MDA를 이용한 다중 에이전트 기반 시스템 개발단계에서 재사용성 향상을 위한 프레임워크

이풍석°, 장수현, 이은석
성균관대학교 컴퓨터공학부 소프트웨어공학 연구실
{pungdol°, nikelion, eslee}@ece.skku.ac.kr

A Framework for Improving Reusability at the Development Process of Multi-Agent based System using MDA

Poong-Seok Lee°, Soo-Hyun Jang, Eun-Seok Lee
SE Lab, School of Computer Eng., Sungkyunkwan University

요 약

최근 유비쿼터스 환경에서 동작하는 지능형 시스템에 관한 관심이 높아지면서, 이러한 지능형 시스템의 개발을 효율적으로 하기 위해 에이전트 기반의 소프트웨어 시스템 개발 방법론 및 지원 도구에 관심이 높아지고 있다. 이러한 시스템들은 에이전트들의 동작환경을 제공하는 에이전트 플랫폼의 사용이 필수적이다. 그러나 실제로 에이전트 기반 시스템을 개발하는 경우 초기 단계에서 가장 적절한 에이전트 플랫폼을 결정하는 것은 어렵다. 또한 개발 중에 다양한 에이전트 플랫폼에 적용 가능한 소프트웨어를 개발해야 하는 경우가 발생할 수 있다. 따라서 본 논문에서는 이러한 문제점을 해결하기 위해 MDA를 기반으로 에이전트 기반 시스템 개발 방법론 및 개발 지원 도구를 제공하고자 한다. 본 논문에서 제안하는 방법을 통해 개발자는 개발 초기 단계에서 결정된 소프트웨어의 아키텍처를 기반으로 다양한 플랫폼에 적용 가능한 에이전트 모델과 소스코드를 생성시킬 수 있다. 본 논문에서는 플랫폼 독립적인 에이전트 모델을 통하여 FIPA-OS와 MTI 에이전트 플랫폼 기반의 소스코드를 생성시키는 실험을 하여 제안 방법론 및 도구의 유효성을 검증한다.

1. 서론

최근 유비쿼터스 환경에서 동작하는 소프트웨어에 관한 요구사항이 높아짐에 따라, 지능형 에이전트 기반으로 소프트웨어를 개발하는 방법에 관심이 높아지고 있다. 에이전트는 객체를 확장시킨 개념으로 하나의 객체는 다른 객체와의 협상을 통해서 전체 시스템의 목적을 달성시킨다. FIPA는 이러한 에이전트 기반 시스템의 개발을 위해 표준 명세서를 제안하였다[1]. FIPA 표준안에 따르면 에이전트들은 에이전트 플랫폼에서 동작하면서, 에이전트 커뮤니케이션 언어를 통해서 상호 커뮤니케이션을 수행한다. 따라서 에이전트를 기반으로 시스템을 개발하기 위해서는 시스템의 목적에 적절한 플랫폼을 선택하여야 한다. 그러나 실제로 에이전트 기반 시스템을 개발하는 경우, 개발 초기 단계에서 적절한 플랫폼을 선택하는 것은 어려우며, 개발자들이 모든 플랫폼에 관한 지식을 갖추는 것도 어렵다. 특히, 여러 플랫폼에서 동작하는 에이전트 기반 시스템을 개발하는 경우, 개발 비용이 매우 높아지게 된다. 이러한 문제를 해결하기 위해 MDA기반 소프트웨어 개발 방법을 에이전트 기반 시스템 개발을 위해 적용하려는 연구가 있어왔다.

전통적인 MDA기반 방법에서 설계자는 소프트웨어가 동작하는 플랫폼을 정의하고 플랫폼 독립 모델로부터 플랫폼 종속 모델로의 변환 규칙을 정의해 주어야 한다.

이러한 방식은 하나의 플랫폼 독립 모델로부터 여러 종류의 플랫폼 종속 모델을 생산시킬 수 있기 때문에 이식성과 재사용성을 증가 시킨다[2][3].

기존에 MDA를 통해 이러한 에이전트 기반 시스템을 개발하고자 하는 노력이 있어 왔지만, 이러한 방식은 플랫폼 독립 모델로부터 정해진 에이전트 플랫폼에 종속 모델 및 소스코드를 개발하기 위해 필요한 모델 변환 규칙을 개발자가 모두 정의 하여야 했다. 그렇게 되면, 개발 과정 중에 여러 플랫폼에 적용하는 부분에 있어서 MDA의 장점인 재사용성을 떨어뜨린다. 따라서, 개발자들에게 모델 변환 과정을 정의하는 단계에서 보다 설계 단계에서 재사용성을 높이는 방안이 요구된다.

본 논문에서는 MDA를 통한 에이전트 기반 시스템을 개발하기 위해, 재 사용 가능한 개발 프레임 워크를 제공하고자 한다. 본 논문에서 제안하는 개발 프레임 워크는 기존 방법들의 단점인 실제 에이전트 설계 및 구현 단계에서의 적용 상 난해한 점과 새로운 에이전트 플랫폼에 대한 모델 변환이 필요할 때마다 변환 프로세스가 필요한 것을 해결하기 위해서, 재사용성을 향상시키기 위한 개발프로세스 및 방법론을 제안하고, 이를 지원하는 라이브러리를 제공한다.

개발자가 모델 변환 규칙을 모두 지정하지 않더라도,

제공된 라이브러리는 소스모델과 타겟모델의 관련성을 지정해 주면, 이를 통해 플랫폼 종속 모델과 소스코드를 생성시켜 준다. 그리고 모델 변환 결과를 평가하기 위해 에이전트 시스템 개발 과정에서 적용되는 FIPA-OS와 MTI(Message Transport Interface)의 두 가지 플랫폼을 통하여, PIM과 PSM의 두 메타모델간 변환의 일관성을 평가하게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 MDA 기반 에이전트 개발 방법론에 대해 알아보고, 3장에서는 기존 방법을 보완한 MDA 기반의 에이전트 시스템 개발 프레임워크에 관하여 기술한다. 4장에서는, 3장에서 제안한 프레임워크를 유비쿼터스 학습 시스템(U-Learning)에 적용한 사례를 통하여 유효성을 평가한 후, 마지막 5장에서는 결론을 내린다.

2. 관련연구

현재 MDA를 적용시킨 에이전트 시스템에 관한 예가 몇 가지 존재한다. 그러나 이 모델들은 추상화된 모델을 사용, 실제 에이전트 설계 및 구현 단계에서의 사용하기 힘든 단점이 존재하거나 새로운 에이전트 플랫폼에 대한 모델 변환이 필요할 때마다 변환 프로세스를 새로 만들어야 하는 문제가 발생한다.

Denis Gracanin와 Ronald D. Snyder이 제시한 방법론은 Cougaar라는 중간단계의 표현형식으로 컴포넌트화 하여 개발하는 방법론을 따르며 커뮤니티 구조를 위한 개발 방법인데, 해당 도메인에 컴포넌트가 존재하지 않을 때 모델 변환 개발자가 Cougaar를 구성하기 위한 추가적 비용과 함께 커뮤니티 구조 정립이 필요하고 새로운 에이전트 플랫폼마다 모델 변환을 처음부터 다시 만들어야 하는 경우가 생긴다[4][5].

Huamonte의 경우 에이전트 시스템을 역할에 따라 에이전트의 행위를 결정하고 역할 기반으로 모델링하는 방법을 제안하였으나 다양한 에이전트 플랫폼에 대한 모델 변환 전략이 포함되어 있지 않다[6].

Gaya Buddhinath은 에이전트 모델을 8가지 개념으로 분류하고 이를 모델링 하는 방법을 제안한다. 그러나 이 방법은 에이전트를 구성하는데 너무 많은 모델링 개념을 표현해야 하며 모델 개발자들의 부담을 높일 수 있다. 이 방법 역시 다양한 플랫폼을 대상으로 하는 모델 변환 전략이 포함되어 있지 않다[7].

본 논문에서는 FIPA 표준안 기반의 플랫폼 독립적인 에이전트 모델을 FIPA-OS[8]와 MTI(Message Transport Interface)[9]의 플랫폼 종속 모델로 변환하는 과정을 통해, 에이전트 모델 변환 시 설계 및 구현의 관점에서 고려해야 하는 요소를 분석하고 이를 바탕으로 다중 에이전트 시스템 모델 변환에 특화된 모델 변환 프레임워크를 제안한다.

3. MDA 기반의 다중 에이전트 시스템 개발 프레임워크

본 장에서는 MDA를 적용한 에이전트 기반 시스템의 설계 및 구현단계에서 다양한 에이전트 플랫폼으로의 원활한 변환 전략을 지원하는 프레임워크를 제안하고, 모델 변환 원리에 대해서 살펴 보도록 한다.

에이전트 기반 시스템에서 적용될 일반적인 MDA 프로세스의 구조는 (그림 1)과 같은 순서로 구성된다.

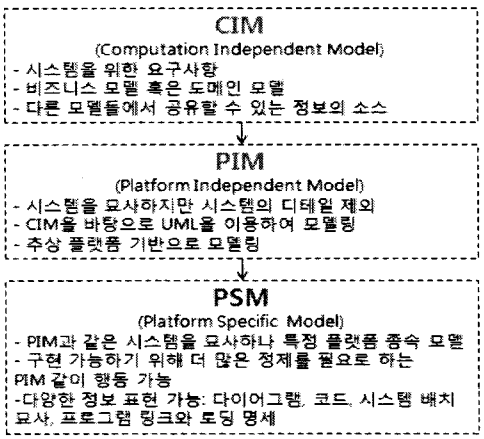

그림 1. 일반 MDA 모델 변환 프로세스

이러한 MDA 프로세스에서 프레임워크는 (그림 2)와 같이 PIM에서 PSM로 메타모델을 변환하는 역할을 한다.

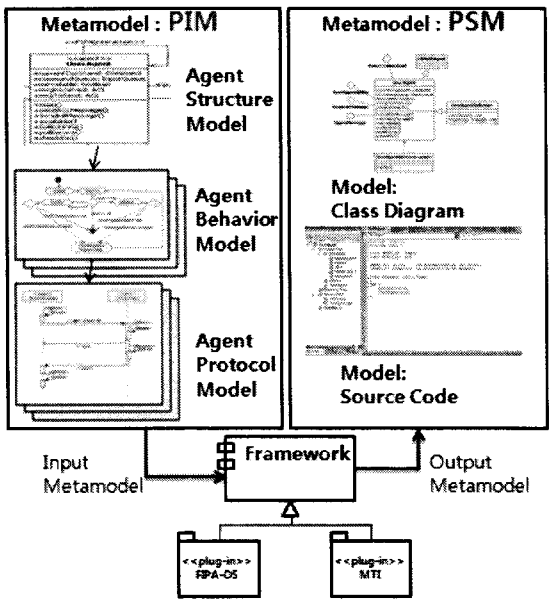

그림 2. 에이전트 기반 시스템 모델 변환 프로세스

(그림 2)와 같이 변환 프레임워크의 입력 메타모델,

즉 플랫폼 독립 모델(PIM)은 에이전트 구조 모델(Agent Structure Model), 에이전트 행위 모델(Agent Behavior Model), 에이전트 프로토콜 모델(Agent Protocol Model)로 정의 한다.

첫째, 에이전트 구조 모델은 에이전트들의 구조와 관계를 표현하는 모델이다. 에이전트만 존재하는 것이 아닌 다른 비 에이전트 클래스와 함께 동작하는 것을 나타내며 에이전트 플랫폼이 객체 지향 언어 중심으로 구현되어 있으므로 본 모델은 객체 지향 언어의 특징을 표현할 수 있는 클래스 다이어그램을 사용한다.

다른 클래스들 간에 에이전트임을 구별하는 요소로 <<agent>> 스테레오타입을 사용하고, 클래스들의 관계는 본래 연관(association)으로 나타내지만 에이전트 사이의 연관은 에이전트 간의 인터렉션 프로토콜을 나타낸다.

둘째, 에이전트 행위 모델은 에이전트가 주변 환경에 영향을 끼칠 수 있는 행동을 나타내고 에이전트 구조 모델에서 <<agent>> 클래스의 오퍼레이션이 에이전트의 행위가 되며, 에이전트에서 계층적으로 구성할 수 있는 가장 복잡한 구조를 가진다. 에이전트 자신의 상태를 변화 시키기도 하며 다른 에이전트와 시스템에 영향을 주는 모델로써 활동 다이어그램을 사용한다.

셋째, 에이전트 프로토콜 모델은 FIPA의 인터렉션 프로토콜 명세서를 인용하여 에이전트간 인터렉션을 나타낼 뿐만 아니라 에이전트의 행위도 포함하여 나타낸다. 하나의 프로토콜을 나타내기 위해 하나의 시퀀스 다이어그램을 사용한다.

위와 같은 최소 3가지 다이어그램 집합이 필요한 이유는 클래스 다이어그램은 에이전트의 행위와 관계에 대해 선언을 나타내고, 선언된 행위와 관계에 대해 정의하는 것이 활동, 시퀀스 다이어그램이기 때문이다.

따라서 클래스, 활동, 시퀀스 다이어그램의 순서로 프레임워크는 PIM을 읽어 들이게 된다.

프레임워크는 새로운 다수의 플랫폼에 대한 변환 규칙을 적용 할 수 있도록, Plug-in 방식으로 구성되어 있고 이벤트 기반으로 PIM을 파악하여, 각 메타모델의 특성에 따라 이벤트 함수를 발생시키게 된다.

가장 먼저 파악되는 클래스 다이어그램은 (그림 3)와 같은 방식으로 이벤트 함수를 호출한다.

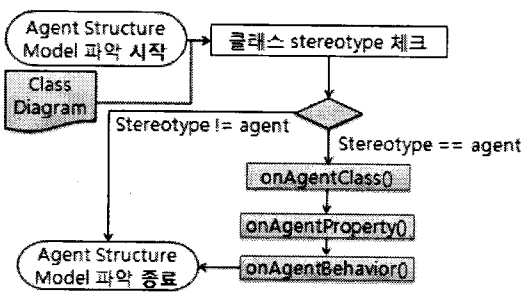

그림 3. 클래스 다이어그램 이벤트 처리 프로세스

onAgentClass()는 <<agent>> 클래스임을 확인하고 가장 먼저 호출되어 이벤트 처리기에 에이전트의 이름을 알려는 이벤트 함수이다. 다음으로 onAgentProperty()와 onAgentBehavior()는 일반적인 클래스에서 에트리뷰트와 오퍼레이션과 동일시 되는 개념으로, 에이전트의 지식과 행위에 관한 처리를 하게 된다.

다음으로 특정 에이전트의 행위에 대한 정보를 담고 있는 활동 다이어그램은 에이전트의 특성에 따라서 하나의 에이전트에 다수가 연관되어 존재할 수 있다.

활동 다이어그램은 초기에 다이어그램 내의 복합 행위 존재 여부를 파악 후 (그림 4)와 (그림 5)의 경우로 분류 되며, 그림의 액션은 이벤트 함수를 나타낸다.

위와 같은 분류의 이유는 특정 플랫폼의 특성에 따른 복합 행위에 따른 대처 방법이 다를 경우를 고려한 것이다.

그리고 PIM 액션의 흐름을 파악하고, 그 내용에 맞는 이벤트 함수가 호출 된다. 그리고 특정 플랫폼에 의존하여 각각의 이벤트 처리 프로세스에서는 출력 메타모델을 기반으로 출력 메타모델을 생산하게 되는데, 클래스 다이어그램을 사용하고 또한 블록 기반의 프로그래밍 형식을 사용한다.

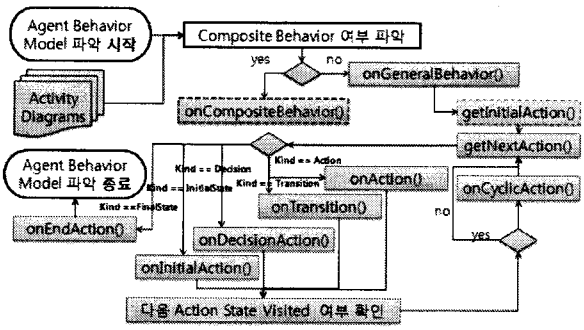

그림 4. General Behavior의 이벤트 처리 프로세스

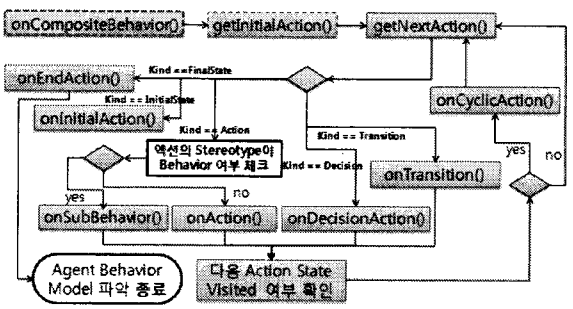

그림 5. Composite Behavior의 이벤트 처리 프로세스

최종적으로 에이전트 간 커뮤니케이션과 인터렉션을 표현한 에이전트 프로토콜 모델인 시퀀스 다이어그램은 다이어그램 하나가 이벤트 함수 하나를 호출한다.

(그림 6)의 이벤트 함수의 입력 매개변수 부분처럼 프로토콜에 따라 지정된 템플릿으로 파악 된다.

```
public void onRequestProtocol(
        String behaviorName, String protocolName,
        String initiatorAgent, String responderAgent,
        String receivingBehaviour, String confirmationBehaviour,
        String responseBehaviour, String resultBehaviour) {
    boolean isRegistered = false;
    while(iter.hasNext()){
        if(iter.next().equals("RequestReceiver")){
            isRegistered = true;
            break;
        }
    }
    if(isRegistered == false){
        receivers.add("RequestReceiver");
        MsgMethodRegister((MSGRECV) RequestReceiver);
    }
    new FipaRequest( behaviorName, protocolName, initiatorAgent,
            responderAgent, receivingBehaviour,
            confirmationBehaviour, responseBehaviour,
            resultBehaviour);
}
```

그림 6. 프로토콜 이벤트 함수의 예

입력 매개변수에 대한 내용은 다음과 같다.

에이전트간 최초 인터렉션을 하는 행위와, 프로토콜 방식, 최초 메시지 전송 에이전트와 응답하는 에이전트, 초기 메시지를 받는 행위에서, 확인하는 행위를 거쳐 최종 메시지를 보내는 행위와 그것에 반응하는 마지막 행위까지 일련 규칙에 따라 파악된다.

그리고 이벤트 함수 하나가 호출될 시 각 프로토콜의 성격에 맞는 클래스를 인스턴스화 하고 동시에 클래스 생성자에 설정된 메소드를 호출하여 다이어그램을 읽는 순차적 단계를 밟게 된다. 그리고 그러한 인스턴스화 된 클래스마다 정의된 서브 클래스나 콜백함수에 의해 메시지에 반응한다.

이러한 프레임워크에 Plug-in으로 사용될 변환 규칙은 FIPA-OS와 MTI 플랫폼에 상관없이 동일 시 처리 된다.

그렇다면 특정 플랫폼 종속적으로 PIM을 변환한다는 것은 결국 이벤트 함수의 내용만 특정 플랫폼의 특성에 맞게 설정하면 된다는 것이다.

특정 플랫폼으로의 이벤트 함수 설정은 변환 규칙과 함께 3.1과 3.2에서 설명 하도록 하겠다.

3.1. FIPA-OS 플랫폼 종속 모델 변환 규칙

본 장에서는 프레임워크에 Plug-in으로 사용될 FIPA-OS 플랫폼 종속 모델로 변환하기 위한 규칙에 대해서 표와 함께 간단히 설명하도록 한다.

FIPA-OS는 자바언어 기반의 플랫폼이며, MTS(Message Transport Service)와 태스크 매니저, 컨버세이션 매니저와 같은 필수적인 요소를 만드는 에이전트 쉘(Agent Shell) 클래스를 상속하면서 에이전트를 구성해 나가는 형태를 가지고 있다.

변환 초기 PIM의 에이전트와 동일한 이름을 가진 에이전트 클래스를 생성한 뒤, 에이전트의 행위에 따른 클래스를 만들어 액션을 기록한다. 또한 프로토콜마다 그 내용을 가진 인터페이스 작성하고, 메인 에이전트 클래스에서 인터페이스를 구현하여 필요한 인터렉션을 가능하게 하고, 각 시퀀스 다이어그램의 정보를 토대로 프로토콜들을 PSM의 인터페이스로 작성해서 구현한다.

FIPA-OS 플랫폼 종속 모델을 위한 기본적인 이벤트 함수의 규칙은 (표 1)과 같다.

표 1. FIPA-OS 변환 규칙

이벤트 함수	내 용
onAgentClass	에이전트 쉘을 상속 받는 동일 명의 클래스 생성, 주요 파일 임포트 및 생성자와 메소드 생성.
onAgentProperty	생성된 PSM 에이전트 클래스로 PIM 클래스의 프로퍼티를 복사
onAgentBehavior	PSM 에이전트 클래스로 PIM 행위를 변형하여 복사 및 선언.
onGeneralBehavior	Behavior 클래스에 메인 메소드 생성 및 에이전트 클래스 인스턴스화 조건 설정. BehaviorClass.action (Behavior 클래스의 참조 클래스) 생성, 내부에 메인 메소드 설정.
onAction	BehaviorClass.action 내 메소드 생성, onAction의 행위 코드 추가.
onDecision	if문으로 다음의 onAction 행위를 BehaviorClass.action 클래스에 추가
onCyclicAction	PSM 코드의 전반적인 수정 요구. do while문으로 수정. 기존의 코드를 stack에 푸시, 코드 삭제 후 팝해서 코딩.
onSubBehavior	SubBehavior의 behavior 클래스 인스턴스를 BehaviorClass.action 클래스의 action 안에 생성.

3.2. MTI 플랫폼 종속 모델 변환 규칙

본 장에서는 프레임워크에서 Plug-in으로 사용될 MTI 플랫폼 종속 모델로 변환하기 위한 규칙에 대해서 표와 함께 간단히 설명하도록 한다.

MTI는 C++ 언어 기반의 인지도가 아직은 높지 않은 플랫폼이고 아직은 계속 개발중인 플랫폼이다.

에이전트는 AID, 프로토콜 및 생명 주기, 기타 정보를 모두 헤더파일과 동적 링크 라이브러리를 이용해 선언 및 정의한다. 변환 초기 PIM의 에이전트와 동일한 이름을 사용하여 에이전트의 선언은 헤더파일, 정의는 cpp파일에 메인 함수를 이용해 표현된다. 그리고 공통 헤더 파일과 동적 링크 라이브러리를 포함하게 된다.

또한 에이전트의 행위는 에이전트의 정의 부분을 담당하는 파일의 메인 함수에 작성되고, 에이전트의 인터렉션은 에이전트 마다 콜백함수를 이용하게 된다.

MTI 플랫폼 종속 모델을 위한 기본적인 이벤트 함수의 규칙은 (표 2)과 같다.

표 2. MTI(Message Transport Interface) 변환 규칙

이벤트 함수	내 용
onAgentClass	PIM과 동일 이름을 갖는 PSM.h, PSM.cpp과 필수 코드 생성.
onAgentProperty	헤더파일에 C++ 언어 형 변수로 복사.
onAgentBehavior	PIM의 행위를 C++ 언어 함수로 헤더 파일에 선언, cpp파일에 차후 정의 가능한 블록 (to do) 생성.
onGeneralBehavior	cpp파일에 콜백함수 선언 및 등록.
onAction	onAction의 행위를 메인 함수에 추가.
onDecision	if문으로 다음의 onAction에 따르는 행위를 메인 함수에 추가.
onCyclicAction	PSM 코드의 전반적인 수정 요구. do while문으로 수정. 기존의 코드를 stack에 푸시, 코드 삭제 후 팝해서 코딩.
onSubBehavior	SubBehavior가 속한 에이전트의 인스턴스를 메인 함수에 생성.

4. 사례 연구

본 장에서 다루게 될 사례는 현재 다양한 에이전트 플랫폼이 존재하고 미래에 새로운 에이전트 플랫폼이 개발되어 나올 수 있는 시점에서 다중 에이전트 기반의 유비쿼터스 학습 시스템(U-learning System)을 개발 하는 상황을 대상으로 한다. 다양한 학습자들이 다양한 에이전트 플랫폼에서 서비스를 요청할 것으로 예상되고 있어, 미래를 확신할 수 없는 상황이므로 본 논문에서 제시된 프레임워크를 이용하여 MDA 개발 프로세스를 적용하기로 한다.

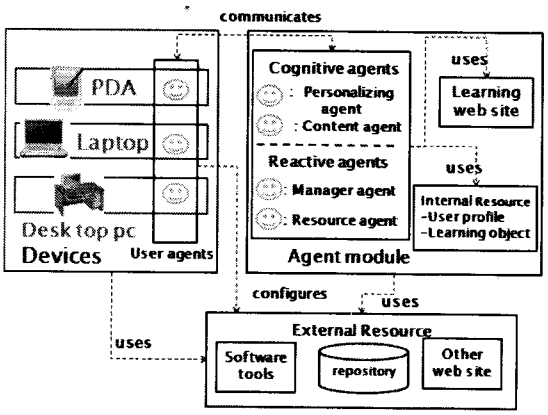

그림 7. U-learning Multi-Agent System

(그림 7)과 같은 구조를 갖는 유러닝 시스템은 학습자 디바이스의 정보와 현재 상태를 파악하여 최적화된 학습환경을 구축하기 위해 서버 에이전트와 인터렉션을 하게 된다. 또한 학습 콘텐트 전송에 있어서도 학습자의 환경을 고려하여 전송하는 학습자 환경을 최적으로 유도하는 학습 시스템이다. 이러한 시나리오를 바탕으로 프레임워크의 변환에 필요한 PIM을 작성하였다. 그러나 본 논문의 제약사항 상 클래스 다이어그램의 학습자 에이전트(User Agent) 부분만을 변환 대상으로 다룬다.

그래서 변환 대상이 되는 PIM은 (그림 8)과 같다.

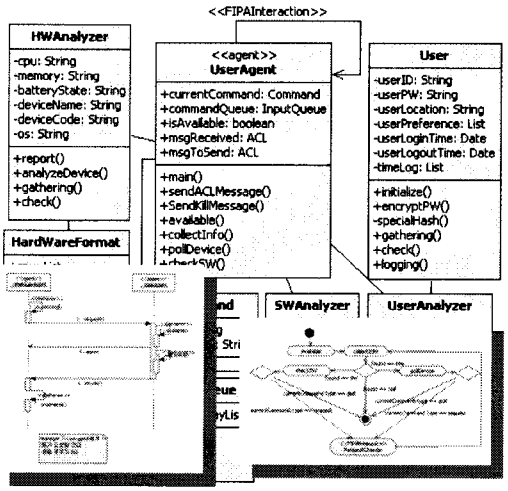

그림 8. PIM 중 사용자 에이전트 부분

FIPA-OS 플랫폼 대상의 프레임워크를 통해 (그림 9)와 같은 결과를 얻었다.

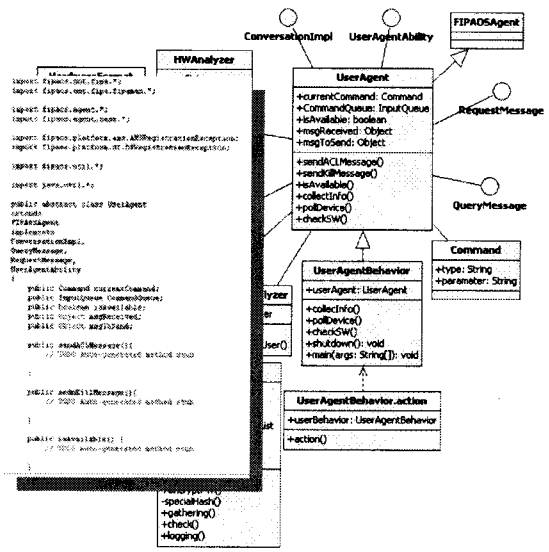

그림 9. FIPA-OS: PSM 중 사용자 에이전트 부분

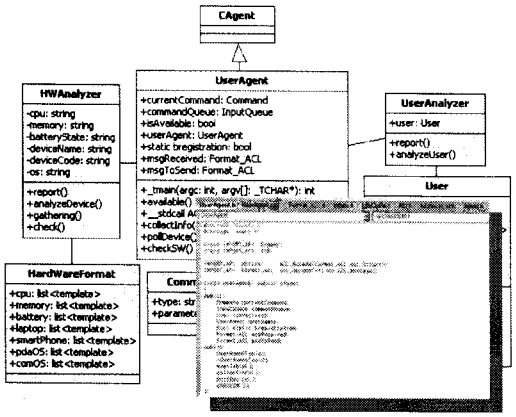

그림10. MTI: PSM 중 사용자 에이전트 부분

MTI 플랫폼 대상의 변환 규칙을 적용한 프레임워크를 통해 (그림 10)와 같은 결과를 얻었다.

비록 프레임워크를 통해 변환을 거친 PSM에서 실행 가능한 것(*executable things*)을 바로 얻은 것은 아니지만, 개발 초기 단계에서 결정된 소프트웨어의 아키텍처를 프레임워크의 Plug-in을 통해 다양한 플랫폼에 적용 가능한 에이전트 모델과 소스코드를 생성시킬 수 있는 전략을 제공한다는 점과 함께, 초기 설계의 재사용성의 향상으로, 개발자뿐만 아니라 투자한 기업에게 있어서도 프레임워크의 효용 가치는 더욱 크게 작용할 수 있다.

하물며 Plug-in 내 이벤트 함수의 설정 시, 더 많은 정제(refinement)를 거친다면 직접적으로 배치할 수 있는 코드로의 변환을 유도하는 것은 불가능한 것이 아니다.

5. 결론

본 논문에서는 에이전트 기반 시스템을 개발하는 단계에서 적용할 플랫폼이 정해지지 않았거나, 다양한 플랫폼을 지원하는 에이전트 기반 시스템을 개발해야 하는 상황에서 개발 비용을 줄이기 위한 MDA 기반 에이전트 개발 프레임워크를 제안하였다.

제안하는 방법을 통해 개발자는 개발 초기 단계에서 결정된 소프트웨어의 아키텍처를 기반으로 다양한 플랫폼에 적용 가능한 에이전트 모델과 소스코드를 생성시킬 수 있었다. 또한 플랫폼 독립적인 에이전트 모델을 통하여 FIPA-OS와 MTI 에이전트 플랫폼 기반의 소스코드를 생성 시키는 실험을 하여 제안 방법론 및 도구의 유효성을 검증하였다.

특히 본 논문에서 제안하는 개발 프레임워크는 플랫폼 종속적인 모델을 얻기 위해서 모델 변환 프로세스와 변환 규칙을 직접 정의하지 않고 모델간의 매핑 관계만 정의되면 플랫폼 독립적 모델로부터 플랫폼 종속적인 모델과 소스코드를 얻을 수 있기 때문에 기존 MDA 기법에 비해 재사용성이 증가함을 알 수 있다.

참고문헌

[1] FIPA, "FIPA Abstract Architecture Specification," http://www.fipa.org

[2] OMG, "MDA Guide version 1.0.1," http://www.omg.org, 12 JUN. 2003

[3] Thomas O.Meservy and Kurt D. Fenstermacher, "Transforming Software Development - An MDA Roadmap," Computer, IEEE VOL. 38, Issue 9, Page(s): 52 – 58, SEP. 2005

[4] Denis Gracanin, H. Lally Singh, Shawn A. Bohner and Michael G. Hinchey, "Model-Driven Architecture for Agent-Based Systems," LNCS VOL. 3228, 2004

[5] Ronald D. Snyder, and Douglas C. MacKenzie, "Cougaar Agent Communities," Proceedings of Open Cougaar 2004, JUL. 2004

[6] Huamonte, J. Smith, K., "The use of roles to model agent behaviors for model driven architecture," southeastCon, 2005. Proceedings of IEEE, APR. 2005

[7] Gaya Buddihinath J., Lin P., and Michael W., "A Model Driven Component-Based Development Framework for Agents," International Journal of computer Systems Science and Engineering, JUL. 2005

[8] FIPA-OS, "FIPA-OS Developers Guide," http://fipa-os.sourceforge.net/index.htm/

[9] Sang Yong Park and Hee Yong Youn, "Efficient Message Transport Interface for Efficient Communication between Agent Framework and Event Service," International Journal of Web and Grid Services, VOL 2, Number 3, pp. 331-352(22), 22 Nov. 2006

[10] Joao Paulo Almeida, Remco Dijkman, Marten van Sinderen, Luis Ferreira Pires, "On the Notion of Abstract Platform in MDA Development," Eighth IEEE International 20-24, Page(s): 253 – 263, SEP. 2004