MDA 모델 변환을 위한 메타모델 정의[1]

이승연[O] 신규상
한국전자통신연구원 컴퓨터.소프트웨어 연구소
{coral[O], gsshin}@etri.re.kr

MDA Model Transformation Using Metamodel

Seungyun Lee[O] Gyu-Sang Shin
Computer Software Technology Laboratory, ETRI

요 약

MDA는 시스템을 구현 플랫폼 및 구현 기술과 독립적으로 설계할 수 있도록 지원하고, 설계된 모델을 다양한 플랫폼으로 매핑할 수 있도록 하여 재사용 및 통합을 용이하게 한다. 하지만, 플랫폼에 독립적으로 설계된 모델을 다양한 플랫폼에 매핑하려면, 해당 플랫폼에 종속적인 모델로 매핑하여야 하고, 추가적으로 생성해야 하는 모델 정보를 파악하여 코드와 일대일 매핑될 수 있도록 하여야 한다. 본 논문은 플랫폼에 독립적인 설계모델을 플랫폼에 맞게 변환하기 위하여 MOF(Meta-Object Facility)에 기반하여 매핑규칙을 정의할 수 있도록 변환 메타모델을 정의하고 이를 EJB 도메인에 적용해본다.

1. 서 론

이종의 컴포넌트 개발 플랫폼 및 다양한 구현 기술의 발달로 이들간의 상호 운용성 및 통합을 지원하기 위하여 OMG에서 채택한 MDA(Model Driven Architecture)[1] 기반 소프트웨어 개발이 대두되고 있다. MDA의 핵심은 잘 정의된 비즈니스 독립적인 모델을 다양한 플랫폼에 맞도록 플랫폼 종속정인 모델로 자동 변환하고 그 변환된 모델을 통해서 코드를 자동 생성함으로써 소프트웨어의 생산성을 높이고 플랫폼 변화에 능동적으로 대처 할 수 있다는 것이다.

모델 변환은 MDA의 주요 개념중의 하나로, 플랫폼 독립 모델(PIM: Platform Independent Model)과 플랫폼 종속 모델(PSM: Platform Specific Model)로 분리된 모델간에 매핑 규칙을 정의하고, 정의된 규칙에 따라 플랫폼에 종속적인 모델을 변환한다. 변환된 초기 PSM은 추가 모델링을 통하여 개발 코드와 일대일 대응관계에 있도록 정의되어 자동으로 코드생성이 가능하다. 모델변환을 위하여 고려해야 할 사항은 플랫폼에 독립적으로 표현된 모델이 플랫폼 종속적인 모델로 변환될 때의 매핑 규칙과 플랫폼 종속적인 추가정보가 무엇인지 파악하는 것이다.

본 논문은 플랫폼에 종속적인 모델의 프로파일 정보를 바탕으로 매핑시 추가적으로 필요한 정보와 매핑 규칙이 무엇인지 파악하고, 이를 표현할 수 있는 일반적인 매핑 모델을 정의한다. 일반적인 매핑 모델을 기반으로 하여 특정 플랫폼 모델로의 변환모델을 작성할 수 있도록 MOF를 기반으로한 메타 모델을 정의하고, 이를 기반으로 특정 플랫폼에 맞는 변환규칙을 정의한다.

[1] 본 연구는 2002년 과학기술부 국가지정연구실 사업으로 수행되었음

본 논문은 다음과 같은 구성을 갖는다. 2장에서는 본 연구와 관련된 개념들을 소개한다. 3장에서는 모델 변환을 위한 메타모델을 정의하고 4장에서는 이를 EJB(Enterprise JavaBeans) 모델에 적용해본다. 5장에서는 결론을 맺는다.

2. 관련 연구

1. MDA(Model-Driven Architecture)

MDA는 이기종 컴퓨팅 환경간의 상호 운용성 및 통합을 지원하기 위하여 OMG에서 제안한 소프트웨어 개발 패러다임이다. MDA는 OMG의 UML[3], MOF[4], CWM[5]과 같은 모델링 표준에 기반하여, 시스템 모델에 대한 정형화된 아키텍쳐를 제안한다. MDA는 시스템의 기능에 대한 명세와 특정 기술 플랫폼상에 구현할 기능에 대한 명세를 구분하여 표현한다.

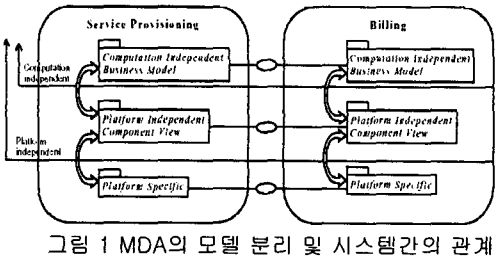

그림 1 MDA의 모델 분리 및 시스템간의 관계

<그림 1>에서 보는 바와 같이, MDA는 시스템 모델을 시스템 모델을 플랫폼에 독립적인 부분(PIM)과 플랫폼에 종속적인 부분(PSM)의 핵심 모델로 분리하며 모델간의 매핑을 통하여 일관성 있는 구조를 갖도록 한다. 시스템

나 파라미터 매핑과 같은 플랫폼 종속적인 정보들을 모델링하도록 한다. PIM과 PSM의 분리는 시스템의 핵심적인 구조는 같게 유지하면서 다양한 플랫폼에 적용하여 구현할 수 있도록 지원한다. 또한, 시스템간의 통합이 PIM 단계에서 명확하게 정의되며 이를 특정 플랫폼으로 매핑하여 실현될 수 있다.

3. 모델 변환을 위한 메타모델 정의

그림 2는 모델간의 변환을 수행하는 방법을 보여준다. 모델들간의 직접 변환은 변환코드를 복잡하게 만들고, 다른 플랫폼 모델로 변환할 때 재사용하기 어렵다. 따라서, 본 논문에서는 모델들을 표현하는 메타모델을 정의하고, 메타모델들 간의 변환모델을 정의한다. 정의된 메타모델을 이용하여, 각 플랫폼에 맞는 입력, 출력 모델과 변환규칙을 정의하고, 정의된 변환규칙은 변환엔진을 통하여 입력모델을 출력모델로 변환한다.

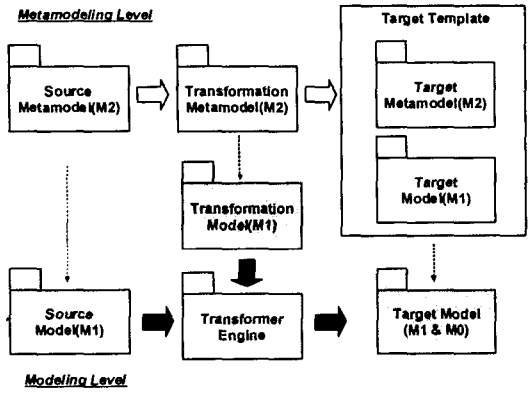

그림2 모델 변환 방법

변환의 대상이 되는 PIM과 PSM은 메타모델을 기반으로 정의되어 있다. 예를 들어, UML은 MOF를 기반으로 하여 시스템을 모델링할 수 있도록 정의되었으며, 구현 환경이나 플랫폼에 특화된 UML 프로파일 역시 MOF를 기반으로 UML 모델을 확장한 것이다. PIM과 PSM을 프로파일에 따라 메타모델을 정의하는 것처럼, 변환규칙에 대해서도 메타모델을 정의한다. 모델의 각 요소가 기반으로 하는 MOF 요소를 기준으로 이들간의 매핑을 지원하기 위하여 필요한 조건은 무엇인지 파악하고 이를 위한 메타모델을 정의해 두면, 해당 플랫폼으로 변환하고자 할 때의 규칙을 작성할 때 메타모델의 조건들을 이용하여 정의할 수 있다. 변환규칙을 정의할 수 있도록 하는 메타모델의 조건은 다음과 같다.

- 일대일 매핑: 입력 모델의 Classifier와 Feature는 출력 모델의 Classifier와 Feature로 매핑된다.
- 일대다 매핑: 입력 모델의 Classifier와 Feature는 출력 모델의 하나 이상의 Classifier와 Feature로 매핑된다.
- 모델 인스턴스 매핑: 일대일 매핑과 일대다 매핑은 MOF를 기반으로 정의한 메타모델 레벨에서 매핑과 실제 시스템에 적용된 모델 인스턴스와의 매핑을 모두 지원해야 한다.

메타모델 레벨에서 정의하는 변환 모델은 위의 세가지를 지원하도록 정의되어야 한다. 즉, 변환하고자 하는 모델들의 요소들간의 일대일매핑 및 일대다 매핑을 지원하고, 각 플랫폼에 종속적인 모델 인스턴스들의 매핑을 지원할 수 있어야 한다. 그림 3은 위의 세가지 조건을 고려하여 정의한 변환 규칙 메타모델이다.

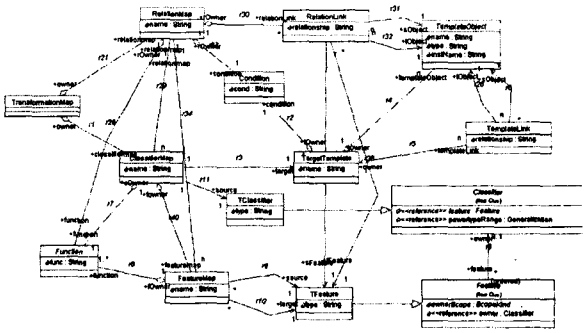

그림 3 변환 규칙 메타 모델

변환 규칙 메타 모델을 살펴보면 다음과 같다.

- Map: PIM을 PSM으로 매핑하는데 필요한 Map은 ClassifierMap, FeatureMap, RelationMap으로 정의한다. ClassifierMap은 매핑하고자 하는 모델의 타입이 Classifier 인 요소가 target 모델의 어떤 요소로 매핑되는지 정의하는 것이고, FeatureMap은 타입이 Feature인 요소의 매핑과 관련된 것이다. RelationMap은 Classifier와 Feature타입의 매핑이 이뤄진 후에, 이들간의 관계가 존재하면, 이 관계를 target 모델에서 매핑시키기 위한 Map으로 일정한 패턴을 따라 매핑된다.
- Function: 각 Map의 변환행위를 기술하는 부분으로 ASL(Action Semantics Language)를 이용하여 기술한다.
- Condition: 각 Map이 수행되기 위한 조건으로 OCL과 ASL을 이용하여 target 모델로 매핑되기 위한 조건이 만족되는지 테스트할 수 있도록 정의한다.
- Target 모델의 Template 표현: 매핑되는 target 모델은 메타모델에서 정의된 요소들의 매핑도 있을 수 있으나, 플랫폼에 종속적인 모델 인스턴스들을 생성하여 매핑해야하는 경우도 있다. 일대다 매핑 및 다양한 형태의 요소들을 표현할 수 있도록 메타모델을 정의한다. Target 모델의 요소들을 표현하기 위한 TemplateObject와 TemplateObject들간의 관계를 정의하는 TemplateLink로 이루어진 하나의 템플릿을 만들수 있도록 메타레벨에서 정의를 하고, 실제 특정 영역으로의 매핑을 적용할 때 필요한

4. 모델 변환 메타 모델의 적용 - EDOC to EJB

3장에서 제안한 모델 변환 메타 모델을 EDOC(Enterprise Distributed Object Computing)[2] 영역에서 모델링된 PIM을 EJB 모델로 변환하는 예제에 적용해 본다. EDOC 프로파일은 분산 시스템을 컴포넌트 기반으로 모델링하는 방법을 제안한 것으로, 프로세스 컴포넌트, 엔터티와 그들간의 관계를 표현한다. 그림 4는 회의 예약 시스템(Meeting Reservation System)의 요소인 Reservation 엔터티를 EDOC 프로파일에 기반하여 모델링한 것이다. <<Entity>>는 <<EntityData>>와 <<Key>>를 가지며 모두 MOF의 Classifier를 Base Element로 하고 있다.

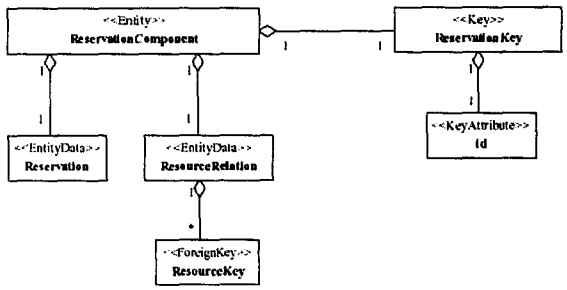

그림 4 회의 예약 시스템의 Reservation 엔터티 모델링

그림 4의 Reservation 엔터티는 EJB 플랫폼에 매핑될 때 엔터티빈으로 매핑될 수 있다. 엔터티빈은 기본적으로 2개의 인터페이스, 구현 클래스, 그리고 PrimaryKey 클래스의 4개의 클래스를 가지며 내부적으로 빈을 생성하고 접근하고 관리하는 메소드들을 가진다. 이는 메타모델 레벨에서 매핑하는 것이 아니라 모델 인스턴스 레벨에서 매핑이 되어야 하므로 모델 변환 메타모델의 Template 요소를 이용하여 target 영역의 매핑모델을 정의할 수 있다.

<<EntityData>>와 <<Key>>는 EJB 플랫폼의 Helper Class와 Primary Key Class로 매핑될 수 있다. 각 클래스의 에트리뷰트와 메소드는 그대로 매핑되며, EJB 플랫폼의 자바형식에 맞춰 표현된다. 매핑된 클래스들은 <<Entity>>에 매핑된 Template요소와 <<PrimaryKey>>나 그밖의 의존관계를 갖게되는데 이는 RelationMap을 통하여 정의된다. Entity와 Key의 관계를 Condition에 정의하고 이를 만족하면 RelationMap의 Function을 수행하도록 하여 target 모델인 EJB 영역에 EntityBean 클래스들과 Primary Key 클래스의 관계를 정의할 수 있다. 그림 5는 EDOC 프로파일의 Entity와 관련 요소를 EJB의 *EntityBean*으로 매핑하는 규칙을 그림 3의 메타모델을 이용하여 작성한 것이다.

Classifier, Feature들을 넣을 수 있도록 한다.

```
MClassifierMap iEntity2EJBBean =
    textent.getMClassifierMap().createMClassifierMap
    ("entity2entitybean");

MTClassifier iSource =
textent.getMTClassifier().createMTClassifier();
iSource.setName("Entity");

MTargetTemplate iTarget =
    textent.getMTargetTemplate().createMTargetTemplate
    ("entitybeanTemplate");

MTemplateObject entitybean =
    textent.getMTemplateObject().createMTemplateObject
    "EJBEntityBean","etri.mof.ejb.uml.javax.ejb.EEJBEnitityB
    ean","");
entitybean.setOwner(iTarget);

MTemplateObject entityimpl =
    textent.getMTemplateObject().createMTemplateObject
    "EJBImplementation", "etri.mof.ejb.uml.javax.ejb.EEJBIm
    plementation", "");
entityimpl.setOwner(iTarget);

MTemplateLink ownedel1 =
    textent.getMTemplateLink().createMTemplateLink
    ("OwnedElement");
ownedel1.setSObject(entityimpl);
ownedel1.setTObject(entitybean);
ownedel1.setOwner(iTarget);

iEntity2EJBBean.setSource(iSource);
iEntity2EJBBean.setTarget(iTarget);
```

그림 5 Entity를 EntityBean으로 매핑하는 규칙

5. 결론

본 논문은 플랫폼에 독립적인 모델을 다양한 플랫폼에 맞게 변환하기 위한 변환 메타 모델을 제안하고 이를 EJB 영역에 적용해보았다. 변환 메타모델을 이용하여 해당 플랫폼에 맞는 변환규칙을 정의하면, 이는 변환 엔진을 통하여 자동으로 PSM을 생성해 낼 수 있다. 앞으로 변환 메타모델의 구성 요소 중 Condition과 Function을 정형화된 기법으로 정의하고 변환 메타모델을 구현한 변환 엔진을 개발하여 다양한 플랫폼에 맞는 PSM을 자동생성할 수 있도록 연구할 것이다.

6. 참고문헌

[1] Object Management Group, " Model Driven Architecture," OMG document ormsc/2001-07-01.
[2] OMG, UML Profile for Enterprise Distributed Object Computing (EDOC), OMG Document: ptc/02-02-05, Feb. 2002
[3] OMG, Unified Modeling Language Specification 1.4, formal/01-09-67, Sept. 2001.
[4]OMG, Meta Object Facility Specification 1.4, April, 2002
[5] CWMPartners, Common Warehouse Metamodel (CWM) Specification, OMG Documents: ad/01-02-{01,02,03}, Feb. 2001.
[6] COMBINE, " Component-based Interoperable Enterprise System Development, " IST-1999-20893.