXECUTE getCurrentPlan $plan;

    ASSIGN $service_name "FindCheaperBook";
    ASSIGN $sub_goal0 "FindBookInfo_known";
    ASSIGN $sub_goal1 "FindAmazonPrice_known";
    ASSIGN $sub_goal2 "FindBNPrice_known";
    ASSIGN $sub_goal3 "ComparePrices_known";

    ACHIEVE FindBookInfo_known "True";
    AND{
        PARALLEL{
            DO{
                OR{
                    ACHIEVE FindAmazonPrice_known "True";
                    RETRIEVVE FindAmazonPrice_known $known;
                }
                {   EXECUTE println "Subgoaling Error!";
                };
            } WHILE: TEST(!= $known "True");
            RETRIEVE BookPrice $CP_AmazonPrice;
            ASSERT CP_AmazonPrice $CP_AmazonPrice;          (p)
        }
        {
            DO{
                OR{
                    ACHIEVE FindBNPrice_known "True";
                    ...

                ASSERT CP_BNPrice $CP_BNPrice;
            };
        }
        {
            DO{
                OR{
                    ACHIEVE ComparePrices_known "True";
                    ...
                                                        (q)
                RETRIEVE CP_BookStore $BookStore;
                RETRIEVE CP_BookPrice $BookPrice;
            };

            UPDATE BookStore $BookStore;
            UPDATE BookPrice $BookPrice;
            EXECUTE println "BookStore is " $BookStore "and BookPrice is " $BookPrice ".";
EFFECTS:
    EXECUTE com.irs.jam.primitives.GetCurrentPlanName.execute $service_name $planList;
    UPDATE (FindCheaperBookProcess_known) (FindCheaperBookProcess_known "True");
}
```

(그림 11) FindCheaperBookProcess.jam

해서는, 기본 서비스 변환과정을 고려하면서, 복합 서비스를 구성하기 위해 사용되는 프로세스들과 실제의 실행을 위해 그들이 호출하는 서비스(또는 프로세스)들, 그리고 구성된 프로세스들 간의 제어 구조를 분석해 볼 필요가 있다. (그림 4)의 주 프로세스 FindCheaperBookProcess은 (그림 10)과 같은 구성 요소 프로세스들과 제어 구조로 구성되어 있다.

(g)(h)(i)는 별도로 존재하는 기본 서비스의 프로세스들을 나타내며, (j)(k)는 복합 서비스를 나타낸다. (j)는 서비스를 이루는 주 프로세스이며 Sequence로 2개의 블록이 연결되어 있다. 첫 번째 블록은 FindBookInfo 프로세스로 구성되고, 두번째 블록은 프로세스들(FindAmazonPrice, FindBNPrice, ComparePrices) 간에 Split~Join으로 구성되어 있다. 이때 FindBookInfo, FindAmazonPrice, FindBNPrice는 메인 프로세스를 구성하는 역할과 함께 실제의 외부 서비스를 호출하는 역할을 수행한다. 그리고 ComparePrices 프로세스는 다시 내부의 ComparePricesProcess를 호출하는데, 이때 ComparePrices Process는 다시 BN 관련 서비스들((l)(m))과 Amazon 관련 서비스들((n)(o)) 간에 If~Then~Else으로 구성되어 있어서, 선택적으로 한쪽의 서비스를 수행한다. 이때 (l)(m)(n)(o)는 ComparePricesProcess의 출력을 담당하는 프로세스들로서, 외부의 서비스를 호출하지 않고 프로세스 내부에 직접 실행 구문을 포함하고 있다. 그러므로 이들 모두를 대상으로 계획수립을 위한 원시 계획들로의 변환과정이 요구된다.

(그림 10)에서 표현된 것들 외에도 OWL-S가 정의하고 있는 제어구조들로는 Split, Unordered, Choice, Iterate, Repeat-Until, Repeat-While이 있으며, 이와 같은 제어 구조들로 표현된 복합 서비스 역시 계획명세 내에서 동일한 수준의 기능을 갖도록 표현되어야 한다. 본 논문에서 제안하는 반응형 계획기법인 JAM은 복합 행위들을 기술하기 위한

〈표 4〉 복합 서비스의 제어구조 변환

(1) 제어 구조 Sequence의 변환
- 입력: Sequence에 의해 차례로 연결된 요소 프로세스 P1, P2에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 절차: P1에 대하여 변환된 JAM 행위를 A1, P2에 대하여 변환된 JAM 행위를 A2라 할 때,
  M = {A1;A2;}

(2) 제어 구조 Split의 변환
- 입력: Split에 의해 차례로 연결된 요소 프로세스 P1, P2에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 절차: P1에 대하여 변환된 JAM 행위를 A1, P2에 대하여 변환된 JAM 행위를 A2라 할 때,
  M = PARALLEL{A1;}{A2;};

(3) 제어 구조 Split+Join의 변환
- 입력: Split에 의해 차례로 연결된 요소 프로세스 P1, P2와 Join에 의해 연결된 프로세스 P3에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 절차: P1에 대하여 변환된 JAM 행위를 A1, P2에 대하여 변환된 JAM 행위를 A2, P3에 대하여 변환된 JAM 행위를 A3라 할 때,
  M = AND{ PARALLEL{A1;}{A2;}}{A3;};

(4) 제어 구조 Unordered의 변환
- 입력: Unordered에 의해 차례로 연결된 요소 프로세스 P1, P2에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 절차: P1에 대하여 변환된 JAM 행위를 A1, P2에 대하여 변환된 JAM 행위를 A2라 할 때,
  M =DO_ALL{A1;}{A2;};

(5) 제어 구조 Choice의 변환
- 입력: Choice에 의해 차례로 연결된 요소 프로세스 P1, P2에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 절차: P1에 대하여 변환된 JAM 행위를 A1, P2에 대하여 변환된 JAM 행위를 A2라 할 때,
  M = DO_ANY{A1;}{A2;};

(6) 제어 구조 If-Then-Else의 변환
- 입력: If조건이 참인 경우에 대하여 수행 대상이 되는 프로세스 P1과, 거짓인 경우에 대하여 수행 대상이 되는 프로세스를 P2에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 절차: P1에 대하여 변환된 JAM 행위를 A1, P2에 대하여 변환된 JAM 행위를 A2라 할 때,
  M = WHEN: TEST(참인 조건){A1;}; TEST(거짓인 조건){A2;};

(7) 제어 구조 Iterate의 변환
- 입력: Iterate에 의해 차례로 연결된 요소 프로세스 P1에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 절차: P1에 대하여 변환된 JAM 행위를 A1이라 할 때,
  M = WHILE: {A1;};

(8) 제어 구조 Repeat-While, Until의 변환
- 입력: While 또는 Until조건이 참인 경우에 대하여 수행 대상이 되는 프로세스 P1에 대한 OWL-S 프로세스 정의
- 출력: JAM의 복합 행위 구조인 M
- 함수: P1에 대하여 변환된 JAM 행위를 A1이라 할 때,
  M = DO{A1;} WHILE:TEST(참인 조건);

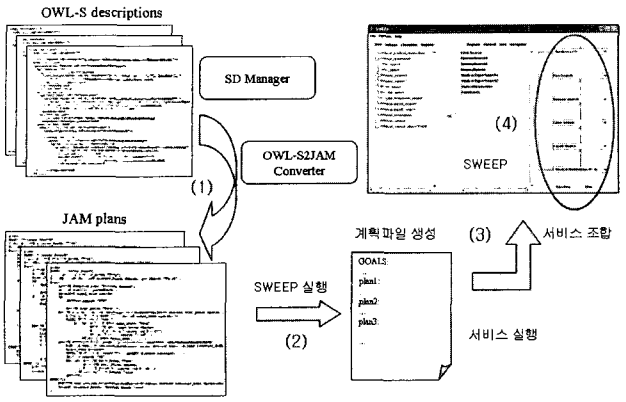
풍부한 제어 구조를 지원한다. 이들을 열거하면, AND, DO_WHILE, DO_ALL, DO_ANY, PARALLEL, TEST, WHEN, WHILE가 있다. 그러므로 이들을 이용하여, OWL-S의 제어구조들을 JAM 계획의 제어구조로 변환하기 위한 알고리즘을 정리하면 다음과 같다.

<표 4>의 복합 서비스의 제어구조 변환 알고리즘을 이용하여 (그림 4)를 JAM 계획으로 변환시킨 결과는 (그림 11)과 같다.

(p)는 아마존 도서가격 검색서비스에 의해 구해진 도서가격정보를 BN도서가격검색 서비스의 출력결과와 구분하여, 가격비교를 위한 프로세스에서 이용된다. (q)는 가격비교 프로세스 수행을 통해 최종 출력될 도서가격과 사이트명을 나타내기 위한 변수로 저장하는 구문을 나타낸다.

4.4 조합과 실행

조합된 결과로서의 OWL-S 서비스는 (그림 4)와 같은 명세를 가질 수 있다. 이때 OWL-S 서비스는 사용자의 요구에 맞는 결과를 제공할 수 있도록 작성된다. 사용자의 요구 입력 방식을 살펴보면, 사용자의 요구를 처리하기 위한 별도의 처리기 또는 처리 언어를 이용하는 방식과 OWL-S로 기술된 사용자 요구를 이용하는 방식 등이 있다. 본 논문에서는 후자의 경우가 보다 더 시맨틱 웹서비스 기술의 취지



(그림 12) 조합과 실행 과정

에 적합한 것으로 가정한다. 조합을 구현하기 위해, 변환된 JAM 계획들로부터 각각의 계획들이 달성할 수 있는 목표들이 제공되면, 사용자가 제공한 OWL-S 서비스 기술명세와의 매치메이킹(Matchmaking) 작업을 통해, 목표를 나타내는 문자열 값은 계획수립을 위한 전체 목표로 설정된다. 이후 정해진 목표와 변환된 JAM 계획들, 그리고 이들로부터 제공되는 각종 지식 정보들을 이용하는 계획수립 과정이 시작된다.

기본적으로 반응형 계획기법들은 실행을 통해 계획을 작성하는 특징을 갖는다. 즉 전체 목표를 달성하기 위한 재료계획들을 하나씩 실행해 나가면서, 정상적으로 수행된 계획들만을 최종 수립된 계획에 포함시키는 과정을 따른다.

이를 과정별로 설명하면, (그림 12)에서와 같이 조합 작업이 시작되면 OWL-S 서비스들로부터 변환된 JAM 계획들로부터 추출된 목표정보들과 사용자에 의해 제공된 OWL-S 기술과의 매치메이킹 과정을 통해, 사용자의 목표가 전체 목표로 지정된 계획파일(a script file)이 작성된다. 실제적으로 계획수립 과정은 계획파일의 실행으로부터 이루어지는데, JAM 계획기에 의해 정해진 전체 목표를 자신의 목표로 갖는 JAM 계획이 먼저 선택된다. 이후 JAM 계획기는 해당 계획의 BODY부에 기술된 행위들을 차례로 수행하는데, 필요에 따라 하부목표 달성과정도 수행한다. 만약 행위 수행 중 에러나 오류 상황 등이 발생하면, JAM 계획기는 수행중인 목표를 달성할 수 있는 다른 계획을 선택한다. BODY부에 기술된 행위들을 정상적으로 마친 JAM 계획은 EFFECTS부에 기술된 바에 따라 현재 계획으로부터 도출된 결과 정보들을 지식정보에 기록하고 계획구성 행위를 수행한다. 이때의 계획구성 행위는 현재의 계획이 전체 목표 달성을 위해 수립된 계획의 일부로서, 구성되었음이 기록되도록 한다. 이러한 과정을 통해 실행되고 기록된 JAM 계획들은 전체 목표 달성을 위한 계획을 구성하게 되며, 사용자에게 리스트 형태로 제공될 수 있다.

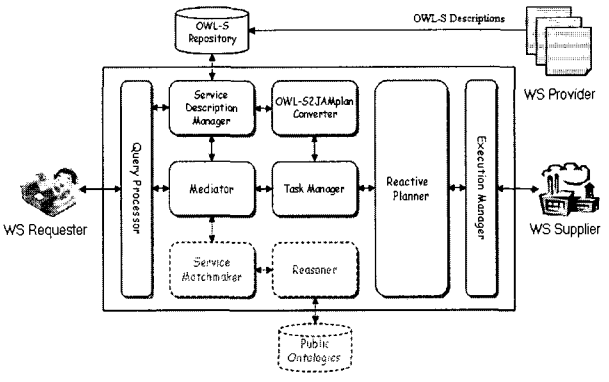
이와 같은 과정을 통해, 반응형 계획기법은 이전에 수립된 계획을 사용할 수 없는 상황에서, 사전에 알지 못했던 문제나 환경의 변화에 의해 제기된 문제들에 반응하여 동적으로 계획을 재수립할 수 있는 능력을 갖는다.

그러나 반응형 계획기법은 기술의 특성상, 계획수립의 결과로서 행위들의 연속된 리스트, 즉 실행을 통해 선택된 구성 계획들에 대한 순차화된 리스트 정보만을 제공한다는 한계를 갖는다. 이러한 한계는 계획기법을 이용한 대부분의 연구들에서 동일하게 발생하는 한계이며, 추후 연구과정을 통해 해결되어야 할 문제이다.

5. 구현 및 실험

5.1 시스템 구조

상기 기술한 내용에 따라, 우리는 시맨틱 웹서비스 조합기로서 (그림 13)과 같은 SWEEP 시스템을 설계하였다. SWEEP은 핵심 구성요소로서 엔진역할을 수행하는 반응형 계획기(Reactive Planner)와, 웹서비스 명세들로부터 자동으로 단위 계획 명세를 생성하는 OWL-S2JAM 계획 변환기(OWL-S2JAMplan Converter)를 가진다. 또한 서비스 검색 및 조합작업을 지원하기 위해 OWL-S로부터 추상화된 정보를 추출하여 저장하는 서비스 명세 관리기(Service Description Manager)와 온톨로지 정보를 대상으로 추론 결과를 돌려주는 추론기(Reasoner), 추론 결과를 이용하여 서비스의 입출력 인수들 간에 매칭 결과를 알려주는 서비스 연결기(Service Matchmaker)가 있다. 그리고 이 외에도 질의 처리기와 여러 모듈들 간에 연결을 담당하는 중개 처리자(Mediator), 사용자의 요구 목표를 전달하고 처리 결과를 돌려주는 작업 관리자(Task Manager), 반응형 계획기를 통해 실행된 서비스들의 상태를 관리하는 실행 관리기(Execution Manager) 등으로 구성된다.

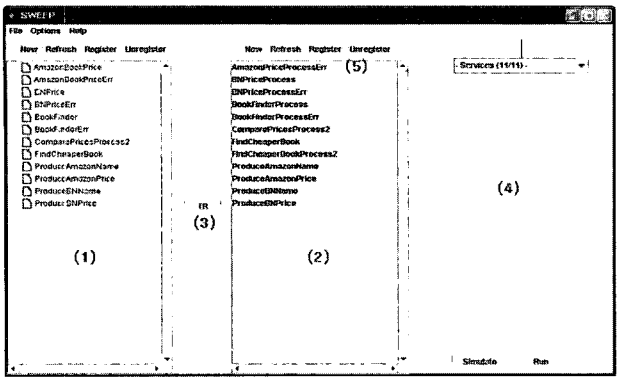


(그림 13) SWEEP 시스템의 구조

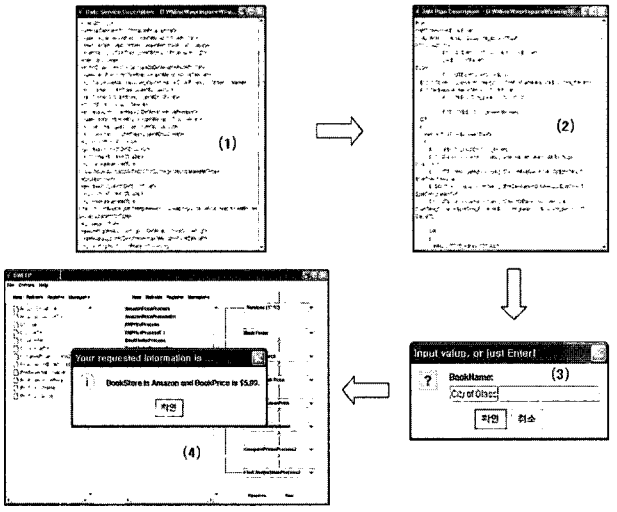
5.2 구현 시스템

SWEEP 은 JAVA1.4.2과 OWL-S API 1.1.0 beta, 그리고 JAM 0.65+0.76i 등을 이용하여 구현하였고, (그림 14)는 SWEEP 의 초기화면을 나타낸 것이다.

(그림 14)에서 (1)은 등록된 OWL-S 서비스들의 리스트를 나타낸다. (2)는 (1)의 서비스 명세파일로부터 JAM 계획으로 변환된 계획들의 리스트정보이며, 이때 (3)의 변환버튼 동작을 통해 OWL-S2JAM 계획 변환기가 동작하게 된다.



(그림 14) SWEEP 초기 화면



(그림 15) 조합 실행 결과

(4)는 사용자의 목표 설정 작업을 수행하고, 설정된 목표에 따라 실제로 조합과정을 보여주는 패널이다. (5)는 (1)또는 (2)에 등록된 서비스 파일들과 계획 파일들을 등록하거나 삭제하는 메뉴를 나타낸다.

5.3 실험

제안된 접근 방법의 적용 가능성과 효과를 알아보기 위해, 우리는 [19]에서 제공되는 예제 웹서비스들을 이용하여, 서비스들의 명세 변환과 시맨틱 웹서비스 조합을 실험하였다. 실제로 SWEEP를 실행하여 조합을 구현한 결과는 (그림 15)의 (4)에서 나타내었다. 이때 (그림 15)의 (1)은 등록된 웹서비스 중의 하나인 AmazonBookPrice.owl의 명세를 팝업 창으로 나타낸 것이며, (2)는 변환된 JAM 계획인 AmazonBookPrice.jam의 명세를 나타낸다.

SWEEP을 실행하게 되면, 우선 (그림 14)의 (2)에 등록된 JAM 계획들을 대상으로 하여, 계획 수립을 위한 계획파일이 자동으로 생성된다. 이때 각각의 JAM 계획들로부터 읽혀진 서비스 정보들은 (그림 14)-(4)의 콤보박스 리스트에 등록되어, 사용자 선택을 통해 목표가 설정된다. 설정된 목표를 대상으로 하여 실행 명령((그림 14)-(4)의 하단부 Run 버튼)을 내리면, JAM의 계획 수립 과정을 통해 웹서비스 조합이 구현된다. 실행 과정에서 사용자 정보의 입력을 요청하는 화면이 (3)과 같이 주어지면 사용자는 해당하는 정보를 입력하게 되고, 최종적으로 (4)와 같이 구해진 정보 및 조합된 결과를 얻게 된다.

서비스 명세변환을 실험하기 위해 최저 도서가격 검색 서비스를 제공하는 명세를 입력으로 받아 실험을 진행하였으며, 결과로서 아래와 같은 JAM 계획 파일들이 생성되었다. 이를 통해 하나의 복합 서비스가 입력되면, 서비스를 구성하는 모든 프로세스들이 JAM 계획수립을 위한 계획들로 변환되어야 함을 나타낸다.

<표 5>에서, 우리는 (그림 10)의 (a)~(i)에 대해 기본 계획들과 복합 계획들로, 그리고 FindBookInfo, FindBNPrice, FindAmazonBookPrice, ComparePrices에 대하여 추상 계획들로 변환된 결과를 나타내었다. 전자의 경우 각각의 프로세스들은 어떠한 형태로든 내부에 직접 함수를 호출하거나, 실행 구문을 포함한다. 그러나 후자의 경우는 아무런 실행 구문을 갖지 않고, 단순히 외부의 프로세스나 내부의 구체화된 다른 프로세스를 호출하는 구문만을 포함한다. 그러므로 우리는 이와 같은 프로세스들을 4.3절에서 기술한 바에 따라, (그림 16)에서 단순 프로세스로 인식하고, 기본 계획으로의 대체될 계획들(Find BookInfo.jam, FindBNPrice.jam, FindAmazonBookPrice.jam)과 복합 계획으로의 대체될 계획들(Compare Prices.jam)을 도출하도록 하였다.

시맨틱 웹서비스 조합을 실험하기 위해 우리는 전자의 실험에서 사용된 기본 계획들과 직접 기본 서비스를 호출하도록 수정한 ComparePricesProcess2.jam, 그리고 에러 상황을 내재시킨 계획들(예. BookFinderProcessErr.jam, BNPrice ProcessErr.jam, AmazonBookPrice ProcessErr.jam), 총 11개의 계획을 이용하였다. 그리고 사용자의 목표 선택과정을 통해 FindCheaperBookPrice을 목표로 설정하여 실험을 진행하였다.

실험 결과를 통해, 반응형 계획을 이용한 조합 도구인

〈표 5〉 최저 도서가격 검색 서비스를 위해 생성된 JAM 계획 리스트

복합 계획	기본 계획	추상 계획
FindCheaperBookProcess.jam ComparePricesProcess.jam	BookFinderProcess.jam BNPriceProcess.jam AmazonPriceProcess.jam ProduceBNPrice.jam ProduceBNName.jam ProduceAmazonPrice.jam ProduceAmazonName.jam	FindBookInfo.jam FindBNPrice.jam FindAmazonBookPrice.jam ComparePrices.jam

```
plan:{
NAME: "FindBookInfo"
GOAL: FindBookInfo_known "true";
PRECONDITION:
    RETRIEVE FindBookInfo_known $known;
    (== $known "False");
BODY:
    EXECUTE getCurrentPlan $plan;
    ACHIEVE BookInfo_known "true";
EFFECTS:
    EXECUTE com.irs.jam.primitives.GetCurrentPlanName.execute $service_name $planList;
    UPDATE (FindBookInfo_known) (FindBookInfo_known "True");
}
```

(그림 16) FindBookInfo.jam

SWEEP이 효과적으로 OWL-S가 제공하는 서비스 온톨로지 명세들을 JAM 계획 명세로 변환시키는 것을 알 수 있었다. 또한 에러 상황에서 동적으로 재계획과정을 진행함으로써, 견고하게 전체 계획을 수립하고 실행함을 알 수 있었다. 반면에 SWEEP은 이와 같은 장점들 외에 몇 가지 한계점들을 가진다. 첫째로 JAM의 한계로 인해, 온톨로지가 정의하고 있는 여러 개념들 간의 의미와 관계를 충분히 반영하지 못한다. 둘째로 사용자의 요구가 기술된 OWL-S 명세를 이용하지 못한다. 셋째로 OWL-S 명세를 갖는 조합된 결과를 생성하지 못하고, 구성된 계획들의 순차화된 리스트 정보만을 제공한다. 이와 같은 문제점들은 추후 연구과정을 통해 반드시 극복되어야 할 과제이다.

6. 결 론

우리는 본 논문에서 자동화된 웹서비스 조합을 위한 계획 기술로서 전통적인 인공지능 계획기법들의 문제점을 제기하고, 반응형 계획기법을 이용한 방법을 제안하였다. 그리고 제기된 문제점들을 대상으로 반응형 계획기법을 이용한 해결 방법을 기술하였다.

우선적으로 복합 제어구조를 제대로 표현하지 못하는 한계에 대하여, OWL-S로 표현되는 기본 서비스, 단순 서비스, 복합 서비스를 효과적으로 표현하고 처리할 수 있었다. 복합 서비스를 제대로 표현하고 계획수립을 위해 이용하기 위해서는 복합 서비스가 담고 있는 복합 제어구조를 얼마나 제대로 잘 변환할 수 있는가가 관건이다. 이에 대해 우리는 반응형 계획기 중의 하나인, JAM의 복합 행위를 표현하기 위한 구조들을 이용하여 기술 명세들에 대한 변환 알고리즘을 작성하였고, 변환된 예를 기술하였다. 다만 복합 제어구조로 연결되는 프로세스들 간의 데이터 흐름 처리에 대해서는 아직까지 완전하게 자동적이지는 못하다. 이는 계속적으로 명세가 바뀌어가고 있는 OWL-S로 인한 문제와 더불어, 계획 기법들이 주로 상태정보를 위주로 계획을 수립하는 기계적 원리만을 따르는 한계에서 기인하는 것으로 예측된다. 두 번째로 풍부한 데이터 타입을 사용하지 못하는 한계에 대하여, 본 논문에서는 아직까지 단순 타입들에 대해서만 정의하고 이용할 수 있었다. 추후 연구를 통해 개선되어야 할 부분이다. 세 번째로 동적 객체 처리를 지원하지 못하는 한계에 대하여 본 논문에서는 반응형 계획기법을 이용함으로써, 동적으로 입출력 되는 데이터들을 효과적으로 처리할 수 있었다. 이는 주로 정보획득 작업을 수행하는 웹서비스들을 대상으로 하여 효과적인 계획수립을 지원할 수 있는 기반이 되는 기능으로 사용될 수 있다. 네 번째로 실시간 조합과 처리를 못하는 한계에 대하여, 반응형 계획기를 이용한 조합 방법은 계획수립과 실행을 동시에 병행하는 특징과 환경의 변화를 인식하고 이를 계획과정에 반영하는 기능을 통해, 불확실성이 내재된 실시간 에러 환경에서도 견고하게 동작하였다.

그럼에도 불구하고 최근 들어 전통적인 계획기법들의 한계점을 보완하려는 많은 연구들이 수행되었는데, 제안한 방법과 비교하여 이들을 간략하게 설명하면 다음과 같다.

첫 번째로, 복합 제어구조 표현이라는 문제에 대하여 최근의 기법들은 PDDL을 이용함으로써, 다양한 제어구조들을 표현할 수 있다. 즉, 반복, 선택, 병렬과 같은 다양한 제어구조로 표현된 복합 프로세스들을 표현할 수 있다. 그러나 JAM에서 제공하는 수준의 다양한 복합 제어구조들을 표현해 내기에는 다소 어려움이 있다. 두 번째로, 풍부한 데이터 타입 사용이라는 문제에 대하여, 최근의 기법들은 전통적인 계획기법들이 제공하는 타입 정의 수준을 넘어, 훨씬 복잡한 수준의 타입을 정의할 수 있게 되었다. 이를 통해 시맨틱 웹서비스의 인터페이스를 구성하는 용어들을 정의하고 있는 온톨로지 정보들을 다소 제한된 범위에서 표현하고 이용할 수 있게 되었다. 하지만 JAM의 경우 아직까지 단순한 수준의 타입만을 정의함으로 인해, 추후 다양한 수준의 타입 정보를 사용할 수 있는 반응형 계획기를 고려해 볼 필요가 있다. 세 번째로, 동적 객체 처리라는 문제에 대하여 전통적인 계획 기술에 기반을 둔 기법들은 계획 실행을 통해야만 알 수 있는 값들을 미리 정의하고 처리한다는 것은 본질적으로 불가능하다. 그러나 계획 단계에서 사전 실행과정을 통해서, 미리 이러한 정보들을 알아내고, 이를 계획 단계에 반영하려고 하는 연구들이 점차로 많이 이루어지고 있는 추세이다. JAM은 임의의 변수들을 정의하고 이용함으로써, 이와 같은 문제에 대해 어느 정도의 해결책을 제공할 수 있으나, OWL-S 서비스가 제공하는 수준만큼의 다양한 객체 처리에는 한계가 있다. 네 번째로, 실시간 계획수립과 실행이라는 문제에 대하여 전통적인 계획 기술 기반의 기법들은

계획 단계와 실행 단계가 분리되어 있음으로 인해, 실시간으로 실세계의 지식 상태를 반영한 계획을 수립할 수 없다. 이와 관련하여 조건적 계획기법(Conditional Planning) 및 의존적 계획기법(Contingent Planning)들이 소개되어, 계획이 실행될 당시의 상황을 고려하여 미리 여러 가능성을 고려하여 계획을 작성하거나, 실행 상황을 고려해 가면서 계획을 작성하려는 연구들이 점차로 많이 수행되고 있다. 그러나 JAM은 계획기 자체가 실행과정을 통해 정상적으로 완료된 원시 계획들만을 대상으로 계획수립 작업을 진행하는 기능과, 실행 후 결과 상태에 대한 인지기능(sensing) 및 이의 반영을 통해 실시간적으로 외부 세계의 변화를 고려한 계획을 수립할 수 있다.

이와 같은 문제점들을 극복하기 위한 방안을 구체화시키기 위한 접근 방법으로서, 본 논문에서는 프로토타입 시스템인 SWEEP을 설계하고 구현하였다. 그리고 SWEEP을 이용하여 e-Commerce 분야의 실재하고 있는 예제 웹서비스들을 대상으로 실험하였다. 실험은 OWL-S로 기술된 웹서비스 기술명세들을 반응형 계획수립을 위한 계획 명세들로 변환하는 과정과 변환된 계획들을 이용한 조합 과정에 대하여 진행하였다. 실험 결과를 통해 불확실성을 가진 웹서비스 환경에서 안정적이고 견고한 조합을 구현할 수 있었다. 결과적으로 본 연구는 전통적인 인공지능 계획 기술이 안고 있는 한계를 일부 극복하고자 한 시도임과 동시에, 현실적인 면을 감안하여 웹서비스 조합을 이루기 위해 적절한 방법을 제안한 연구라고 할 수 있다.

참 고 문 헌

[1] A. G. Hernandez, A. El Fallah-Seghrouchni and H. Soldano, "Learning on BDI Multi-agent Systems", Computational Logic in Multi-Agent Systems Fort Lauderdale, USA, Jan 6-7, 2004.

[2] B. Srivastava, J. Koehler, "Planning with Workflows-An Emerging Paradigm for Web Services Composition", In ICAPS 2004 Workshop on Planning and Scheduling for Grid and Web Services, Whistler, British Columbia, Canda, June. 2004.

[3] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. Mcllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "Owl-s: Semantic markup for web services," http://www.daml.org/services/owl-s/1.1/overview/, 2004.

[4] J. Rao and X. Su., "A Survey of Automated Web Service Composition Methods", In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, San Diego, California, USA, July.6, 2004.

[5] Joachim Peer, "Web Service Composition as AI Planning - a Survey", Technical Report, Univ. of St.Gallen, 2005.

[6] Koehler, J., Srivastava, B., "Web Service Composition - Current Solutions and Open Problems", In ICAPS 2003 Workshop on Planning for Web Services, 2003.

[7] Kuter, Ugur and Sirin, Evren and Nau, Dana and Parsia, Bijan and Hendler, James, Whistler, "Information gathering during planning for web service composition," In Workshop on planning and scheduling for web and grid services at icaps04, Canada, 2004.

[8] Lavindra de Silva and Lin Padgham, "Planning as needed in BDI Systems," Proceedings of ICAPS 2005, Monterey, California, June. 2005.

[9] M. J. Huber, "Jam: A BDI-theoretic mobile agent architecture," in Agents, pp.236-243, 1999.

[10] M. Klusch, A. Gerber, and M. Schmidt, "Semantic Web Service Composition Planning with OWLS-Xplan," First International Symposium on Agents and the Semantic Web, 2005.

[11] M. Solanki and C. Abela, "The Landscape of Markup Languages for Web Service Composition," May. 2003.

[12] P. Godefroid and F. Kabanza, "An Efficient Reactive Planner for Synthesizing Reactive Plans," In Proc. of AAAI-91, pp.640-645, Anaheim, CA, 1991.

[13] R. Jensen and M. Veloso, "Interleaving deliberative and reactive planning in dynamic multi-agent domains," In Proceedings of the AAAI Fall Symposium on Integrated Planning for Autonomous Agent Architectures. AAAI Press, 1998.

[14] Rubén Lara, Dumitru Roman, Axel Polleres, Dieter Fensel, "A Conceptual Comparison of WSMO and OWL-S," European Conference on Web Services (ECOWS 2004), Erfurt, Germany, pp.254-269, Sep.27-30, 2004.

[15] Sirin, Evren and Parsia, Bijan and Wu, Dan and Hendler, James and Nau, Dana, "HTN planning for web service composition using SHOP2," Journal of Web Semantics 1 (4), pp.377-396, 2004.

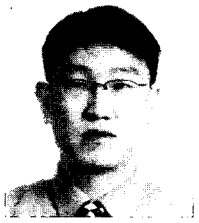
[16] S. McIlraith and T.C. Son, "Adapting golog for composition of semantic web services," In Proc. Of KR-02, Morgan Kaufmann, pp.482-496, 2002.

[17] DAML-S and Related Technologies, http://www.daml.org/services/owl-s/1.0/survey.pdf

[18] Congo.com, http://www.daml.org/services/owl-s/1.1/examples.html

[19] OWL-S Services, http://www.mindswap.org/2004/owl-s/services.shtml

[20] Web Services Architecture W3C Working Group Note 11 Feb 2004, http://www.w3.org/TR/ws-arch/



진 훈

e-mail : bioagent@kyonggi.ac.kr

1998년 경기대학교 전자계산학과(학사)

2000년 경기대학교 대학원 전자계산학과 (이학석사)

2007년 경기대학교 대학원 전자계산학과 (이학박사)

2007년~현재 경기대학교 정보과학부 외래교수

관심분야 : 시맨틱웹서비스, 바이오인포매틱스, 멀티에이전트시스템, 인공지능

김 인 철

e-mail : kic@kyonggi.ac.kr

1985년 서울대학교 계산통계학과(학사)

1987년 서울대학교 대학원 전자계산학과 (이학석사)

1995년 서울대학교 대학원 전자계산학과 (이학박사)

1989년~1995년 경남대학교 전산통계학과 조교수

1993년~현재 경기대학교 전자계산학과 교수

2003년~2004년 미시간 주립대학교 방문교수

관심분야 : 인공지능, 시맨틱 웹서비스, 지능형 에이전트, 지능로봇 등