모델 재사용을 위한 모델변환 접근방법

특집 07

목 차

고승원 · 한정수 · 송영재
(경희대학교 · 백석대학교)

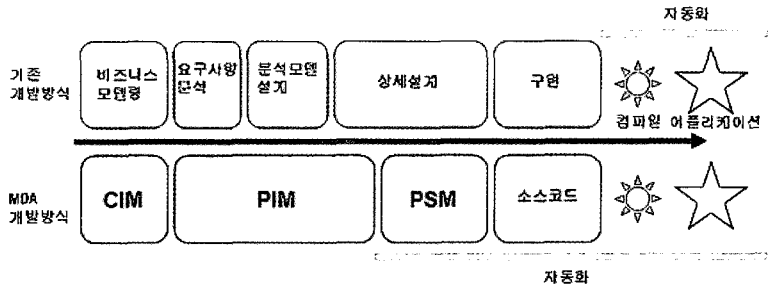
1. 서 론

소프트웨어 개발에 있어서 영원한 화두일 수 밖에 없는 재사용에 대한 그 대상은 함수단위의 모듈에서부터 클래스에서의 재사용, 그리고 컴포넌트 재사용을 지나 이제는 소프트웨어 개발 프로세스 중에서 분석단계나 설계단계에서 생성된 모델 차원에서의 재사용을 꾀하고 있다. 특히 이와 관련해서 한번 모델을 생성해놓으면 여러 특정 플랫폼에서 재사용이 가능한 MDA(Model Driven Architecture)개념을 바탕으로 모델변환에 대한 중요성이 인식되며, 이에 대한 다양한 형태의 모델변환방법들이 연구되고 있다[1]. 따라서 현재까지 진행되어온 모델변환기법들에 대해서 정리하고 이를 지원하는 지원도구들에 대해서 OMG MOF 2.0 QVT에서의 평가항목을 지원하고 있는지에 대해서 살펴본다. 특히, 모델To모델 변환기법을 통해서 모델변환이 다양한 도메인에서 활용될 수 있는 가능성을 가지고 있으며, 이미 이에 대한 활발한 연구가 진행되고 있다.

2. MDA 와 모델변환

기존 개발방식은 분석, 설계 작업을 통해 생성된 모델을 개발자가 직접 구현하는 데 반해 MDA 개발방식은 개발 작업의 중심이 플랫폼 독립적인 모델인 PIM(Platform Independent Model)을 구축하는 것에 맞춰져있고, 이후에는 MDA 도구를 사용해 PIM모델을 PSM모델(Platform Specific Model)로 변환 작업을 해서 최종 소스코드, 애플리케이션을 만들어내는 것이다. 따라서, (그림 1)에서와 같이 MDA 개발방식의 중심에는 PIM이 있고 MDA개발방식을 지원하는 도구와 모델변환작업이 애플리케이션을 만들어내는데 중요한 역할을 한다[6].

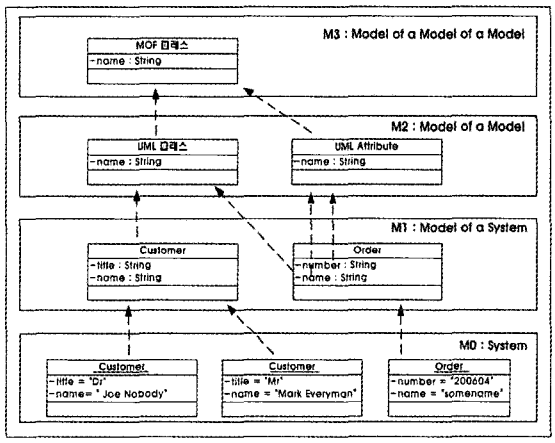
(그림 1)에서도 나타나듯이 기존 개발방식에 비해 MDA 개발방식의 차이점은 무엇보다 소프트웨어 개발 단계에서 생성되는 모델 중심의 개발이 진행된다는 데 있다. 기존 개발방식에서의 비즈니스 모델링에 해당되는 단계는 MDA 개발방식에서는 CIM(Common Information Model)



(그림 1) 기존 개발방식과 MDA 개발방식과의 차이점

로 나타내며, 소프트웨어 개발 단계의 요구분석과 분석모델에 대한 설계단계는 MDA개발방식에서는 PIM모델을 생성한다. 그러나 MDA 개발방식에서는 플랫폼과 독립적인 PIM모델 생성도 중요하지만 이렇게 생성된 모델에 대해서 플랫폼 독립적인 PSM 모델로의 변환이나 그 사이의 간격이 너무 클 때에는 중간모델(Intermediate Model)을 두어서 소프트웨어 개발 단계에서의 그 간격을 최소화 할 수 있다. 이와 같이 MDA 개발방식에서 모델변환을 지원하기 위해서는 CIM, PIM, PSM모델에 대한 메타모델의 생성 및 메타모델 요소에 대한 정의가 중요하다. 아래 (그림 2) 는 OMG 에서 MDA 개발방식에서 생성될 수 있는 모델에 대해서 모델정의를 위한 계층구조를 보여주고 있다. 이 OMG 4 Layer Architecture를 살펴보면 시스템에서의 실제 구현레벨에서 사용되는 인스턴스 모델인 M0 계층부터 우리가 UML모델로 모델링을 하는 M1 계층, 그리고 이 UML모델에 대한 메타모델계층인 M2계층에서는 모델요소를 정의하고 그 상위에 MOF(Meta-Object Facility)로 메타모델언어를 정의하고 있다. 또한 (그림 2)에서의 화살표는 해당 상위 클래스의 인스턴스가 된다는 의미이다[5].

다음 장에서는 모델변환 프레임워크, 즉 MDA 개발방식에서의 일반적인 모델변환구조와 현재 다양하게 연구가 진행되고 있는 모델변환접근방법에 대해서 알아본다.



(그림 2) OMG 4 Layer Architecture

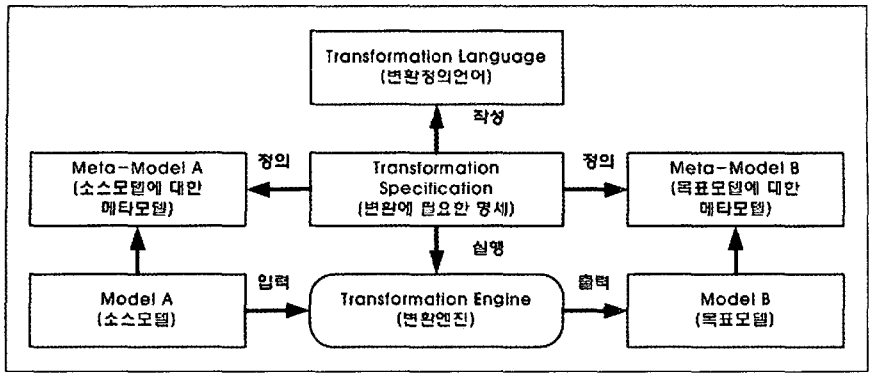
3. 모델변환 접근방법

3.1 모델변환 프레임워크

최근 2년 사이에 다양한 형태로 진행되고 있는 모델변환접근방법들은 기존의 대표적인 모델변환 형태인 UML 모델로부터 코드생성으로부터 벗어나 다양한 형태의 모델에서 모델로의 변환 연구가 활발하게 진행되고 있다[2].

(그림 3)에서의 MDA 개발방식에서의 기본적인 모델변환 패턴을 보기 전에 먼저, 모델변환에 필요한 요소들을 정의하면 아래와 같다[3].

1. 메타모델(Metamodel) 정의
   : 메타모델을 이용한 모델변환에서 필요한 소스모델 및 목표모델에 대한 메타모델 요소 정의



(그림 3) MDA 개발방식에서의 기본적인 모델변환 패턴

2. 메타언어(MOF) : 메타모델을 정의하기 위한 언어
3. 변환정의언어(Transformation Definition Language) : XSLT 나 ATL 과 같은 변환규칙을 정의하는 언어.

위와 같은 모델변환에 필요한 요소들을 바탕으로 (그림 3)에서 보면 먼저 어떠한 대상이 되는 모델에 대한 메타모델과 목표모델에 대한 메타모델을 정의하고, 두 모델 간에 변환규칙을 변환정의언어(Transformation Language)로 정의한다. 그리고 이 변환규칙을 구현한 모델변환엔진(Transformation Engine)을 통해서 소스모델을 목표모델로의 변환이 이루어진다[7].

여기에서 여러 변환정의언어에 대한 개발이나 패턴이나 그래프를 이용한 변환규칙에 대한 연구가 다양하게 진행되고 있다[1]. 모델변환 접근방법은 크게 분류하면 Model to Code 접근방법과 Model to Model 접근방법 두 가지로 구분된다.

3.2 Model to Code 접근방법

소스모델로부터 코드생성이 가능한 변환 메카니즘으로 기존의 변환도구들이 대표적으로 지원하던 UML 모델로부터 자바나 C++와 같은 소스 생성 기능을 제공한다.

메타언어를 정의하기 위한 언어인 MOF 를 통한 새로운 메타모델 정의를 하지 않고 이미 정의된 모델요소 타입을 표현하기 위해 JAVA 클래스로 정의하며, 새로운 모델요소 타입이 요구될 때, 이 JAVA 클래스를 서브클래싱해서 UML 모델을 나타내는 자바클래스로 모델변환을 하는 Visitor-based Approach 와 Template 이라고 부르는 소스모델의 내부 폼에 사용자가 어떠한 필드나 조건등을 입력함으로써 이에 대한 이벤트 API 함수가 소스코드를 생성하는 Logic 을 수행하는 Template-based Approach 가 있다.

3.3 Model to Model 접근방법

같은 메타모델이나 아니면 다른 메타모델로부터 정의되는 소스모델과 목표모델 사이에서의 모델변환 어프로치이다. 대부분의 현재 존재하는 MDA 도구는 단순히 Model to Code 변환, 즉 PIM 모델로부터 PSM모델(구현코드)로의 변환만 지원하며, 모델 간의 변환이 필요한 가장 큰 이유는 PIM 모델과 PSM 모델간의 추상화 차이가 클 때 Target PSM 모델을 곧바로 생성하는 것보다 중간에 Intermediate Model 을 생성하는 것이 보다 쉽게 PIM 모델과 PSM 모델간의 연결이 수월해진다. 기존의 Model to Model 접근방법은 아래의 네 가지가 있다.

3.3.1 Direct-manipulation Approach

내부 모델표현과 이를 제어하는 API 를 포함

하고, Object-oriented framework 처럼 보통 구현이 되며 모델변환을 지원하기 위한 추상클래스의 구현을 통해 최소한의 인프라스트럭처를 지원한다. 이 접근방법은 사용자가 직접 Java 와 같은 프로그래밍 언어를 통해서 변환규칙을 구현한다.

3.3.2 Relational Approach

소스모델과 목표모델 요소간의 Relation 을 Constraints 을 이용해서 명세한다. 이러한 명세는 mapping rule 을 포함하며, 바로 실행 및 생성을 지원하는 게 아니라 mapping rule 에 대한 수학적인 접근이 주된 아이디어이다.

3.3.3 Graph-transformation based Approach

소스모델과 목표모델의 요소를 정의하는 그래프를 생성하며 UML과 유사한 일종의 그래프 모델을 PSM 모델로 변환하는 사례가 프로세스 그래프와 플로우 그래프를 이용해서 그래프 변환을 수행한다. ATOM, GreAT, UMLX, BOTL, VIATRA와 같은 모델변환방법이 그 예이다.

3.3.4 Structure-driven Approach

소스모델과 목표모델의 메타모델 정의에 있어서 모델요소에 대한 Attributes 들과 Reference set, 즉 변환에 대응하는 API 함수 등에 대한 계층구조로 목표모델을 생성하여 모델변환을 수행한다. 앞선 Template based Approach에서의 template과 같이 주로 사용된다.

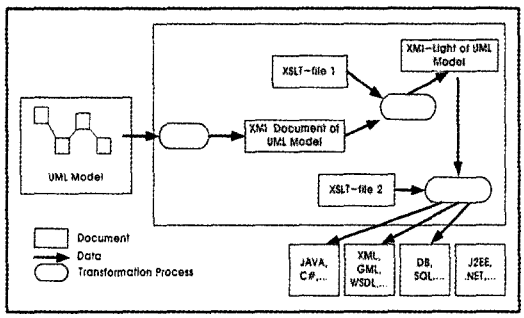
4. 모델변환지원도구

이 장에서는 위에서 알아본 다양한 모델변환 방법들을 가지고 이를 지원하기 위한 도구들에 대해서 알아본다. 또한 OMG MOF2.0 QVT (Query/View/Transformation)에서 제공하는 모델변환도구들에 대한 평가항목을 바탕으로 모델변환도구들에 대해서 장단점을 비교해본다.

4.1 UMT(UML Model Transformation)

COMBINE and ACEGIS 프로젝트에서 프로토타입으로 개발된 모델변환도구이며, 2004년 6월에 V.081버전까지 개발되었다[2]. 또한, UMT 도구에서는 메타모델에 대한 정의와 변환규칙을 명세하기 위해서 XML, XMI, XSLT 를 이용한 모델변환 및 코드생성을 지원하며, 모델변환엔진의 입력이 되는 소스모델로 UML모델을 사용하며, 이를 WSDL, XML Schema, Java Interface, IDL, EJB, 포맷으로 모델To코드 변환이 가능하다.

(그림 4)에서보면 소스모델인 UML 모델을 XMI 포맷으로 코드변환이 이루어지고, 이 XMI 도큐먼트와 모델변환규칙을 정의하고 있는 XSLT 파일1 과 같이 XMI-Light 라는 중간모델로 모델변환이 수행된다. 또한 XMI-Light모델과 플랫폼에 맞는 Java 나 XML 스키마 혹은 WSDL 등으로 코드변환을 하기 위해서 코드변환규칙을 정의하는 XSLT 파일2와 같이 모델To 코드 변환을 수행한다. 아래에 있는 내용은 소스모델과 중간모델, 목표모델 사이에서 이루어지는 프로세스에 대한 설명이다[9].



(그림 4) UMT의 모델변환 프로세스

1. Source model
   : UML 을 통해서 생성한 UML 소스모델에 대해서 모델변환을 위해 XMI 포맷으로 생성
2. Intermediate Model

: UMT 에서는 XMI-Light 라는 모델로 부르며 소스모델과 목표모델 사이에서 목표모델로 모델변환을 하기 위한 중간모델로 모델To 모델 변환이 가능하다.
UMT Transformer 에 의해서 자바와 같은 소스코드로의 변환도 지원한다.

3. Transformer
: Single-file Transformer와 Multi-file Transformer 두 가지 타입의 XSLT Transformer를 사용한다.

4.2 MTL(Model Transformation Language)

MTL(Model Transformation Language)은 모델변환엔진이면서 모델변환을 정의하는 언어이기도 하다. INRIA 연구소 Triskell 팀에서 BasicMTL 이라는 메타모델 언어개발과 더불어 이 언어를 지원하는 모델변환엔진을 개발하였다. MTL에서는 메타모델을 이용해서 여러 형태의 DSL(Domain Specific Language)로 작성된 소스모델에 대해서 모델변환이 가능하며 자바가상머신의 개념을 응용해서 메타모델 언어로 작성된 메타모델을 읽어서 자바 소스코드를 생성하며 마치 일종의 컴파일러 역할을 수행할 수 있게 개발하였다. MT Engine의 주요특징은 먼저, 모델변환엔진 자체에 대해서 객체지향 디자인을 바탕으로 해서 구현이 되었으며, 코드 재사용을 통해 변환엔진에 대한 다양한 형태로의 적용이 수월하다라는 장점이 있다. 또한 MTL은 다중상속을 지원하며, BasicMTL을 통해서 소스모델이나 목표모델에 대한 메타모델을 정의하고자 할 때, 메타모델 요소에 대한 정의가 가능하다. 그리고 MTL에서는 메타모델 저장소를 두어서 마치 UML 프로파일처럼 메타모델 요소로 정의된 메타모델을 저장소에 저장, 메타모델에 대한 재사용을 지원하고 있다. 메타모델에 대한 검색기능이나 패턴을 이용한 메타모델 합성기술등을 이용하면 모델재사용을 지원하는 좋은 모델변환도구로 개선될 수 있다.

또한, MTL에서는 UMT 변환도구와 마찬가지로 UML 소스모델은 XMI로 변환되어 기술되며, UML 모델과 Java 코드로의 변환 사이에서의 간격을 줄일 수 있는 UMT XMI-Light와 같은 중간모델(Intermediate Model)은 따로 사용하지 않는다. 따라서 변환규칙을 정의할 때 보다 쉽고 명확한 명세가 어렵고, 복잡한 변환규칙에 대해서는 정의할 수 없는 변환정의언어 자체에서 다른 변환정의언어에 비해서 미흡하다는 점이 단점이다. MTL에서의 모델변환을 수행하는 변환엔진, 즉 Transformer는 Java로 구현되었다.

4.3 ATL(The Atlas Transformation Language)

ATL(Atlas Transformation Langauge) 역시 MTL과 마찬가지로 변환정의언어이면서 변환엔진으로 MDA 프레임워크를 이용한 모델변환을 수행하며, 낭트대학 INRIA 연구소에서 개발하였다. GMT 프로젝트의 한 파트로써 ATL 프로젝트가 진행되었으며, ATL에서는 이클립스 플랫폼을 기반으로 UML모델을 ATL 변환규칙 정의언어를 이용해서 XMI 포맷으로 변환한 뒤 이를 다시 자바클래스 코드로 변환하는 프로세스를 통해서 모델변환을 수행한다.

ATL 은 무엇보다도 MTL이 가지고 있었던 변환규칙 정의가 보다 명확하고 이해하기 쉬우며 복잡한 변환규칙에 대한 정의가 가능하다, 또한 변환모델에 대해서 재사용 및 합성을 지원하는 장점을 가지고 있다. ATL로 기술되는 ATL 파일은 크게 메타모델의 이름 및 소스모델로부터 생성할 목표모델을 정의하는 Header와 입력이 되는 메타모델의 특정부분에 대한 명세가 이루어지는 Helper 그리고, 변환규칙을 정의하고 있는 Rule 의 세부분으로 구성되어 있다. 아래 코드는 이 중에서 Headers 및 Rule 에 대한 내용을 보여주고 있다. Headers 부분을 살펴보면 단순히 소스모델인 UML모델로부터 변환엔진을 거쳐서

생성될 목표모델, 즉 Java 코드로 변환된다는 것을 정의하고 있으며, Rule에서는 이를 위한 구체적인 변환규칙을 정의하고 있다. 즉, 입력이 되는 UML모델에서의 클래스 이름이 자바코드에서 클래스 이름으로 변환되고, 해당 클래스가 추상클래스인지, 아니면 공용클래스(Public class)인지, 또는 어떤 패키지 내에 포함되어 있는지를 Rule의 C2C에서는 UML클래스에서 자바클래스로 변환되는 매핑정의를 기술한다.

```
Headers
module UML2JAVA
create OUT : JAVA from IN: UML;

Rule C2C {
from e : UML!Class
to out : JAVA!JavaClass {
        name<-e.name,
        isAbstract<-e.isAbstract,
        isPublic<-e.isPublic(),
        package<-E.namespace
      }
   }
```

ATL에서의 모델변환의 입력이 되는 소스모델은 다른 변환도구들과 마찬가지로 UML모델이며, 이를 XMI모델로 변환하며, 변환규칙을 정의하는 ATL언어와 이 언어로 생성된 변환규칙 파일인 ATL파일을 적용해서 소스모델과 목표모델로의 변환을 수행한다.

4.4 모델변환지원도구 비교평가

MDA를 지원하는 여러 UML 변환도구에 대한 평가는 OMG MOF 2.0 QVT에서 제공하는 변환도구들에 대한 평가항목은 보면 아래와 같다[4].

1. Self containing : 어떻게 변환이 이루어지는가에 대한 이해 정도가 충분하게 도큐먼트가 지원되는가?
2. Scalability : 변환규칙에 대한 확장성
3. Simplicity : 변환규칙이 이해하기 쉽고 명확한가?
4. Bi-directional mapping : 양방향 모델변환 매핑을 지원하나?
5. Ease of Adoption : 제공하는 변환 메커니즘이 적용하기 쉬운가?
6. Provide an abstract syntax for the transformation language : 변환규칙 정의언어가 어느 정도의 추상화 레벨을 제공하나?
7. Support composition and reuse : 변환모델에서의 재사용 및 합성 지원여부
8. Provide complete example : 변환도구에서 사용예제 지원여부
9. Support complex transformation scenarios : 복잡한 변환규칙 시나리오에 대한 지원여부

위의 평가항목을 바탕으로 모델변환도구인 UMT, MTL, ATL에서의 평가항목에 대한 내용을 만족하는지 안하는지, 혹은 지원하는지에 대해서 평가하면 <표 1>과 같은 결과를 얻을 수 있다.

<표 1> UMT,MTL,ATL에 대한 평가결과

평가항목	UMT	MTL	ATL
변환규칙에 대한 이해와 관련된 도큐먼트 지원	Y	Y	Y
변환규칙에 대한 확장성	Y	Y	N
변환규칙에 대한 명확성	Y	N	Y
양방향 모델변환 매핑 지원	Y	N	N
제공하는 변환메커니즘의 적용성	Y	Y	Y
풍부한 조건명세 지원	Y	Y	Y
변환정의언어의 추상화레벨 지원	N	Y	Y
변환모델에 대한 재사용 및 합성 지원	N	N	Y
변환도구에서 사용예제 지원	Y	Y	Y
복잡한 변환규칙 지원	N	N	Y

<표 1>를 통해서 보면 변환규칙을 정의하고 있는 언어를 지원하는 ATL 이나 MTL 은 XMI나 XSLT를 통해서 변환규칙을 정의하는 UMT

에 비해 변환규칙 정의에 대한 추상화레벨을 제공하는 장점을 가지지만, 양방향 모델변환 매핑을 지원하지 않는다는 단점 역시 가지게 된다. 아마도 변환규칙언어 자체가 가지고 있는 문제점들은 점차 개선되어지고 있으며, 특히 ATL 변환도구의 경우에는 변환규칙언어에 대한 완성도 및 변환모델에서의 재사용 및 합성에 대한 지원 등을 통해 다른 변환도구에 비해서 보다 우수한 장점을 지녔다고 볼 수 있다. 따라서 추후에 모델변환정의언어를 개발한다던지, 모델변환지원도구를 개발하고자 할 때에는 위의 평가항목에서 나타난 결과를 통해서 변환언어 자체에서 추상화 지원 및 이해와 명확성, 적용성을 지원하느냐의 여부와 생성된 변환모델을 가지고 모델변환엔진에서 양방향 매핑을 지원하는지와 변환모델에 대한 재사용 및 합성에 대한 지원, 그리고 MTL 변환도구에서처럼 메타모델에 대한 저장소를 두어서 메타모델에 대한 재사용을 지원하는 여러 기능 등을 가진다면 보다 우수한 모델변환도구로 개발될 수 있다.

5. 결 론

소프트웨어 개발 시에 생성되는 모델에 대해서 재사용을 위한 모델변환은 소프트웨어 개발 프로세스 상에서 요구분석단계와 설계단계 사이의 간격이나 설계단계와 구현단계 사이에서의 간격을 최소화할 수 있는 소프트웨어 개발을 지원하기 위한 방법이다. 앞서 살펴본 바와 같이 이미 MDA 개발방식을 통해서 모델차원에서의 재사용 가능한 메타모델들과 이렇게 생성되는 모델에 대한 여러 형태의 모델변환연구는 최근 들어 활발히 진행되고 있으며, 본 논문에서는 모델변환접근방법에 대해서 알아보고 이를 지원하는 여러 도구들에 대해서 알아보았다. 특히, 모델변환도구에서의 평가항목을 바탕으로 앞으로 모델변환정의언어 및 모델변환도구를 개발하고자 할 때 UMT나 MTL 및 ATL 변환도구에서의 장점 등에서 나타난 바와 같이 변환규칙을 정의하는 언어에 대한 이해도와 명확성, 변환규칙에 대한 확장성이나 복잡한 변환규칙에 대한 적용성을 지원하는지 고려되어야 하며, 변환모델에 대한 재사용 및 합성 기능과 양방향 모델변환매핑의 지원, 그리고 모델 재사용을 위한 메타모델 저장소 등을 변환도구에서의 지원 등이 요구된다. 또한 다양한 도메인과 플랫폼을 가진 임베디드 소프트웨어 개발은 재사용이 필요하면서도 다양한 플랫폼에 적용하기 어려운 한계점을 가진다[8]. 이러한 임베디드 소프트웨어 개발에 모델변환을 적용한다면 여러 시뮬레이션 환경에서 그 대상이 되는 모델 생성 및 테스트모델로의 변환등의 모델변환기법 등에 적용 할 수 있는 형태로 연구가 계속되어야 할 것이다.

참고문헌

[1] Classification of Model Transformation Approaches, Krzytof Czarnecki and Simon Helsen, OOPSLA'03.

[2] Wein Evaluation of UML Model Transformation Tools. Wensheng Wang, Jun, 2005.

[3] A Model-Driven Approach to Non-Functional Analysis of Software Architectures, James Skene and Wolfgang Emmerich, ASE'03.

[4] MOF 2.0 Query / Views / Transformation RFP, OMG, 2002.

[5] MDA Explained, The model driven architecture: practice and promise, Kleppe,Warmer,Bast, Addison-Wesley, 2002.

[6] Model Driven Architecture, David S.

Frankel, OMG Press, 2002.
[7] Framework for model transformation and code generation, Jon Oldevik, Arnor Solberg, Brian Elvesater, Arne J.Berre, EDOC'02.
[8] Model-Integrated Development of Embedded Software, Gabor Karsal, Janos Sztipanovits, Akos Ledeczi, Ted Bapty, Proceeding of the IEEE Vol.91, No1, 2003.
[9] http://umt-qvt.sourceforge.net

저자약력

고 중 원

2002년 경희대학교 컴퓨터공학과(학사)
2004년 경희대학교 컴퓨터공학과(석사)
2006년-현재 경희대학교 컴퓨터공학과(박사과정)
관심분야 : MDA, CBD, 디자인패턴, 모델변환, 임베디드 시스템
이 메 일 : jwko@khu.ac.kr

송 영 재

1969년 인하대학교 전자공학과(학사)
1976년 일본 Keio대학교 전산학과(석사)
1980년 명지대학교 전산학과(박사)
1976년- 현재 경희대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어 재사용, CASE 도구
이 메 일 : yjsong@khu.ac.kr

한 정 수

1990년 경희대학교 전자계산공학과 (공학사)
1992년 경희대학교 전자계산공학과 (공학석사)
2000년 경희대학교 전자계산공학과 (공학박사)
2001년- 현재 백석대학교 정보통신학부학부 교수
관심분야 : CBD, 컴포넌트 관리
이 메 일 : jshan@bu.ac.kr