제 26 회 한국정보처리학회 추계학술발표대회 논문집 제 13 권 제 2 호 (2006. 11)

MDA 기반의 다중 에이전트 기반 시스템 개발*

장수현, 윤현상, 이은석
성균관대학교 전기전자컴퓨터 공학과
e-mail : nckelion@ece.skku.ac.kr

MDA-Based Multi-Agent based System Development

Suhyeon Jang, Hyunsang Youn, Eunseok Lee
Information Communication Engineering, SungKyunKwanUniversity

요 약

MDA 는 OMG 에서 제안한 소프트웨어 개발 방법론으로써, 플랫폼 독립적인 모델에서 플랫폼 종속적인 모델로의 모델 변환을 사용하여 소프트웨어의 이식성, 재사용성 등을 향상 시킬 수 있다. 다중 에이전트 기반 시스템을 개발하는 데 MDA 를 적용하는 것은, 다양한 유비쿼터스 환경에서 동작하는 지능형 시스템 개발을 효율적으로 지원한다. 최근 MDA 를 적용하여 다중 에이전트 기반 시스템을 개발한 연구가 있다. 그러나 이 연구는 에이전트 플랫폼이 고려되지 않고 모델 변환이 자동화 되지 못한 단점이 있다. 본 논문에서는 OMG 에서 표준화한 방법으로 UML 을 모델을 기술하고 모델 변환 규칙을 적용한다. 이를 통해 하나의 모델로부터 다양한 에이전트 플랫폼으로의 이식성을 얻을 수 있다. 이를 평가하기 위해 이전 연구에서 개발한 전자상거래 시스템의 시나리오를 가지고 에이전트 시스템을 구현하고 그 유효성을 입증하였다.

1. 서론

소프트웨어 개발에 OMG(Object Management Group)의 MDA(Model-Driven Architecture)를 적용하는 것은 모델로부터 자동으로 코드를 생성하여 소프트웨어 생산성이 증가하고, 하나의 모델로부터 다양한 에이전트 플랫폼에 맞게 변환이 가능하기 때문에 이식성이 향상된다. 모델을 기반으로 하기 때문에 유지보수 역시 쉬워진다[6]. 이런 이유로 소프트웨어 개발에 MDA 를 적용하는 사례가 늘어나고 있다.

에이전트 시스템은 자율적으로 상황을 인식하고 판단하는 지능적인 컴포넌트이다. 그리고 다중 에이전트 기반 시스템은 다른 에이전트들과의 통신을 통해서 서로 협력하여 목표를 달성하는 에이전트들의 집합이다[7]. 다중 에이전트 기반 시스템은 유비쿼터스 환경에서 다양한 환경정보를 인식하고 판단하는 자율적인 서비스를 제공하기 위해 사용된다.

최근에 다중 에이전트 기반 시스템 개발에 MDA 를 적용시킨 사례[4]에서는 에이전트 시스템의 플랫폼 특징적인 요소를 고려하지 않았다. 또한 모델의 표현 방법을 자체적인 표현방법을 사용 하였으며 표준적인 모델 변환 규칙을 사용하지 않았다.

다중 에이전트 기반 시스템은 다양한 환경에서 동작할 수 있기 때문에 각 시스템에 맞는 에이전트 플랫폼상에서 동작해야 한다. 에이전트 플랫폼을 사용함으로써 개발자는 자신이 원하는 플랫폼에서 에이전트를 쉽게 구현할 수 있다. 특히 FIPA(Foundation for Intelligent Physical Agent)에서는 에이전트 플랫폼을 위한 표준안[8]을 제시했는데, 여기에는 에이전트 플랫폼이 가져야 하는 일반적인 기능과 구조를 설명하고 있다. 하지만 FIPA 표준을 기반으로 다양한 종류의 에이전트 플랫폼이 제공되고 있으며 각각의 에이전트 플랫폼은 다중 에이전트 기반 시스템을 개발하는 개발자에게 자신들만의 고유한 라이브러리를 제공하기 때문에 개발자는 목표로 하는 에이전트 플랫폼에 관한 지식을 가지고 있어야 한다.

MDA 는 시스템의 동작환경에 독립적인 모델을 정의하고 목표한 플랫폼에 의존적인 모델로 변환한다. 플랫폼 의존적인 모델은 다시 프로그래밍 코드로 변

* 본 연구는 교육인적자원부에서 주관하는 2 단계 BK21 사업의 정보기술 분야 밀착형 산학협력 인재양성 사업에 의해 수행하였음.

환될 수 있다. MDA 의 이런 특징은 각기 다른 에이전트 플랫폼 고유의 라이브러리를 기반으로 시스템의 소스코드를 생성시킬 수 있다.

본 논문에서 다중 에이전트 기반 시스템의 공통적인 요소를 UML 모델로 표현하는 방법을 정의하고 에이전트 플랫폼이 고려된 모델 변환 규칙을 정의하였다. 또한 이 모델과 규칙을 OMG 에서 표준화한 방식으로 MDA 에 적용하는 방법에 대해 제안한다.

2. 관련연구

MDA 는 모델간의 변환을 통하여 소프트웨어 개발을 수행하는 접근 방법이다. MDA 는 이를 통해 이식성, 상호 운용성 그리고 재사용성을 향상시킬 수 있다.[1] MDA 에서는 다음의 4 가지의 접근방법을 제공한다. 1) 시스템을 플랫폼 독립적으로 명시한다. 2) 플랫폼을 정의한다. 3) 시스템을 위해 특정한 플랫폼을 선택한다. 4) 명시된 시스템을 특정 플랫폼에 맞게 변환한다.[1] MDA 는 이런 작업을 수행하기 위한 기반 기술의 집합이다.

MDA 에서는 세 가지 모델을 정의하고 있다. CIM(Computation Independent Model)은 시스템의 요구사항을 표현한다. 그리고 데이터 프로세스에 관한 정보를 숨기고 시스템이 하고자 하는 것을 명확히 하기 위한 관점을 제공한다. CIM 은 도메인 모델이라고 불리기도 하며 도메인 전문가가 주로 사용하는 모델이다.

PIM(Platform Independent Model)은 플랫폼 독립적인 관점을 제공하는 모델이다. 플랫폼이란 인터페이스나 패턴을 통해서 기능의 집합을 제공하는 모든 서브시스템이나 기술이다.[1] 따라서 PIM 에서는 대상 시스템이 플랫폼 특징적인 기술이 포함되지 않은 플랫폼 중립적인 표현법을 사용한다.

PSM(Platform Specific Model)은 모델 변환의 목표가 되는 모델로서 시스템이 최종적으로 동작할 플랫폼에 특징적인 기술이 포함된 관점을 제공한다. 예를 들면 CORBA Component Model 의 PSM 은 CORBA 플랫폼을 사용하기 위해 나타나는 구성요소를 모델링 하고 이 모델로부터 CORBA 코드를 생성할 수 있다.

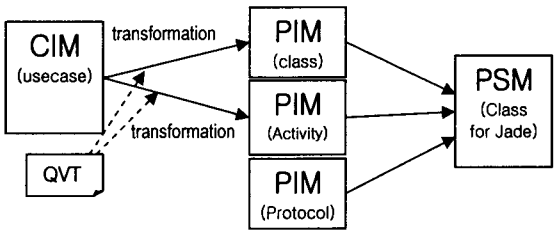
MDA 는 목표한 PSM 을 얻기 위해 CIM 이나 PIM 으로부터 모델 변환을 시작한다. QVT(Query, View and Transformation)는 메타모델(MOF[2])을 기반으로 한 범용 모델 변환 언어로써 MDA 의 모델 변환을 수행할 수 있다[3]. OMG 에서 표준화한 이 모델 변환 언어는 OMG 의 OCL(Object Constraint Language)[5]을 포함하여 모델 변환을 형식적으로 기술이 가능하다. QVT 는 크게 세 부분으로 구성되어 있다. Query 는 대상모델의 특정 구성요소를 선택하는 방법을 제공하고 View 는 목표모델의 표현을, 그리고 Transformation 은 두 모델 사이의 변환규칙을 기술한다.

[4]에서는 에이전트 시스템 개발에 MDA 를 적용하는 방법을 제안했다. 이 연구에서는 에이전트들을 다중 에이전트 기반 시스템에서의 역할별로 구분하여 에이전트 Society 를 구성했다. 그리고 각 에이전트의 행위를 표현하기 위해 행위를 하나씩 추가하는 플러그-인 방식의 모델링을 수행한다. 그러나 이 연구는 제안 시스템이 복잡해서 완전히 이해하고 있는 개발자만이 사용할 수 있고 XML 기반의 자체적인 모델 표현 방식을 사용하여 UML 같은 일반적인 목적의 시스템 표현 방법에 익숙한 개발자는 쉽게 적용하기 힘들뿐만 아니라 일반 모델링 도구에서 사용할 수 없기 때문에 모델의 공유가 힘들다. 마지막으로 에이전트 플랫폼에 대한 고려가 없어서 FIPA 에서 제안하는 에이전트 플랫폼을 기반으로 동작하는 에이전트 시스템에 적용할 수 없다.

본 논문에서 우리는 다중 에이전트 기반 시스템을 개발하기 위해 각 모델을 UML 을 기반으로 표현현하는 방법을 제안하고, 에이전트 플랫폼이 고려된 모델 변환 규칙을 QVT 로 기술하는 접근법을 제안한다.

3. 제안 시스템

본 장에서 우리는 UML 로 표현된 에이전트 모델로부터 QVT 를 사용하여 모델 변환을 실시하는 MDA 접근법을 사용한다. 제안 시스템은 다음 두 가지 특성을 가진다. 1)에이전트 플랫폼 독립적인 모델과 플랫폼 특징적인 모델의 표현 방법을 정의하고 2)QVT 를 통해 제안된 모델들 사이의 변환 규칙을 기술한다.

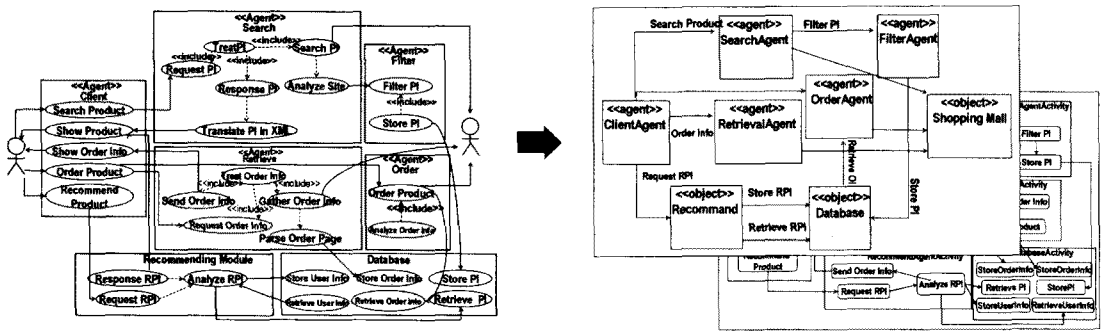


(그림 1) 에이전트 시스템을 위한 MDA 프로세스

본 방법론에서는 CIM 을 시작으로 PIM 을 거쳐서 최종적인 PSM 을 얻어내는 기본적인 모델 변환을 사용한다. (그림 1)에서는 모델의 변환 과정을 보여주고 있다. 하나의 CIM 은 다중 에이전트 기반 시스템의 아키텍처를 위한 클래스 다이어그램과 에이전트의 행위를 나타내는 활동 다이어그램으로 변환된다. 여기에 개발자는 에이전트의 프로토콜을 기술해 주는 시퀀스 다이어그램을 추가해 주고 나머지 두 모델에 대한 수정 작업을 한다. 이렇게 수정된 세 가지의 다이어그램을 조합해서 최종 PSM 을 만들어 내게 된다. 각 모델의 변환은 QVT 에 의해 기술되어 QVT 와 UML 을 다룰 수 있는 모델링 도구에 의해 변환이 수행된다.

제안 시스템을 설명하기 위해 전 연구에서 개발했던 다중 에이전트 기반의 전자 상거래 시스템의 시나리오를 통해서 프로세스를 표현하고자 한다.

여기서 대상으로 하는 전자 상거래 시스템[7]은 에이전트를 통해 현존하는 다양한 전자 상거래 웹 사이트를 분석하여 상품 검색을 수행할 수 있다. 또한 고객이 원하는 물품을 보여주고 이와 비슷한 제품들을 고객에게 추천하고 고객의 주문정보를 통해 해당 상



(그림 2) CIM 과 PIM

을 고객에게 추천하고 고객의 주문정보를 통해 해당 상 거래 시스템에 상품을 주문하는 일을 수행한다.

이 전자 상거래 시스템에 대한 CIM 을 작성하면 (그림 2)와 같다. 웹 사이트를 분석하여 상품을 검색하는 작업을 수행하는 Search 에이전트가 존재하며 이 에이전트는 검색 결과를 XML 형태로 고객에게 전달한다. 또한 검색된 제품 정보는 Filter 에이전트에 의해 페이지의 필요 없는 정보를 제거하고 데이터베이스에 제품 정보를 저장한다.

Retrieve 에이전트는 사이트로부터 제품 주문에 대한 정보를 수집하며 이를 바탕으로 고객이 주문을 하면 Order Agent 가 해당 상거래 시스템에 상품을 주문한다. Recommending Module 은 상품을 추천하는 개인화 서비스를 제공하며 전체 시스템에서 수집하고 분석한 정보는 Database 에 저장된다.

CIM 은 최초의 유스케이스 다이어그램으로 작성하며 이 모델에는 에이전트를 식별할 수 있는 'agent' 스테레오타입을 정해 주어야 한다. PIM 에서는 에이전트 시스템의 구조와 행위 그리고 프로토콜을 정의한다. 에이전트의 구조는 클래스 다이어그램으로 표현되며 행위는 활동 다이어그램으로 표현한다. 그리고 프로토콜은 시퀀스 다이어그램으로 표현한다. 본 시나리오를 바탕으로 만들어진 (그림 2)의 CIM 은 (그림 3)에서 표현된 QVT 에 의해 클래스 다이어그램과 활동 다이어그램으로 표현되는 PIM 으로 변환된다. 변환된 결과는 (그림 2) 에 표현되어 있다.

다음은 CIM 에서 PIM 으로 변환하는 모델 변환 규칙의 일부분이다.

[step1] 시스템을 구분하는 'classifier'는 클래스로 변환한다.
[step2] 만일 'classifier'가 'agent' 스테레오타입을 가지면 변환된 클래스도 'agent' 스테레오타입을 가진다.
[step3] Usecase 사이의 관계는 해당 Usecase 를 가지고 있는 'Classifier'가 변환된 클래스들의 관계로 변환한다.
[step4] Class Diagram 의 association 의 이름은 변환 이전의 association 의 target 의 이름으로 한다.
[step5] 각 association 마다 하나의 sequence diagram 을 생성한다.
[step6] 각 에이전트 마다 하나의 activity diagram 을 생성한다.

step1 과 step2 를 QVT 로 표현하면 (그림 3)과 같다. OCL 을 이용해서 각 클래스와 관계를 나타내는 변수에 CIM 의 모델을 선택하고 mapping 을 통해 PIM 으로 변환되는 과정을 보여주고 있다..

변환된 PIM 은 에이전트와 에이전트가 아닌 일반 객체를 구분하고 에이전트 사이의 통신을 관계를 통해서 표현한다. 개발자는 여기에 표현된 관계에 프로토콜 이름을 추가해 준다. 프로토콜 중에는 FIPA 표준에서 정한 11 개의 Agent Interaction Protocol 이 포함된다. 이 프로토콜은 에이전트가 통신할 때 사용할 수 있는 메시지의 순서를 시퀀스 다이어그램으로 표현하고 있다. 개발자가 PIM 의 클래스 다이어그램에 Interaction Protocol 을 정해 주면 이에 대한 시퀀스 다이어그램을 자동으로 생성해 준다. 하지만 만일 FIPA 에서 정하지 않은 독자적인 프로토콜을 사용하고자 할 때에는 시퀀스 다이어그램을 개발자가 직접 작성해 주어야 한다.

```
init{
  var classmembers : OrderedSet(uml::kernel::PackageableElement) :=
    ClassesFromModel(model)->asOrderedSet();
  var assocmembers : OrderedSet(uml::kernel::PackageableElement) :=
    assocFromModel(model)->asOrderedSet(); }
object{ ownedMembers:=classmembers -> union(assocMembers); }
mapping classifierToClass(in input: uml20::usecases::UseCaseClassifier):
uml20::classes::Class{
    object{
        name := if input.stereotypes = OrderedSet{ 'agent' }
                then input.name+'agent'
                else input.name endif;
        stereotypes := if input.stereotypes = OrderedSet{ 'agent' }
                then OrderedSet{ 'agent' }
                else OrderedSet{ 'object' } endif;
} }
```

(그림 3) CIM 에서 PIM 으로의 QVT

PSM 에서는 PIM 에서 정의된 세 가지 모델을 취합하여 플랫폼에서 바로 사용 가능한 클래스 다이어그램을 만들어 낸다. 이 때 클래스 다이어그램은 다중 에이전트의 구조와 어떤 에이전트와 통신해야 하는 지 등을 통해 에이전트가 통신하는 대상을 설정할 수 있고, 활동 다이어그램을 통해서 각 에이전트의 행위를 나타내는 클래스를 구성할 수 있다. 마지막으로 외부 시스템이나 에이전트와 통신하는 행위에 대해서 시퀀스 다이어그램을 참조하여 외부 개체와 통신하는 프로토콜을 위한 클래스를 만들 수 있다. 여기서는 PSM 모델의 대상 플랫폼은 JADE 를 사용하였다. JADE 는 JAVA 를 기반으로 한 다중 에이전트

<표 1> 프로세스 단계별 자동화 및 비자동화 정도

	요구사항 분석	시스템 분석	시스템 설계	세부 설계
자동화		- 에이전트 아키텍처 생성 - 활동 다이어그램 생성 - 프로토콜 생성	- 행위에 관련된 클래스와 그 관계 - 프로토콜에 관련된 클래스 생성	- 기본코드 생성
비자동화	- 초기모델 생성 - 에이전트 식별	- 사용자 정의 행위와 프로토콜 추가 - 통신 프로토콜 결정	- 구현상 추가적인 클래스 다이어그램	- 알고리즘 코드

플랫폼으로써 FIPA 에서 권고하는 에이전트 플랫폼의 기능을 만족하고 있다. 세 가지의 다이어그램은 다음 규칙에 의해 PSM 으로 변환된다. 여기서는 지면의 제약으로 변환규칙 중 일부분을 표현하고 있다.

[step1] 'agent'스테레오타입을 가진 클래스는 하나의 에이전트 클래스로 변환하고 CompositeBehaviour 클래스를 가진다.

[step2] activity diagram 의 각 activity 는 같은 이름의 behavior 클래스로 변환되고 step1 에서 생성한 Behaviour 의 서브클래스로 포함된다.

[step3] activity diagram 의 분기점은 분기하는 Behaviour 내부에 분기점을 만든다.

[step4] 하나의 sequence diagram 은 하나의 protocol 클래스로 만들어 지며, 프로토콜 내에서 하나의 메시지 전달은 send 및 receive 를 포함하는 하나의 메소드로 변환된다.

다른 에이전트 플랫폼을 대상으로 한 모델 변환 역시 해당 플랫폼에 대한 지식을 가진 사람이 QVT 로 모델 변환을 작성하는 것은 UML 메타모델에 대한 이해만 있다면 큰 무리 없이 작성할 수 있다.

4. 평가

본 장에서는 전자 상거래 시스템 개발의 결과물을 통해 제안한 모델의 유효성을 평가하고, MDA 기반의 개발 프로세스의 자동화를 평가한다.

소프트웨어 설계 및 구현의 프로세스는 기본적으로 요구사항 분석, 시스템 분석, 시스템 설계, 세부 설계의 4 가지 과정을 거친다[9]. <표 1>은 다중 에이전트 기반의 전자 상거래 시스템 구현 중 위 4 개의 단계에서의 자동화 되는 부분과 자동화 되지 않는 부분을 표현한다.

위 <표 1>에서 보는 바와 같이 본 논문에서 제안한 접근 방법은 시스템 분석단계, 시스템 설계단계, 세부 설계단계에서 모델을 상당부분 자동 생성하기 때문에 개발에 필요한 시간과 노력을 절감할 수 있다..

<표 2>는 결과 클래스에서 자동으로 생성된 클래스 개수와 개발자가 직접 추가해 주어야 했던 클래스 개수를 나타낸다. 클래스 개수가 개발자의 작업량과 완전히 비례하지는 않는다. 하지만 설계자의 입장에서 바라보면 설계의 완성도와 관련이 깊다.

<표 2> 모델 변환을 통해 생성된 클래스와 사용자가 추가한 클래스

생성된 클래스	추가한 클래스	수정한 클래스	전체 클래스
20	6	10	26

20 개의 클래스가 자동 변환에 의해서 생성되었다. 결과적으로 자동화에 의한 클래스 생성율은 76 퍼센트 정도 되었으며 이는 개발자가 초기 모델에서 더 많은 부분을 추가할수록 생성율은 높아질 것이다. 추가된 클래스는 Recommending Module 에서 사용자의 개인화 서비스를 담당하는 부분과 Search 에이전트의 웹 페이지 분석 등의 특별한 분석 알고리즘을 기술하는 부분이었다.

제안된 시스템은 다중 에이전트 기반 시스템 개발을 위한 프로세스 중 개발자의 단순 작업을 줄일 수 있었으며 그 결과물이 전체 클래스의 76 퍼센트를 생성할 수 있어서 다중 에이전트 기반 시스템 개발을 간소화 할 수 있었다.

5. 결론

본 논문에서는 에이전트 플랫폼을 고려한 MDA 기반 개발 방법론을 제안하였다. 모델 변환을 위해 CIM 으로부터 PIM 을 거쳐 PSM 을 표현할 수 있는 UML 기반의 설계방법을 제안하고 이 모델들 사이의 변환 규칙을 정의하였다. 제안한 MDA 프로세스는 다중 에이전트 기반의 전자 상거래 시스템을 통해서 평가 하였으며 그 결과 프로세스를 자동화 하고 프로젝트의 결과물을 다수 생성함으로써 다중 에이전트 기반 시스템 개발에 시간과 노력을 감소시켰다.

참고문헌

[1] Mariano Belaunde et. al, "MDA Guide Version 1.0.1", http://www.omg.org/docs/omg/03-06-01.pdf

[2] Object Management Group, "Meta Object Facility (MOF) Core Specification", http://www.omg.org/cgi-bin/doc?ptc/2004-10-15

[3] Stephen J. Mellor, Kendall Scott et. al, "MDA Distilled: Principles of Model-Driven Architecture", Addison Wesley 2004

[4] Denis Graanin, H. Lally Singh et. al, "Model-Driven Architecture for Agent-Based Systems", FAABS 2004, LNAI 3228, pp. 249–261, 2005.

[5] Object Management Group, "UML 2.0 OCL Specification", http://www.omg.org/docs/ptc/03-10-14.pdf

[6] anneke Kleppe, "MDA Explained The Model Driven Architecture : Practice and Promise", Addison-Wesley

[7] Eunseok Lee, Jionghua Jin, "A Next Generation Intelligent Mobile Commerce System", LNCS 3026, pp320-331 2004

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

[9] Sommerville, "Software Engineering 7th", Addison Wesley