디자인패턴 기반의 에이전트 지향 개발방법론

이학진[1], 윤현상[2], 이은석[3]
성균관대학교 컴퓨터공학과[1 2 3]
{s9832033[1],wizehack[2], eslee[3]}@selab.skku.ac.kr

Agent Oriented Methodology Based on Design Pattern

Hak Jin Lee[1], Hyun Sang Youn[2], Eun Seok Lee[3]
Dept. of Computer Engineering, Sungkyunkwan University [1 2 3]

요 약

최근 에이전트 기반 시스템을 만들기 위한 기술로서, 에이전트 지향 개발 방법론 (Agent Oriented Methodology)이라는 영역이 출현하여, 여러 연구 기관에서 에이전트 시스템을 만드는데 이를 적용한 성공 사례가 늘고 있다. 그러나, 이들을 가지고 복잡한 다중 에이전트 시스템을 개발하는 일은 여전히 경험이 많지 않은 개발자들에게 있어서 어려운 일에 해당된다.

본 논문에서는 개발자에게 에이전트 설계 시 부과되는 이러한 어려운 점을 대한 해결책으로서, 개발자들에게 기능에 따라 분류된 디자인 패턴을 지원하여 다중 에이전트 시스템 설계 및 개발을 지원하는 개발방법론을 제안한다. 또한, 우리는 요구사항 분석부터 코드 생성까지 개발 프로세스에 반영하는 지원 도구를 제안한다. 이 CASE 도구는 단계별 자동화 기능뿐만 아니라, 설계 과정에서 개발자들에게 디자인 패턴을 고려할 수 있는 기능을 제공한다. 이를 통해, 우리는 에이전트 시스템 개발자 및 에이전트에 대한 지식이 부족한 이들에게 멀티에이전트 시스템 설계와 개발을 보다 쉽고 빠르게 해결할 수 있도록 돕는다.

본 논문에서는 여행 도우미 시나리오(Travel Assistant Scenario)을 제안 개발 방법에 따라 적용시켰다. 최종적으로, 보다 쉬운 방식으로 에이전트 소스 코드를 생성해냈으며, 이를 통해 제안 개발방법론이 개발자들의 다중 에이전트 기반 시스템의 개발에 대한 부하를 줄여 줄 수 있다는 결론을 내렸다.

Keyword : Multiagent system, Agent-oriented Software Engineering, Design pattern

1. 서론

에이전트 기반 시스템은 소프트웨어에 요구되는 동적인 환경에서의 다양한 서비스 제공이라는 측면에서 그 응용 분야가 산업 및 공공부문으로 점차 확대되어가고 있으며[1], 다양하게 사용되고 있다. 또한, 이와 같은 에이전트 기반 시스템을 만들기 위한 개발방법론에 대한 연구도 수년간 활발히 진행되어 왔고, 실제 이를 적용한 많은 성공 사례들이 나타나고 있다[2, 3].

또한, 에이전트 기반 시스템의 개발을 지원하기 위해 ABLE(Agent Building and Learning Environment)[4], JADE(Java Agent DEvelopment Framework)[5] 와 같은 다양한 프레임 워크가 개발되었다. 이들을 이용하여, 에이전트 기반 시스템들의 구현을 쉽게 할 수 있게 되었다.

일반적으로, 에이전트 기반시스템을 개발하기 위해, 경험이 많지 않은 개발자들의 경우, 이러한 개발 방법론을 이용하여 에이전트를 개발할 때 여러 문제에 봉착하게 된다. 일관성이 결여된 설계는 에이전트 시스템의 신뢰도를 떨어뜨릴 수 있다. 이러한 문제들을 해결하기 위해 개발자들에게는 디자인 패턴들이 제공될 필요가 있다.

디자인 패턴 [6]은 소프트웨어 설계 시에 발생하는 공통된 설계 문제에 대해 유용한 해결책들을 기술하고 있으며, 그러한 패턴은 익숙하지 않은 설계자에 의해 이해되고 활용될 수 있다. 본 논문에서 우리는 설계 경험이 적은 개발자들을 지원하는 패턴 기반 개발 방법론을 제안한다. 또한 제안

프로세스 및 방법론에 기초한 개발 도구를 제공하여 에이전트 기반 시스템을 보다 효율적으로 개발할 수 있도록 지원한다.

본 논문은 다음과 같이 구성된다. 우리는 2 장에서 관련 연구를 요약할 것이다. 3 장에서는 본 논문에서 제안한 개발 방법론과 개발 도구를 소개하고 4 장에서 여행 보조자(Traveling Assistant) 시나리오 내에서 에이전트 개발을 위해 우리의 도구를 적용시킨다. 끝으로, 5 장에서는 본 논문에서 소개된 우리의 도구를 정리하고, 향후 연구 분야에 대해 밝힐 것이다.

2. 관련연구

하나의 에이전트는 소프트웨어 공학 영역에서 객체(Object)의 확장이라고 할 수 있다. 에이전트는 객체(Object)의 모든 속성을 가지고 있는데, 자율성 (Autonomy), 지능 (Intelligence), 교섭 (Negotiation) 그리고 이동성 (mobility)과 같은 추가적인 속성까지도 지니고 있다.

2-1 Tropos

Tropos[2]는 에이전트 지향 소프트웨어 시스템을 만들기 위한 개발 방법론으로서, 소프트웨어 개발의 모든 단계(초기 분석으로부터 실제 구현에 이르기 까지)에서 에이전트의 개념과 모든 연관된 인지적 개념 (e.g Goals 과 plans)을 사용한다. 또한, Eric Yu 에 의해 제안된 i* 모델링 프레임 워크를 적용하여, 초기 요구사항 분석을 모델링을 지원하고 있다. i*는 Actor, Goal, 그리고 Actor 의 Dependency 의 세 가지 개념을 제공하며, 이들을 초기/후기 요구사항, 아키텍처 설계, 그리고 상세 설계를 하는데 근본으로써 사용한다. 이러한 특징은 해당 소프트웨어가 동작해야 하는 환경, 소프트웨어와 사용자, 에이전트간에 발생하게 되는 상호작용들에 대해 더 자세히 파악할 수 있도록 도와준다. 그러나 Tropos 는 독자적인 표기법을 이용하여 에이전트를 모델링 하기 때문에 개발자는 추가적으로 표기법을 학습해야 하며, i* 모델링 방식은 표기 형식 자체가 매우 복잡하기 때문에 설계서를 이용한 개발자들간의 의사소통에 지장이 발생한다.

2-2 INGENIAS

INGENIAS [3]는 전체 개발 프로세스 중 요구사항 분석, 설계, 구현을 지원하는 개발 방법론이다. INGENIAS 는 요구사항 분석 단계에서 5 가지 (Environment, Interaction, Organization, Agents, Tasks and goals)의 뷰 포인트(View Point)를 이용하여 요구사항을 분석한다. 설계단계에서는 AUML 을 부분적으로 사용함으로써 다중 에이전트 시스템의 개발을 지원한다. 그리고 마지막으로 JADE [5] 기반의 소스코드를 생성해낸다. 그러나 여전히 익숙하지 않은 표기법으로 개발자들에게 추가적인 학습 부담이 있고, 에이전트 기반의 시스템 개발에 익숙하지 않은 사용자의 경우에는 설계단계에서 발생 가능한 문제점에 대해서 충분한 해결책을 제시하지 못한다.

2-3 디자인 패턴

디자인 패턴 [6]은 실제 소프트웨어 개발에서 중요하면서도 유용하게 인식되어 왔다. 디자인 패턴은 확실한 재사용성과 유지 능력을 목표로 가지고 설계 문제에 대해 입증된 해결책을 기술하고 있다. 객체지향언어를 이용하여 시스템을 개발하는 경우 설계 단계에서 이용할 수 있는 많은 패턴들에 관한 연구가 있어왔다. 또한, 디자인 패턴을 사용하는 일은 에이전트 설계에 있어 유용하게 쓰인다. 본 논문에서 우리는 에이전트 기반의 시스템의 특성에 맞게 설계 단계에서 필요한 패턴의 종류를 정의하고 설계 단계에서 이러한 패턴을 기반으로 에이전트를 개발하는 방법론을 제안하고자 한다.

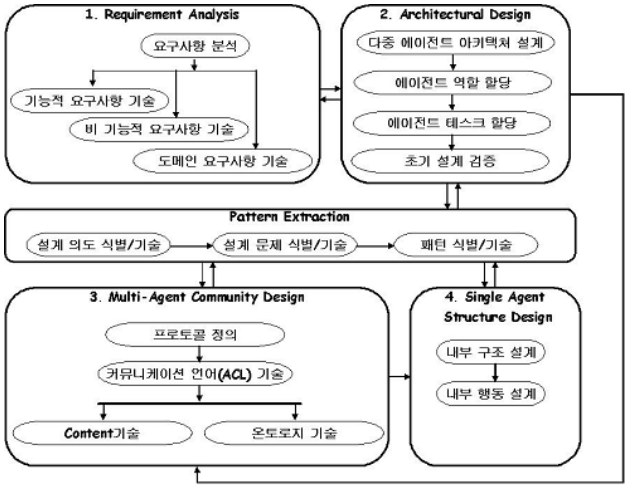
3. 디자인 패턴 기반의 에이전트 개발 방법론

3-1 개발 방법론

본 논문에서 우리는 다중 에이전트 시스템에 관한 개발 방법론을 제안하고 이른 지원하는 지원 도구를 소개할 것이다. 본 논문에서 제안한 방법론을 통하여 개발자는 설계 단계에서 발생 가능한 문제점에 관해 디자인 패턴을 적용함으로 인해 설계 단계에서 발생할 수 있는 문제점을 효과적으로 해결할 수 있다.

3-1-1 개발 프로세스

우리는 전체 개발 프로세스 중 요구사항 분석 단계에서 상세 설계 단계까지를 지원하는 개발 방법론을 제안한다. 개발 방법론에 기초한 개발 프로세스는 아래 [그림 1]과 같다.



[그림 1] 에이전트 개발 프로세스

(1) Requirement Analysis Phase: 이 단계에서는 고객으로부터 받은 초기 요구사항을 분석한다. 개발자는 유스케이스(use case)다이어그램을 이용해서 고객의 요구사항을 분석하고 이를, 기능적, 비 기능적, 도메인 요구사항으로 분류하여 기술한다.

(2) Architectural Design Phase: 이 단계는 다중 에이전트 기반 시스템의 아키텍처를 설계하는 단계로서, 개발자는 분석된 요구사항들을 에이전트에 할당하여 시스템을 구성하는 에이전트를 식별한다. 그런 후 시스템을 구성하는 에이전트들 간의 관계와 역할을 기술한다. 에이전트 간의 역할을 기술하고 나면 전체 시스템을 구성하는 각 에이전트들에 행동(Behaviour)를 할당하게 된다. 마지막으로 이렇게 구성된 에이전트 기반 시스템이 이전 단계에서 분석한 고객의 요구사항을 만족 시키는지 검증하는 작업을 한다.

(3) Multi-Agent Community Design Phase: 이 단계에서 개발자는 에이전트간의 커뮤니케이션에 관한 상세 설계 작업을 수행한다. 개발자는 에이전트간에 상호작용을 위해 프로토콜을 정의하고, ACL 메시지 및 컨텐츠, 온토로지를 기술한다.

(4) Single-Agent Structure Design Phase: 마지막으로 개발자는 하나의 에이전트 내부를 설계하고 내부에서 이루어지는 활동들에 관해서 기술하게 된다.

3-1-2 디자인 패턴의 적용

다중 에이전트 기반 시스템은 에이전트간의 상호 작용을 통해서 고객이 원하는 서비스를 지원하는 형태로 설계되어야 한다. 그러나 이러한 설계 경험이 없는 개발자의 경우 설계단계에서 많은 문제점에 직면하게 된다. 우리는 이것을 지원하고자 초기 아키텍처 설계부터 상세 설계까지 디자인 패턴을 사용할 수 있는 방법을 제안한다. 본 논문에서 제안하는 패턴은 다음과 같다.

(1) Architectural 패턴: 초기 다중에이전트 시스템 아키텍처를 설계하는 단계에서 개발자의 의도와 설계상에 발생할 수 있는 문제점을 해결하기 위해 필요로 한다. 개발자는 Architectural 패턴을 이용하여 개발자는 에이전트를 추출하고 그들 간의 관계를 설정하는 과정에서 개발자의 의도를 명확히 반영한 설계 작업을 수행할 수 있다.

(2) Protocol 패턴: 다중에이전트 기반 시스템에서 프로토콜은 둘 이상의 에이전트간에 커뮤니케이션을 위해 사용된다. 그러나 이러한 프로토콜을 개발경험이 적은 개발자가 설계하는 것은 추후 시스템 운용 중에 문제점을 야기시킬 수 있다. FIPA에서는 둘 이상의 에이전트간 커뮤니케이션을 위해 FIPA Interaction Protocol 을 정의하였다. 개발자는 패턴화된 FIPA Interaction Protocol 을 통하여 에이전트간 통신 프로토콜을 쉽게 정의할 수 있다.

(3) Single-Structural 패턴: 하나의 에이전트를 구성하는 내부 모듈들은 기존 객체 지향 개발방식과 같이 구성될 수 있다. 따라서 하나의 에이전트의 내부를 설계하기 위해 개발자는 기존 객체지향기반의 시스템설계를 위해 사용된 설계패턴들을 이용할 수 있다.

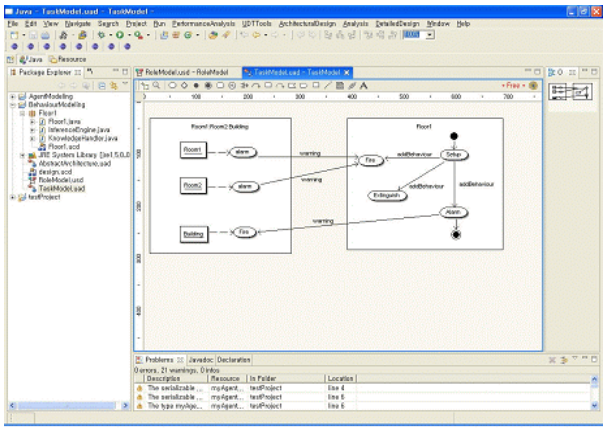
위에서 정의한 패턴의 구분은 [그림 1]에서 소개한 각 프로세스의 설계단계에서 이용될 수 있다. Architectural Design 단계에서 개발자는 Architectural 패턴을 이용하여 다중 에이전트 기반 시스템의 구조를 설계할 수 있고, Multi-Agent Community Design 단계에서는 Protocol 패턴을 이용하여 에이전트간에 이루어지는 통신 프로토콜을 정의할 수 있다. 마지막으로 Single-Agent Structural Design 단

계에서 개발자는 Single-Structural 패턴을 이용하여 각 에이전트의 내부 구조를 설계하는 단계에서 발생할 수 있는 문제점을 쉽게 해결할 수 있다.

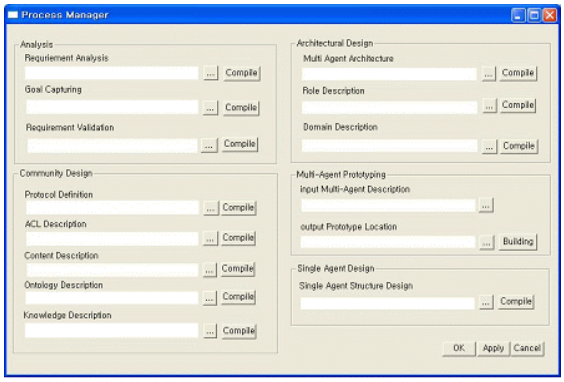
제안된 설계패턴을 이용하기 위해 개발자는 [그림 1]에서 설명한 바와 같이 먼저, 설계 의도와 설계단계에서 발생한 문제점을 식별하고 이를 기술해야 한다. 이를 통해 개발자는 이러한 문제를 해결할 수 있는 적절한 패턴을 찾을 수 있다.

3-2 지원 도구

본 논문에서는 제안 방법론을 지원하는 CASE 도구를 개발하였다. 이 도구는 위 방법론에 기초하여 개발의 상당부분을 자동화 해주고, 개발 프로세스를 관리할 수 있도록 도와준다. 또한 최종적으로 소스코드의 상당 부분을 생성하기 때문에 구현 단계에서 수행해야 하는 작업의 부담을 크게 줄일 수 있다. [그림 2]는 Eclipse3.1 기반 환경에서 플러그인 되어 실행되는 우리의 개발도구를 보여준다. 메뉴바와 툴바 부분을 보면 기존 Eclipse 보다 확장된 메뉴와 툴을 제공하는 것을 알 수 있다.

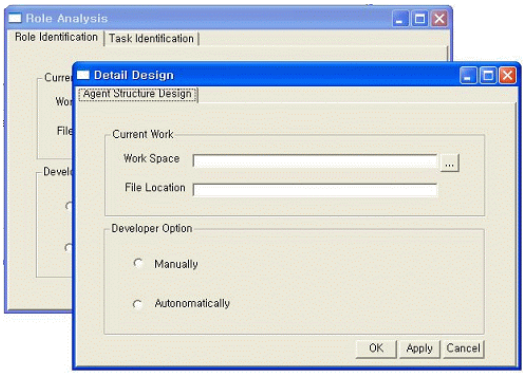


[그림 2] Eclipse 기반 CASE 도구



[그림 3] Process Manager

[그림 3]은 본 논문에서 제안한 CASE 도구에서 개발 프로세스 및 결과물을 관리하는 Process Manager 를 나타낸다. Process Manager 를 통해서 개발자는 각 단계에서 작업한 결과물에 관해서 검증 및 관리를 할 수 있다.



[그림 4] Process Phase Manager

[그림 4]은 CASE 도구 중 Process Phase Manager 를 나타낸다. 이것은 위 [그림 1]에서 소개한 개발 프로세스의 각 단계별 작업을 수행하는데 필요한 기능들을 제공한다. Process Phase Manager 를 통해서 개발자는 각 단계별 작업의 상당 부분을 자동화 할 수 있고 필요한 설계패턴을 사용할 수 있다.

4. 적용 및 평가

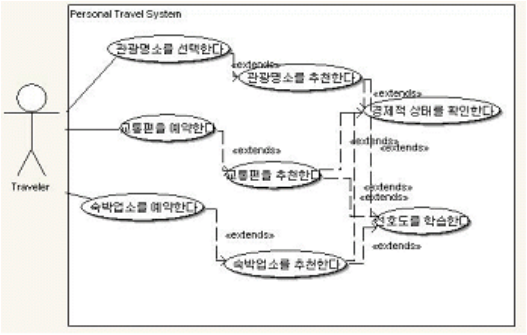
4-1 시나리오

이번 장에서, 우리 개발 방법론의 유효성을 증명하기 위하여 간단한 시나리오를 통해 에이전트 개발 과정을 시뮬레이션 하였다. 본 논문에서 우리는 여행 도우미 시스템 개발을 위해 우리의 도구를 적용시킨다. 초기 시나리오는 다음과 같다. 사용자는 모바일 단말기를 이용하여 다양한 여행 정보 및 교통, 숙박 정보를 얻는다. 또한 자신이 여행하기 원하는 지역이 선정되면 여행 도우미 시스템을 통해 그 지역의 관광 명소를 추천 받을 수 있고, 사용자의 경제 상태 및 선호도를 고려해서 적절한 가격의 교통 및 숙박시설을 추천 받을 수 있다. 사용자는 한 두 번의 단순한 클릭만으로 이 모든 일들을 간단히 해결할 수 있다.

4-2 개발 방법론의 적용

앞의 초기 시나리오를 통해 우리는 요구사항을 분석하고 기술하는 작업을 수행한다. 먼저 유스케이스(use case)를 이용하여 요구사항을 분석하는 작

업을 수행한다. [그림 5]는 CASE 도구를 통해서 요구사항을 분석하는 것을 나타내는 그림이다.



[그림 5] 목표 decomposition 과 에이전트 추출

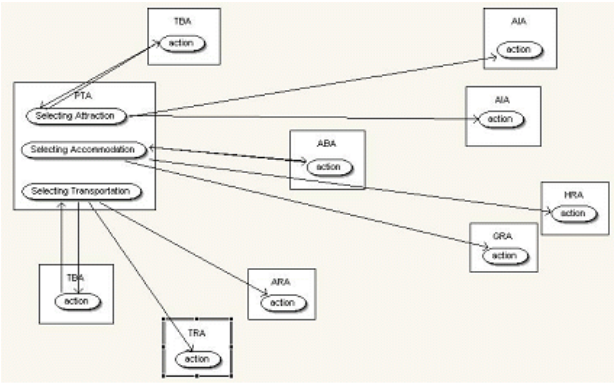
추출된 유스케이스를 기반으로 개발자는 요구사항을 기술할 수 있다. 일반적으로 하나의 유스케이스를 통하여 여러 가지 기능적 요구사항이 기술되고, 각 기능적 요구사항에 관하여 비 기능적 요구사항들이 추출되게 된다. 또한 도메인 요구사항의 경우 그 분야의 비즈니스 전문가를 통하여 도메인 요구사항을 기술하게 된다.

요구사항 분석이 완료되면 개발자는 Architectural Design 단계를 수행한다. 이 단계에서 개발자는 기능, 비 기능, 도메인 요구사항을 토대로 자신의 설계의도와 설계문제점을 찾는다. 이 과정을 통해서 개발자는 적절한 다자인 패턴을 선택할 수 있다. 아래 [그림 6]은 설계 의도에 관한 식별 및 기술 작업을 나타낸다.

Design 1	Intent	예약 가능한 숙박업소 정보를 동적으로 확장시키고 싶다
	Problem	1. 모든 숙박 업소가 자체적인 예약 시스템을 가지고 있는것은 아니다
		2. 사용자가 필요로 하는 모든 숙박업소 정보를 항상 가질 수 없다
		3. 사용자의 디바이스에 모든 숙박업소 정보를 업데이트할 수 없다
		4. 숙박업소 종류별로 예약 서비스를 제공하길 원한다(예: 호텔, 민박…)
⋮	⋮	⋮

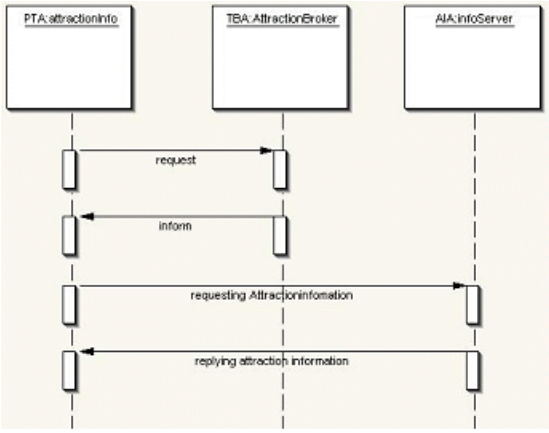
[그림 6] 설계 의도에 관한 식별 및 기술 작업

개발자는 선택된 패턴을 기반으로 에이전트를 추출하고 그들 간의 관계를 기술하는 작업을 수행한다. 아래 [그림 7]과 같이 시스템 아키텍처가 완성되면 개발자는 각 에이전트 들의 역할(Role)과 행동(Task)를 할당하는 작업을 한다. 역할(Role)과 행동(Task)은 각 에이전트에게 기능적 요구사항을 할당함으로 이루어진다.

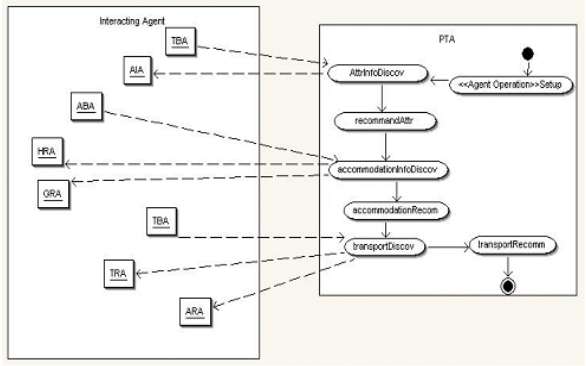


[그림 7] 다중 에이전트기반 시스템 아키텍처

아래 [그림 8]과 [그림 9]는 각각 역할(Role)과 행동(Task)을 나타낸다.



[그림 8] Role 다이어그램



[그림 9] Task 다이어그램

아키텍처 설계의 검증을 위해 우리는 다음과 같은 에이전트 기반 시나리오를 작성하고 이 시나리오를 분석하여 **4-1** 절에서 기술한 초기 시나리오를 만족시키는지 검증 한다. 다음은 에이전트 기반으로 작성한 시나리오의 일부이다.

"Personal Travel Agent 는 하나의 Broker 에이전트에 접속을 하여, 여행 정보와 가격을 요청해준다. Broker 에이전트는 여러 서비스 제공자 에이전트들에게 이러한 여행 정보를 요청하기 위해 접촉하게 된다. Broker 에이전트는 Personal Assistant Agent 가 자신의 이동 단말기 디스플레이를 통해 HTML 문서를 볼 수 없기 때문에, HTML 정보를 XML 정보로 변환하여 Personal Assistant Agent 로 다시 변환된 정보를 보고해주게 된다. Personal Assistant Agent 는 개인화된 정보를 기반으로 한 추천 방식을 제공하기 위해 여행 정보를 필터링....(중략)....."

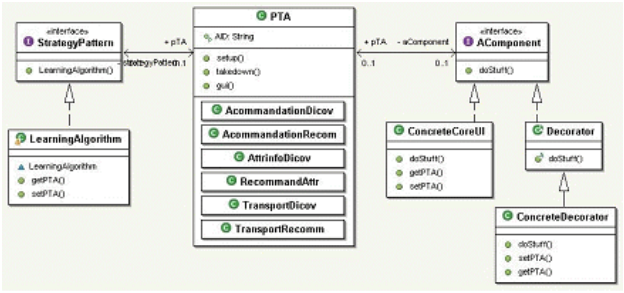
Architectural Design 단계가 끝나면 개발자는 다중 에이전트 커뮤니티에 관한 상세설계단계로 넘어가게 된다. 이 관점에서 개발자는 위에서 기술한 에이전트의 역할(Role)을 FIPA Interaction Protocol 을 기반으로 구체화 시킨다. 개발자는 특정 프로토콜을 선택하여 위 Role 을 기술한 다이어그램과 같은 UML 의 메시지 시퀀스 다이어그램을 이용하여 프로토콜을 기술하고 프로토콜에서 기술한 각 메시지를 ACL 형태로 기술한다. 이 과정에서 필요로 하는 온토로지와 Contents 언어가 선택된다. 아래 [그림 10]은 PTA (Personal Travel Agent)가 HRA (Hotel Reservation Agent)에게 보내는 메시지 중 하나를 보여준다.

```
(cfp
  :sender (agent-identifier :name PTA)
  :receiver (set (agent-identifier :name HRA))
  :content
    "((action (agent-identifier :name PTA)
     (reserve room 2))
    (any ?x (and (= (price suit) ?x) (< ?x 2))))"
 :ontology Hotel
 :language fipa-sl)
```

[그림 10] PTA 가 HRA 에게 보내는 CFP 메시지

개발자는 PTA 과 HRA 간에 사용할 프로토콜로 FIPA Contract Net 프로토콜을 사용 하였으며, 위 [그림 10]에서 개발자는 그 중 일부인 cfp (Call for Proposal) 메시지를 기술한 내용이다. 위 내용은 PTA 에이전트가 HRA 에이전트에게 suit room 2 개에 관한 제안을 요청하는 것이다. contents 언어는 fipa-sl 을 이용하였으며 온토로지는 Hotel 이라는 온토로지를 사용하였다.

마지막으로 개발자는 에이전트의 내부구조를 설계하게 된다. 이 때 역시 에이전트의 구조에 관한 적절한 패턴이 선택되고 이 패턴을 이용하여 개발자는 개발자의 의도를 반영한 에이전트를 개발할 수 있다. 아래 [그림 11]은 PTA 의 내부 구조를 나타난다.



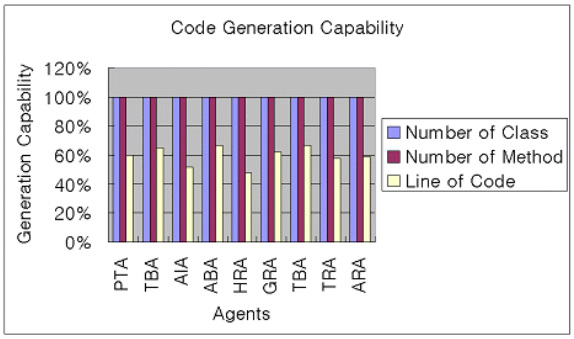
[그림 11] PTA 의 내부 구조

[그림 11]에서 PTA 를 구성하기 위해 GoF 의 Decorator 패턴과 Strategy 패턴이 사용되었다. Decorator 패턴을 이용한 부분은 에이전트의 UI 를 구성하는 부분이고 Strategy 패턴은 PTA 의 학습 및 추론하는 부분을 담당한다. 또한 중심이 되는 PTA 클래스는 JADE 문법을 사용하여 내부 구조를 설계하기 이전에 이미 개발 도구에서 코드를 자동화 하여 생성 시키는 부분이다. 위 구조 설계가 끝나면 개발자는 최종적으로 위 구조를 구성하는 내부 클래스들간의 커뮤니케이션을 메시지 시퀀스 다이어그램으로 기술하게 되고 CASE 도구는 이러한 설계서를 기반으로 구현에 근접한 소스코드를 생성시킨다. 비록 우리가 개발한 CASE 도구가 100%의 소스코드를 생성시키진 못하지만 이 도구를 이용하여 구현단계에서 발생하는 비용을 상당부분 줄일 수 있을 것이다.

4-3 평가

본 장에서는 본 논문에서 제안한 방법론을 기반으로 설계한 Personal Travel Assistance System 에 관해서, 본 논문에서 제안한 CASE 도구의 소스코드 생성 능력을 평가하였다. 평가 결과는 다음 [그림 12]와 같다. 여기서 제안 도구를 통하여 개발하였을 경우 Class 와 Method 의 100%를 모두 생성시킬 수 있다. 그러나 소스코드 생성의 경우는 모두

70%미만의 생성능력을 보여줬다. 이것은 제안한



[그림 12] 제안 CASE 도구의 소스코드 생성 능력

개발 도구를 이용하여 개발하였을 경우 JADE API 를 사용하는 Agent 를 나타내는 클래스의 경우는 90%의 코드 생성 능력을 보여주지만 그 외의 경우 대부분 코드의 골격만을 생성해주기 때문이다.

5. 결론 및 향후 연구과제

본 논문에서 우리는 제안한 방법론을 기반으로 Personal Travel Assistance System 을 개발하였다. 본 논문에서 제안한 방법론을 통해 개발자는 요구사항 분석부터 설계 단계까지의 에이전트 개발 프로세스를 지원받을 수 있으며, 디자인 패턴을 이용하여 설계단계에서 발생하는 문제점을 해결할 수 있다. 또한 제안하는 CASE 도구를 통해 개발 프로세스의 이전 단계에서 다음단계로 넘어가는 경우 작업 산출물의 일부를 자동화 하여 개발자의 부담을 줄여주며 마지막으로 소스코드를 생성시켜서 구현 부분에서의 부담을 줄여준다.

그러나 전체 소스코드의 70%이하로 생성시키기 때문에 여전히 구현단계에서 상당한 비용이 발생된다. 또한 설계 패턴을 선택하는 부분에 있어서 아직은 개발자의 능력과 노력이 요구된다. 향후 우리는 이러한 부분을 보완하여 보다 완성도 있는 개발 방법론과 지원도구를 제안할 것이다.

[참고문헌]

[1] M. Younas, K-M Chao, R. Anane, “The M-Commerce Transaction Management with Multi-Agent Support”, Proceedings of the17th IEEE International Conference on Advanced Information Networking and Applications (2003)

[2] Fausto Giunchiglia, J. Mylopoulos, A. Perini, "The Tropos software development methodology: processes, models and diagrams", Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent systems (2002)

[3] Jorge J. Gómez-Sanz, R. Fuentes, J.J. Gomez-Sanz, R. Fuentes, “Agent Oriented Software Engineering with INGENIAS”, Fourth Iberoamerican Workshop on Multi-Agent Systems (2002)

[4] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, Y. Diao, “ABLE: A toolkit for building multiagent autonomic systems”, IBM Systems Journal, Vol 41, No 3, (2002)

[5] G. Caire, “JADE Tutorial JADE Programming For Beginners”, TILAB, formerly CSELT, 04 December (2003)

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Professional Computing Series (1994)